University Mohammed Premier

National School of Applied Sciences of Oujda

End of Year Project

Windows Malware analysis

Produced by:

Nasreddin EL HAFI

Mohammed HADI

Supervised by:

M. Mohammed Amine Koulali

Jury :

Pr. Youssef Jabri

Presented on July 5, 2021

Abstract

If you're a vigilant Windows user, you sometimes may see a file with a strange or suspicious name that you'd like to investigate. Or, if you're an ethical hacker or are on the incident response team of an organization, you may be tasked with analyzing files to determine whether they're legitimate or malicious. Either way, you need a way to be able to differentiate good code and software from malicious varieties.

Malware Analysis holds an important part in cybersecurity for defence and mitigation as it's peer spectrum and with more and more sophistication of malware we might fall behind to detect, understand and mitigate malware in the future.

There are a few key reasons to perform malware analysis:

- Malware detection — To better protect your organization, you need to be able to identify compromising threats and vulnerabilities.
- Threat response — To help you understand how these threats work so you can react accordingly to them.
- Malware research — This can help you to better understand how specific types of malware work, where they originated, and what differentiates them.

Table of contents

Abstract 1

Table of contents 2

Figure table 3

Chapter I. Introduction 7

- Introduction to malwares 7
- Malware analysis 10
- Phases of Malware Attack 13

Chapter 2. Malware Analysis Stages (Vipasana Ransomware) 18

- Vipasana Ransomware 19
- Stage One: Fully Automated Analysis 20
- Stage Two: Static Properties Analysis / static Analysis 25
- Stage Three: Interactive Behavior Analysis /Dynamic Analysis 33
- Stage Four: Manual Code Reversing 47

Conclusion 62

Acronyms 63

References 64

Figure table

- Figures

Figure 1: Common types of malware

Figure 2: The Four Stages of Malware Analysis.

Figure 3: MITRE ATT&CK components

Figure 4: Executable General informations

Figure 5: Mitre ATT&CK Matrix

Figure 6: Execution on Any.Run

Figure 7: Process Graph

Figure 8: Behaviour Activities

Figure 9: Malware Initial Assessment

Figure 10: Desktop while malware execution

Figure 11: Desktop After Execution

Figure 12: Graphical Representation of malware execution

Figure 13: Regshot Result

Figure 14: ApateDNS After Execution

Figure 15: Wireshark during Execution

Figure 16 : Encryption functions detection using PEiD

Figure 17 : Encryption Algorithm

Figure 18 : Changing Desktop Image

- Tables

Table 1 : start of CMD.exe

Table 2 :Changes the autorun value in the registry

Table 3 : Steals credentials from Web Browsers

Table 4 :Changes the desktop background image

Table 5 : Sections

Table 6 : Imports

- Outputs

output 1: memory dump processes list

Output 2: Vipasana Keys list

Output 2: Vipasana Mutexes list

- Code Blocks

Code 1: RSA Keys Obfuscation

Code 2: Removal of concatenated random String

Code 3: file name after encryption

Code 4 : Assembly Code calling function used for malware persistence

Code 5 : PRNG to generate 2048 characters

Code 6 : Generating static global key stream

Code 7: Assembly Code setting up the extensions

Code 8 : Looping over available drives to encrypt

Code 9 : Update internal state

Code 10 : Encryption Algorithm

Chapter I. Introduction

I. Introduction to malwares

1. History of malware

The term malware was first used by computer scientist and security researcher Yisrael Radai in 1990. However, malware existed long before this. One of the first known examples of malware was the Creeper virus in 1971, which was created as an experiment by BBN Technologies engineer Robert Thomas. Creeper was designed to infect mainframes on ARPANET. While the program did not alter functions -- or steal/delete data -- it moved from one mainframe to another without permission while displaying a teletype message that read, "I'm the creeper: Catch me if you can." Creeper was later altered by computer scientist Ray Tomlinson, who added the ability to self-replicate to the virus and created the first known computer worm.

The concept of malware took root in the technology industry, and examples of viruses and worms began to appear on Apple and IBM personal computers in the early 1980s before becoming popularized following the introduction of the World Wide Web and the commercial internet in the 1990s. Since then, malware, and the security strategies to prevent it, have only grown more complex.

2. What is Malware and How do malware infections happen?

Malware, also known as malicious code, refers to a program that is covertly inserted into another program with the intent to destroy data, run destructive or intrusive programs, or otherwise compromise the confidentiality, integrity, or availability of the victim’s data, applications, or operating system. Malware is the most common external threat to most hosts, causing widespread damage and disruption and necessitating extensive recovery efforts within most organizations.

