



Channel Islands

CALIFORNIA STATE UNIVERSITY

---

## **Malware Persistence Mechanisms**

A Thesis Presented to

The Faculty of the Computer Science Department

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

*Student Name:*

Zane GITTINS

*Advisor:*

Dr. Michael SOLTYS

July 2020

---

© Year

Zane Gittins

ALL RIGHTS RESERVED

APPROVED FOR MS IN COMPUTER SCIENCE



Nov 23, 2020

---

Advisor: Michael Soltys

Date

---

Reza Abdolee

Date

---

Socrates Frangis

Date

APPROVED FOR THE UNIVERSITY

---

Name

Date

## Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Malware Persistence Mechanisms, Masters Thesis

---

Title of Item

Malware, persistence, cybersecurity

---

3 to 5 keywords or phrases to describe the item

Zane Gittins

---

Author(s) Name (Print)



---

Author(s) Signature

Date

# Malware Persistence Mechanisms

Zane Gittins

November 23, 2020

## **Abstract**

In the public imagination Cybersecurity is very much about malware, even though malware constitutes only part of all the threats faced by Cybersecurity experts. However, malware is still one of the best methods to gain persistent access and control of a target system. There are many methods to deploy malware to a target system, a common method is a well socially-engineered phishing attack that deceives a user to gain a foothold on a system. Once the attacker gains a beachhead in the victim's network, it may be used to download additional payloads and exploit vulnerabilities, to gain more control and access within a network. Using malware as their foothold, attackers are able to to conduct reconnaissance, gather intelligence (e.g., exfiltration of intellectual property) or simply inflict damage or extortion (e.g., ransomware). All of this has to be done in a way that allows an attacker to retain access for as long as possible; the ability to do so is

called *persistence*, and this thesis examines some of the different techniques used by malware authors to accomplish persistence in an ever evolving landscape. In the second section of this thesis we propose an architecture for detecting malware persistence mechanisms, and give examples to detect the malware that we cover in the first section.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Contributions</b>	<b>4</b>
<b>3</b>	<b>Persistent malware</b>	<b>6</b>
3.1	Emotet . . . . .	7
3.2	Ocean Lotus Symantec DLL Hijacking . . . . .	16
3.3	TrickBot . . . . .	24
3.4	Ocean Lotus — Explorer-COM Hijack . . . . .	28
3.5	Agent Tesla . . . . .	35
3.6	Search Protect . . . . .	46
<b>4</b>	<b>Detection Architecture</b>	<b>52</b>
4.1	Windows Event Logs . . . . .	56
4.2	Windows Event Forwarding . . . . .	59
4.3	Apache Kafka . . . . .	61
4.4	Logstash . . . . .	62
4.5	Elasticsearch . . . . .	64
4.6	Kibana . . . . .	65
4.7	Elastalert . . . . .	67
4.8	Application of Architecture . . . . .	72



5 Conclusion	74
References	79

## List of Figures

1	Pyramid of Pain . . . . .	7
2	Steps that lead to an Emotet infection. . . . .	11
3	Vulnerable LoadLibraryW Call . . . . .	19
4	Functions Imported from DLL . . . . .	19
5	Library Exports . . . . .	22
6	DNS Request . . . . .	24
7	Process Monitor . . . . .	34
8	COM Export . . . . .	35
9	MZ file signature . . . . .	36
10	Steganography . . . . .	37
11	Agent Tesla Structure . . . . .	37
12	SetWindowsHookEx . . . . .	40
13	Hook Function . . . . .	40
14	vkCode . . . . .	41
15	Screen Capture . . . . .	41
16	Fakemail Agent Tesla . . . . .	44
17	Visualization Layer Diagram. . . . .	53
18	Detection Architecture Diagram. . . . .	56

# 1 Introduction

Malware authors continue to seek more advanced methods to maintain Persistence on target systems. We define *persistence* as the method by which malware survives a reboot of the victim operating system, and is a key element of attacks that require attackers to Pivot through a network to accomplish their objective. *Pivoting* is the process by which an attacker moves from a compromised system that they control to otherwise inaccessible systems [35]. Traditional methods for persistence are increasingly detected by defenders and anti-virus software. This thesis seeks to give a deep dive on a subset of persistence mechanisms used by malware. We start with traditional persistence mechanisms used by criminal elements, and then analyze more sophisticated persistence mechanisms believed to be utilized by nation state actors. These more advanced persistence mechanisms are harder for defenders to identify, and are less likely to be discovered by defenders. More obscure persistence methods do not appear in common tools used by defenders such as Autoruns. *Autoruns* is a utility that is apart of Microsoft's system internals toolkit, which covers a wide range of auto-start locations [23]. These techniques are generally deployed first by nation state actors, and later make their way into criminally operated malware. One example of this is the technique known as application shimming, with the first known public sample as Black Energy, which had application shimming capabilities added as early as April 2013 [29]. This technique was later discovered in the banking trojan

Gootkit in 2014, and Dridex in 2015. The terminology of *Advanced Persistent Threat* denotes a state-sponsored group with superior capabilities and funding that gains unauthorized access to a system and remains undetected for an extended period of time. These groups continue to attack their targets, even after failures, and often seek to deceive, deny, degrade, destroy, and disrupt their adversaries. The advances in malware development made by sophisticated groups often make their way into the hands of criminals, and then become commodities on the Internet available at little or no cost to less sophisticated actors. In the following sections we discuss malware samples and the persistence techniques they use. At the end of each section we map the persistence technique to the Mitre Attack framework. *Mitre Attack* is an industry standard knowledge base for attack tactics and techniques. Mitre Attack is organized in categories of tactics. The categories are initial access, execution, persistence, defense evasion, credential access, discovery, lateral movement, collection, command and control, exfiltration, and impact. The subject of this thesis is the techniques used by malware authors to achieve persistence, and therefore all map to the persistence tactic of Mitre Attack. As of 2020, Mitre Attack currently lists 63 persistence techniques [24], and there are undoubtedly more techniques not listed within Mitre’s framework. In this thesis We cover examples of three common techniques in the persistence category, and three more advanced techniques. In the second section of this thesis, we propose an architecture for detecting persistence mechanisms used by malware. First, we provide a generic architecture, and then

we discuss open source projects that can be used to satisfy this architecture. Lastly, we dicuss how this architecture can be used to detect the malware samples presented in the first section.

## 2 Contributions

This thesis provides analysis that focuses on the persistence mechanism of several malware samples. All samples except the Ocean Lotus COM hijacking sample have been analyzed previously, however these samples were not analyzed with a focus on their persistence mechanism. By highlighting malware persistence mechanisms, we believe that malware eradication efforts are made easier for defenders, as removing a persistence mechanism will stop the malware from surviving a reboot of the victim operating system. In some of the samples we analyzed, we provide custom tools and scripts to aid in analysis, these scripts are available on Github. For example, in the case of Trickbot, we provide a python script which uses regular expressions to unwind obfuscated JavaScript. The JavaScript is obfuscated by creating a mapping from numeric digits to characters, the python script scans an obfuscated JavaScript payload, creates a dictionary to unwind the encoding, and then applies this dictionary to the payload. In the analysis of AgentTesla, We provide a python program named Fakemail, which is a DNS and SMTP server used to capture the data exfiltrated by AgentTesla and identify command and control servers used by AgentTesla.

We discovered a malware sample that uses COM Hijacking for persistence using the malware sandbox Hybrid Analysis. This is important as it identifies a new persistence technique likely used by the Ocean Lotus threat group [6] that was not previously publically known to the security community.

Identifying this sample as possibly tied to Ocean Lotus allows defenders to sweep their environments for this technique, and possibly identify Ocean Lotus activity within their environments. Understanding threat groups allows organizations to prioritize defenses, and create strategies that seek to limit the chances of success for attackers.

In the last section, we propose an architecture for detecting the malware analyzed in the first section. This architecture makes use of open source and free to use projects, and is a strong starting point for organizations who wish to gain better optics into security events within their environments at little cost.

We seek to explain each topic thoroughly, and make each section approachable, however this thesis assumes some knowledge of various operating system and networking concepts. Specifically the malware analyzed in this thesis all targets the Windows system, so to get the most out of this thesis the reader should have a basic understanding of the Windows operating system and networking fundamentals. A high level understanding of DNS, SMTP, and TCP/IP will greatly help when reading this thesis. Several of the tools created to aid in the analysis of malware samples were written in Python, so having an understanding of the language will help when reading the code snippets within this thesis. Additionally, a high level understanding of assembly is useful in understanding some of the techniques discussed, such as web injections.

### 3 Persistent malware

In this section we are going to examine six representative samples of malware: Emotet, Ocean Lotus Symantec DLL Sideload, TrickBot, Ocean Lotus — Explorer-COM Hijack, Agent Tesla, and Search Protect. Search Protect is often classified as a potentially unwanted program, but we included it in our analysis because it was a good representative sample of a lesser used persistence technique. We start each section with a brief summary of the file type that carries the malware, together with its SHA256 signature. *SHA256 Signatures* are unique strings generated by the SHA256 hashing function, hashing functions are one-way functions that return an output of a set size given an arbitrary input. Hashing functions are commonly used in computer security because they allow for the unique identification of files. We include the hashes of the samples we analyzed so that our analysis may be verified, and so the reader may follow along in analyzing malware while reading this thesis. In the cases of the malware we analyzed, defenders using the SHA256 hash to identify and subsequently block the malware is a poor technique, this is because these samples are constantly updated by their authors, or were used in targeted attacks and likely compiled for each campaign. When developing detections it is best to look at the techniques, tactics, and procedures of the attackers and build detections around them that are not easily changed by malware authors. It is for this reason that We concentrate our analysis on the persistence mechanism, which allow defenders to better detect existing



malware as well as prevent malware infections in the future. This idea is well illustrated by David Bianco’s Pyramid of Pain, which illustrates that indicators such as file hashes and IP addresses are easy for an actor to change, however techniques and procedures are more difficult [1].

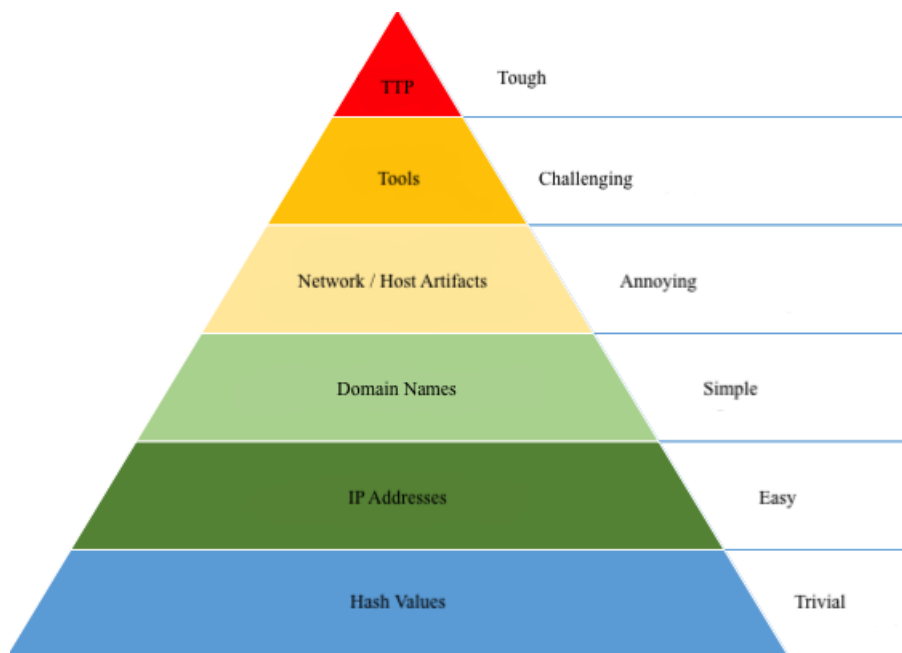


Figure 1: Pyramid of Pain

### 3.1 Emotet

Word Document

SHA256 94926C8520049F7EE51334D699DFC63EB3DB7DDD9C29946161689E1E33BFC0F5

Emotet, also known as Geodo, is an information collection malware that is used to install additional malware on victims. At the time of this writing,

Emotet targets solely Microsoft Windows systems. Emotet was first seen in 2014, and at that time its capabilities largely focused on stealing banking credentials by using Web Injections. *Web Injections* are a technique where malware intercepts Windows API functions called by the browser. The process of intercepting a function is called *hooking*. There are many forms of hooking, such as inline-hooking, and Import Address Table (IAT) hooking. *The Import Address Table* is a table loaded in memory which keeps track of the addresses of dynamic link libraries loaded within an executable [3]. An attacker can modify the address of a module in the Import Address Table, thereby intercepting a function call. In order to modify the Import Address Table attackers must calculate its position in memory, this can be done by inspecting the portable executable (PE) header of the application. The PE format is a file format specification used by executables and dynamic link libraries on Windows systems. Generally in the case of Web Injections, inline-hooking is used. *Inline-Hooking* is where a function in the victim process is patched, in memory, to have a JMP instruction at the beginning, that redirects to a new function that the attacker has written to the memory of the victim application. This allows an attacker to intercept function parameters, and modify the result of the function. A commonly hooked function to perform Web Injections on Windows is `HttpSendRequestA`. By intercepting this function, the malware can scan HTTP requests for sensitive data such as credit card numbers and login details, and send these to the operator of the malware, all occurring silently to the infected user.

