

ANALYSIS AND CATEGORIZATION OF DRIVE-BY DOWNLOAD
MALWARE USING SANDBOXING AND YARA
RULESET

by

MOHIT SINGHAL

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2019

Copyright © by Mohit Singhal 2019

All Rights Reserved



Acknowledgements

I would like to thank Professor David Levine, my supervisor who gave me an opportunity to work under him and encouraged me throughout the work. It is his consistent support and motivation that helped me to complete this thesis.

I would also like to thank my committee members Dr. Jiang Ming and Dr. Manfred Huber, for taking time out of their busy schedule and attending my thesis defense.

I would like to extend my sincere thanks to my parents, who are always there supporting and motivating me. I would also like to extend my thanks to my friends who were always there to support me and motivate me throughout my research. I would also like to express my deepest gratification to Dr. Cynthia Manzano, my therapist who helped me and motivated me.

Lastly I would like to extend my thanks to the team at VMRay, who were generous enough to extend my free trial for three months.

April 18, 2019

Abstract

ANALYSIS AND CATEGORIZATION OF DRIVE-BY DOWNLOAD MALWARE USING SANDBOXING AND YARA RULESET

Mohit Singhal, MS

The University of Texas at Arlington, 2019

Supervising Professor: David Levine

With the increase in the usage of websites as the main source of information gathering, malicious activity especially drive-by download has exponentially increased. A drive-by download refers to unintentional download of malicious code to a user computer that leaves the user open to a cyberattack. It has become the preferred distribution vector for many malware families. Malware is any software intentionally designed to cause damage to a user computer.

The purpose of this research is to analyze the malware that were obtained from visiting approximately 100,000 malicious URLs and then running these binaries in sandboxes and then analyzing their runtime behavior with a software tool (YARA) to categorize them and classify what malware family to which they belong. Out of the 1414 program executables (binaries) that were captured, 1000 binaries were executed and 99 were identified as false-positive. Out of the 1414 binaries that were extracted 959 of them were executable, 48% of the binaries were extracted from websites that were hosted in the US. We also found that 105 binaries had the same name but different hashes that is, they were not identical. Out of the 901 binaries, 867 of them were identified as Trojan Horse and we were able to

identify 53 type of malware families, with one particular family, Kyrptik, having 176 malware belonging to it which is about 19% and about 4% of the malware families were not identified.

Table of Contents	
Acknowledgements.....	iii
Abstract.....	iv
List of Illustrations.....	vii
List of Tables	ix
Chapter 1 Introduction	10
Chapter 2 Background	15
Chapter 3 Related Works	20
Chapter 4 System Design.....	23
Chapter 5 Experimental Setup and Implementation	29
Chapter 6 Observations.....	35
Chapter 7 Conclusion.....	51
Chapter 8 Future Work	52
References.....	53
Biographical Information.....	57

List of Illustrations

Figure 1-1 Drive-by Download Architecture.....	10
Figure 2-1 Keylogger hook.....	15
Figure 2-2 Malware creating a backdoor and then receiving data.....	17
Figure 2-3 Malware sample is rebooting the system and then running in a hidden window.....	18
Figure 4-1 Host-Only networking.....	23
Figure 4-2 Cuckoo sandbox architecture	25
Figure 4-3 VMRay Analyzer architecture	26
Figure 5-1 Proposed architecture	29
Figure 5-2 Viper web based interface.....	31
Figure 5-3 Binary information	32
Figure 5-4 Percentage of binaries extracted from different countries	33
Figure 6-1 Number of websites that were crawled.....	35
Figure 6-2 Number of binaries collected	36
Figure 6-3 Ph0neutria working.....	37
Figure 6-4 Cuckoo sandbox result	38
Figure 6-5 Screenshots of the execution of binary	38
Figure 6-6 Static analysis of a binary	39
Figure 6-7 Strings output	40

Figure 6-8 Strings output when sample is not encrypted.....	40
Figure 6-9 Signatures of the binary	41
Figure 6-10 Report generated from a false-positive sample.....	41
Figure 6-11 Report generated from VMRay.....	42
Figure 6-12 Dynamic analysis report.....	43
Figure 6-13 Process tree from the execution of binary.....	44
Figure 6-14 Process information.....	45
Figure 6-15 Information about the execution of binary.....	45
Figure 6-16 Information from VirusTotal.....	46
Figure 6-17 Function log obtained from VMRay	47

List of Tables

Table 5-1 Number of binaries captured	32
Table 6-1 Categorization of malware	48
Table 6-2 Number of malware families identified.....	48

Chapter 1

Introduction

With the increase in the number of websites as the main source of information gathering, malicious activity has also increased exponentially. According to an Internet security threat report from Symantec [1], one in ten websites are malicious. Since the Internet has become a main source of gathering information, attackers are using it to spread malware using “exploit-as-a-service” model which is based on “drive-by download” attacks. Gone are the days when the user had to click “accept” in order to download or install a software update to become infected, the user now just has to visit or “drive by” a webpage, without stopping to click or accept any software, and the malicious code will be downloaded in the background to the user device [2]. Exploits are normally the errors in the software development process that leave holes in the software’s built-in security that cybercriminals can then use [22]. Attackers rather than creating their own exploit kits, pay for an existing exploit kits like Blackhole and Eleonore to do the “dirty work” of exploiting the victim’s browser. A typical drive-by download is shown in Fig 1-1. [3]

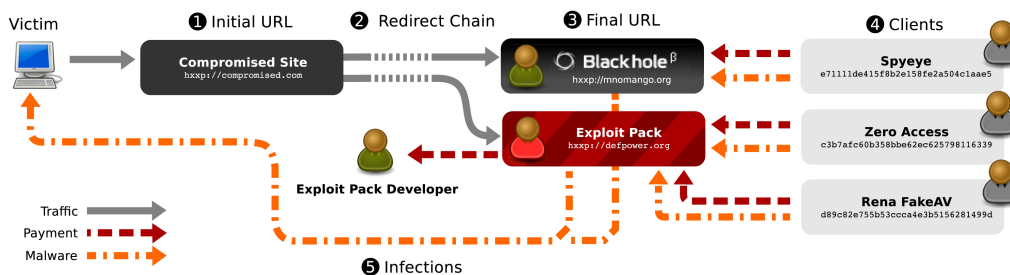


Figure 1-1 Drive-by Download Architecture [3]

As we can see in fig 1-1, we can see that the victim is visiting a compromised site which is also known as landing site which is often setup as a legitimate website. The landing website then redirects the victim’s browser to a chain of intermediate pages to the final URL which is the exploit site which hosts the exploit kit. The attack code is downloaded from this exploit site and is executed by the victim’s web browser. Then, the victim’s browser forcibly downloads additional malware from the malware download site.

Malware is a term used to describe software that accomplishes the deliberately harmful intent of an attacker [4]. Basically, malicious attacks are done in order to steal critical user data or to control the victim's computer from a remote location or to damage the compromised system. There has been an exponential increase in the number of new malware that are affecting computers since 2015. According to AV-TEST [6], total number of malware's increased from 47 million to 882 million from 2010 to 2019. There are different types of malware, each classified by their unique and specific capabilities. Some of the popular types of malware are given below.

Trojan Horses: This is the most commonly seen type of malware. Trojans masquerade as a normal software trick the victims into installing it. For instance, once installed, it will execute the payload which can either steal user data or financial information, or it can give access to the attacker to remotely control the system.

Spyware: This is designed to monitor and gather user's activity without their knowledge. They often spread through the Trojan Horse payload. Keyloggers are the most common type of spyware which is used to collect user's keystrokes.

Downloader: This is designed to have only the functionality of downloading content. The downloaded content can vary from case to case but they can compromise of, but not limited to, configuration/command information, miscellaneous files, misleading applications [5].

Viruses: These are designed to replicate and then spread by hiding inside a normal computer program or file. The file/program that the virus has infected is called the host. When this host is executed, the virus also get executed. Viruses normally disable system defenses, steal user data and in some cases, can even corrupt the boot sector which is used by the computer to load the operating system and other programs that are necessary.

Botnet: A botnet is a collection of Internet-connected devices, which may include PCs, servers, mobile devices and Internet of things devices that are infected and controlled by a common type of malware. Infected devices are controlled remotely by threat actors, often cybercriminals, and are used for specific functions, so that the malicious operations stay hidden to the user [7].

Remote Access Trojan (RAT): A remote access Trojan (RAT) is a malware program that includes a back door for administrative control over the target computer. RATs are usually downloaded invisibly with a user-requested program -- such as a game or sent as an email attachment. Once the host system is compromised, the intruder may use it to distribute RATs to other vulnerable computers and establish a botnet [8].

Riskware: Riskware is the name given to legitimate programs that can cause damage if they are exploited by malicious users – in order to delete, block, modify or copy data, and disrupt the performance of computers or networks [9].

Backdoor: A backdoor is a means to access a computer system or encrypted data that bypasses the system's customary security mechanisms. Attackers often use backdoors that they detect or install themselves as part of an exploit. In some cases, a worm or virus is designed to take advantage of a backdoor created by an earlier attack [10].

Ransomware: It is a special type of malware, which infects the victim's computer and limits the victim's access to the system and its files until a ransom is paid. It does this by encrypting the victim's files and locking the user's desktop and then displaying the user a message which demands ransom and the way to pay that ransom.

According to [38], malicious cyber activity cost the U.S. economy between \$57 billion and \$109 billion in 2016. Verizon's data breach investigations report noted that 75% of recent cyber incidents and breaches were caused by outsiders, while 25% were performed by internal actors. According to [38], on an average a median firm with a market capitalization of \$12 billion, lost \$498 million per adverse cyber event. PricewaterhouseCoopers in 2014 reported that as many as 71% of cyber compromises go undetected.

Given the exponential increase in the number of new malware being discovered [6], and the huge losses suffered by firms, it is necessary that these malware should be analyzed and categorized so that their signatures can be fed into the available anti-virus software, in order to protect users from these malicious software's. In this paper, we take a much deeper dive into the analysis of these malicious entities by checking their effect and behaviors on the system using both static and dynamic analysis.

There are two basic approaches to malware analysis: static and dynamic analysis. Static analysis encompasses examining the sample without executing it, whereas dynamic analysis involves running the sample in a contained and isolated environment called a “sandbox”. A sandbox is a security mechanism for running untrusted programs in a safe environment without fear of harming real systems [13]. In static analysis, the sample is examined on its static properties like header details, hashes, signatures. The sample is also disassembled using a disassembler to look into its instructions to see what it does. In contrast, in dynamic analysis the sample behavior is observed while it is being executed in a contained environment. This provides new insights into the working of the sample as we can see any changes in the system or any odd behavior. Through dynamic analysis an analyst can see track any changes in the registry, whether any existing file was modified or was added/deleted, if there was any new process that was created, whether the sample tried to connect to the Internet. As stated earlier, dynamic analysis is normally performed in a contained environment such as sandbox. By using a sandbox, we can easily go back to a previous state and we also eliminate the possibility of the sample affecting the host machine and infecting it.

In this paper, we analyze the binaries (which could be executables, documents, pdf) that were collected after visiting approximately 100,000 malicious websites using a malware crawler. A malware crawler is a software which visits links of websites in order to extract meaningful information from them. These binaries were stored in a labeled database and then executing them in Cuckoo Sandbox, VMRay Analyzer Cloud and the sandboxes available in VirusTotal, we analyzed the static and dynamic behaviors and then using a software tool (YARA) categorizing them and classifying what malware family to which they belong.

Out of the 1414 program executables (binaries) that were captured, 1000 binaries were executed and 99 were identified as false-positive. Out of the 1414 binaries that were extracted 959 of them were executable, 48% of the binaries were extracted from websites that were hosted in the US. We also found that 105 binaries had the same name but different hashes that is, they were not identical. Out of the 901 binaries, 867 of them were

identified as Trojan Horse and we were able to identify 53 type of malware families, with one particular family, Kryptik, having 176 malwares belonging to it which is about 19%, and about 4% of the malware families were not identified.

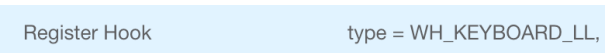
Chapter 2

Background

Malware authors are always trying to find new methods to conceal a malicious executable in a form that can evade existing detection method such as anti-virus search. In order to do that, they create new naming conventions, obfuscating the code in an image or making it a macro in a Microsoft Word document or an Excel sheet. According to Symantec, office macro downloaders and the use of powershell scripts increased 1,000 percent [1] over a period of two years. Different malware uses different techniques to increase the potential harm to the victim's computer. For example, they can create a process in a hidden window, inject its script into other running processes, it can disable meta files, it can corrupt the master boot sector of the computer, it can store store the keystrokes of the user, encrypt user's file and establish a communication with a command-and-control servers. In addition to the above, the proposed system assumes that a binary is indeed a malware it should also perform one or more of the following activities.

Registry Changes: The Windows registry is a very easy target for any malware. It is a collection of configuration settings for the Windows operating systems. One of the widely reported registry changes is the addition into the "Autostart Extensibility Points (AESPs)". Normally malware might add itself to AESPs to store keystrokes so that it can capture the password in plain text or to store other password's which can be used by the attacker to either cause financial fraud or to get hold of the system remotely. In order to do that the malware has to manipulate the

"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" registry key. Figure 2-1, shows the hook that is used by keylogger to monitor the keystrokes.



```
Register Hook                                type = WH_KEYBOARD_LL,
```

Figure 2-1 Keylogger hook

In order to gain persistence, malware normally manipulate the registry key that was given earlier. They can also achieve persistence by hooking the malicious software to a particular Winlogon event like logon, logoff, startup and shutdown. A hook is a place and usually an interface provided in packaged code that allows a programmer to insert customized programming [23].

They do that by making changes to the

“HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\” registry key.