Malware authors use a variety of physical and virtual means to spread malware that infects devices and networks. For example, malicious programs can be delivered to a system with a USB drive or can spread over the internet through drive-by downloads, which automatically download malicious programs to systems without the user's approval or knowledge. Phishing attacks are another common type of malware delivery where emails disguised as legitimate messages contain malicious links or attachments that can deliver the malware executable file to unsuspecting users. Sophisticated malware attacks often feature the use of a command-and-control server that enables threat actors to communicate with the infected systems, exfiltrate sensitive data and even remotely control the compromised device or server.Emerging strains of malware include new evasion and obfuscation techniques that are designed to not only fool users but security administrators and antimalware products as well. Some of these evasion techniques rely on simple tactics, such as using web proxies to hide malicious traffic or source IP addresses. More sophisticated threats include polymorphic malware

that can repeatedly change its underlying code to avoid detection from signature-based detection tools; anti-sandbox techniques that enable the malware to detect when it is being analyzed and to delay execution until after it leaves the sandbox; and fileless malware that resides only in the system's RAM to avoid being discovered

3. Malware types

The following figure represents the common types of malware

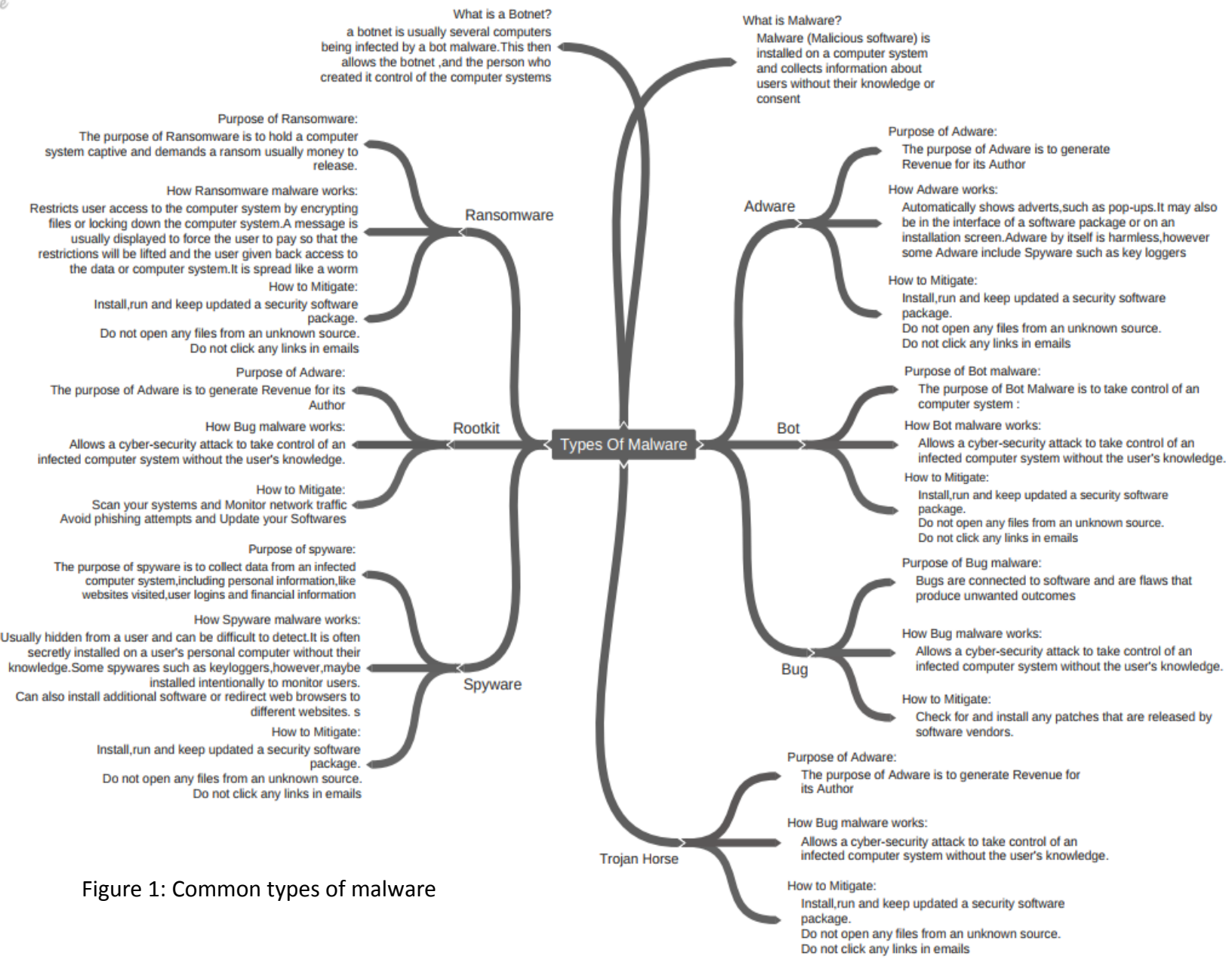


Figure 1: Common types of malware

4. How to detect malwares ?

A user may be able to detect malware if they observe unusual activity such as a sudden loss of disc space, unusually slow speeds, repeated crashes or freezes, or an increase in unwanted internet activity and pop-up advertisements. Antivirus software may also be installed on the device to detect and remove malware. These tools can provide real-time protection or detect and remove malware by executing routine system scans.