In the original versions of Emotet, initial access to target systems was gained through email campaigns. These emails often contained the malicious executable, or contained a link that tempted victims to download and run a malicious executable [13]. The executable would manipulate victims' browsers to show fake content overlaid on top of webpages. This technique was used to steal credentials to sensitive accounts, such as username and password pairs for online banking websites.

Until 2015, Emotet was for sale on public forums [16], after which its sale became private. Since Emotet became private, its capabilities have shifted to support reconnaissance, and serve as a first stage in an infection chain that may lead to additional malware being installed on victim systems. Emotet has been observed deploying the trojans Dridex and Qakbot, information stealer Trickbot, as well as the Ryuk ransomware. These malware samples are operated by a wide range of actors, suggesting that Emotet operators sell the access they obtain to other criminals. This model has been referred to as access as a service, following the naming convention given to recent cloud technologies. *Ryuk* is a ransomware tailored to target corporate environments [12], due to the variance in ransom demands, CrowdStrike believes that Ryuk operators calculate the price of a ransom per victim. This suggests that Ryuk operators have maintained a presence in victim networks and have thoroughly enumerated networks to determine their cost. Outsourcing the initial access stage to Emotet operators allows Ryuk ransomware actors to focus their efforts on the ransomware itself, and is a concerning development

which shows that malware authors are becoming increasingly specialized. This specialization may be a quality that makes Emotet popular in criminal communities, not worrying about initial access allows criminals to focus on value generating strategies, such as ransomware, fraudulent transactions, and theft of valuable credentials.

At the time of this thesis, Emotet still uses malicious emails to gain initial access, however these emails now contain Microsoft Office documents with embedded Visual Basic macros. *Visual basic macros* allow users to extend Microsoft Office applications by using the Visual Basic programming language, which can be used to automate tasks and provides a rich feature set. However, these Macros are commonly used by attackers to execute malicious code when an office document is opened by a user. Using the Visual Basic functions `AutoOpen` and `Document_Open` allows for malicious code to execute when a Microsoft office document is opened. Microsoft Word supports five auto macros, `AutoExec`, `AutoOpen`, `AutoOpen`, `AutoClose`, and `AutoExit` [22]. These functions can be used to run Visual Basic code automatically, and are often abused by malware authors. For example, the `AutoOpen` function runs malicious code whenever the document is opened. In the case of this Emotet sample, the `Document_open` function is used, which causes visual basic code to be executed when the malicious document is opened by the victim, and the victim enables Macros for the document. The visual basic Macros used in this sample were heavily obfuscated, junk instructions were inserted and the names of variables were randomized, furthermore, execu-

tion flows through several unnecessary functions before reaching real code. Obfuscation can make it more difficult for automatic malware scanners to identify the sample as malicious, and also make it more difficult for malware analysts to determine the result of the Macros. Code to execute malicious PowerShell is embedded within several form variables within the document. Form variables can be accessed by Visual Basic and are often used by malware authors to store malicious content within a document. On execution of the macro, the contents of several form variables are appended together to create a call to `PowerShell.exe` with Base64 encoded PowerShell code as an argument. As soon as these Macros are executed, the PowerShell code is launched. *PowerShell* is a Windows command-line shell built on top of the .NET Framework, which accepts and returns .NET objects. In this Emotet loader visual basic Macros embedded within an Office document are used to execute PowerShell code on the victim system, and this PowerShell code downloads the next stage of the malware. Because the arguments for `PowerShell.exe` are constructed at runtime it makes it more difficult for static analysis engines to detect the visual basic code as malicious. Figure 18 displays the process.



Figure 2: Steps that lead to an Emotet infection.

`PowerShell.exe` can be called with the parameter `-EncodedCommand`

with Base64 encoded PowerShell code as the value. This is a powerful feature, but it is often abused by malware to hide the content of a payload within a Base64-encoded string. There are many techniques for hiding and obfuscating PowerShell. Some of these techniques include inserting escape characters, using PowerShell aliases, and abusing how each Cmdlet returns a .NET object to make it difficult to statically analyze the code. *Cmdlets* are commands in PowerShell that typically return a .NET object to another cmdlet in the pipeline. An example of a cmdlet is the `Get-ChildItem` cmdlet, which can be used to list the contents of a directory, much like the `ls` command on Unix. Inserting escape characters into PowerShell code does little to deter a determined defender, however it does make it more difficult for anti-virus programs, which may be searching for specific strings, from identifying a sample as malicious. PowerShell aliases allow for shortened versions of PowerShell cmdlets, for example the cmdlet `Invoke-Expression` can be shortened to `IEX`. Attackers may use expressions like `Get-Command` which returns a PowerShell object, and can be invoked, to further obfuscate scripts. These are only a few of the techniques that may be used to obfuscate PowerShell code. A defense against malicious scripts in Windows is AMSI, the Antimalware Scan Interface. AMSI allows applications to interface with Windows security data. This data includes PowerShell scripts, user access control prompts, memory scans, and more [17]. There are bypasses that malicious actors may use to disable AMSI, a perpetuation in the cat and mouse game of malware detection. One technique proposed by Cyberark disables

AMSI by loading a C# dll into the malicious process and modifying the memory of the `AmsiScanBuffer` function. This technique patches the length variable in `AmsiScanBuffer` to be 0 by xoring it with itself, this causes the `AmsiScanBuffer` to fail to detect malicious PowerShell code [11].

Emotet invokes PowerShell with a Base64-encoded payload. This PowerShell payload attempts to download the Emotet binary from five unique command and control servers. *Command and Control* servers are systems controlled by attackers which are used to communicate with malware on compromised systems. The use of five command and control servers provides this stage of the attack some resiliency; if any of these servers are successfully contacted, then the other servers under the attacker's control are not needed, and hence not contacted. Due to this, defenders only looking at network traffic may miss some of the command and control servers and fail to have a complete understanding of the attacker's infrastructure.

If PowerShell successfully downloads the next stage from a command and control server then a Windows executable named `377.exe` is saved with the contents returned by the attackers server. The name of this executable differs between samples, however in all cases we analyzed executables contained only numeric names. However, other Emotet executables have been reported with names containing a wide range of characters, this should not be used as any sort of indicator of an Emotet infection. In our sample, `377.exe` is started by PowerShell by making a call to the `.NET System.Diagnostics.Process` class. When run with the name `377.exe`, and administrative permissions,

the executable saves an exact copy of itself to the following location:

`C:\Users\TargetUserName\AppData\Local\monthmaker`

Where `TargetUserName` is the name of the infected user. The executable name varies between samples, however the directory is consistent in the samples that we analyzed. `377.exe` then starts the copy of itself called `monthmaker.exe` and deletes itself from the victim's disk. `monthmaker.exe` then creates a service for persistence. Windows Services allow for the creation of executables that run for an extended duration, and that can be set to run when a computer boots, similar to `systemd` on Unix. The service created by `monthmaker.exe` has the path where `monthmaker.exe` was saved, and has a description that is an exact copy of a legitimate Microsoft Windows description for a service, in one case the description was copied from Bitlocker's service. By using the same description as the legitimate Bitlocker service, the malicious service may be more difficult to spot by defenders. *Bitlocker* is a encryption feature in Microsoft Windows [5]. The service start type is set to automatic, which will cause `monthmaker.exe` to start each time the operating system does. There are five types of start types for Windows services, automatic, boot, disabled, manual, and system. The automatic start type starts the service at system startup. The boot type starts drivers, and is only valid for driver services. The disabled start type is for services that are inactive on the system. The manual start type allows users and applications to manually start the service, but the service will not



be started automatically by the system. Lastly, the system start type is for drivers started by the `IOInitSystem` function [18]. The most commonly used start type by malware is the automatic start, because it allows malware to start each time the system does, *Rootkits* are malware designed to give an attacker super level access, and thus run in the kernel, and in Windows may be implemented as services with the start type of boot or system. The name of the service created by our sample was `monthlymaker` however this varies from sample to sample. In the Mitre Attack Framework service creation is assigned the identifier T1050, and is a widely used technique by many malware strains.

Defenders can detect variants of Emotet that use services for persistence by monitoring for Windows event ID 7045 in the Windows system event log, on Windows 2008R2 and later systems. *Event logs* are files on Windows systems that store events that occur on a system. Events include security data, error logs, access logs, and many more [21]. Event ID 7045 is generated on a Windows `System` whenever a new service is installed on a system. Event ID 7045 contains the service name, image path, service type, and account name that the service runs under. The image path is the location of the executable that the service will run, in this case the image path stores the file path to the `monthlymaker.exe` executable. Another event ID that can be used to detect service creation is event ID 4697 in the security log. This event ID contains additional information compared to event ID 7045, namely, it contains the user account and SID that the service will run under. Event ID 4697

is not enabled by default, and will require `success` auditing to be enabled for the auditing category `Audit Security System Extension`. By carefully monitoring this persistence mechanism in their environments, defenders can create a reliable method for detecting variants of Emotet that use malicious service creation for persistence. Additionally, monitoring for irregular service creation is a strong method for identifying numerous other malware samples and malicious actors that make use of this technique. In order to effectively monitor service creations defender's require a strong understanding of their current environment, because service creation is common for legitimate programs. It will also be necessary for defenders to centralize event logs into one location, to run analytics and rules against. Common solutions to this are security information and event management systems (SIEM), there are a number a free and open source tools that can also aid defenderes in centralizing logs such as Elasticsearch, Winlogbeat, Apache Kafka, Kibana, Elastalert, Wazuh, Security Onion, Windows Event Forwarding, and many others. Combining these technologies, defenders can create a strong solution to centralize and alert on irregular service creation within their environments.

## 3.2 Ocean Lotus Symantec DLL Hijacking

Vulnerable Symantec RasTLS Application

SHA256 F9EBF6AEB3F0FB0C29BD8F3D652476CD1FE8BD9A0C11CB15C43DE33BBCE0BF68

Ocean Lotus Dynamic Link Library

SHA256 06DEC0082EAC094DC0B4B3DE8854F190F1D3112DADA0D414D9A085A0EE309199

Ocean Lotus is an advanced persistent threat, also known as SeaLotus, APT-C-00 and APT32, that has been followed closely by security firms such as ESET. Ocean Lotus is believed to be backed by the Vietnamese government, as their campaigns closely align with Vietnamese interests. Lotus targets have included companies, governments, and dissidents. Most known targets of Ocean Lotus have been located in and around Southeast Asia [7]. Ocean Lotus is also known for targeting foreign private sector companies with operations in Vietnam [2]. Known private sector targets have included companies from the United States, Germany, China, and the Philippines. Ocean Lotus makes heavy use of spear phishing emails in their campaigns, these emails have the goal of getting the victim to interact with a malicious attachment so that Ocean Lotus can infect the victim with one of their payloads. Attachments have included Microsoft office documents with malicious Macros, as well as executables with names that are written to appear like they are documents.