Connections to a remote host or non-responsive IP addresses: One of the ways malicious agents are able to get control of the victim’s system is generally by having a direct connection to the victim’s system and that is achieved by the malware executing a payload that pings an IP address that belongs to the malware author. One common type of attack is the creation of a backdoor. In a recent report by Symantec, Asus software updates were hijacked by attackers and they were able to install a backdoor on thousands of computers [11]. Fig 2-2 shows malware creating a backdoor and then receiving data.

Information	Value
Remote Address	217.195.152.27
Remote Port	54727
Local Address	192.168.0.254
Local Port	49160
Data Sent	549 bytes
Data Received	0 bytes

Operation	Additional Information	Success	Count	Logfile
Create	protocol = IPPROTO_TCP, address_family = AF_INET, type = SOCK_STREAM	✓	1	FN
Bind	local_address = 192.168.0.254, local_port = 0	✓	1	FN
Connect	remote_address = 217.195.152.27, remote_port = 54727	✓	1	FN
Send	flags = NO_FLAG_SET, size = 549, size_out = 549	✓	1	FN DATA
Close	type = SOCK_STREAM	✓	1	FN

Information	Value
Remote Address	217.195.152.27
Remote Port	21
Local Address	192.168.0.254
Local Port	49159
Data Sent	191 bytes
Data Received	715 bytes

Operation	Additional Information	Success	Count	Logfile
Create	protocol = IPPROTO_TCP, address_family = AF_INET, type = SOCK_STREAM	✓	1	FN
Connect	remote_address = 217.195.152.27, remote_port = 21	✓	1	FN
Receive	flags = NO_FLAG_SET, size = 1024, size_out = 321	✓	1	FN DATA
Send	flags = NO_FLAG_SET, size = 29, size_out = 29	✓	1	FN DATA
Receive	flags = NO_FLAG_SET, size = 1024, size_out = 55	✓	1	FN DATA
Send	flags = NO_FLAG_SET, size = 18, size_out = 18	✓	1	FN DATA
Receive	flags = NO_FLAG_SET, size = 1024, size_out = 43	✓	1	FN DATA
Send	flags = NO_FLAG_SET, size = 14, size_out = 14	✓	1	FN DATA
Receive	flags = NO_FLAG_SET, size = 1024, size_out = 23	✓	1	FN DATA
Send	flags = NO_FLAG_SET, size = 5, size_out = 5	✓	1	FN DATA
Receive	flags = NO_FLAG_SET, size = 1024, size_out = 34	✓	1	FN DATA

Figure 2-2 Malware creating a backdoor and then receiving data

Changes in the System: Malware families such as Emotet, Ramnit and various others [24, 25] make changes to the operating system, either modifying the registry keys or dropping new files or crashing running processes. They also can stop crucial Windows services such as disabling the Windows security center or killing the .NET framework [26]. One of the commonly seen behavior from malware samples were that after they

were executed in any of the sandboxes, they would suddenly reboot the system and then start their execution in a hidden window. Fig 2-3 shows the same.

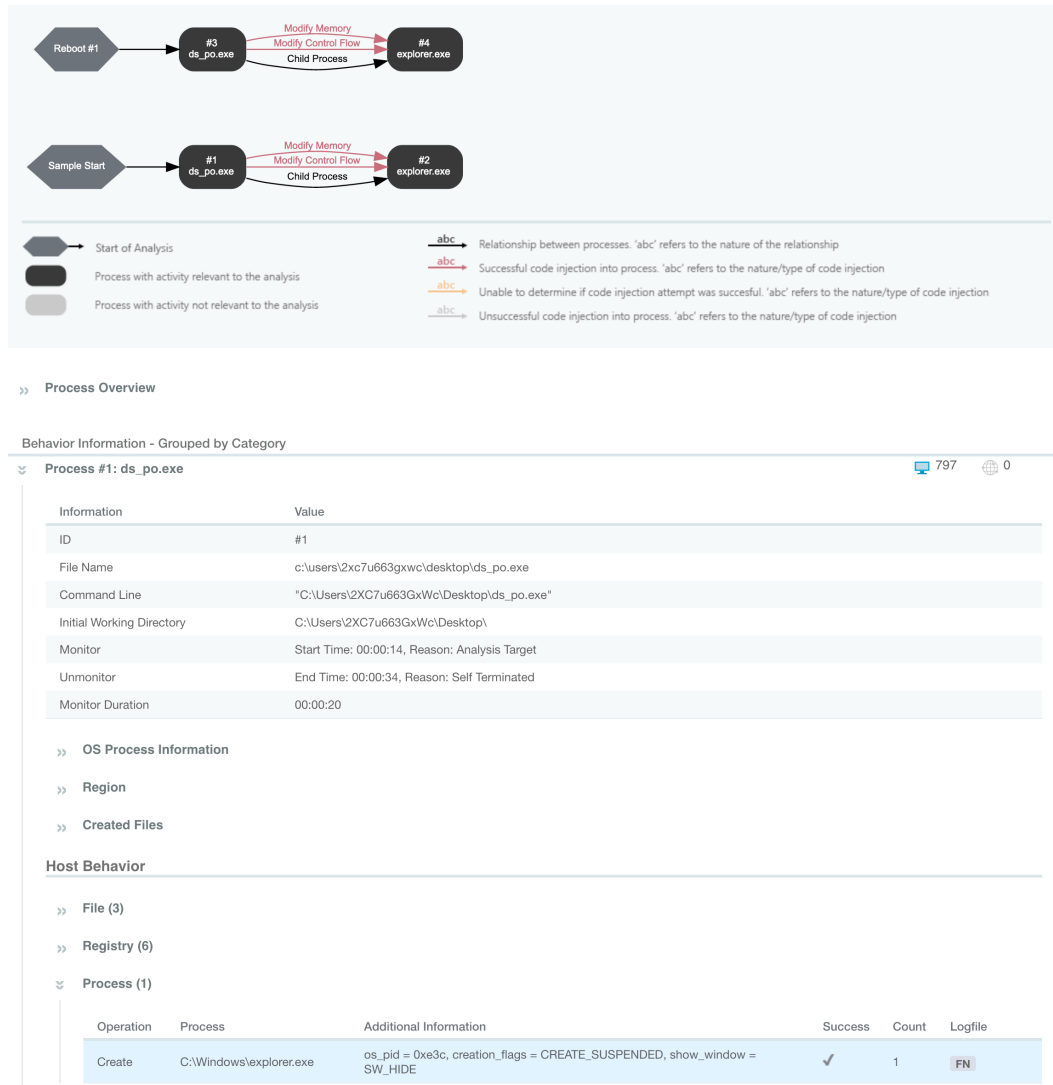


Figure 2-3 Malware sample is rebooting the system and then running in a hidden window

Identification using YARA ruleset: This is an extremely powerful tool that is aimed at (but not limited to) helping malware researchers to identify and classify malware samples [12]. While executing in the binaries in Cuckoo sandbox [13] and VMRay Analyzer

Cloud [14], they were able to match the behavior of some of the samples with the existing YARA rules.

In this work, we were able to use all the features that are listed above and able to categorize malware based on the effect they were having on a system.

Chapter 3

Related Works

- In [32] Matthew et al. assesses the efficiency and effectiveness of the existing tools for automated malware analysis. The main aim was to use six existing tools to study how malware persists on the infected system. The author tested the tools by using malware samples that were extracted from various sources and then studied which techniques of persistence was detected by which tool.

One of the limitation of this research was that the number of malware samples that were extracted for the study. They only considered 33 samples in order to study the tools. The other limitation was that, the author never considered using sandboxes in order to get a much better perspective on the persistence methods. In this research, the authors executed the malware for three minutes and after that would extract data from the tools that the authors are considering for the research. The authors never considered the possibility that malware author are using “Sleep” functionalities in order to evade possible detection.

- In [33] Bailey et al. assesses the efficiency of existing anti-virus software’s to detect and provide meaningful information about the malware samples that were collected. The authors provide evidence that different anti-virus software characterize malware in a way that is inconsistent across anti-virus products. The aim of authors in this research was to come up with a novel solution to identify the malware not only based on the number of hits on various anti-virus software but also by looking into the system changes. This paper provides many solid insights.

The limitations of this research were that the authors only considered a small amount of information. The authors never measured the extensive amount of information that can be extracted by disassembling the code, in order to get more insights, as well as that they also didn’t consider the possibility of

obfuscated code and as well as the possibility of sleep functionalities used by malware authors.

- In [34] Tanaka et al. investigate the rise of exploit-as a service ecosystem. The main aim of this research was to analyze the malware download sites focusing on the time series variation of malware by examining their SHA1 hashes daily for a period of one and a half years. The authors daily examined approximately 43,000-malware download URLs in order to study their behavior and to investigate malware distribution network. The authors were able to categorize the variation into three categories i.e. unchanged, every time changed and changed occasionally.

The limitation of this research was that the authors were only considering if there were any changes in the hashes and not looking if the changed malware were in any way different in their behavior then the previously discovered malware. Also, the choice of using SHA1 in order to compare hashes is questionable, because in 2015, a collision attack on SHA1 was detected.

- In [35] Nappa et al. investigated the hosting and distribution of drive-by downloads, however the main aim of their research was to understand how drive-by download operates? In order to study this, the authors created an infrastructure to track individual exploit servers over time, collecting and classifying the malware that they distribute.

Limitation in this research was the authors were classifying the malware based on icons, network and screenshots that were obtained while the malware executed in an isolated contained environment. They executed the samples in a virtual machine rather than in a sandbox, hence limiting themselves to some of the basic ways to classify a malware.

- In [3] Grier et al. investigated the emergence of exploit-as-a-service model in the drive-by download ecosystem. The authors performed their analysis on about 77,000 malicious URLs. They were able to extract about 10,000 binaries from these URLs which they then executed in a contained environment.

The authors classified the binaries solely on the results that they got from executing the malware. They never considered the possibility of using YARA rules, as that would have decreased the number of unknown binaries.

- In [36] Moshchuk et al. studied the threats from malicious spyware that exist on the web. They used a crawler to do a large-scale, longitudinal study of the web, by crawling 18 million URLs. They found spyware in approximately 13% of the 21,200 executable that they tested.

The authors in this research determined the existence of a spyware by running a scan of an anti-spyware tool called “AdAware”. In addition, the authors considered executing the sample in a sandbox if it triggered some of the trigger conditions. The authors did not take into consideration if the binary was doing any networking activity as part of a trigger condition.

Chapter 4

System Design

In this chapter, we describe the techniques for a binary to be identified as a malware sample. The ultimate goal of the proposed system is to identify which binaries are malicious in nature and which are false-positive. A false-positive is when a program/file is not malicious in nature but shows those characteristics. As the proposed system is using Cuckoo sandbox and VMRay Analyzer Cloud, it is virtually impossible that any data will be lost as Cuckoo reverts back to its original state and VMRay has multiple virtual machines that are working with no interaction between them and after the binaries have executed reverts back to its original state.

We first describe the malware aggregation stage in the proposed system. Then, we mention the need of a virtualized environment, after that we describe what settings were being used to execute the binaries in the sandboxes that were used, and then we describe on what parameters the static and dynamic analysis were performed on the summaries that we got from the sandboxes. Finally, describing how YARA ruleset were applied on the summaries to manually classify whether the binary was a false-positive or the binary was malicious in nature, if that is the case then what is the type of malware and to which family does the malware belong.

Virtualized Environment: One of the most important things while executing any malware sample is that it doesn't affect the analysts machine. So, for that analysts always use virtual machines in order to curb that problem. In the proposed system, we are using a Host-Only networking which is depicted in fig 4-1 [13].

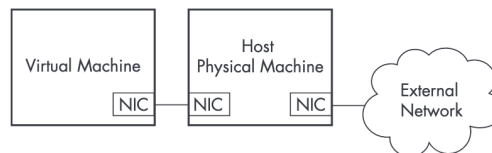


Figure 4-1 Host-Only networking [13]

Using the above stated architecture, it creates a separate private LAN between the host OS and the guest OS [13]. Since this connection is not connected to the Internet, the malware sample is contained in the VM but it is still allowed network connectivity.

Malware Aggregation: One of the most difficult job for any malware analyst especially the one's focusing on drive-by download is the number of websites that exist on the Internet. According to one survey there are about 1.8 billion websites on the world-wide web [14]. Now going to all of them and determining whether they contain malware samples or not is not possible. However, there are some malware URL feeds such as Malshare, CleanMX, Cymon, Malc0de [27, 28, 29, 30] to name a few, that makes it easier to collect malware samples.

Our malware crawler (ph0neutria) sources the samples straight from these URL feeds [15]. One of the interesting aspect of this malware crawler is that it stores all the extracted binaries in a viper database [16], which is discussed more in chapter 5. Ph0neutria has plugins for the above URL feeds. One thing that the malware crawler aims to do is to only crawl to those URL feeds which are frequently updated. Using this crawler, we were able to crawl approximately 100,000 URL's and was able to collect 1414 binaries.

Execution Environment: Sandboxing is used in malware analysis, to run an unknown and untrusted application or file inside an isolated system and get the details of what that application/file does to that test system.

In order to differentiate whether the binary that we extracted is indeed malicious in nature, the proposed system uses two of the leading sandboxes which are available in the market i.e. Cuckoo sandbox and VMRay Cloud Analyzer. The proposed system also makes use of the sandboxes that are present in VirusTotal. Cuckoo sandbox is a leading open source automated malware analysis system. It's used to automatically run and analyze files and collect comprehensive analysis results that outline what the malware does while running inside an isolated operating system [17]. Fig 4-2 shows the Cuckoo sandbox architecture. Cuckoo sandbox provides the following in its result [17]:

- Traces of calls performed by all processes spawned by the malware.
- Files being created, deleted and downloaded by the malware during its execution.
- Memory dumps of the malware processes.
- Network traffic trace in PCAP format.
- Screenshots taken during the execution of the malware.
- Full memory dumps of the machines.