In 2018, the security firm ESET released a whitepaper [8] which described how Ocean Lotus deployed a backdoor which made use of a flaw in the Symantec Network Access Control application to maintain persistence and bypass security products. In the ESET report, Ocean Lotus delivers their payload through email, using names that entice their targets to open a malicious executable. One example of the initial executable from the ESET report is "Mi17 Technical issues – Phonesack Grp.exe", the name is crafted in a way

to convince the target that this is not an executable, but a document. When opened, these files execute their payload as well as open a document on the victim's computer. The title of one of the documents includes the word Mi17 [8]. The Mi17 is a Russian made helicopter, this suggests their targets may be interested in the Mi17, as phishing documents are usually crafted to be enticing to their targets. This backdoor has only been observed effecting targets running Microsoft Windows. The flaw used by Ocean Lotus to maintain persistence is a common technique, which makes use of the lack of validation when the Symantec Network Access Control application loads a dynamic link library. This vulnerability is a flaw overlooked by the developers of this version of the Symantec Network Access Control application. This technique is known as DLL-sideload, and has identifier T1073 in Mitre Attack. *Dynamic link libraries*, abbreviated as DLLs, are libraries that contain code that can be used by multiple programs simultaneously. *DLL-Sideload* is a technique where an attacker causes an unintended library to be loaded. By placing a malicious DLL with the same name that the Symantec product expects, and by exporting the same functions as the legitimate library, Ocean Lotus is able to force the Symantec Network Access Control application to load a malicious dynamic link library. It appears that Ocean Lotus deletes the legitimate symantec library, and replaces it with their own malicious library. Because this version of the Symantec application does not check the digital signature of the `RasTls.dll` library it loads, attackers are able to force the application to execute their malicious code. The vulnerability oc-

curs when the Symantec Application attempts to use `LoadLibraryW` to load the `RasTls.dll` with no validation:

```
local_4 = DAT_00410358 ^ (uint)local_414;  
GetWindowsDirectoryW(local_414, 0x104);  
pHVar1 = LoadLibraryW(L"RasTls.dll");  
*(HMODULE *) (unaff_ESI + 4) = pHVar1;
```

Figure 3: Vulnerable `LoadLibraryW` Call

When the Symantec Network Access Control application attempts to call one of the functions imported from the malicious library, the Ocean Lotus code executes. Instead of the legitimate function embedded within the real dynamic link library, a malicious function is imported and executed. The Symantec application can be seen importing these functions from the `RasTls.dll` here:

```
pVar2 = GetProcAddress(*(HMODULE *) (unaff_ESI + 4), "RasEapGetInfo");  
*(FARPROC *) (unaff_ESI + 0xc) = pVar2;  
if (pVar2 != (FARPROC) 0x0) {  
    pVar2 = GetProcAddress(*(HMODULE *) (unaff_ESI + 4), "RasEapFreeMemory");  
    *(FARPROC *) (unaff_ESI + 0x14) = pVar2;  
    if (pVar2 != (FARPROC) 0x0) {
```

Figure 4: Functions Imported from DLL

It does not appear that the malicious library maintains any of the original functionality of the original library, and therefore may cause instability in the Symantec application.

AppLocker is a tool created by Microsoft to enable application whitelisting. AppLocker can be used to block unknown applications from running on a system, and is a powerful feature for defending Windows systems [19].

For example, AppLocker can be configured to only allow executables signed by Microsoft and a few other trusted vendors to run on Windows systems. If code signed by Symantec is whitelisted in AppLocker then this technique could be used to bypass AppLocker as the malicious code will be running under the context of the legitimately signed Symantec executable. Furthermore, using side-loading as a method of persistence makes it more difficult for defenders to detect the malware, because it is executing in the context of a trusted software security vendor.

To ensure that the Symantec Network Access Control application starts each time the operating system reboots, the malware modifies a Registry key in the current user Registry hive:

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\run
```

*The Windows Registry* is a hierarchical database that stores settings used by the Windows operating system and Windows applications. The Windows Registry is composed of multiple registry hives, which are the top-level container in the registry. Each user on the system has an HKCU registry hive, which is stored in a file on disk called NTUSER.DAT, data is copied between NTUSER.DAT and the HKCU registry hive stored in memory. NTUSER.DAT is a hidden file located at:

```
C:\Users\USERNAME
```

Where USERNAME is the name of a user on the system. The HKCU registry hive is modifiable by the current user, without administrative priv-

ileges. There is another registry hive, HKLM, which contains system wide configurations. There is only one HKLM Registry hive per system, and it requires administrative privileges to modify. The registry key modified by Ocean Lotus, is used to start the Symantec application each time that an infected user logs in, and is in the HKCU registry hive. Modifying registry run keys has identifier T1060 in the Mitre Attack framework, and is a common technique used by malware samples. What makes Ocean Lotus' use of this technique unique, is that they are starting a legitimate application each time the user logs in, and abusing a vulnerability to subvert that application. The HKCU registry hive is writable without administrative permissions and will launch Symantec Endpoint Protection when the user who was infected logs on to the system. The value stored in this registry key is the path to where the Symantec application, `rastlsc.exe`, is located on disk:

```
C:\Users\Username\AppData\Roaming\Symantec Endpoint Protection\  
12.1.671.4971.104a\DeviceAssociationService\rastlsc.exe
```

Each time a user logs in, `rasltsc.exe` is started, and once started, it imports the malicious Ocean Lotus library. Then, when `rasltsc.exe` attempts to use one of the imported functions, malicious shellcode stored in a file `SysLog.bin` — located in the same directory — is executed by code located at the function imported from the malicious library. This technique functions because the malicious `rastls.dll` exports the same five functions that the legitimate library exports, and because the application does not

properly validate the imported library. Using the tool Ghidra, a decompiler and disassembler developed by the United States National Security Agency, we were able to find the exported functions of the Ocena Lotus library:

10001090	RasEapGetCredentials	JC	Payload
	RasEapFreeMemory		
10001010	JGE	Payload	
	RasEapInvokeInteractiveUI		
100011f0	JNP	Payload	
	RasEapGetInfo		
10001100	JNZ	Payload	
	RasEapCreateConnectionProperties2		
100010f0	JMP	Payload	

Figure 5: Library Exports

All exported functions from the malicious library call the same malicious function. *Shellcode* is position independent code, written in assembly language, which traditionally launches a shell, but may take any action. *Position independent* code executes properly despite not knowing its relative address. Typical assembly code does not have this restriction, and may require loading at a specific memory location to execute correctly. A shell is an interactive command line interpreter, an attacker may use this shell to conduct further stages of an attack on the victim system [33]. Shellcode need not end in a shell, it may be used to install malware on the victim system, or used to crash the target system. Shellcode is often used by exploits, but has gained traction in malware author communities because it can often be used to bypass endpoint protection products, such as traditional anti-virus and endpoint detection and response (EDR). One example of using shellcode to bypass anti-virus is a technique presented by Brian Fehrman of Black Hills



Information Security [9]. His technique uses the `installutil.exe` program to run shellcode when it executes malicious C# code. This can be used to bypass application whitelisting because `installutil.exe` is often a trusted program on Windows systems.

The shellcode used by Ocean Lotus is a backdoor which communicates over TCP port 25123 [8]. The shellcode has an interesting defense mechanism, the clearing of its File Signature. The shellcode makes use of function `RtlZeroMemory` to clear the MZ file signature of the Symantec application in memory. *RtlZeroMemory* takes a pointer to a block of memory, as well as a length, it then fills this block, up to the length, with zeros. *File signatures* are the bytes at the beginning of a file, used to identify a file. Windows executables and dynamic link libraries start with the two bytes, 0x4D 0x5A, which in ASCII is MZ. Windows executables are also known as PE files, and the MZ file signature is the first few bytes of a PE executable. Some security solutions scan memory for this file signature to identify Windows executables. Clearing the MZ file signature may prevent some security solutions from scanning the application in memory, and thus failing to identify the injected shellcode. Automatic Memory Dumping may also fail because of this defense mechanism [8]. *Memory dumping* is the process of writing a region of memory to disk. Automatic memory dumping scans memory for file signatures that correspond to executables, and dumps these executables to disk. An example of C++ code that can clear the MZ file signature can be found below, this is not the exact code used by Ocean Lotus, but is an

example of the MZ signature removal technique:

#### Listing 1: Clear MZ File Signature

```
VirtualProtect((LPVOID)peHeaderAddress, peHeaderSize, PAGE_READWRITE, oldProtect);
RtlZeroMemory((PVOID)peHeaderAddress, sizeofHeader);
VirtualProtect((LPVOID)peHeaderAddress, peHeaderSize, oldProtect, oldProtect);
```

First the protection of the MZ File Signature in memory is changed to `PAGE_READWRITE`, next the block of memory is filled with zeroes using `RtlZeroMemory`, lastly the block is reset to it's previous memory protections prior to the initial change.

To communicate, the malware chooses between one of three subdomains in its configuration and prepends a subdomain that is generated from the computer name of the victim system [8].

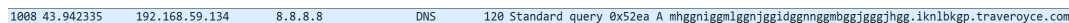


Figure 6: DNS Request

If this DNS request is successful than the malware attempts to communicate over TCP with a command and control server. The communication is encrypted using RC4, however the key is prepended to the data, and thus the communication is accessible with some basic scrutiny of the packets. *RC4* is a stream cipher, which is used to encrypt plaintext in one character chunks [14].

### 3.3 TrickBot

HTML Email Attachment

SHA256 7AEE90A79191DBB914C77D886C14D5BCAF217CE9C046B85407F69A6F9AB7BB73

Trickbot is a malware aimed at stealing valuable information from the targets that it infects. Trickbot is a *modular malware*, that is, the initial executable does not contain all of the malware's functionality, instead modules are downloaded from the Trickbot command and control server based on the data sent by the client. These modules typically perform specialized actions, such as stealing passwords from victim's browsers. This model has several advantages. First, it allows the malware authors to distribute new modules to clients when they see fit. Secondly, if a Trickbot module is detected by an anti-malware solution, only the module may be removed, leaving the primary Trickbot infection behind. Trickbot is well known for its ability to steal passwords saved in browsers, including Google Chrome (note that Google Chrome, when the user is not logged into their Google account, keeps all passwords in a sqlite database encrypted using a key that can be retrieved by making a Windows API call in the context of the current user), Firefox, as well as stealing saved information from Microsoft Outlook.

Trickbot samples have been observed using the following method to gain initial access. A malicious email is sent to targets, this email contains an `.html` attachment. If the user downloads and opens the `.html` attachment they are presented with a web page with a single line of text, that prompts them to download a Microsoft Word document. The Word document is embedded within the html page in Base64. Opening this word document results in embedded visual basic code being executed via a malicious macro,

the purpose of this code is to construct a JScript script, which is placed in the same directory as the word document. *JScript* is Microsoft's implementation of JavaScript, and can be executed natively on Windows systems. The VBA code ends by executing the JScript by calling Microsoft's Wscript tool. WScript also known as Windows Script Host, is a tool by Microsoft that allows users to execute scripts in a variety of languages.

The JScript used by Trickbot is heavily *obfuscated* — a common technique used by writers of malware —, however using Python code we were able to extract its contents; see the code snippet in Listing 2. Note that the obfuscation, although difficult for a human to parse, can be unwound with simple regular expressions, as show in the Python code.

Listing 2: Python script for de-obfuscating the Trickbot JScript

```
import re

def matches(regex , text):
    pattern = re.compile(regex)
    return pattern.search(text)

payload = "payload.js"
data = open(payload , "r")
payload_data = data.read()
data.close()

new_dict = {}

payload_data = payload_data.split("\n")
```

```

for line in payload_data:

    if matches("[0-9]+:\\\\_\\\\", line):

        print(line)

        data = int(re.search(r'\d+', line).group())

        val = re.findall(r'"([^\"]*)"', line)

        new_dict[data] = val

    elif matches("[0-9]+\\\\\\\\\\\\([0-9]+", line):

        data = map(int, re.findall(r'\d+', line))

        data = list(data)

        key = data[len(data)-1]

        print(new_dict[key])

        new_dict = {}

```

The Trickbot JScript attempts to download the next stage of the payload by making an HTTP GET request to a Trickbot command and control server. If the request is made with the proper parameters then Trickbot is downloaded to the target system. If the GET request does not contain the expected parameters then the request is rejected by the command and control server and the next stage of the malware is not downloaded. Trickbot copies itself to %APPDATA%\Roaming\mslibs\ and chooses a random lowercase alpha string for its name. This was the behaviour observed by the sample we analyzed, other variants of Trickbot may copy themselves to alternate locations, and use different file names. These therefore should not be used in writing detections,

as they can be easily modified by the malware authors. Detection rules should focus on techniques and not indicators of compromise which may be brittle and specific to a single variant of a broader malware family.

*Windows Scheduled Tasks* are a feature that allows for running a executable or script whenever a given trigger is met, this is similar to `cron` on Unix. Trickbot creates a scheduled task named `Ms Libraries` with two different triggers. The first trigger runs the trickbot executable every time the user logs on. The second trigger will run the executable every 9 minutes for the next 415 minutes on the day that the scheduled task was created. This is likely to make multiple attempts in case initial connection to the Trickbot command and control servers fail. Scheduled task creation has identifier T1053 in the Mitre Attack framework. When a scheduled task is created, event ID 4698 is logged in the security event log. This event ID contains information about the new scheduled task, some important fields for defenders are `Command` which contains the command that will be executed by the scheduled task, and `Enabled` which shows if the scheduled task is active. In our case, the `Command` field contained the path to the trickbot executable located at `%APPDATA%\Roaming\mslibs\`. Defenders that closely monitor scheduled tasks can identify Trickbot samples that make use of this persistence mechanism.

### 3.4 Ocean Lotus — Explorer-COM Hijack

COM Hijack Library

SHA256 C9CC360E5DEA70E62F1A6564B258962E72FCF808292FBB6861486356A5EF8BB9

By searching for the IP address 198.50.234.111 from the Ocean Lotus report by ESET in Hybrid Analysis, we were able to discover another sample that may be used by the Ocean Lotus group. This IP address was an indicator of compromise in the 2017 report by ESET on Ocean Lotus, and identified as a command and control server by Unit42 in 2019. *Indicators of compromise* are pieces of data that can be used to identify a specific group or malware. This sample was uploaded publicly in November of 2018, and therefore may be tied to Ocean Lotus. If this sample is not tied to Ocean Lotus it still serves as a clear example of a persistence technique that we have not yet discussed. *Hybrid Analysis* is a free to use malware sandbox, that runs a given executable, or opens a specified url inside of a virtual machine and records events that occur inside of the virtual machine for a specified timeframe. Hybrid Analysis is currently owned by the security firm CrowdStrike and provides rich search capabilities, including the ability to find malware samples using Yara rules. *Yara* is an open source tool backed by VirusTotal for classifying and identifying malware. Combining the in-depth features of Yara rules with the large number of malware in Hybrid Analysis, we were able to identify this sample. During the analysis of this sample the IP address 198.50.234.111 resolved to DNS name `ristineho.com` which in February of 2019, was identified as a command and control server for Kerndown by Palo Alto's Unit42 [30]. *Kerndown* is a downloader used by Ocean Lotus to install additional malware such as Cobalt Strike. *Cobalt Strike* (<https://github.com/cobaltstrike/cobalt-strike>):

`//www.cobaltstrike.com/`) is a publicly available command and control framework used by red teams to legally test the security of organizations, however it has also been used by actors in real attacks [30]. The Kerrdown sample we found communicating with `ristineho.com` was:

SHA256    860f165c2240f2a83eb30c412755e5a025e25961ce4633683f5bc22f6a24ddb6

Our contribution to the community is the identification of another sample communicating with the `ristineho.com` domain that is using COM Hijacking for persistence, as of June 2020, this is not a listed technique for Ocean Lotus in Mitre Attack or in Unit42’s post on Kerrdown. COM hijacking may be used by Ocean Lotus to persist the Kerrdown downloader, which could be used to inject a Cobalt Strike beacon into the memory of a victim host whenever the hijacked COM class is loaded. *Beacon* is the payload generated by Cobalt Strike, and is favored by red teams for its ability to create communication which can blend into normal traffic, using techniques such as DNS to relay commands. Cobalt Strike can use DNS to communicate to a command and control server by making unique DNS requests to a domain controlled by the operator, because the operator controls the authoritative name servers for the domain, all unique DNS requests are forced to be answered by the authoritative name server. The name server replies with an IP address which corresponds to the requested domain, the data that makes up this IP address is interpreted as a command by the Cobalt Strike beacon. For example, a reply of 0.0.0.0 indicates that there are no tasks for the



Beacon [25]. Note that the actor can change the Cobalt Strike reply for no tasks to any IP address by modifying the variable `dns_idle` in their command and control profile in Cobalt Strike. Additionally, actors can use the variable `dns_max_txt` to limit the size of TXT records, and `maxdns` to limit the length of DNS names [27]. Using these options to limit the size of TXT records and DNS name length will greatly increase the stealth of Beacon, however comes at the cost of the speed of data relayed to the beacon, as the smaller space in each DNS request means that less can be relayed to and from the command and control server. Tasks can be communicated to the beacon using different modes, DNS A record responses, DNS AAAA record responses, or TXT records [27]. DNS TXT records are the fastest, with DNS AAAA and A each being progressively slower. An example of this communication is a Cobalt Strike beacon with a command and control server that is authoritative for `malware.c2.com`. To communicate the beacon first crafts a DNS query to the authoritative server and prepends a unique identifier, for example `1234.malware.c2.com`[26], this DNS request is made to the local DNS server. Because the local DNS server does not know the answer to this query it asks a top level root CA. The root CA will not know the answer for `1234.malware.c2.com` so it replies to the local DNS server with a DNS server that is authoritative for `.com`. The `.com` dns server will not know the answer to the query, but will reply with the DNS server that is authoritative to `c2.com`. This server does not know the answer to the query, but replies with the DNS server authoritative for `malware.c2.com`, which is the Cobalt

Strike team server. The server authoritative for `malware.c2.com` knows the 4 byte answer to the DNS A record query, and replies to the local DNS server, the local DNS server forwards the response to the Cobalt Strike beacon, and thus is able to send data to the Cobalt Strike beacon. This covers communication from the team server to the beacon, but for the Cobalt Strike beacon to send data to the team server it generates long DNS requests (200+ characters) that only the authoritative server can respond to. Within these requests is the data that the beacon wishes to send to the team server. The maximum length of requests can be shortened, however this requires more DNS requests to be made to send data.

This malware sample makes use of a technique known as Component Object Model (COM) hijacking. The *Component Object Model* is a standard for creating platform independent software components, and is the basis of Microsoft's OLE and ActiveX technologies. COM servers provide access to COM objects through pointers to interfaces. One example of a COM class the reader may be familiar with is the `Wscript.Shell` class commonly used in Visual Basic. COM servers are implemented in the form of Windows dynamic link libraries. For a dynamic link library to be a valid COM server it must export two functions at a minimum, `DllGetClassObject`, and `DllCanUnloadNow`. `DllGetClassObject` is called when an application needs to access a COM object, the server which exports `DllGetClassObject` returns a pointer to the interface for the COM object. `DLLCanUnloadNow` is called when the application is no longer using any COM objects, and unloads the

COM server library. COM objects are loaded by using the Windows Registry. Specifically, a COM object is searched for by looking at the registry keys present in:

System Hive    HKLM:\\SOFTWARE\\Classes\\CLSID

User Hive      HKCU:\\SOFTWARE\\Classes\\CLSID

The DLL path for the COM server is stored within the registry key InProcServer32, which is a subkey of either the system or user hive. The user registry hive is queried before the system hive. This can pose a serious issue, if an attacker creates or modifies a user hive key then their COM server will be loaded instead of the legitimate COM server which may be located in the system hive. Because the user hive is modifiable without administrative privileges, by the current user, an attacker can cause malicious COM servers to be loaded. In the Hybrid Analysis report for this malware sample we noticed the registry value for a COM object was modified, the CLSID for this COM object is:

CLSID    0E5AAE11-A475-4C5B-AB00-C66DE400274E

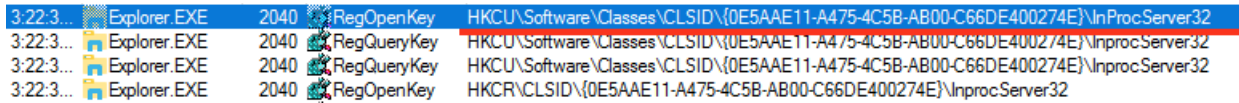
The Location of the legitimate registry key is HKLM:

HKLM:\\SOFTWARE\\Classes\\CLSID\\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}

The Location of the malicious registry key is HKCU:

HKCU:\\SOFTWARE\\Classes\\CLSID\\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}

The legitimate CLSID contains the registry value `%SystemRoot%\system32\Windows.Storage.dll` within the subkey `InProcServer32`. Which is a legitimate COM server used by Windows Explorer (`explorer.exe`). `Explorer.exe` is a core part of the Windows operating system, and it runs under each user interactively logged into a system. By creating a similar registry key in HKCU that contains a path to a malicious DLL, the malware is able to perform COM hijacking. This will cause the malware to be loaded any time an executable is launched from the task bar. By using the Windows system internals tool, Process Monitor, we were able to verify that Explorer attempts to load the COM server:



3:22:3...	Explorer.EXE	2040	RegOpenKey	HKCU\Software\Classes\CLSID\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}\InProcServer32
3:22:3...	Explorer.EXE	2040	RegQueryKey	HKCU\Software\Classes\CLSID\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}\InProcServer32
3:22:3...	Explorer.EXE	2040	RegQueryKey	HKCU\Software\Classes\CLSID\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}\InProcServer32
3:22:3...	Explorer.EXE	2040	RegOpenKey	HKCR\CLSID\{0E5AAE11-A475-4C5B-AB00-C66DE400274E}\InProcServer32

Figure 7: Process Monitor

We then used the tool Ghidra to verify that the malicious dynamic link library exports `DllGetClassObject`, which is necessary to perform COM hijacking of Windows Explorer. Component object model hijacking is identified as T1122 in the Mitre Attack Framework. To detect this technique defenders can use `sysmon` to monitor event ID 13, Registry value modification events. Monitoring for new subkeys of the CLSID key will allow defenders to detect new COM servers.

```
HRESULT DllGetObject(IID *rclsid, IID *riid, LPVOID *ppv)
{
    FUN_180001000();
    return 0x0;
}
```

Figure 8: COM Export

### 3.5 Agent Tesla

#### Agent Tesla Dropper

878F50F5965B5C795BE1E1D7A12CE6155DC6FDE4ED127E8839EF1E4EE66BD708

Agent Tesla is a publicly available, for purchase malware, that enables threat actors to steal passwords saved in browsers, collect keystrokes, and take screen captures of victim computers. Agent Tesla has been around since at least 2014, and has been used by criminal groups such as Silver Terrier in attacks against companies [31]. In this section we analyze a sample of Agent Tesla used in a campaign in 2020. Agent Tesla targets solely the Microsoft Windows operating system, and is focused on user workstations from which it can steal user information.

Agent Tesla employs layers of obfuscation and XOR encryption. The XOR function is a Boolean function which on input  $(x, y)$ , where  $x, y$  are bits, returns 1 if and only if exactly one of  $x, y$  is 1. This function is ubiquitous in cryptography as it is reversible and easily used to encrypt a stream of data. To decode the Agent Tesla sample we opened the sample in DnSpy. *DnSpy* is an open source .NET assembler and debugger. Upon inspecting the sample we found code which executed a method named `cor41`, which was

loaded from a resource within the Agent Tesla sample. *.NET Resources* are non-executable data that is included within the application. An example of a resource is a image used by the interface of the application. Using DnSpy we viewed the suspicious resource in the DnSpy hexadecimal editor, and noticed the bytes MZ in the ASCII view, these two bytes are the File Signature for Windows executables and dynamic link libraries. Noting that this was likely an executable or dynamic link library embedded as a resource, we extracted the resource.

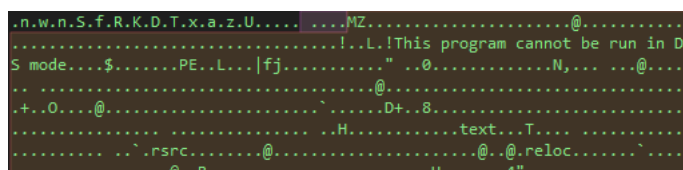


Figure 9: MZ file signature

Once extracted to disk, we discovered that the embedded resource was a dynamic link library that is used to convert another resource, a PNG image, into an executable. This technique, known as *steganography*, involves hiding one file within another. In this case, Agent Tesla operators have hidden the next stage of the malware within a XOR encrypted executable embedded within a PNG resource. The PNG resource looks like the following when viewed in DnSpy:

An overview of the structure of the executable can be found in Figure 11.

Using PowerShell and its ability to invoke C# code we were able to use the functions in the malware author's library to decode the image and produce

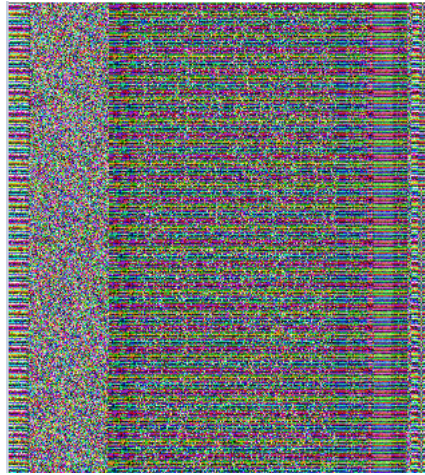


Figure 10: Steganography

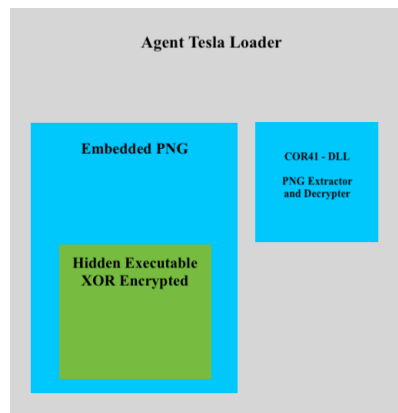


Figure 11: Agent Tesla Structure

an executable. The PowerShell code to decrypt the PNG resource is provided on a Github repository for this thesis (<https://github.com/zaneGittins/AgentTeslaStegDecoder>) but a significant snippet is provided in Listing 3. This code converts non-black pixels into byte arrays. This data is still XOR encrypted, and must go through an additional decryption to obtain the malware executable.

Listing 3: PowerShell code to decrypt the PNG resource