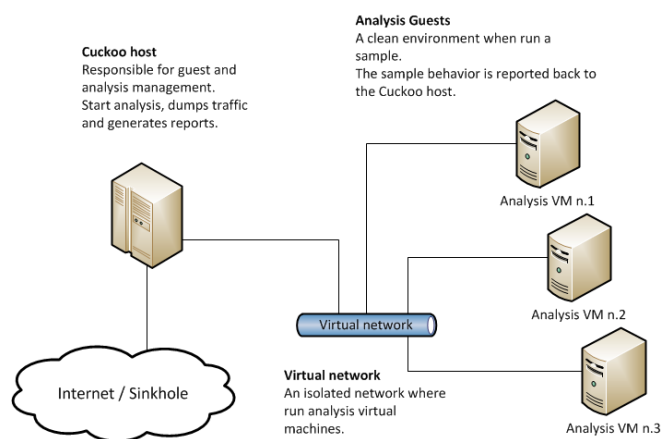


Figure 4-2 Cuckoo sandbox architecture [17]

VMRay Analyzer is an automated malware detection and analysis framework that utilizes dynamic, static, and reputation-based analysis to generate comprehensive behavior reports of unknown software [18]. Fig 4-3 shows the VMRay Analyzer architecture. One of the advantage of using VMRay Analyzer is that it has inbuilt VMRay Threat Identifier (VTI) which automatically identify and flags malicious behavior.

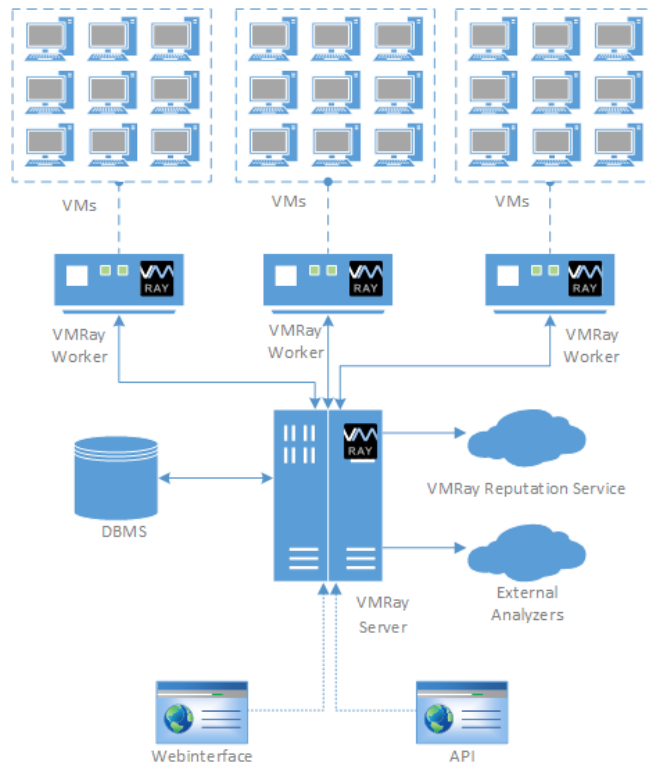


Figure 4-3 VMRay Analyzer architecture [18]

Static and Dynamic analysis: In order to classify whether the binary is indeed malicious in nature, we need to perform static and dynamic analysis on the sample. Static analysis involves examining the sample without running it, while in dynamic analysis the sample is executed. Since, Cuckoo sandbox and VMRay analyzer already do both the static and dynamic analysis, we only have to infer the results and extract the key behaviour from the analysis.

Obfuscating or packing of a malware sample is one of the widely-used techniques that malware authors use. Using this technique, it's hard to detect malware. One of the important tool to detect this is by using PEiD [31]. If a binary is packed or obfuscated, then it could be malicious in nature. This is already addressed in the proposed system, as the sandboxes gives this information and is one of the parameters that was considered while analyzing the sample. The other parameters that we considered in order to classify whether a binary is malicious or not is if the binary was making any changes to the system

such as abnormal number of files and processes being created, deletion or modification of a system file/service, whether it was disabling any services such as Windows update center or Windows security center, whether it was encrypting user files or changing the master boot record. The other parameter that was focused was while the binary was running if it was establishing any network connections, was it sending data out or receiving any data, was it downloading additional components. Lastly, we also considered if the binary was making any registry changes.

YARA ruleset: In the final step of the proposed system, we are extracting YARA rules to classify whether the binary in question, a false-positive or a malicious in nature. If the binary is malicious in nature then, what is its type and which malware family does it belong to.

YARA ruleset here comes in handy to help classify the sample. It is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples. Using YARA one can create descriptions of malware families (or whatever one want to describe) based on textual or binary patterns [12]. Using this we were able to compare the extracted parameters that we got from static and dynamic analysis with the YARA rules, in order to describe if the binary was malicious and to which family does the malware belong. Below is the YARA rule for Emotet:

```

rule Emotets {
  meta:
  author = "pekeinfo"
  date = "2017-10-18"
  description = "Emotets"
  strings:
  $mz = { 4d 5a }
  $cmovnz={ 0f 45 fb 0f 45 de }

  $mov_esp_0={ C7 04 24 00 00 00 00 89 44 24 0? }

  $_eax={ 89 E? 8D ?? 24 ?? 89 ?? FF D0 83 EC 04 }

  condition:
  ($mz at 0 and $_eax in( 0x2854..0x4000)) and ($cmovnz or $mov_esp_0)
}

```

As we can see the part highlighted in the yellow are the strings that can be found while doing the disassembling of the sample. If we see these strings and if they are according to the condition that is highlighted in blue, we can conclude that the sample belongs to the malware family Emotet.

Chapter 5

Experimental Setup and Implementation

In order to collect binaries through drive-by download, Fig 5-1 shows the proposed architecture.

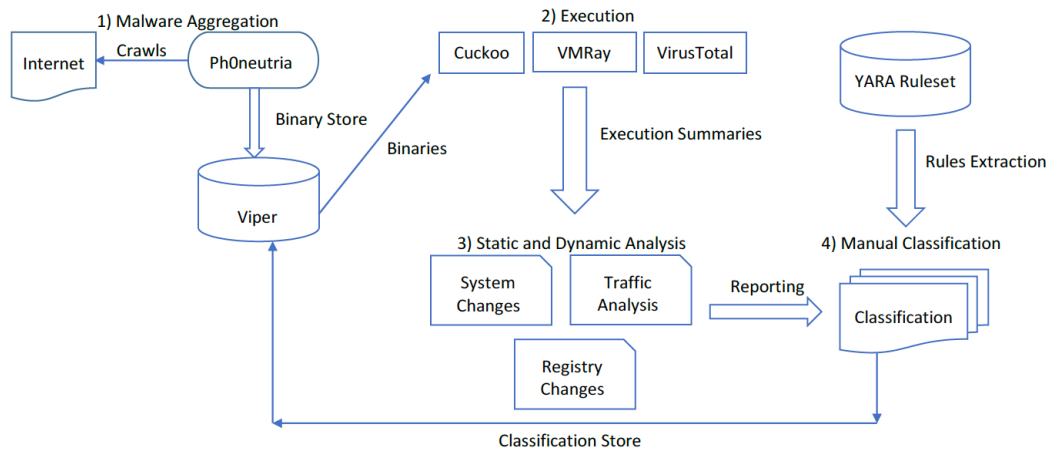


Figure 5-1 Proposed architecture

Ph0neutria is the malware crawler, it is using plugins that contain the websites reference name that it crawls and extract binaries and stores them into the Viper database. The crawler is executed everyday so that it can add newly discovered binaries into the Viper database. Then exporting the binaries to the three sandboxes i.e. Cuckoo, VMRay and VirusTotal (that are available in it) which will give us the analysis report and then performing static and dynamic analysis from those reports to monitor any system changes, traffic analysis and registry changes. Then using the YARA ruleset we are manually classifying each sample with its respective malware family and then storing the result in the Viper database.

In order to capture binaries that maybe malicious in nature, we used a malware crawler called Ph0neutria. We used Ubuntu to execute the crawler. As discussed in the previous chapter ph0neutria has in-built plugins which contain the location of the

websites to crawl and from which to extract binaries. We will now discuss each of those plugins in detail here.

- **CleanMX:** CleanMX is one of the most popular malware website database collected and verified since February, 2006. As of April 9th, 2019, it has a total of 10 million URL's which contains both malicious and clean links. In order to extract the data, one has to register one's user agent.
- **Cymon:** Cymon is one of the largest open trackers of malware, phishing, botnet, spams and many more. This plugin gets its feed from the following websites: Abuse.ch, Bambenek Consulting C2, Cyber Crime Tracker, Malc0de, URLVir and VX Vault. In order to use this plugin, we created our username and password and then updated that in the plugin.
- **Hybrid:** Hybrid analysis is a free online malware analysis service. In other words, it is an online analysis tool which uses the Falcon sandbox. The primary purpose of this plugin in the malware crawler is to check if the malware family is blacklisted and if it is then save the information as a tag on the Viper database.
- **Malshare:** Malshare is a free malware repository.
- **Open Threat Exchange (OTX):** OTX is the largest open threat intelligence community that enables collaborative defense with actionable, community-powered threat data [20].
- **Shodan:** Shodan is a search engine that lets user search various types of computers connected to the Internet. The primary purpose of this plugin is to find the C2 servers or commonly known as Command and Control server hosting malware.

URLhaus which is part of Abuse.ch, used to produce a list of malware URL's.

One of the important feature of Ph0neutria is the ability to enable or disable any plugin. In this experiment, collection of binaries was a crucial part of the research. Ph0neutria has a seamless integration of Viper database which is where the crawler is storing the captured binaries.

Viper Database: Viper is a binary analysis and management framework. It's fundamental objective is to provide a solution to easily organize a collection of malware and exploit samples. It has two types of interfaces: one is the console access and the second is a web based interface. In this research, we used the web based interface because of ease of use. The web based interface is implemented using the open source Python web framework Django [16]. Fig. 5-2 shows the web interface of Viper database.

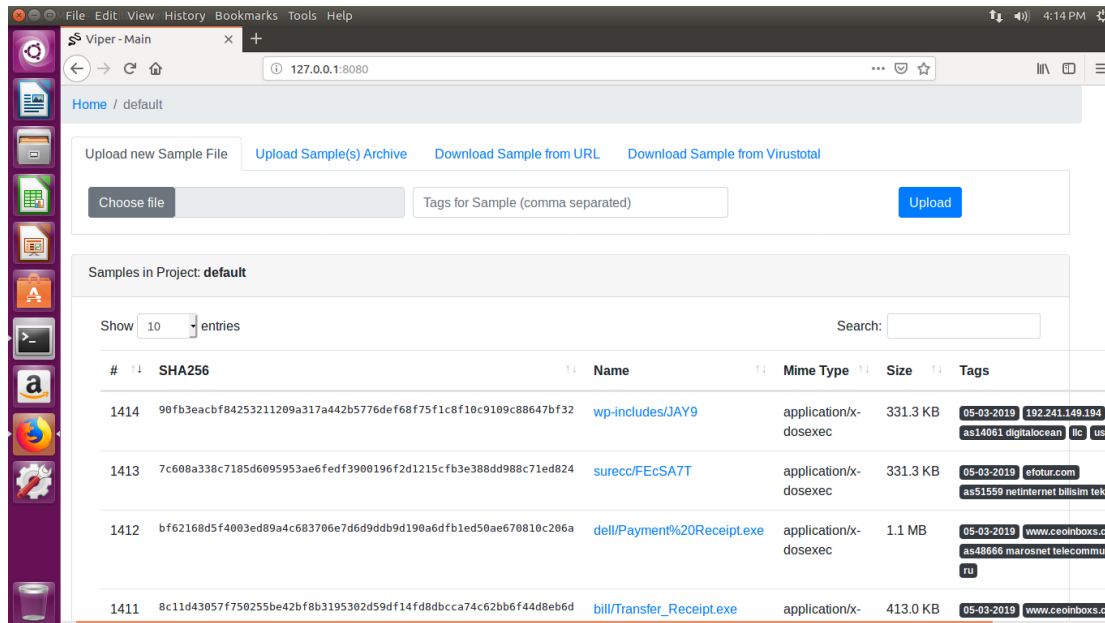


Figure 5-2 Viper web based interface

The web interface gives us the option to create our own projects, add/modify YARA rules. As shown in fig. 5-2, we are using the default project. The binaries that were extracted by the malware crawler are stored here. It is a labeled database with the SHA256 of the binary, the name of the binary, the type, its size and the tags which consists of the date when it was added and the website from which it was extracted.