```
public static byte[] FromBitmap(Bitmap cor23) {  
    ArrayList arrayList = new ArrayList();  
    checked {  
        int num = cor23.Size.Width - 1;  
        for (int i = 0; i <= num; i++) {  
            int num2 = cor23.Size.Height - 1;  
            for (int j = 0; j <= num2; j++) {  
                Color pixel = cor23.GetPixel(i, j);  
                Color color = Color.FromArgb(0, 0, 0, 0);  
                bool flag = !pixel.Equals(color);  
                if (flag) {  
                    arrayList.InsertRange(arrayList.Count, new byte[] {  
                        pixel.R,  
                        pixel.G,  
                        pixel.B }); } } }  
        return (byte[]) arrayList.ToArray(typeof(byte)); } }
```

In the PowerShell code we also perform XOR decryption using a copy of the C# code analyzed via DnSpy; see Listing 4. Once the image is extracted and XOR decrypted, we obtain the final payload. The key is a sixteen character key, where an individual character of the key is XOR'd with the ciphertext, and rotates by using the modulus operation to loop the key across the entire ciphertext.



Listing 4: PowerShell code for XOR decryption

```
public static byte[] XOR(byte[] cor30) {  
    byte[] array = new byte[cor30.Length - 16 - 1 + 1];  
    Array.Copy(cor30, 16, array, 0, array.Length);  
    int num = array.Length - 1;  
    for (int i = 0; i <= num; i++) {  
        byte[] array2 = array;  
        int num2 = i;  
        array2[num2] ^= cor30[i % 16]; }  
    return array; }
```

Because of the ease in reverse engineering .NET code, authors have utilized control flow flattening to make it difficult to analyze. *Control flow flattening* is a method for obfuscating code. In this case the malware author's use switch statements inside of a for loop to make it difficult to determine the flow of execution. When a block of code in a switch statement is executed, the program returns to the beginning of the for loop, until the final block, which exits the for loop. This allows the malware authors to re-arrange the code inside of switch statements within the for loop, which makes it difficult to read. The end of each switch statement determines the next switch statement that will be executed on the subsequent iteration of the for loop.

When first run this payload checks for the mutex, `qaxmCJedRGq`, if it exists then the malware does not run. *Mutexes* are objects that allow programs to share the same resource, by temporarily locking that resource to prevent

multiple processes from accessing a resource simultaneously. Mutexes are commonly used by malware to check if a computer is already infected, additionally mutexes can be used by defenders to identify compromised systems. Agent Tesla uses a mutex to prevent multiple instances of Agent Tesla running on the victim system. This mutex is likely not shared with other variants of Agent Tesla, and can only be used to uniquely identify this particular sample.

If run with administrative privileges the malware disables Windows Defender by setting several registry keys. Through Registry manipulation, Agent Tesla disables the Windows Defender AntiSpyware, OnAccessProtection, BehaviourMonitoring, and TamperProtection keys. These changes cause Windows Defender to be inactive on the victim system.

To capture keystrokes of the victim system Agent Tesla hooks WH\_KEYBOARD\_LL by using the Windows API call `SetWindowsHookEx`, which captures all keys that users enter on the victim system.

```
// Token: 0x0600009A RID: 154
[DllImport("User32.dll", CallingConvention = CallingConvention.StdCall, CharSet = CharSet.Auto, EntryPoint = "SetWindowsHookEx")]
private static extern int vt(int eg, jkp.vx.vg ex, IntPtr et, int gx);
```

Figure 12: SetWindowsHookEx

```
private int jrl(int jrq, IntPtr jrf, IntPtr jra)
```

Figure 13: Hook Function

This hook function takes three variables that are necessary for keyboard hooks. This first variable, `jra` is a code that determines how the next hook

handles the return of this hook, the next variable `jrf` is a pointer to a keyboard input notification, this can be used to determine if a key is pressed down. The third variable `jra` is a pointer to a struct which contains the keycode of the pressed key, a hardware scan code, and a timestamp for the event. This allows the Agent Tesla malware to steal sensitive keystrokes, such as passwords entered into web pages, and exfiltrate them by one of Agent Teslas several channels.

```
case 8U:
{
    jkp.vx.ve ve = ve2;
    jkp.vx.bg bg;
    object obj2 = Marshal.PtrToStructure(jra, bg.GetType());
    jkp.vx.bg bg2;
    ve((checked(Keys)((obj2 != null) ? ((jkp.vx.bg)obj2) : bg2).vkCode));
    num = 3024804571U;
    continue;
}
```

Figure 14: vkCode

```
Bitmap bitmap;
Graphics graphics = Graphics.FromImage(bitmap);
Graphics graphics2 = graphics;
Point point = new Point(0, 0);
Point upperLeftSource = point;
Point upperLeftDestination = new Point(0, 0);
graphics2.CopyFromScreen(upperLeftSource, upperLeftDestination, blockRegionSize);
memoryStream = new MemoryStream();
```

Figure 15: Screen Capture

To take screen captures of the victim computer, Agent Tesla uses `.NETGraphics.CopyFromScreen` from the `System.Drawing` namespace. The image data is stored in a memory stream and exfiltrated via email. Capturing screenshots can allow Agent Tesla actors to learn how users operate, what applications they use, and what data they interface with, potentially increasing the revenue they can generate from an infected system. This helps operators of the malware to gain a clearer understanding of their target environments, by watching what

systems users interact with, they can identify high value targets for further malware deployments, such as cryptominers and ransomware. Additionally, these screenshots may contain sensitive data of what is currently open on a victim's system.

Once sensitive data is collected Agent Tesla must exfiltrate this data to its operators. Agent Tesla has several methods for communicating with command and control servers, the technique we observed was SMTP. To communicate Agent Tesla uses the simple mail transfer protocol (SMTP) to send an email, which contains data that Agent Tesla wishes to exfiltrate. In the sample we analyzed, the from and to address of the email were the same, and credentials for the email address were embedded within the executable. The sample we analyzed connected to port 587 on the destination server to exfiltrate data via email. The subject line for each email appears to correspond to the type of information exfiltrated, the username, and the system name. For example,

PW\_SYSTEMNAME/USERNAME

Where PW may denote the exfiltrated data is a password, SYSTEMNAME is the name of the compromised system, and USERNAME is the name of the compromised user. The body of the email contains a timestamp, the username of the infected user, system name of the infected system, and operating system details. Emails exfiltrating passwords, use the following format for the body of the email:

Time: TIMESTAMP  
User Name: OSUSERNAME  
Computer Name: COMPUTERTNAME  
OSFullName: OSINFO  
CPU: CPUINFO  
RAM: RAMINFO  
URL: URLINFO  
Username:USERNAME  
Password:PASSWORD  
Application:APPLICATION

Where TIMESTAMP, OSUSERNAME, COMPUTERTNAME, OSINFO, CPUINFO, RAMINFO, URLINFO, USERNAME, PASSWORD, APPLICATION are values assigned by the malware during exfiltration. To capture this information we developed a python tool called Fakemail, (<https://github.com/zaneGittins/Fakemail>). This tool changes DNS to point to the local host 127.0.0.1, responds to DNS queries on a specified port, and responds to SMTP traffic on a given port. For compatability with windows it is necessary to call Fakemail with the `-d 53` commandline argument, which causes Fakemail to listen for DNS queries on port 53. Port 53 is the standard port for DNS traffic. For our sample, we used the paramters `-p 587` to have Fakemail listen for SMTP traffic on port 587. With these configurations when the query for the email server `weldonsqfe.com` is made by Agent Tesla,

our server replies with 127.0.0.1. This allows us to identify the malicious domain used by Agent Tesla, as well as to redirect the SMTP traffic to our Fakemail SMTP listener. The malware therefore connects back to 127.0.0.1 instead of the malicious domain and attempts to send an email via SMTP. Because Fakemail responds to SMTP traffic we were able to gather the data and format that Agent Tesla uses to exfiltrate data. Fakemail was written to respond to programs that use the .NET class `SmtpClient`, and therefore does not implement the entire SMTP protocol. Agent Tesla is written in .NET and uses the class `SmtpClient` for communication, which makes Fakemail the perfect tool to intercept Agent Tesla SMTP exfiltration. Using Fakemail we were able to intercept this email and its contents, displayed in the figure below.

```
Sending '354 start mail input, end with <CRLF>.<CRLF>' to client.
Breaking recieved end.
Received mail data.
MIME-Version: 1.0
From: amandayang@weldonsqfe.com
To: amandayang@weldonsqfe.com
Date: 13 Jun 2020 14:08:20 -0700
Subject: PW_Researcher/ARES-WK01
Content-Type: text/html; charset=us-ascii
Content-Transfer-Encoding: quoted-printable

Time: 06/13/2020 14:08:18<br>User Name: Researcher<br>Computer Name: ARES-WK01<br>OSFullName:=
Microsoft Windows 10 Enterprise 2016 LTSB<br>CPU: Intel(R) Core(TM) i9-8950HK=
CPU @ 2.90GHz<br>RAM: 4095.49 MB<br><br>URL:https://accounts.google.com/signin/v2/challenge/pwd<br>=
=0D=0AUsername:test555<br>=0D=0APassword:Spring2020!#<br>=0D=0AAApplication:Chrome<br>=
=0D=0A<br>=0D=0A
.
Sending '250 OK' to client.
```

Figure 16: Fakemail Agent Tesla

Research into the server-side of Agent Tesla suggests that cyber-criminals do not directly view these emails, but instead they are parsed and ingested into a database and displayed to the actor via a web browser. Because the contents of these emails are not encrypted in any way, defenders who

have access to full network data can determine the severity of a compromise by dissecting the command and control traffic between an Agent Tesla sample and a command and control server. Two open source tools capable of enabling defenders to inspect this communication are Security Onion (<https://securityonion.net/>) and Moloch (<https://molo.ch/>).

Analyzing the payload we found that Agent Tesla uses the following Registry key to maintain persistence:

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run
```

Registry run keys have identifier T1060 in Mitre Attack, and are a common technique for malware persistence. This key is in the current user registry hive, HKCU, and will cause the Agent Tesla sample to run each time the affected user logs in. The HKCU registry run keys are editable by the current user, and therefore Agent Tesla does not require administrative privileges for this method of persistence. To detect Agent Tesla defenders should alert on SMTP communication to untrusted domains, as well as monitor for the creation of startup registry keys. Detecting SMTP exfiltration requires application level protocol identification, as many Agent Tesla samples do not use the typical port 25 to communicate over SMTP. To identify new registry run keys, defenders can use the tool Sysmon, a service and driver by Microsoft that can be used to monitor a wide range of security events and generate Windows event logs. Event ID 13 in Sysmon is logged when a value is set in the Windows registry. This event ID can be used to monitor when

the path to the Agent Tesla payload is added as a value to a run key. To determine if a machine is compromised by Agent Tesla without Sysmon installed prior to an infection, defenders can use tools such as Kansa, OSQuery, GRR, and Velociraptor to query large numbers of hosts in real time for the current entries in their Registry hives.