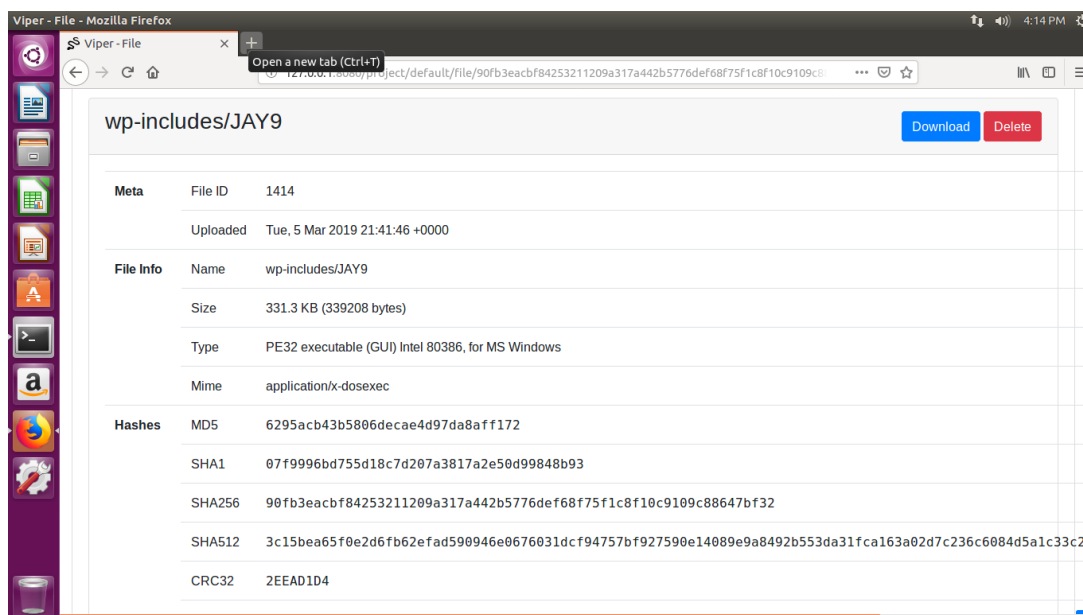


Figure 5-3 Binary information

Fig. 5-3 shows the information that is stored regarding each binary in the database. As one can see, one can download or delete the sample from the database, see the hashes of the binaries that are computed when that binary is being stored in the database. We captured 1414 binaries in total.

Table 5-1 Number of binaries captured

Type	Count
Rich Text Format	21
Zip	175
Executable	959
Microsoft Word Document	200
Microsoft Excel Sheet	2
Octet Stream	57

Table 5-1 shows the types of binaries that were captured and the respective count. One of the interesting aspects of the database are the tags associated with the binaries. One can get information about the binaries such as when was the binary added to the database, the website it was extracted from and which country hosted the website.

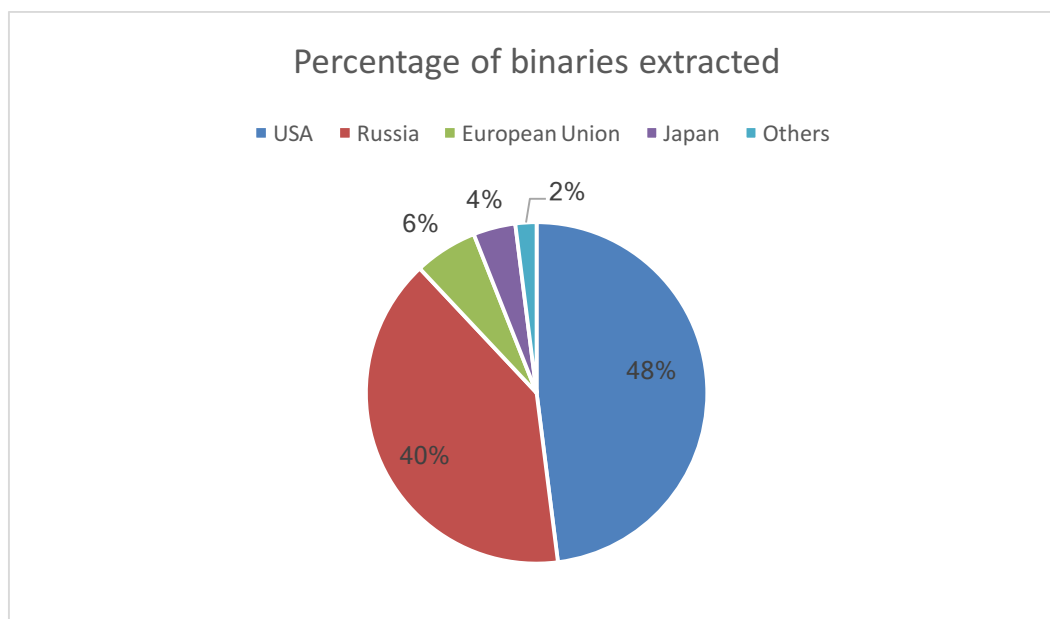


Figure 5-4 Percentage of binaries extracted from different countries

Fig. 5-4 shows the percentage of binaries extracted from different countries, 48% of the binaries were extracted from websites which were hosted in the US. Out of the 1414 binaries that were extracted, 1000 of the binaries were tested in sandboxes. One reason to test 1000 binaries, is to see which of the malware type is most popular and with more binaries under scrutiny that gives us a better idea of the behavior of malware on that OS.

Once the binaries were downloaded on the local machine, the binaries were then executed in Cuckoo sandbox and VMRay Cloud Analyzer. In order to have uniformity in the execution operating system, we selected Windows 7 SP 2 (32 bit) for both Cuckoo and VMRay. We used Windows 7 (32 bit) for Cuckoo sandbox, because it works well with it. One of the tricks that malware authors employ is to check whether they are being executed on a sandbox is to see whether there is any fake network being setup using InetSim [21]. In order to minimize that possibility, we configured the sandboxes to host-only, which will still give the binaries limited DNS, IRC and HTTP traffic so that they can communicate with C2 server making it look like a genuine network.

Each binary was executed and immediately afterwards the execution was over, a report was generated and the sandboxes were rolled back to its original clean state. Upon

getting the report, the main focus was to extract behavior of the binaries and also to classify on a preliminary stage whether the binary was a false-positive or malicious in nature which will be explained in detail in chapter 6.

The final stage was to classify the binary to see whether it was malicious or not, by using YARA rules. They were applied to the summaries which were extracted from the reports generated by running in the sandboxes. If the binary was malicious in nature, we were looking at the behavior of the malware to identify what type of malware it is which is outlined in chapter 1 and the family to which the malware belongs.

Chapter 6

Observations

We started the malware aggregation phase from the 12th of January 2019 and we ended the aggregation phase on 5th of March 2019 when there were no new binaries being discovered and stored in the database. In the malware aggregation phase we were able to capture and store 1414 binaries. Fig. 6-1 shows the number of websites that were crawled in 52 days.

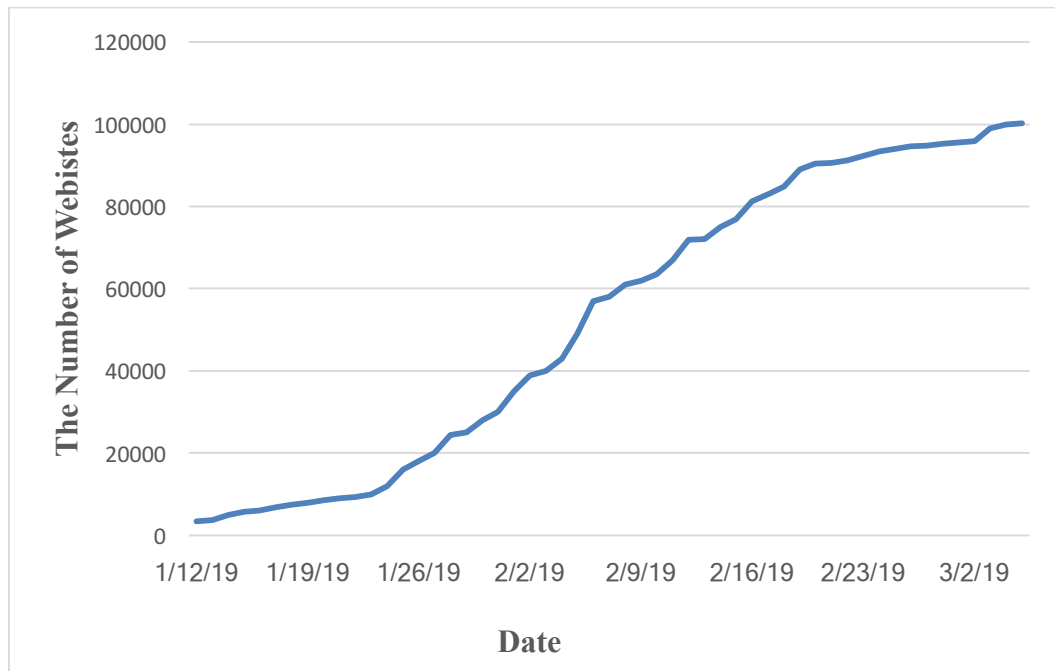


Figure 6-1 Number of websites that were crawled

As we can see from fig. 6-1, there is an exponential growth in the number of websites that were crawled.

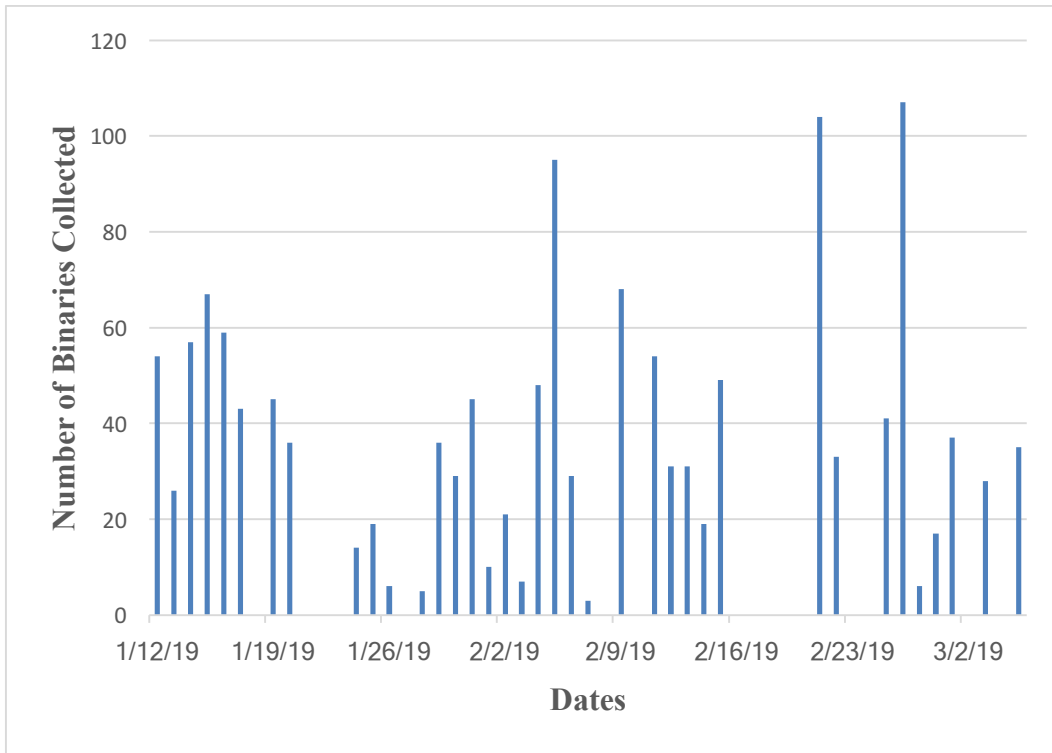


Figure 6-2 Number of binaries collected

Fig. 6-2 shows the number of binaries that were collected and stored over a period of 52 days. Fig. 6-3 shows the working of ph0neutria which; is first checking the hash of the binary that it just pulled with the existing entries, and if it finds that, then it is deleting the binary, if not, then it will add that binary to the database.

```
root@ubuntu: /opt/ph0neutria
2019-02-13 13:37:36 ubuntu core.file_utils[3820] INFO Removing file: /opt/ph0neutria/tmp/GSGYs7PDes7AKtkthCc3Gjda4vZlFUT
2019-02-13 13:37:36 ubuntu core.malware_utils[3820] WARNING URL: http://34.220.101.62/US/Invoice/yDnsy-UffTS_ZK-Iy/ did not successfully download. Continuing...
2019-02-13 13:37:36 ubuntu core.file_utils[3820] INFO Adding to cache: http://206.189.154.46/En_us/Info/New_invoice/tpDs-xIodr_VDgMFSO-s9d/
2019-02-13 13:37:36 ubuntu core.file_utils[3820] INFO Making HEAD request to: http://206.189.154.46/En_us/Info/New_invoice/tpDs-xIodr_VDgMFSO-s9d/
2019-02-13 13:37:38 ubuntu core.file_utils[3820] INFO HEAD request to http://206.189.154.46/En_us/Info/New_invoice/tpDs-xIodr_VDgMFSO-s9d/ did not return a Content-Length header. Attempting GET.
2019-02-13 13:37:40 ubuntu core.file_utils[3820] INFO Downloaded as temporary file: /opt/ph0neutria/tmp/rWSG18MPq9lEothUWHZysFYvsZEedFf. Beginning processing...
2019-02-13 13:37:40 ubuntu core.file_utils[3820] INFO File with hash 87476cb142b08b99b38551267bc4c4012d3878b5dd3e12ddcc6e640df0248cc0 identified as type: application/xml
2019-02-13 13:37:40 ubuntu core.file_utils[3820] INFO Removing file: /opt/ph0neutria/tmp/rWSG18MPq9lEothUWHZysFYvsZEedFf
2019-02-13 13:37:40 ubuntu core.malware_utils[3820] WARNING URL: http://206.189.154.46/En_us/Info/New_invoice/tpDs-xIodr_VDgMFSO-s9d/ did not successfully download. Continuing...
2019-02-13 13:37:40 ubuntu core.file_utils[3820] INFO Adding to cache: http://3.120.147.8/download/9428618769/sary-0cZ_cEYzUU-2u/
2019-02-13 13:37:40 ubuntu core.file_utils[3820] INFO Making HEAD request to: http://3.120.147.8/download/9428618769/sary-0cZ_cEYzUU-2u/
2019-02-13 13:37:43 ubuntu core.file_utils[3820] INFO HEAD request to http://3.120.147.8/download/9428618769/sary-0cZ_cEYzUU-2u/ did not return a Content-Length header. Attempting GET.
2019-02-13 13:37:43 ubuntu core.file_utils[3820] INFO Downloaded as temporary file: /opt/ph0neutria/tmp/embDGPKxZp0fZUPhqf1bPVSCGARxEXgv. Beginning processing...
2019-02-13 13:37:43 ubuntu core.file_utils[3820] INFO File with hash 7d13b50b4660f44796587f9c06cc69f08e4b42a6b72a841206e8d29f768054d identified as type: application/xml
2019-02-13 13:37:43 ubuntu core.file_utils[3820] INFO Removing file: /opt/ph0neutria/tmp/embDGPKxZp0fZUPhqf1bPVSCGARxEXgv
2019-02-13 13:37:43 ubuntu core.malware_utils[3820] WARNING URL: http://3.120.147.8/download/9428618769/sary-0cZ_cEYzUU-2u/ did not successfully download. Continuing...
2019-02-13 13:37:43 ubuntu core.file_utils[3820] INFO Adding to cache: http://18.223.20.43/EN_en/xerox/Invoice_number/LaejY-Xt_sgrNPE-YD/
2019-02-13 13:37:43 ubuntu core.file_utils[3820] INFO Making HEAD request to: http://18.223.20.43/EN_en/xerox/Invoice_number/LaejY-Xt_sgrNPE-YD/
2019-02-13 13:37:44 ubuntu core.file_utils[3820] INFO HEAD request to http://18.223.20.43/EN_en/xerox/Invoice_number/LaejY-Xt_sgrNPE-YD/ did not return a Content-Length header. Attempting GET.
2019-02-13 13:37:44 ubuntu core.file_utils[3820] INFO Downloaded as temporary file: /opt/ph0neutria/tmp/LZp7FaDnfx8HnrR9WX1b6xZUjoOrriGq. Beginning processing...
2019-02-13 13:37:47 ubuntu core.file_utils[3820] INFO File with hash 87476cb142b08b99b38551267bc4c4012d3878b5dd3e12ddcc6e640df0248cc0 identified as type: application/xml
2019-02-13 13:37:47 ubuntu core.file_utils[3820] INFO Removing file: /opt/ph0neutria/tmp/LZp7FaDnfx8HnrR9WX1b6xZUjoOrriGq
2019-02-13 13:37:47 ubuntu core.malware_utils[3820] WARNING URL: http://18.223.20.43/EN_en/xerox/Invoice_number/LaejY-Xt_sgrNPE-YD/ did not successfully download. Continuing...
2019-02-13 13:37:47 ubuntu core.file_utils[3820] INFO Adding to cache: http://18.221.1.168/corporation/Rthgy-VE_DqQJ-lP/
2019-02-13 13:37:47 ubuntu core.file_utils[3820] INFO Making HEAD request to: http://18.221.1.168/corporation/Rthgy-VE_DqQJ-lP/
2019-02-13 13:37:48 ubuntu core.file_utils[3820] INFO HEAD request to http://18.221.1.168/corporation/Rthgy-VE_DqQJ-lP/ did not return a Content-Length header. Attempting GET.
```