### 3.6 Search Protect

Search Protect

SHA256 6D5048BAF2C3BBA85ADC9AC5FFD96B21C9A27D76003C4AA657157978D7437A20

Windows Shims allow programs created on older operating systems, such as Windows XP, to run on newer systems such as Windows 10. Windows shims are libraries that apply fixes which allow older programs to run on newer systems. Using the application compatibility administrator tool a user can create a Shim database file by specifying a target executable, which the tool will filter for based on file name, file location, file size, and others [29]. Once the application is specified a user can select several modes and or fixes. *Fixes* are changes that a shim can apply to a target executable. *Modes* are bundles of fixes, for example the mode WIN7RTM which is a group of fixes to ensure compatability with Windows 7 programs. There are over eight hundred fixes available in Microsoft's compatability administrator utility. Some of the pre-defined fixes for shims include InjectDll, DisableNX, and RedirectEXE. These are only a small number of the predefined fixes avail-



able for creating shims using Microsoft’s application compatability administrator tool (<https://docs.microsoft.com/en-us/windows/deployment/planning/compatibility-administrator-users-guide>). The `RedirectEXE` fix is of use to attackers because it allows for redirection of an executable while maintaining the privileges of the original executable. This technique has been used by the malware Black Energy, which targeted Ukrainian SCADA systems, to bypass user access control (UAC) prompts [29][34]. *User Access Control* was a feature introduced in Windows Vista and Server 2008, that requires a prompt to gain administrative privileges, child processes are the exception, they inherit the administrative token from their parent process. This feature can help to prevent malware from launching with administrative privileges, because it requires the administrator to accept the UAC prompt, Black Energy uses an application compatability shim to bypass this security feature. Black Energy also used the fix `DisableNXShowUI` to disable data execution prevention (DEP). *Data Execution Prevention* marks memory regions as non-executable, this prevents shellcode from being executed in a process, and requires attackers to use more advanced techniques, such as return oriented programming (ROP). *Return Oriented Programming* is a method by where an attacker chains together instructions already present in the target programs memory to achieve a goal, instead of injecting shellcode. This can be done when the attacker controls the instruction pointer, (EIP) on 32 bit programs, (RIP) or 64 bit programs. Each gadget is an instruction typically followed by a return instruction, which allows the actor to chain

together multiple gadgets by redirecting execution to a new gadget after the completion of the previous gadget. This is a powerful technique for bypassing protections such as data execution protection (DEP), where the attacker may not be able to write and execute the memory of the target application.

One of the most valuable shim fixes for persistence is `InjectDll`, which was designed to pre-load libraries into an executable. From an attacker's perspective, this can be used to inject a malicious library into an executable, and thereby gain persistence by injecting and executing the malicious library whenever the shimmed executable is launched. If a commonly launched executable is chosen to shim, attackers can obtain a reliable method of persistence.

When a program in Windows is executed the application compatability database is queried to determine if the program requires a shim. The shim-cache, a component of the application compatability database, has been used by the Windows forensic community to determine what programs have executed on a Windows system. However, shims can also be used maliciously to accomplish evasion, process manipulation, persistence, denial of service, obfuscation, and in-memory patching of applications [29].

Windows shims are typically installed using a native program called `sdbinst.exe`. This program registers a shim database file, which has the extension `.sdb`, by creating subkeys under the following registry locations:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB
```

It requires administrative access to make changes to the HKLM Registry hive. Normally, shims are installed using the `sdbinst.exe` program, which causes a listing to be created for the shim as an installed program on the system. In the case of the financially motivated criminal group FIN7, they use the `sdbinst.exe` to install their malicious shim, but use a name of Microsoft KB2832077 in an attempt to hide among Windows patches. This name follows the naming convention for Windows patches, even though it is actually a malicious application shim. However, malware that manually creates these keys itself, instead of using `sdbinst.exe` has one key advantage, the listing for the shim does not appear in the Windows add and remove program interface, and therefore is more difficult to identify by defenders.

A program known as **Search Protect**, often classified as a potentially unwanted program by anti-virus, uses shims to maintain persistence on victim hosts. Search Protect uses the `InjectDll` fix to inject a malicious dll into the processes `chrome.exe`, `firefox.exe`, `iexplorer.exe`, and `software_removal_tool.exe`. This causes Search Protect to start whenever one of these common internet browsers is started.

Using the tool `python-sdb` (<https://github.com/williballenthin/python-sdb>), it is possible to parse and view a shim database file. Using this tool to view the Search Protect shim database revealed the following:

```
<STRINGTABLE>
  <STRINGTABLE_ITEM type='string'>2.1.0.3</STRINGTABLE_ITEM>"
  <STRINGTABLE_ITEM type='string'>Apps32</STRINGTABLE_ITEM>"
  <STRINGTABLE_ITEM type='string'>VC32Ldr\x04\x02</STRINGTABLE_ITEM>"
```

```

<STRINGTABLE_ITEM type='string'>InjectDll</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>\\\\.\\globalroot\\systemroot\\apppatch\\nbin\\vc32loader.dll</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>chrome.exe</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>ch</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>&lt;Unknown&gt;</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>*</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>explorer.xxx</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>ex</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>firefox.exe</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>ff</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>iexplore.exe</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>ie</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>software_removal_tool.exe</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>sr</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>software_reporter_tool.exe</STRINGTABLE_ITEM>"
<STRINGTABLE_ITEM type='string'>sr2</STRINGTABLE_ITEM>"
</STRINGTABLE>'

```

This shim loads the dynamic link library located at:

```
\\\\.\\globalroot\\systemroot\\apppatch\\nbin\\vc32loader.dll
```

into the programs `chrome.exe`, `explorer.exe`, `firefox.exe`, `iexplore.exe`, and `software_removal_tool.exe`.

To install the `.sdb` file Search Protect uses the native windows tool `sdbinst.exe`:

```
c:\windows\system32\sdbinst -q C:\Users\admin\AppData\Local\Temp\VC_browsers32.sdb
```

The `-q` switch runs the `sdbinst` program in quiet mode, which automatically accepts all prompts, this is useful for adversaries who may be running sd-

isnt without interaction, such as launching `sdbinst` from an office macro. Application shimming has identifier T1138 in the Mitre Attack framework. To detect application shimming defenders should monitor modifications to the Registry key `AppCompatFlags`, and its subkeys `Custom` and `InstalledSDB`. To do this defenders can use event ID 13 of Microsoft's `sysmon` tool. Additionally, defenders can use event ID 1 from `sysmon`, `process-created`, for `sdbinst.exe`, however this is not a strong a detection because it is possible to modify these registry keys without the use of Microsoft's `sdbinst.exe` utility.

## 4 Detection Architecture

In this section we propose an architecture for detecting malware persistence techniques in Windows environments. This architecture is for collecting and alerting on Windows events from endpoints. There are several methods for detecting malicious actors presence on endpoints, such as anti-virus and endpoint detection and response (EDR), user behaviour and entity analytics (UEBA), network security monitoring (NSM), and system information and event management (SIEM). To bolster the likelihood of detection defenders should layer these technologies, and conduct pro-active human driven hunting activities to search for signs of attackers within their environments. Combining business specific knowledge, such as where sensitive data is stored, what data storage services are legitimate, and the roles of various users within an organization are critical to detecting adversaries. Defenders can prevent attackers from completing their goals by detecting adversaries, fully scoping intrusions, and removing adversaries access before adversaries accomplish their objective. This requires defenders to have strong optics into their environments and analysts who understand what to look for on the technical side, combined with the knowledge of business critical assets and data.

The architecture we discuss is a general architecture that will be useful in detecting a wide range of security events. This section is primarily focused on the application of multiple open source projects to create a robust detection platform. Before we discuss the specific open source projects, it is important

to discuss a general architecture that could be applied in the future, should any of the specific open source projects we discuss no longer be supported. In Figure 17 we show such an architecture.

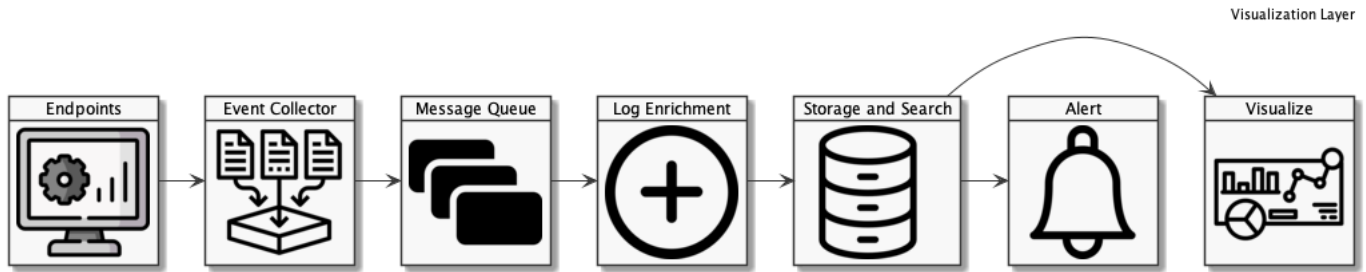


Figure 17: Visualization Layer Diagram.

Figure 17 shows each generic component of our detection architecture. Endpoints are the systems where we seek to discover malware persistence mechanisms, they are the source of data. Endpoints in this architecture are configured to send logs to an Event Collector. The Event Collector in turn sends events to a high performance message queue that can handle a large volume of data. Events from the queue are then enriched by a log enrichment component that adds metadata to specified logs. Next, these logs are stored in a high performance database. Two components connect to the database. The first component alerts analysts of logs that match rules. The second component allows analysts to search for and visualize data.

Our proposal is not novel, and is close to the architecture used by the Hunting ELK (HELK) project by Roberto Rodriguez (<https://github.com/Cyb3rWard0g/HELK>). The HELK project is a powerful way to quickly deploy a security logging stack, and many of the ideas in our project were

influenced by Roberto Rodriguez’s work. HELK includes two additional technologies not present in our proposed architecture, Apache Spark, and Jupyter Notebooks. Apache Spark can provide data science capabilities, and Jupyter Notebooks can be used to create powerful run books for hunting for malicious activity. Jupyter notebooks also allow for a nice way for defenders to document an investigation, as each section of a notebook shows the progress of an investigation, and the defenders understanding of an adversary at a given point in time. These technologies were not included in our architecture for simplicity, however because we use Apache Kafka, these two technologies could be integrated into this stack, which would make the architecture very close to the HELK project [32].

Although our proposed architecture is similar to HELK, we seek to provide additional details on exact event ids to monitor, as well as an overview of how the various technologies work together to create a strong log centralization, hunting platform, and alerting solution. *Hunting* is the process of proactively searching for adversaries within an environment based on techniques, tactics, and procedures of known malicious groups. Using this stack defenders will be able to hunt for adversaries by writing queries in Kibana. Additionally we give examples of queries and rules that can be used to detect the malware presented in the first section of this thesis.

All of the technologies used in this infrastructure are free to use, however the paid licenses and support are worth consideration for production environments, as it will allow for a larger feature set, and increased security of each



component, as well as additional features useful for detecting adversaries. It is critical to properly defend this architecture, as failure to do so can allow adversaries to disrupt the optics of defenders into their environments. The architecture we propose uses an ELK stack, ELK stands for Elasticsearch, Logstash, and Kibana. Additionally we add Apache Kafka for dealing with spikes of data that may otherwise adversely effect Logstash, and Elastalert to write rules that alert on security events in Elasticsearch. There are no recommendations regarding sizing or number of nodes per technology in this thesis, as these numbers are highly dependent on the amount of data being ingested and the rate that this data is ingested. Special considerations should be taken for production environments to ensure that this technology stack does not have a single point of failure. Many of the components we propose such as Apache Kafka, Logstash, and Elasticsearch can be run in clusters, which should reduce the likelihood of a single component failing, and defenders losing visibility into security events. The HELK project combines all of these technologies into an easily deployable docker container, however it is also useful to understand how each of these technologies work independently so that they can be deployed as standalone instances for environments that require more powerful nodes and easier customization options to tailor the architecture to the business needs of an organization. Additionally, an in depth understanding of this logging stack will allow for integrating with new technologies in the future, and troubleshooting each component.



is a tool for executing Windows binaries on remote systems, used by both system administrators and adversaries. Using `OriginalFileName` will allow for alerts on PsExec to trigger even if an adversary modifies the name of the PsExec executable. Event ID 1 also includes the hash of the executable, which can be used to search for malware samples within an environment. Furthermore, event ID 1 also includes more details on the parent process of the executable than the native event ID 4688 does. Event ID 4688 only includes the path of the parent process, while event ID 1 includes the process identifier of the parent process, as well as the command line arguments of the parent process. Command line arguments can be critical information to determining what an actor did while in control of a victim system. In the case of the Search Protect malware, analyzing the arguments provided to the `sdbinst.exe` application would be critical to identifying malicious activity, furthermore using the parent process information defenders can identify the process that launched `sdbinst.exe` and installed the malicious application shim. In our proposed architecture, windows event logs are forwarded from hosts to a Windows event collection (WEC) server. If using event ID 4688 instead of event ID 1 then it is crucial that defenders enable command line logging. Neither event ID 4688 or commandline logging are enabled by default in Windows domains. To enable 4688 process creation events navigate to the following path in group policy and change `Audit Process Creation` from `Not Configured` to `Enabled`.

Computer Configuration > Policies > Windows Settings

> Security Settings > Advanced Audit Configuration > Detailed Tracking

To enable command line logging navigate to the below path and change Include command line in process creation events to Enabled [20].

Administrative Templates\System\Audit Process Creation

Daniel Panay from FireEye stated that Mandiant, FireEye’s digital forensics and incident response (DFIR) team, rarely sees command line logging enabled in their customer’s environments [28]. This is critical for defenders to enable as it provides important context to process creation events. One example of this is Emotet, which launches `powershell.exe` with a Base64 encoded payload as an argument. By enabling command line logging for process creation, defenders can catch the malicious Base64 executed by Emotet. This is critical to detection, but also plays an important role in forensics, as the malicious document may be deleted from disk by the attacker, and event logs may be the only record which contains a complete copy of the malicious PowerShell that was executed on a compromised endpoint. Another example is the use of the `sdbinst` program to install a malicious application shim, without command line logging it is difficult to determine if the installed shim is malicious or not. Use of the `-q` switch makes the usage of `sdbinst` more suspicious, and may warrant additional investigation by defenders, such as examination of the parent process and user of the process launching `sdbinst`.

## 4.2 Windows Event Forwarding

All of the windows event we discussed in the last section are written to the local system that they were generated on. Events are written to `.evtx` files in the `C:\Windows\System32\winevt\Logs` directory. In order to alert and hunt for security events it is necessary that we centralize event logs and get them off of endpoints as soon as possible. Events on compromised endpoints may be deleted or modified, so it is critical that these are sent to a secure location. There are two methods for collecting event logs, installing an agent on each endpoint, or using Windows event forwarding (WEF).

*Windows event forwarding* allows for Windows workstations and servers to forward event logs to a destination server. Windows event forwarding allows for two models, source initiated (push) or collector initiated (pull). We recommend using the source initiated model, as it does not require configuration of each host in the collector and therefore scales more easily. The Windows event collection (WEC) server can have multiple subscriptions configured, these subscriptions dictate which hosts to accept events from, the destination log to collate events to, as well as the type of events, and event ids to accept. Windows event collectors should be configured to store event logs (`.evtx` files) on a separate hard drive. This will help to reduce the chance that the disk write speed becomes a bottleneck for collected events. Because our architecture will forward events off of the collector to an Apache Kafka topic in realtime, the size of the forwarded event logs on the collector can

be kept small (1-5G), which will greatly increase the performance of the collector. Large log files will lead to high RAM utilization and slower write times for incoming events. Forwarded event logs are by default all stored in one log file, however this can be modified to store events in unique event logs, which allows for more easily diagnosing issues on the Event Collector, especially when multiple log sources may use the same numerical event ids. Using separate log files should also increase performance, as each log file can be kept at a small size, and one noisy event channel should not effect the performance of other event channels.

Windows event forwarding can be deployed with group policies to easily configure large groups of endpoints to send logs to a central server. Using windows event forwarding has several advantages over directly running a log collection agent on each host. First, it allows administrators to focus on a single log collection technology rather than managing the log collection software on each host. Furthermore, forwarding Windows event logs off of hosts in realtime makes it more difficult for actors to remove event logs before they reach their destination. For example, attackers can clear windows event logs, thereby denying defenders evidence. There have also been techniques proven to be able to delete individual event records, a capability not supported by Microsoft, which greatly hampers analysis. It is paramount that defenders get logs off of endpoints as fast as possible to limit the capability of attackers to disrupt their optics. Logs collected on the Windows event collection server should be forwarded using winlog-

beat (<https://www.elastic.co/beats/winlogbeat>), a tool by Elastic, to an Apache Kafka host.

### 4.3 Apache Kafka

*Apache Kafka* is a technology for ingesting data into Kafka topics. *Topics* in Kafka are streams of records, in our case, each record will be a security event sent from winlogbeat on a WEC server. Events from our Apache Kafka topics will be collected by Logstash, which will in turn send our events to Elasticsearch for search and storage. It would be possible to use this architecture without Apache Kafka, and instead configure winlogbeat to send events directly to Logstash. This may be a suitable approach for smaller environments that do not need to worry about large spikes of events and do not want to add the complexity of Apache Kafka. Apache Kafka also allows for the addition of other technologies into this architecture, such as Apache Spark, and Jupyter Notebooks.

Apache Kafka uses a consumer, producer architecture, that is, *producers* subscribe to Kafka topics and add events, and *consumers* subscribe to topics and read events. Apache Kafka is an important technology to include in our detection stack because it prevents Logstash from being overwhelmed in the event of a spike in the number of events sent by winlogbeat from our WEC server. Additionally, if Logstash goes down, it can later be brought back online and continue to read data from Kafka as a consumer from its last position in the topic. There another advantage to using Kafka, which

is a tool called Kafkacat. Kafkacat allows for replaying events into a Kafka topic, this is an invaluable ability to have as it allows us to save samples of malicious activity and replay them into our detection architecture to test alerts and train new analysts. One such project already exists called Mordor, (<https://github.com/hunters-forge/mordor>) which contains numerous pre-recorded security events that can be replayed using Kafkacat.

## 4.4 Logstash

*Logstash* is a tool for ingesting, modifying, formatting, and sending data to a selected output. In the case of our architecture Logstash is configured to read data from a Kafka topic and output this data to Elasticsearch. Logstash can also filter out noisy events, and integrate with other systems. For example, Logstash can be used to integrate a wide range of security products into our logging stack, such as anti-virus alerts, and firewall logs. There are a wide variety of plugins for Logstash, written in the Ruby programming language, available to integrate additional logs. It is also easy to write your own Logstash plugin to interface with an API and send results to Elasticsearch. Logstash has a wide array of input plugins that allow for connecting data to our Elasticsearch instance. Existing Logstash plugins also allow for ingesting common file formats such as csv files, this is a powerful feature as many tools currently support output to csv. Additionally, Logstash has output plugins which allow for the output of data to other platforms besides Elasticsearch, this allows flexibility and defenders to send logs ingested by



this platform to other platforms such as a user entity and behaviour analytics (UEBA) solution. Logstash is therefore a critical component of the logging architecture. Logstash can also be configured to ingest network data such as Zeek logs, formerly known as Bro, and Cisco eStreamer logs, which help to facilitate network security monitoring (NSM), and offer further context into malicious activity. A powerful solution for capturing and analyzing network data is Security Onion, which uses the ELK stack as well. Security Onion, much like HELK, provides an out of the box deployment for collecting security data, unlike HELK, Security Onion is focused on network data, but can also be used to ingest Windows event logs. Ingesting network data into our architecture will allow for correlation between host based data and network data, and therefore provide a clearer picture of an environment. Network data can be especially powerful, because a properly configured network tap will contain an entire record of what transpired on the network, whereas host based data can be dependent on proper configuration of each host. Endpoint systems may not be configured to collect the logs necessary to detect an adversary, whereas network data contains all network information, it is for this reason that network data is often called the great equalizer of log sources. It is critical that defenders ingest and analyze both host and network data. In production environments, multiple Logstash instances can be used to read data from an Apache Kafka cluster.

## 4.5 Elasticsearch

*Elasticsearch* is a search technology that can be used to store and query large volumes of data. By default, Elasticsearch listens on port 9200. Events sent to Elasticsearch in JSON format will be automatically parsed and searchable by field, therefore it is preferable that all events are sent in JSON format where possible. Elasticsearch is a powerful component because it provides close to realtime search capabilities over vast amounts of data, and therefore enables defenders to hunt for adversaries in large data sets. Elasticsearch also scales easily, so it is quick to add additional nodes to handle an increase in log intake, such as new log sources, or additional endpoints added to an environment. Elasticsearch allows for storing a wide-range of data, including structured and unstructured data. The windows event logs read from our Kafka topics will be in JSON format, and thus are easily stored and searched for in Elasticsearch. It is recommended that defenders configure Elasticsearch to store at least thirty days of logs, however the longer that data can be kept, the better for post incident analysis. *Dwell time* is the time it takes for defenders to discover a breach internally. FireEye reported that the mean Dwell time among their customer's in 2020 was 56 days [10]. Therefore, if possible, defenders should look to extend the time that security data is stored for to as long as possible in order to enable searching data for past intrusions. Many organizations may experience much longer dwell times, even lasting several years, therefore security data should be kept for long periods of time if

possible. Storing data for long periods should be considered when designing this architecture, as storage will be important to maintaining records for a reasonable amount of time. This data need not be kept in memory in Elasticsearch, but can instead be sent to cold storage, and restored if it is needed at a later time to investigate an incident.

## 4.6 Kibana

*Kibana* is a web interface for interacting with Elasticsearch. Kibana allows users to write queries in Lucene syntax to search for data, this is a powerful tool for defenders as it will allow us to visualize the data collected, as well as discover malicious events within an environment. For example, the below Lucene search can be used to identify Pass-The-Hash attacks.

```
event_id: 4624 AND auth_process: seclogo AND logon_type: 9
```

This event is generated on the source host, therefore this query only provides optics into Pass-The-Hash attacks that occur on a defender monitored system, and not Pass-The-Hash attacks that originate from a un-monitored system. In other words, this is a source host detection, there are similar detections that can be configured on the destination host of a Pass-The-Hash attack. *Pass-The-Hash* is a method whereby an attacker passes a stolen NTLM password hash to challenge-response authentication on a Windows system. Because the hash of a password is required to authenticate, and not the password itself, it is not necessary for the attacker to crack the stolen

hash. This allows attackers that have administrative access to a system to steal hashes from other users who have logged into that system, and use these hashes for authentication on other systems. Using this technique an attacker can move throughout a victim environment using the privileges that correspond to the user of the stolen hash. Hashes are kept on systems from a variety of actions, one example is an administrator opening an interactive RDP session on a computer, and then not properly closing the session. If default settings are in place, a password hash for the administrator will be stored on the system and accessible by anyone with administrative privileges on that system. This would allow a local administrator to steal the password hash of a domain administrator, and then conduct pass-the-hash attacks, thereby expanding their access. One defense against pass-the-hash attacks is the built-in security group **Protected Users** which prevents credentials from being cached, Kerberos from using DES/RC4, and several other protections for accounts. Adding users to this group and then rotating their password, from before their addition to the group, can be an effective strategy to protecting privileged accounts from pass-the-hash.

The variant of Emotet we analyzed in a previous section used Windows services for persistence. A query to detect the service creation technique used by Emotet is as simple as searching for `event_id: 4697`, however Windows services are used by many legitimate applications, so it will be necessary to filter out known good processes. To do this the following syntax can be used:

```
event_id: 4697 AND NOT process_directory: KNOWNPATH
```

Where `KNOWNPATH` can be replaced by the paths to legitimate services used within an environment. The sample of Trickbot we analyzed used scheduled tasks for persistence. Searching for scheduled task creation used by Trickbot is similar to searching for service creation, except event ID 4696 is used instead.

## 4.7 Elastalert

*Elastalert* is a tool developed by Yelp which allows for writing rules that trigger on data in Elasticsearch. Using Elastalert defenders can write rules that trigger alerts whenever certain events occur within their environments. Elastalert can be configured to query Elasticsearch at a given interval, this interval can be as short as several seconds, or as long as many weeks. Elastalert is a powerful tool for automating queries and sending alerts to Email, Slack, Microsoft Teams, and many more destinations. Elastalert also integrates with powerful outputs such as Atlassian's Jira, which allows for creation and assignment of a ticket in response to an alert. This allows for better tracking and resolution of alerts by defensive teams. One example of an Elastalert rule is alerting on bogon IP addresses returned by the Cobalt Strike DNS beacon:

```
alert:
```

```
- debug
```

```
- ms_teams
```

```
description: Adversaries may communicate using a common, standardized application
```

```
layer protocol such as HTTP, HTTPS, SMTP, or DNS to avoid detection by blending
in with existing traffic. This rule detects Cobalt Strike DNS.
filter:
- query:
    query_string:
        query: (event_id:"22" AND DNSResolvedIP:("0.0.0.0"))
index: winlogbeat*
name: Bogon-DNS-Response
priority: 3
realert:
    minutes: 0
type: any
```

Using Elastalert it is possible to write a wide range of rules to trigger alerts on security related events. Elastalert supports many rule types, the above rule is an **ANY** rule, which means that any event that matches the query will trigger an alert. Another powerful type of alert are cardinality rules, which trigger when a number of unique values for a specified field is reached within a given time range. This can enable rules for triggering on attacks such as when one IP address is used to attempt to login to multiple accounts within a week, which may be an indicator of a password spraying attack. *Password spraying* is where an attacker attempts to use one password against many accounts, the goal of the attacker is not to gain access to a specific account, but to gain access to any account. Password spraying is often more

difficult to detect then brute force attacks, because it is rare for password sprays to trigger account lockouts. To bypass this detection attackers can rotate the IP addresses that they use for each login attempt, however this increases the technical effort on the side of the attacker. These alerts can be configured to trigger on unusual scheduled task creation, service creation, and run key modification. Elastalert uses yaml files for it's rules, however another alternative is to write rules in the Sigma language. Sigma is a universal rule language for security information and event management systems (SIEM) (<https://github.com/Neo23x0/sigma>). Rules in Sigma can be converted to a wide range of commercial SIEMs. This makes it easier to share rules with others, and allows rules that are written to not be tied to any one platform. In the event that the writer switches from this architecture to another, for example a commercial SIEM, the rules they spent time developing can likely be converted if they were written in Sigma.

An exaple if a Sigma rule can be found below:

```
title: Process LSASS Access
logsource:
  product: windows
  service: sysmon
tags:
  - attack.credential_access
  - attack.t1030
detection:
```

```
selection:
    EventID: 10
    TargetImage:
        - "C:\\Windows\\System32\\lsass.exe"
    Access:
        - 0x1010
        - 0x1F0FFF
        - 0x1F1FFF
        - 0x1F2FFF
        - 0x1F3FFF
        - 0x1FFFFFF
    condition: selection
falsepositives:
    - Legitimate applications.
level: high
```

This sigma rule detects applications accessing the `lsass.exe` process with suspicious access permissions. This can be used to detect malicious tools which steal credentials from memory by reading data from the `lsass.exe` process. One such tool is `Mimikatz.exe` which is commonly used by penetration testers and real adversaries for credential access, which accesses the memory of `lsass.exe` with the permissions `0x1010`.

It is important for defenders to know what attacks look like, and identify gaps in their alerting and monitoring capabilities. One tool that greatly aids this process is Atomic Red Team by Red Canary (<https://atomicredteam.com>).



io/). Atomic Red Team allows defenders to run sample attacks which correspond to techniques in the Mitre Attack framework. Using Atomic Red Team, defenders can run examples for many of the persistence techniques used by adversaries. This allows defenders to identify gaps in the logs that they collect, make changes in their log collection process, write rules for Atomic Red Team events, and educate new analysts by showing them examples of real techniques. When writing new detection rules defenders should be careful to not write brittle rules that only detect attacks run by Atomic Red Team. An alternative to Atomic Red team is Caldera(<https://github.com/mitre/caldera>), which can also be used to run example attacks to test detection and optics. Ideally defenders should make use of multiple adversary simulation technologies, as well as real red teams to ensure that they have sufficient logging, alerting, and training. Using multiple technologies will help defenders stray away from writing rules that are tool specific and may fail if the adversary makes minor changes in their toolset. It is important for defenders to create rules that are difficult for adversaries to avoid without changing techniques and tactics, this will increase the cost of the attack for the adversary as well as increase the chance that an adversary makes a mistake that leads to their detection.

When writing rules defenders should try to keep in mind capability abstraction, proposed by the SpectreOps detection team[4]. *Capability abstraction* is the idea that the core features of adversary tools are abstracted, often by multiple levels, from the users of those tools. By taking a deep look into

multiple tools used to accomplish the same technique, defenders can write rules that target the underpinnings of a tool, and therefore these rules are likely to work for updated versions of the tool, and other tools that use the same technical underpinnings. Detection rules written with capability abstraction in mind are much less brittle, and do not alert on indicators such as specific strings, which can easily be changed by attackers. In the blog post by SpectreOps, *Capability Abstraction*, the team reveals that three tools used for Kerberoasting, *Invoke-Kerberoast*, *Rubeus* and *Mimikatz* all use the same RPC call. *Kerberoasting* is an attack against Microsoft's Kerberos implementation, created by Tim Medin, where an attacker with a valid ticket granting ticket (TGT) requests many service tickets (TGS), because portions of the TGS are encrypted with the password of the service, attackers can conduct an offline brute force against TGS tickets to obtain passwords for service accounts [15]. One important note is that while lower levels of abstraction apply to more tools, they also generally have higher false-positive rates and may capture legitimate activity as well.

## 4.8 Application of Architecture

In order to validate the above architecture we created detection rules and loaded them into Elastalert. We then ran each sample in an isolated environment. The environment contained an instance of each component of our detection architecture, where data flowed from an endpoint to our detection stack.

Our detection architecture was able to successfully detect each sample analyzed within this paper, additionally, our detections targeted the persistence mechanisms used by the malware samples, making it more difficult for malware authors to evade. In the case of Emotet we alerted on Windows event id 7045, which detects when Emotet creates a new service. The Ocean Lotus DLL Sideload sample was detected by monitoring for changes to registry run keys, which generates an event id 12 event. Trickbot was detected by alerting on new scheduled task creation, event id 4698. The second Ocean Lotus sample, using COM Hijacking, was detected by alerting on event id 12 events to COM registry keys. Agent Tesla was detected in a similar manner to the first Ocean Lotus sample, by alerting on event id 12 events to Windows run registry locations. Lastly, Search Protect was detected by alerting on values set within the shim database using event id 13.

By alerting on persistence mechanisms defenders can create rules that are more difficult for malware authors to evade. Furthermore, a single rule that alerts on a persistence mechanism can be useful in detecting a wide array of malware samples.

## 5 Conclusion

For all the mythology and Hollywood glamour surrounding malware, all malware is limited to a finite number of persistence mechanisms. In the case of Emotet a Windows service is created. In the case of Trickbot scheduled tasks are leveraged. In Ocean Lotus Symantec DLL Sideload, the malware takes advantage of the lack of validation in loading of dynamic link libraries (DLLs); in the case of Ocean Lotus — Explorer-COM Hijack, the malware rides the Component Object Model by manipulating a key in the user hive; in the case of Agent Tesla, the malware payload was embedded in a PNG file, using a modern variation of the art of steganography, and starting the malware by leveraging Windows run registry keys. Search Protect abuses Windows application shims, typically used for backwards compatibility, to inject itself into commonly used web browsers.

In the second section of the thesis we covered an architecture that can be used to detect the malware presented in this report. The architecture uses Windows event forwarding to send events to Apache Kafka, those events are ingested by Logstash, and then indexed in Elasticsearch. Defenders can then hunt for malicious activity using Kibana, a web interface for interacting with Elasticsearch, and Elastalert to write rules that trigger on security events in Elasticsearch. This architecture is only one of several detection layers that should be applied to detecting adversary activity. Additional important technologies are user and entity behavioural analytics (UEBA), network secu-

rity monitoring (NSM), and anti-virus and endpoint detection and response (EDR) products. Using these technologies, combined with regular review by trained analysts will greatly increase the ability of defenders to detect a wide range of malware and actors within their environments.

We do not want to minimize the challenges of defending systems against malware. Malware can be ingenious and persistent, and the number of persistence mechanisms used by malware are vast. In this thesis we examined a subset of the different techniques malware authors use in order to gain persistence: the ability for the malware to survive a reboot of the system. The techniques described depend on the malware's ability to cloak itself as a legitimate component of the target system. In order to do so, the malware examined makes use of Windows features and takes advantage of validation vulnerabilities, especially in the loading of Dynamically Linked Libraries (DLLs).

## Acknowledgments

This work was completed by the author for a masters thesis in Computer Science at California State University at Channel Islands, under the supervision of Dr. Michael Soltys. We are thankful for the direction, guidance, and classes offered by Dr. Michael Soltys that allowed us to complete this thesis. The security class offered by Dr. Soltys provided us with a firm footing in security fundamentals and encryption mechanisms, as well as taught code-breaking, an invaluable technique when analyzing obfuscated malware.

We would also like to thank Dr. Reza Abdolee for his course on Ethical Hacking, during which we developed some of the ideas for this paper.

We are grateful to our colleagues at Haas Automation, Meissner Filtration, and California State University Channel Islands for discussions about these topics. We are also grateful to Sam Decanio and Kimo Hildreth for comments on the draft of this thesis.

## References

- [1] David Bianco. The pyramid of pain, 2013.
- [2] Nick Carr. Cyber espionage is alive and well: Apt32 and the threat to global corporations, 2017.
- [3] Satyajit Daulaguphu. A journey towards an import address table (iat) of an executable file, 2019.
- [4] SpecterOps detection team. Capability abstraction, 2020.
- [5] Romain Dumont. Bitlocker overview, August 2016.
- [6] Romain Dumont. Apt32, December 2017.
- [7] Romain Dumont. APT32, 2017.
- [8] ESET. Oceanlotus: Old techniques, new backdoors. Technical report, ESET, March 2018.
- [9] Brian Fehrman. How to bypass application whitelisting av, 2016.
- [10] FireEye. Mtrends 2020 insights from the front lines, 2020.
- [11] Avi Gimpel. Amsi bypass redux.
- [12] Alexander Hanel. Big game hunting with ryuk: Another lucrative targeted ransomware, 2019.

- [13] Thomas Hungenberg. Emotet, trickbot, ryuk – ein explosiver malware-cocktail, 2019.
- [14] Klein. Attacks on the rc4 stream cipher, 2008.
- [15] Tim Medin. Attacking kerberos: Kicking the guard dog of hades, 2014.
- [16] Adam Meyers. Meet crowdstrike’s adversary of the month for february: Mummy spider, 2018.
- [17] Microsoft. Antimalware scan interface (amsi).
- [18] Microsoft. Servicestartmode enum.
- [19] Microsoft. Applocker overview, 2017.
- [20] Microsoft. Command line process auditing, 2017.
- [21] Microsoft. Event logging, May 2018.
- [22] Microsoft. Auto macros, 2019.
- [23] Microsoft. Autoruns for windows v13.98, 2020.
- [24] MITRE. Persistence, 2020.
- [25] Raphael Mudge. Malleable command and control.
- [26] Raphael Mudge. Red team ops with cobalt strike (2 of 9): Infrastructure.
- [27] Raphael Mudge. Red team ops with cobalt strike (3 of 9): C2.



- [28] Daniel Pany. Using real-time events in investigations, 2020.
- [29] Sean Pierce. Malicious application compatibility shims, 2015.
- [30] Vicky Ray and Kaoru Hayashi. Tracking oceanlotus' new downloader, kerrdown, 2019.
- [31] Peter Renals. Silverterrier: 2019 nigerian business email compromise update, 2020.
- [32] Roberto Rodriguez. What the helk? sigma integration via elasticsearch, 2018.
- [33] SentinelOne. Malicious input how hackers use shellcode, 2019.
- [34] Michael Soltys. Cybersecurity in the aws cloud, March 2020.
- [35] Murugiah Souppaya Tatu Ylonen, Paul TurnerKaren Scarfone. Security of interactive and automated access management using secure shell (ssh), October 2015.