Figure 6-3 Ph0neutria working

After collecting and storing the binaries in the Viper database, we looked for samples that have the same name but different hashes. The aim was to see if they were extracted from the same website and if they were, then when and how many times the hash value was changed. The goal was to identify whether malware authors were monitoring the traffic and/or creating new malware, with the same name. We were able to identify a total of 105 binaries which had the same name and they were extracted from the same website but had different hashes. The mean difference between the first store of binary and the last store of the same binary but with different hashes was found to be 17 days.

After the samples were analyzed for the hashes, they were executed in the sandboxes.

Cuckoo sandbox: The binaries were uploaded to the sandbox using the default settings that were previously set. In this we used Internet routing so that the binary is tricked that it is being executed on a user machine rather than in a sandbox.

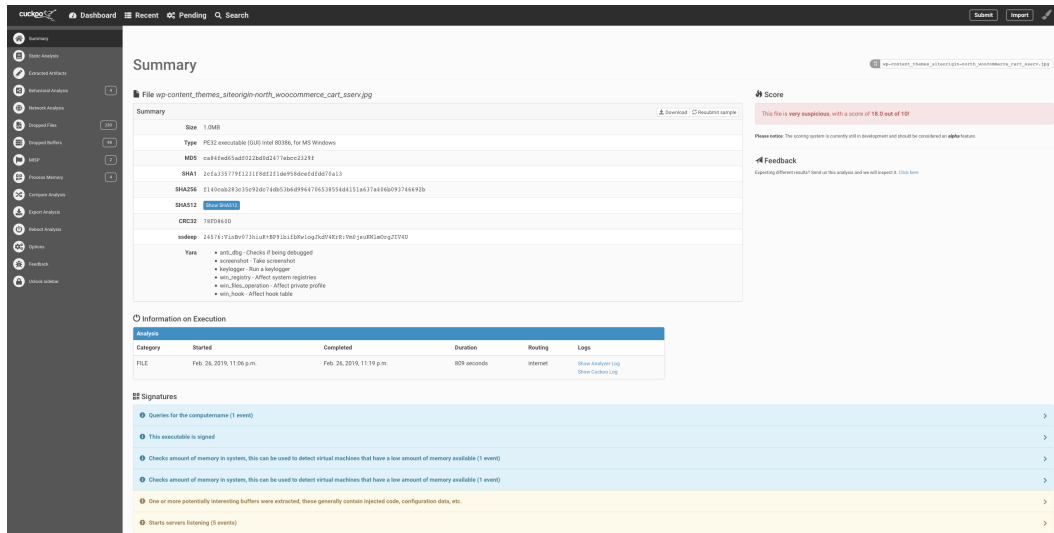


Figure 6-4 Cuckoo sandbox result

Fig. 6-4 shows the summary from executing the sample. We can see that one is able to get the details regarding the sample like the size, type, various hashes and if it was able to match with any YARA rules. In addition to this one can see the duration (time) the binary executed and one can also get the screenshots from the execution which are shown in fig. 6-5.

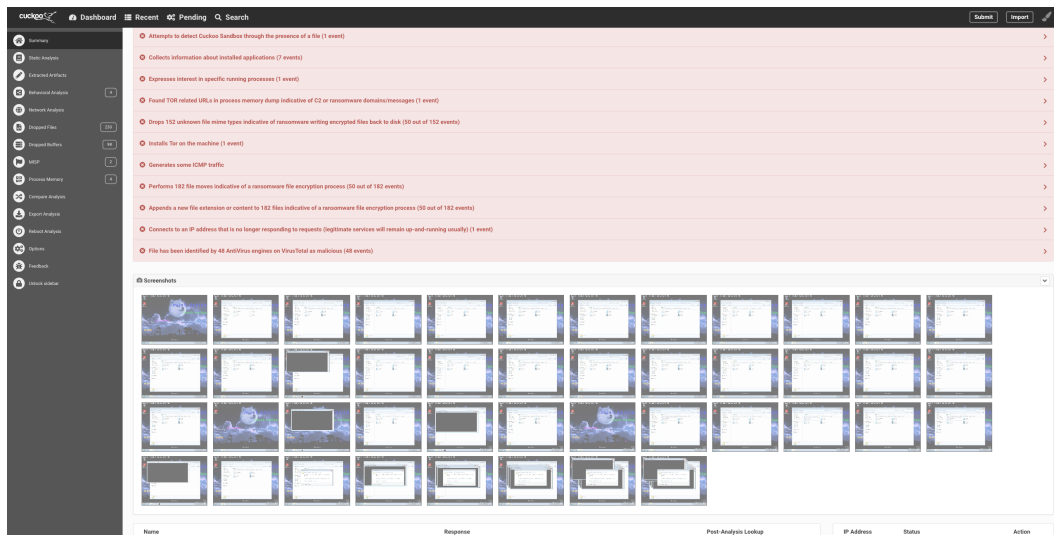


Figure 6-5 Screenshots of the execution of binary

One of the classical indicators whether the sample is encrypted or not is entropy. Entropy is the randomness collected by an operating system or application for use in

cryptography or other uses that require random data. Encrypted or compressed data/file has a higher value of entropy whereas a typical file/data will have a lower value of entropy.

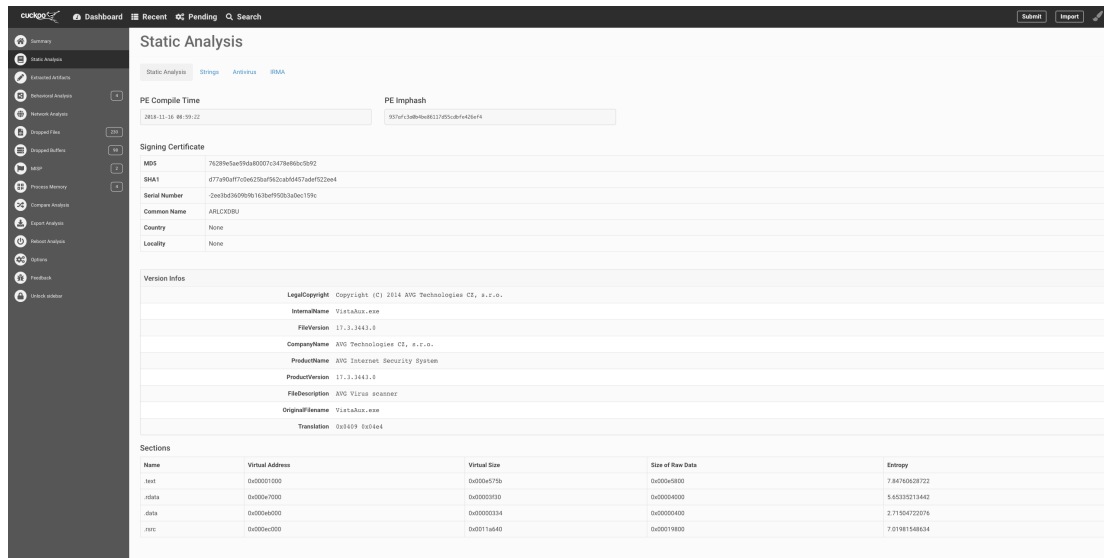


Figure 6-6 Static analysis of a binary

Fig.6-6 shows the static analysis of the binary. We can see that the .text and .rsrc sections have a higher entropy than the other section so we can say that the binary may be encrypted. In order to confirm this, we can see the strings. Strings gives us information such as the DLL's being imported by the sample and any other significant thing such as if the binary is trying to connect to a remote server or if the binary is collecting user's keystroke. DLL is a dynamic link library file format used for holding multiple codes and procedures for Windows programs. DLL files were created so that multiple programs could use their information at the same time, aiding memory conservation [37]. Fig. 6-7 shows the strings output of the above sample.

sample doing to the system after it was executed. Fig 6-9 shows some of the signatures that were identified.

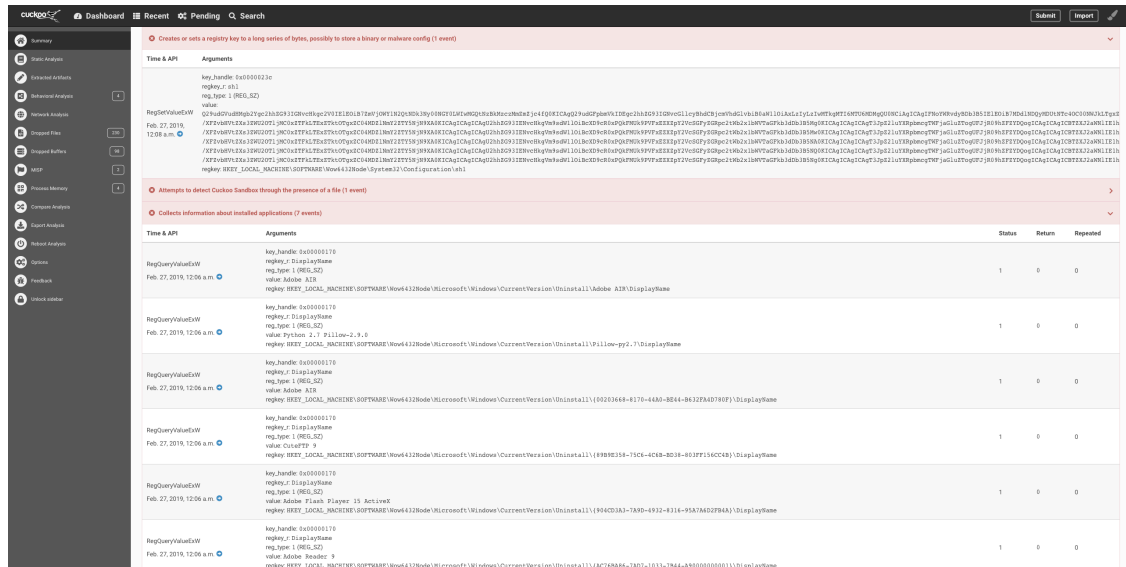


Figure 6-9 Signatures of the binary

As shown in fig.6-9, using these signatures we can make some assumptions regarding what the binary is trying to do and we can make an assumption that the binary belongs to this malware type.

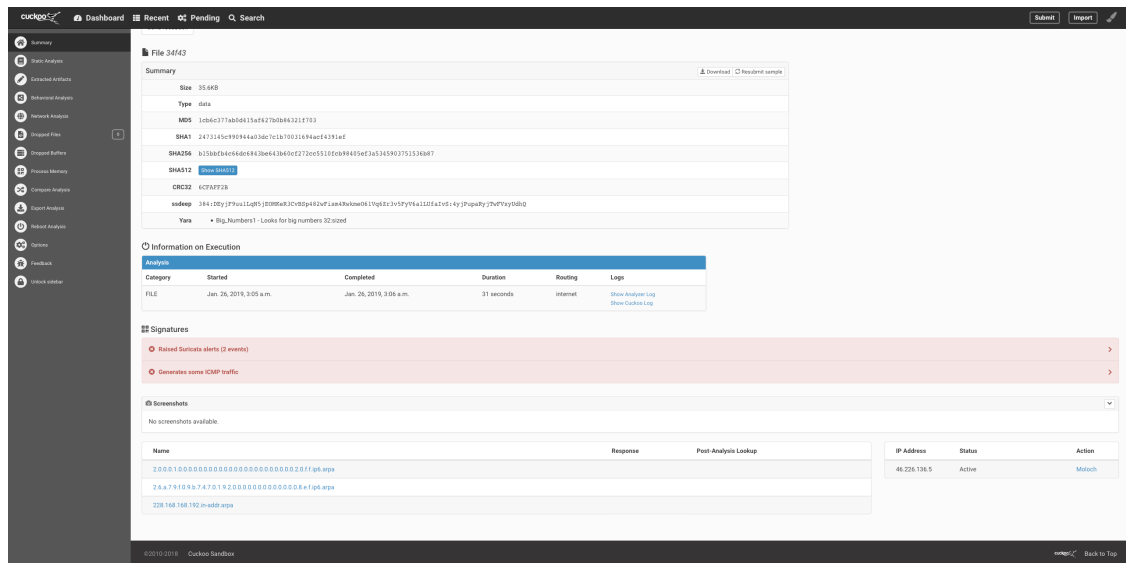


Figure 6-10 Report generated from a false-positive sample

Fig. 6-10 shows the report from a false-positive sample. As we can see that there are no screenshots of the execution of the sample as well as there are no meaningful signatures that can give an idea regarding the working of the sample.

VMRay Cloud Analyzer: One of the advantage of using VMRay is that it gives us access to VirusTotal through API key which is provided free of cost by VirusTotal. As mentioned earlier, we are also using the sandboxes that are provided by VirusTotal.

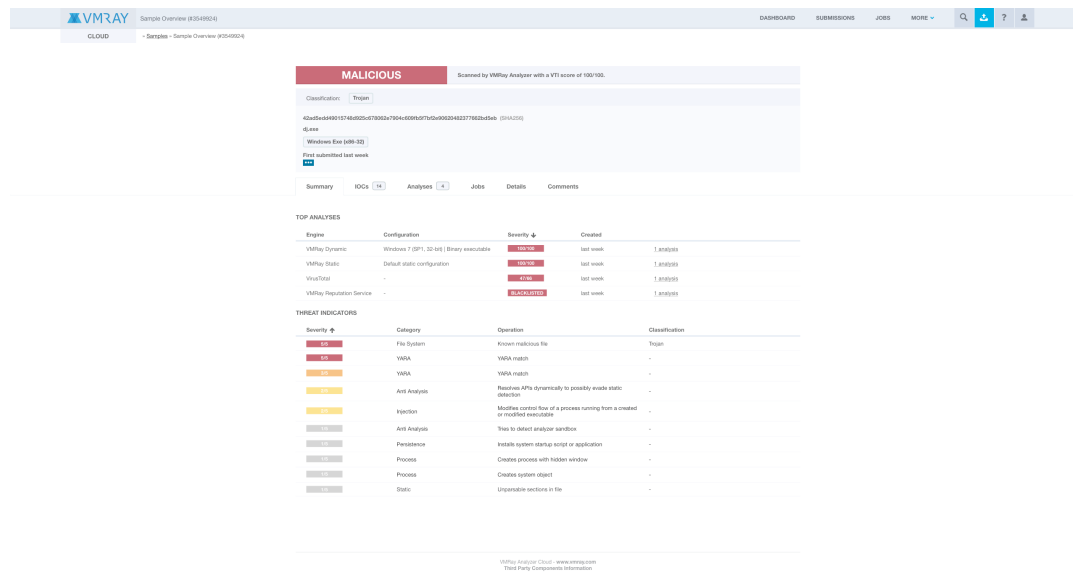


Figure 6-11 Report generated from VMRay

As we can see from fig. 6-11, we can see the report generated by VMRay. We can see that there are different analysis that this sandbox provides such as the dynamic analysis, static analysis, report from VirusTotal and the reputation report from VMRay. We can also see the various threat indicator which are same as what we saw in the report that was generated by Cuckoo sandbox. Fig. 6-12 shows the dynamic analysis report.

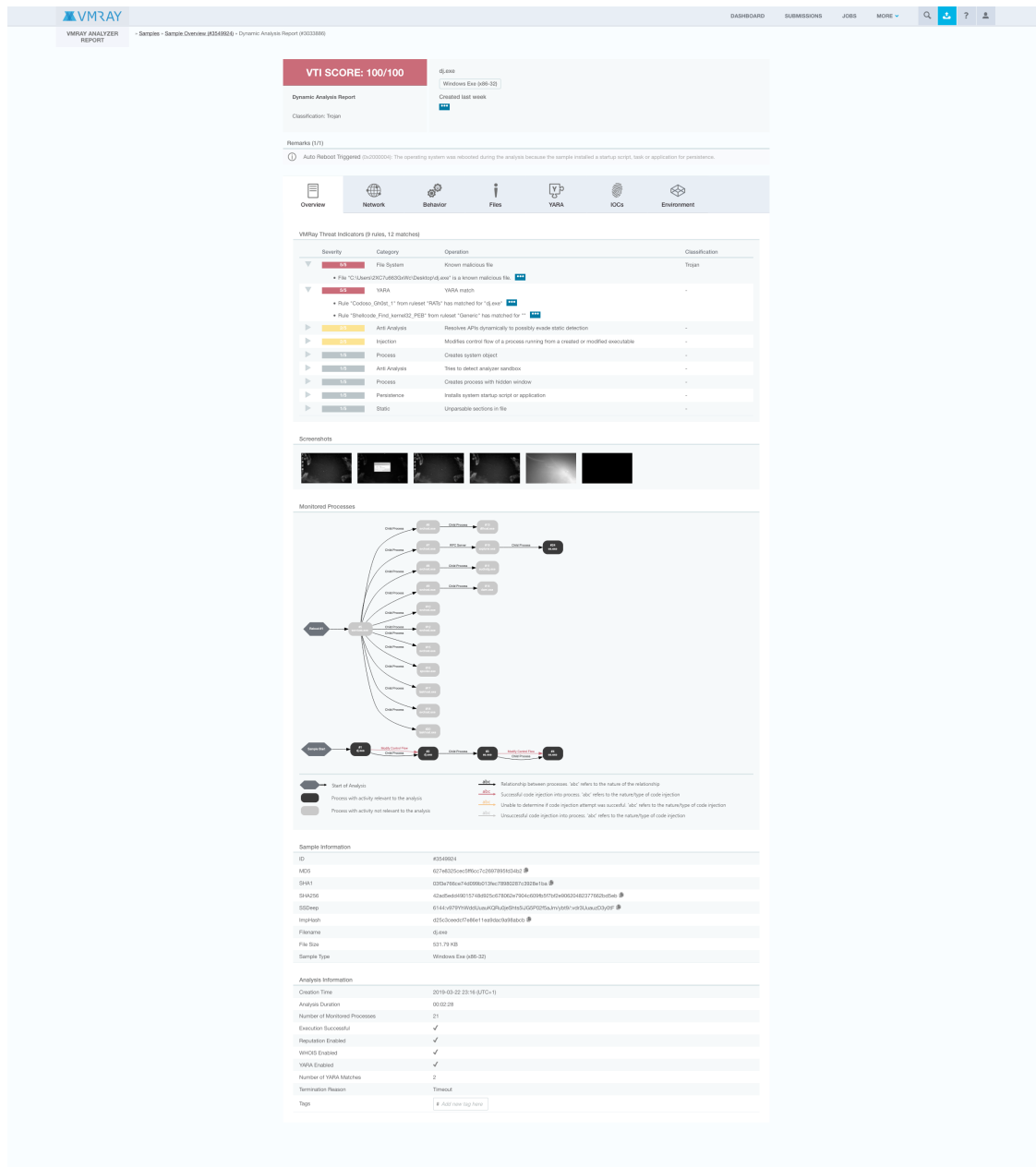


Figure 6-12 Dynamic analysis report

As we can see in fig. 6-12, we can see the screenshot of the execution of the binary, we can see the processes that the malware created while executing and then we can see the information about the sample.

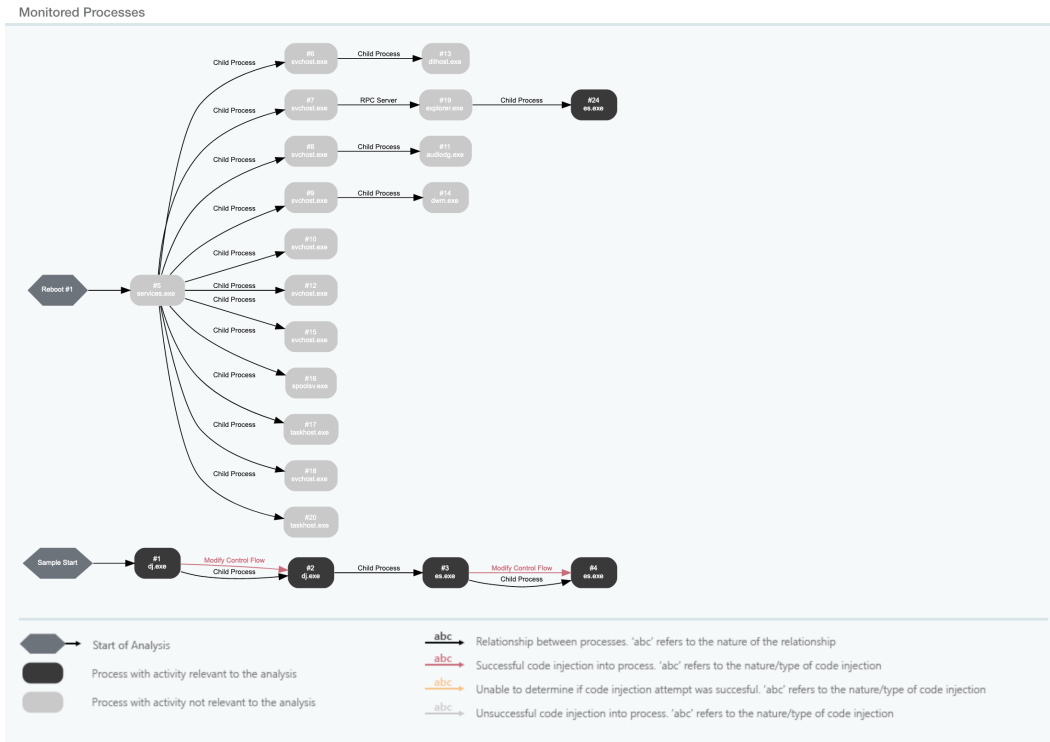


Figure 6-13 Process tree from the execution of binary

Fig. 6-13 shows the process that were created from the execution of the binary. Using this, we can go to the process and see what the process was doing and what changes it made to the system. Fig. 6-14 shows the information that was given about the process “dj.exe”.

Process #1: dj.exe

Information	Value
ID	#1
File Name	c:\users\2xc7u663gxcw\desktop\dj.exe
Command Line	"C:\Users\2XC7u663GxWc\Desktop\dj.exe"
Initial Working Directory	C:\Users\2XC7u663GxWc\Desktop\
Monitor	Start Time: 00:00:16, Reason: Analysis Target
Unmonitor	End Time: 00:00:36, Reason: Self Terminated
Monitor Duration	00:00:19

>> OS Process Information
 >> Memory Dumps
 >> Dropped Files

Host Behavior

>> File (6)
 >> Registry (2)
 >> Process (1)
 >> Thread (3)
 >> Module (142)
 >> Window (14)
 >> Keyboard (1)
 >> System (13)
 >> Mutex (1)
 >> Environment (1)

Figure 6-14 Process information

We can get information such as the dropped files which are files that the malware might use when it is executing, the memory dump and we can get information about the host such as what registry did the malware query or what processes did it create. Fig. 6-15 shows the same.

VMRAY DASHBOARD SUBMISSIONS JOBS MORE

Monitor Duration: 00:00:19

>> OS Process Information
 >> Memory Dumps
 >> Dropped Files

Filename	File Size	Hash Values	VMRay Match	Actions
c:\users\2xc7u663gxcw\desktop\temp-18715aawckb8h7s.tmp	18.00 KB	MD5: 191344756b13028ac4d114c037d97f SHA1: 4c2c71e070105014e05010410332e0b7c7c7c SHA256: 313c2c4e9567f4a6d48464c2d62d0391475416c88-3ac2333 File ID: 852960 19175D5976UACD7848B84911496b487C2L_H0C2824L	X	🔍

Host Behavior

>> File (6)

Operation	Filename	Additional Information	Success	Count	Logfile
Get Info	STD_INPUT_HANDLE	type = file_type	X	1	[FILE]
Get Info	STD_OUTPUT_HANDLE	type = file_type	X	1	[FILE]
Get Info	STD_ERROR_HANDLE	type = file_type	X	1	[FILE]
Open	STD_INPUT_HANDLE	-	✓	1	[FILE]
Open	STD_OUTPUT_HANDLE	-	✓	1	[FILE]
Open	STD_ERROR_HANDLE	-	✓	1	[FILE]

>> Registry (2)

Operation	Key	Additional Information	Success	Count	Logfile
Open Key	HKY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows	-	X	2	[FILE]

>> Process (1)

Operation	Process	Additional Information	Success	Count	Logfile
Create	C:\Users\2XC7u663GxWc\Desktop\dj.exe	os_pid = 3403, creator_pid = CREATE_SUSPENDED, show_window = 0, PID: 3403	✓	1	[FILE]

>> Thread (3)

Operation	Process	Additional Information	Success	Count	Logfile
Get Control	C:\Users\2xc7u663gxcw\Desktop\dj.exe	os_pid = 3403	✓	1	[FILE]
Get Control	C:\Users\2xc7u663gxcw\Desktop\dj.exe	os_pid = 3403	✓	1	[FILE]
Resume	C:\Users\2xc7u663gxcw\Desktop\dj.exe	os_pid = 3403	✓	1	[FILE]

Figure 6-15 Information about the execution of a binary

As we saw in fig. 6-11, we can make some classification about the binary, such as what is the type of malware.

The screenshot shows the VirusTotal Analysis interface. At the top, there is a header "VirusTotal Analysis" and a button that says "Click here to open VirusTotal analysis page". Below this is a table with four columns: "AV Product", "Name", "Version", and "Update". The table lists various antivirus products and their detection results for a specific binary.

AV Product	Name	Version	Update
ALYac	Trojan.GenericKD.31697725	1.1.1.5	20190305
AVG	Win32:Malware-gen	18.4.3895.0	20190305
Acronis	suspicious	1.0.1.40	20190222
Ad-Aware	Trojan.GenericKD.31697725	3.0.5.370	20190305
AegisLab	-	4.2	20190305
AhnLab-V3	Trojan/Win32.Injector.R255715	3.14.1.22785	20190305
Alibaba	-	0.1.0.2	20180921
Antiy-AVL	Trojan/Win32.VBKryjetor	3.0.0.1	20190305
Arcabit	Trojan.Generic.D1E3AB3D	1.0.0.837	20190305
Avast	Win32:Malware-gen	18.4.3895.0	20190305
Avast-Mobile	-	190305-02	20190305
Avira	HEUR/AGEN.1039208	8.3.3.8	20190305
Babable	-	9107201	20180918
Baidu	-	1.0.0.2	20190215
BitDefender	Trojan.GenericKD.31697725	7.2	20190305
Bkav	-	1.3.0.9899	20190304
CAT-QuickHeal	Trojan.Multi	14.00	20190304
CMC	-	1.1.0.977	20190305
ClamAV	Win.Trojan.VBGeneric-6862313-0	0.101.1.0	20190305
Comodo	Malware@#bvd2tos1wwk3	30525	20190305
CrowdStrike	win/malicious_confidence_80% (W)	1.0	20190212
Cybereason	-	1.2.27	20190109

Figure 6-16 Information from VirusTotal

As mentioned earlier, fig. 6-16 shows the information that we got from VirusTotal. We can see the entire report by clicking on the tab. In order to get the final classification regarding the binary, we extracted the function log from VMRay. Fig. 6-17 shows part of the report.

<pre> lpUsedDefaultChar=0x0 returned 1149 [0019.988] RtlAllocateHeap (HeapHandle=0x2a0000, Flags=0x0, Size=0x400) returned 0x2a0700 [0019.989] WideCharMultiByte (In: CodePage=0x0, dwFlags=0x0, lpWideCharStr=ALLUSERSPROFILEC:\ProgramData, cchWideChar=1149, lpMultiByteStr=0x2a0700, cbMultiByte=1149, lpDefaultChar=0x0, lpUsedDefaultChar=0x0 out: lpMultiByteStr=ALLUSERSPROFILEC:\ProgramData, lpUsedDefaultChar=0x0) returned 1149 [0019.988] FreeEnvironmentStringsW (penv=0x121900) returned 1 [0019.988] RtlAllocateHeap (HeapHandle=0x2a0000, Flags=0x0, Size=0x400) returned 0x2a0c00 [0019.988] GetStartupInfoA (In: lpStartupInfo=0x12f904 out: lpStartupInfo=0x12f904, lpReserved=0, lpDesktop=WINSTA0\Default, lpTitle=C:\Users\12c7d663poc\Desktop\1\1.exe, dwX=0x0, dwY=0x0, dwXSize=0x0, dwYSize=0x0, dwCountChars=0x0, dwFILLtrIbute=0x0, dwFlags=0x1, wShowWindow=0x1, cbReserved=0x0, lpReserved=0x0, hStdInput=0xffffffff, hStdOutput=0xffffffff) returned 0x0 [0019.988] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0019.988] GetFileType (hFile=0x0) returned 0x0 [0019.989] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0019.989] GetFileType (hFile=0x0) returned 0x0 [0019.989] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0019.989] GetFileType (hFile=0x0) returned 0x0 [0019.989] SetHandleCount (lNumber=0x20) returned 0x20 [0019.989] GetACP () returned 0x64 [0019.989] GetCPInfo (In: CodePage=0x64, lpCPInfo=0x12f9fc out: lpCPInfo=0x12f9fc) returned 1 [0019.989] GetModuleFileNameA (In: hModule=0x0, lpFileName=0x724c528, nSize=0x104 out: lpFileName=C:\Users\12c7d663poc\Desktop\1\1.exe (normalized: C:\Users\12c7d663poc\Desktop\1\1.exe)) returned 0x24 [0019.990] HeapFree (In: hHeap=0x2a0000, dwFlags=0x0, lpMem=0x2a0700 out: Heap=0x2a0000) returned 1 [0019.990] GetModuleHandleA (lpModuleName=KERNEL32) returned 0x75aa0000 [0019.990] GetProcAddress (Module=0x75aa0000, lpProcName=IProcessorFeaturePresent) returned 0x75af7005 [0019.990] IProcessorFeaturePresent (ProcessorFeature=0x0) returned 0 [0019.990] RtlAllocateHeap (HeapHandle=0x2a0000, Flags=0x0, Size=0x800) returned 0x2a1000 [0019.991] CreateEventA (lpEventAttributes=0x0, lpManualReset=1, lpInitialState=1, lpName=0x0) returned 0x0 [0019.991] CreateMutexA (lpMutexAttributes=0x0, lpName=0x0) returned 0x0 [0019.991] InitializeObjectEx (lpName=0x0) returned 0x0 [0019.991] GetModuleHandleA (lpModuleName=0x0) returned 0x400000 [0019.991] GetModuleFileNameA (In: hModule=0x72940000, lpFileName=0x72a6c8, nSize=0x104 out: lpFileName=C:\Windows\ </pre>	<pre> lpUsedDefaultChar=0x0 returned 1149 [0019.988] RtlAllocateHeap (HeapHandle=0x2a0000, Flags=0x0, Size=0x400) returned 0x2a0700 [0019.989] WideCharMultiByte (In: CodePage=0x0, dwFlags=0x0, lpWideCharStr=ALLUSERSPROFILEC:\ProgramData, cchWideChar=1149, lpMultiByteStr=0x2a0700, cbMultiByte=1149, lpDefaultChar=0x0, lpUsedDefaultChar=0x0 out: lpMultiByteStr=ALLUSERSPROFILEC:\ProgramData, lpUsedDefaultChar=0x0) returned 1149 [0019.988] FreeEnvironmentStringsW (penv=0x121900) returned 1 [0019.988] RtlAllocateHeap (HeapHandle=0x2a0000, Flags=0x0, Size=0x400) returned 0x2a0c00 [0019.988] GetStartupInfoA (In: lpStartupInfo=0x12f904 out: lpStartupInfo=0x12f904, lpReserved=0, lpDesktop=WINSTA0\Default, lpTitle=C:\Users\12c7d663poc\Desktop\1\1.exe, dwX=0x0, dwY=0x0, dwXSize=0x0, dwYSize=0x0, dwCountChars=0x0, dwFILLtrIbute=0x0, dwFlags=0x1, wShowWindow=0x1, cbReserved=0x0, lpReserved=0x0, hStdInput=0xffffffff, hStdOutput=0xffffffff) returned 0x0 [0019.988] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0019.988] GetFileType (hFile=0x0) returned 0x0 [0019.989] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0019.989] GetFileType (hFile=0x0) returned 0x0 [0019.989] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0019.989] GetFileType (hFile=0x0) returned 0x0 [0019.989] SetHandleCount (lNumber=0x20) returned 0x20 [0019.989] GetACP () returned 0x64 [0019.989] GetCPInfo (In: CodePage=0x64, lpCPInfo=0x12f9fc out: lpCPInfo=0x12f9fc) returned 1 [0019.989] GetModuleFileNameA (In: hModule=0x0, lpFileName=0x724c528, nSize=0x104 out: lpFileName=C:\Users\12c7d663poc\Desktop\1\1.exe (normalized: C:\Users\12c7d663poc\Desktop\1\1.exe)) returned 0x24 [0019.990] HeapFree (In: hHeap=0x2a0000, dwFlags=0x0, lpMem=0x2a0700 out: Heap=0x2a0000) returned 1 [0019.990] GetModuleHandleA (lpModuleName=KERNEL32) returned 0x75aa0000 [0019.990] GetProcAddress (Module=0x75aa0000, lpProcName=IProcessorFeaturePresent) returned 0x75af7005 [0019.990] IProcessorFeaturePresent (ProcessorFeature=0x0) returned 0 [0019.990] RtlAllocateHeap (HeapHandle=0x2a0000, Flags=0x0, Size=0x800) returned 0x2a1000 [0019.991] CreateEventA (lpEventAttributes=0x0, lpManualReset=1, lpInitialState=1, lpName=0x0) returned 0x0 [0019.991] CreateMutexA (lpMutexAttributes=0x0, lpName=0x0) returned 0x0 [0019.991] InitializeObjectEx (lpName=0x0) returned 0x0 [0019.991] GetModuleHandleA (lpModuleName=0x0) returned 0x400000 [0019.991] GetModuleFileNameA (In: hModule=0x72940000, lpFileName=0x72a6c8, nSize=0x104 out: lpFileName=C:\Windows\ </pre>
<pre> [0055.519] malloc:Alloc (This=0x754566bc, cb=0xc) returned 0x698500 [0055.519] GetCurrentThreadId () returned 0x0 [0055.519] SetWindowsHookExA (ldHook=1, lpfn=0x729a1e09, hmod=0x0, dwThreadId=0x0) returned 0x120141 [0055.519] RtlAllocateHeap (HeapHandle=0x1bb0000, Flags=0x0, Size=0x14) returned 0x1bb2078 [0055.519] RtlAllocateHeap (HeapHandle=0x1bb0000, Flags=0x0, Size=0x80) returned 0x1bb2090 [0055.519] RtlAllocateHeap (HeapHandle=0x1bb0000, Flags=0x0, Size=0x10) returned 0x1bb2120 [0055.519] RtlAllocateHeap (HeapHandle=0x1bb0000, Flags=0x0, Size=0x2c) returned 0x1bb2130 [0055.519] GetClassInfo (In: Instance=0x72940000, lpClassName=VBMSvcComMgr, lpWndClass=0x12f9dc) out: lpWndClass=0x12f9dc) returned 0 [0055.520] RegisterClassA (lpWndClass=0x12f9dc) returned 0x7ac137 [0055.520] CreateWindowExA (dwStyle=0x0, lpClassName=VBMSvcComMgr, lpModuleName=0x0, dwStyle=0x80000000, X=2147483648, Y=2147483648, nWidth=2147483648, nHeight=2147483648, hWndParent=0x0, hMenu=0x0, hInstance=0x72940000, lpParam=0x0) returned 0x6004c [0055.520] DeWindowProcA (hWnd=0x6004c, Msg=0x81, wParam=0x0, lParam=0x12f904) returned 0 [0055.520] DeWindowProcA (hWnd=0x6004c, Msg=0x83, wParam=0x0, lParam=0x12f904) returned 0 [0055.520] DeWindowProcA (hWnd=0x6004c, Msg=0x1, wParam=0x0, lParam=0x12f904) returned 0 [0055.520] DeWindowProcA (hWnd=0x6004c, Msg=0x5, wParam=0x0, lParam=0x0) returned 0 [0055.520] DeWindowProcA (hWnd=0x6004c, Msg=0x3, wParam=0x0, lParam=0x0) returned 0 [0055.520] SetWindowLongA (hWnd=0x6004c, nIndex=0, dwNewLong=29040790) returned 0 [0055.520] RtlAllocateHeap (HeapHandle=0x1bb0000, Flags=0x0, Size=0x30) returned 0x1bb2170 [0055.520] RtlAllocateHeap (HeapHandle=0x1bb0000, Flags=0x0, Size=0x10) returned 0x1bb2180 [0055.520] RtlAllocateHeap (HeapHandle=0x1bb0000, Flags=0x0, Size=0x10) returned 0x1bb2190 [0055.520] RegisterClipboardFormatA (lpzFormat=Object Descriptor) returned 0x0 [0055.520] RegisterClipboardFormatA (lpzFormat=Link Source Descriptor) returned 0x0 [0055.520] RegisterClipboardFormatA (lpzFormat=Embed Source) returned 0xc0bb [0055.520] RegisterClipboardFormatA (lpzFormat=Embedded Object) returned 0xc0ba [0055.520] RegisterClipboardFormatA (lpzFormat=Link Source) </pre>	<pre> Thread: id = 5 os_tid = 0xcdc [0068.958] GetVersion () returned 0x1db10186 [0068.959] GetModuleHandleA (lpModuleName=kernel32.dll) returned 0x75aa0000 [0068.959] GetProcAddress (Module=0x75aa0000, lpProcName=IsNT) returned 0 [0068.959] HeapCreate (lOptions=0x0, dwInitiaSize=0x1800, dwMaximumSize=0x0) returned 0x150000 [0068.959] VirtualAlloc (lpAddress=0x0, dwSize=0x400000, flAllocationType=0x2000, flProtect=0x4) returned 0x11a0000 [0068.959] VirtualAlloc (lpAddress=0x11a0000, dwSize=0x10000, flAllocationType=0x1000, flProtect=0x4) returned 0x11a0000 [0068.961] GetCurrentThreadId () returned 0x0 [0068.961] GetCommandLineA () returned=\\0\1\ProgramData\ves.exe [0068.961] GetEnvironmentStringsW () returned 0x171920 [0068.961] WideCharMultiByte (In: CodePage=0x0, dwFlags=0x0, lpWideCharStr=ALLUSERSPROFILEC:\ProgramData, cchWideChar=1149, lpMultiByteStr=0x0, cbMultiByte=0, lpDefaultChar=0x0, lpUsedDefaultChar=0x0 out: lpMultiByteStr=0x0, lpUsedDefaultChar=0x0) returned 1149 [0068.961] RtlAllocateHeap (HeapHandle=0x150000, Flags=0x0, Size=0x400) returned 0x150800 [0068.961] WideCharMultiByte (In: CodePage=0x0, dwFlags=0x0, lpWideCharStr=ALLUSERSPROFILEC:\ProgramData, cchWideChar=1149, lpMultiByteStr=0x150700, cbMultiByte=1149, lpDefaultChar=0x0, lpUsedDefaultChar=0x0 out: lpMultiByteStr=ALLUSERSPROFILEC:\ProgramData, lpUsedDefaultChar=0x0) returned 1149 [0068.962] FreeEnvironmentStringsW (penv=0x171920) returned 1 [0068.962] RtlAllocateHeap (HeapHandle=0x150000, Flags=0x0, Size=0x400) returned 0x150c00 [0068.962] GetStartupInfoA (In: lpStartupInfo=0x12f904 out: lpStartupInfo=0x12f904, lpReserved=0, lpDesktop=WINSTA0\Default, lpTitle=C:\ProgramData\ves.exe, dwX=0x0, dwY=0x0, dwXSize=0x0, dwYSize=0x0, dwCountChars=0x0, dwFILLtrIbute=0x0, dwFlags=0x0, dwShowWindow=0x1, cbReserved=0x0, lpReserved=0x0, hStdInput=0xffffffff, hStdOutput=0xffffffff, hStdError=0xffffffff) returned 0 [0068.962] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0068.962] GetFileType (hFile=0x0) returned 0x0 [0068.962] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0068.962] GetFileType (hFile=0x0) returned 0x0 [0068.962] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0068.962] GetStdHandle (hStdHandle=0xffffffff) returned 0x0 [0068.962] GetFileType (hFile=0x0) returned 0x0 [0068.962] SetHandleCount (lNumber=0x20) returned 0x20 </pre>

Figure 6-17 Function log obtained from VMRay

Fig. 6-17 shows the information regarding the binary. We can see that the binary did and what all process that it created/modified and deleted. We also see what changes the binary made to the registry of the system and also if there was any network connection. The importance of this functional log is that we will use this to finally classify the binary as malicious or false-positive using the YARA rules.

Evaluating false-positive: One of the way that we evaluated whether the sample was a false-positive was using the result that were obtained from the result of sandboxes. We also considered a sample as false-positive if only one sandbox showed that the binary was malicious but the other two sandboxes returned that the binary was not malicious. Out of

the 1000 samples that were analyzed 99 of them were false-positive with about 98% of them as octet-stream.

Using the YARA rules we were able to identify the following types of malware which are shown in table 6-1.

Table 6-1 Categorization of malware

Type	Count
Trojan Horse	867
Spyware	723
Downloader	756
Virus	34
Botnet	320
Remote Access Trojan (RAT)	325
Riskware	202
Backdoor	763
Ransomware	34

As we can see in Table 6-1, Trojan Horse was identified in the most prevalent followed by downloader. Table 6-2 gives the categories and their count that were identified in the research.

Table 6-2 Number of malware families identified

Family	Count	Family	Count
CVE-2017-0199	3	Msilperseus	2
Trickbot	3	Banload	7
Fareit	21	Qhost	1
Grp	6	Emotet	150
Injector	77	Androm	12
Coinminer	8	Shade	138
Rnd	1	Cqc	4

Table 6-2 Continued

Yakes	3	Donoff	1
Stealer	6	Dhjogse	3
Formbook	5	Com	2
Guildma	1	Liusky	1
Inject	3	Lokibot	6
Ramnit	4	Cridex	3
Rdn	41	Nanocore	4
Kryptik	176	Obfuse	14
Miner	4	Tiny	1
Ulise	14	Auebm	1
Generic	7	Agentwdr	1
Lnk	1	Strictor	7
Coinstyal	1	Sload	42
Disclipboard	1	Blocker	4
Valyria	10	Frs	10
CVE-2017-11882	15	Mbt	4
Nymeria	4	Quasar	2
Nemesis	3	Diple	1
Loki	5	Vbkryjetor	4
Nanobot	9	Unknown	38

As we can see from the table above, 38 sample family were unknown, 176 malware belong to the family Kryptik, 150 malware belong to Emotet and 138 malware belong to Shade.

Kryptik is the most prevalent malware family followed by Emotet. As per reported in [1] Emotet increased its market share in 2018 by 16%, however we showed that Kyrptik

was the preferred choice of family by malware authors rather than Emotet as it consisted of 19%, whereas Emotet consisted of 15%.

Chapter 7

Conclusion

On examination of the current landscape of drive by downloads, fueled both by availability of exploit-as-a-service and compromised websites directing users to malicious domains, it is evident that drive-by download is the preferred distribution vector for most of the malware families. The purpose of this thesis was to examine these risks and to identify these risks. In this work, we collected 1414 binaries through a malware crawler which visited and extracted these binaries by visiting approximately 100,000 websites. Upon collecting those binaries, we downloaded 1000 of the binaries and then used 3 sandboxes (Cuckoo, VMRay, VirusTotal) to study their behavior in an isolated contained environment. Using those results, we were able to identify that 10% of the binaries were false-positive.

After the identification of false-positive binaries, we did analysis on the reports that were provided from the sandboxes to get information about their behavior in the sandbox. The main aim was possibly to identify what type of malware where the binary belong. The last part was to use this information and apply YARA rules to find the family and the type of malware. Out of the 1414 program executables (binaries) that were captured, 1000 binaries were executed and 99 were identified as false-positive. Out of the 1414 binaries that were extracted 959 of them were executable, 48% of the binaries were extracted from websites that were hosted in the US. We also found that 105 binaries had the same name but different hashes that is, they were not identical. Out of the 901 binaries, 867 of them were identified as Trojan Horse and we were able to identify 53 type of malware families, with one particular family, Kyrptik, having 176 malwares belonging to it which is about 19% and about 4% of the malware families were not identified.

Chapter 8

Future Work

The main aim of this thesis was to analyze and identify drive-by download malwares through the use of sandbox and YARA ruleset. In the future, we would like to employ machine learning in order to minimize the fraction of the number of malware with unknown families. Previous literature has used a random forest algorithm as the main machine learning algorithm to classify samples. We would like to compare various algorithms to find which give us the better result. Also, we would like to study the economic impact that the victims have to incur. In the study that was conducted in [38], authors excluded the small and medium-sized firms from their analysis. We would like to study that because according to the study done in [39], 884 small-business owners, 42% of the respondents experienced breach or a cyber-attack.

We would also like to extend the study of the malwares that were extracted with the same name but with different hashes. As we saw in our research that the mean difference between the binary first store in the database to its second store with same name but with a different hash was 17 days. According to [34], the researchers did not test the hypothesis that even if the hashes were different, were the malware different in nature? We would like to employ our proposed system to test the samples to further classify, if the malware obtained with the same name but with different hashes were giving exactly the same result.

References

- [1] 2019 Internet Security Threat Report. <https://www.symantec.com/security-center/threat-report> (Accessed on March 28th, 2019)
- [2] Drive-by download. <https://securingtomorrow.mcafee.com/consumer/family-safety/drive-by-download/> (Accessed on March 23th, 2019)
- [3] C. Grier, L. Ballard, J. Caballero, N. Chachra, C.J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M.Z. Rafique, M.A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, G.M. Voelker *Manufacturing, Compromise, The Emergence of Exploit-as-a-Service* Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2012)
- [4] Moore, D., Shannon, C., Voelker, G., and Savage, S. *Internet Quarantine: Requirements for Containing Self-Propagating Code*. San Diego, California: University of California Press, 2003
- [5] Report about downloader. <https://www.symantec.com/security-center/writeup/2002-101518-4323-99> (Accessed on March 28th, 2019)
- [6] Malware statistics and trends report. <https://www.av-test.org/en/statistics/malware/> (Accessed on March 20th, 2019)
- [7] Botnet definition. <https://searchsecurity.techtarget.com/definition/botnet> (Accessed on March 28th, 2019)
- [8] Remote access trojan definition <https://searchsecurity.techtarget.com/definition/RAT-remote-access-Trojan> (Accessed on March 28th, 2019)
- [9] What is riskware? <https://usa.kaspersky.com/resource-center/threats/riskware> (Accessed on March 28th, 2019)
- [10] Backdoor definition <https://searchsecurity.techtarget.com/definition/backdoor> (Accessed on March 28th, 2019)

- [11] Asus software hack
https://motherboard.vice.com/en_us/article/pan9wn/hackers-hijacked-asus-software-updates-to-install-backdoors-on-thousands-of-computers (Accessed on March 29th, 2019)
- [12] YARA documentation <https://yara.readthedocs.io/en/v3.8.1/> (Accessed on December 14th, 2018)
- [13] Practical Malware Analysis Book
- [14] Number of Websites in 2019. <https://tekeye.uk/computing/how-many-websites-are-there> (Accessed on March 30th, 2019)
- [15] Ph0neutria malware crawler. <https://github.com/phage-nz/ph0neutria> (Accessed on December 30th, 2018)
- [16] Viper database <https://viper-framework.readthedocs.io/en/latest/index.html> (Accessed on January 12th, 2019)
- [17] Cuckoo sandbox documentation <https://cuckoo.sh/docs/index.html> (Accessed on December 28th, 2018)
- [18] VMRay analysis <https://www.vmrays.com/products/cloud-malware-analysis/> (Accessed on January 14th, 2019)
- [19] VirusTotal official website. <https://www.virustotal.com/#/home/upload> (Accessed on January 14th, 2019)
- [20] OTX official website <https://www.alienvault.com/open-threat-exchange> (Accessed on February 14th, 2019)
- [21] Inetsim. <https://www.inetsim.org/> (Accessed on April 02nd, 2019)
- [22] Exploit. <https://www.avast.com/c-exploits> (Accessed on February 18th, 2019)
- [23] What is Hook? <https://whatis.techtarget.com/definition/hook> (Accessed on January 14th, 2019)
- [24] Emotet malware. <https://www.us-cert.gov/ncas/alerts/TA18-201A> (Accessed on February 18th, 2019)

- [25] Ramnit. <https://www.symantec.com/security-center/writeup/2010-0119222056-99> (Accessed on January 29th, 2019)
- [26] Microsoft .NET framework. <https://dotnet.microsoft.com/> (Accessed on March 14th, 2019)
- [27] Malshare official website. <https://malshare.com/> (Accessed on March 28th, 2019)
- [28] Clean-MX official website. <https://support.clean-mx.com/cleanmx/viruses.php> (Accessed on April 3rd, 2019)
- [29] Cymon io official website. <https://cymon.io/> (Accessed on January 25th, 2019)
- [30] Malc0de official website. <http://malc0de.com/dashboard/> (Accessed on March 10th, 2019)
- [31] PEiD. <https://www.aldeid.com/wiki/PEiD> (Accessed on February 18th, 2019)
- [32] Matthew S Webb *Evaluation tool based automated malware analysis through persistence mechanism Detection* 2016
- [33] Bailey, M., Oberheide, J., Mao, Z. M., Jahanian, F., and Nazario, J. *Automated classification and analysis of Internet malware*. RAID 10th International Symposium Proceedings. Gold Coast, Australia: Springer-Verlag Berlin Heidelberg, 2007
- [34] Tanaka, Y., Akiyama, M., and Goto, A. *Analysis of malware download sites by focusing on time series variation of malware* Journal of Computation Science 22 (2017) 301-313
- [35] Nappa, A., Rafique, M., and Caballero, J. *The MALICIA dataset: identification and analysis of drive-by download operations* Int. J. Inf. Secur. 24 (1) (2015) 15-33
- [36] Moshchuk, A., Bragin, T., Gribble, S.D., and Levy, H.M. *A crawler based study of spyware on the web* Network and Distributed System Security Symposium, San Diego, CA, February 2006

[37] What is DLL? <https://whatis.techtarget.com/fileformat/DLL-Dynamic-link-library-file> (Accessed on April 9th, 2019)

[38] The Cost of Malicious Cyber Activity to the U.S. Economy. <https://www.whitehouse.gov/wp-content/uploads/2018/03/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf> (Accessed on April 10th, 2019)

[39] National Small Business Association. 2015. Year-End Economic Report. <https://www.nsba.biz/wp-content/uploads/2016/02/Year-End-Economic-Report-2015.pdf> (Accessed on April 10th, 2019)

Biographical Information

This report belongs to Mohit Singhal. Mohit has obtained a Bachelor's degree in Computer Science and Engineering and Master of Science degree in Computer Science and he has successfully defended his master's thesis in information security under the supervision of Professor. David Levine.

Mohit is interested in pursuing his career in the field of information security, cyber security and malware analysis. He has worked on several projects in software security area involving static analysis tools, malware analysis and various open source platforms such as Cuckoo sandbox, Kali Linux, IDA Pro, IDS/IPS, Wireshark, etc.

Mohit was appointed as a Graduated Teaching Assistant/Grader for the Fall 2018 and Spring 2019 semester and had successfully conducted the labs along with the CTF competition as a part of information security (CSE 4380, 4380/5380).

Mohit explored his expertise in the field of information security by working as a Research Assistant in Summer 2016 and taking a practical malware analysis course in spring 2018 which improved his thought process and gave an insight into his research in the malware analysis.

Mohit is interested in pursuing his career in the area of computer security and is looking forward to obtain a doctorate degree in the field of information security especially in the area of malware analysis and detection.