Windows Defender, for example, is Microsoft anti-malware software included in the Windows 10 operating system (OS) under the Windows Defender Security Center. Windows Defender protects against threats such as spyware, adware and viruses. Users can set automatic "Quick" and "Full" scans, as well as set low, medium, high and severe priority alerts.

II. Malware analysis

1. Definition & importance of malware analysis

Malware Analysis is the study or process of determining the functionality, origin and potential impact of a given malware sample and extracting as much information from it. The information that is extracted helps to understand the functionality and scope of malware, how the system was infected and how to defend against similar attacks in future.

Objectives:

- To understand the type of malware and its functionality.
- Determine how the system was infected by malware and define if it was a targeted attack or a phishing attack.
- How malware communicates with attacker.
- Future detection of malware and generating signatures.

If you’re a vigilant Windows user, you sometimes may see a file with a strange or suspicious name that you’d like to investigate. Or, if you’re an ethical hacker or are on the incident response team of an organization, you may be tasked with analyzing files to determine whether they’re legitimate or malicious. Either way, you need a way to be able to differentiate good code and software from malicious varieties.

There are a few key reasons to perform malware analysis:

- Malware detection — To better protect your organization, you need to be able to identify compromising threats and vulnerabilities.
- Threat response — To help you understand how these threats work so you can react accordingly to them.
- Malware research — This can help you to better understand how specific types of malware work, where they originated, and what differentiates them.

2. The Four Stages of Malware Analysis

When you learn how to write and read code, you do so little by little. Malware analysis is much the same. It's a process that you approach through a series of formulated steps that become increasingly complex the further you go.

There are four stages to malware analysis, often illustrated using a pyramid diagram that increases in complexity as you go deeper into the process as illustrated in Figure 2. For the sake of ease, we'll break down each of the four stages of malware analysis from the ground up.

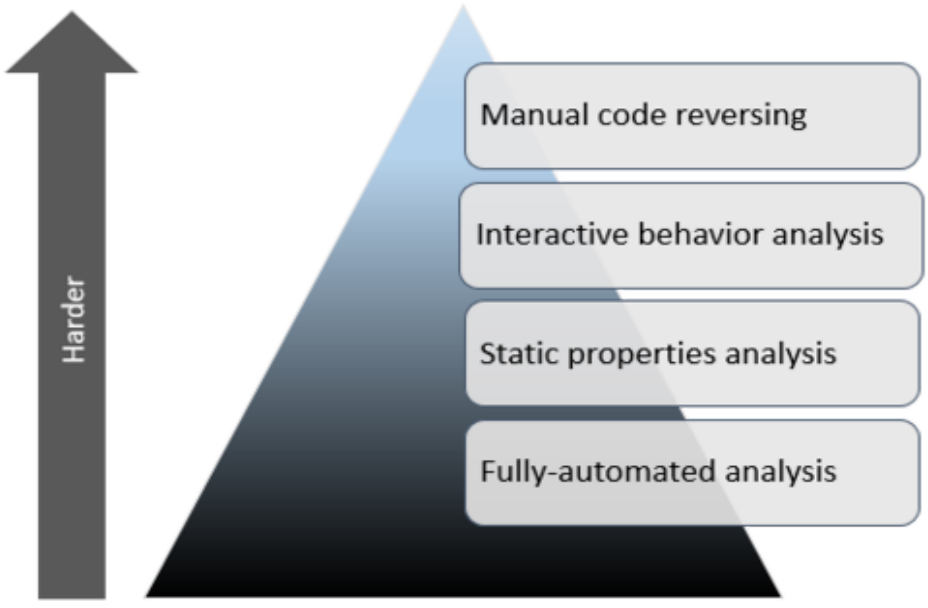


Figure 2: The Four Stages of Malware Analysis.

a. Stage One: Fully Automated Analysis

Automated malware analysis refers to relying on detection models formed by analyzing previously discovered malware samples in the wild. This is the most suited method to process malware at scale and quickly assess the repercussions of a sample on the network infrastructure.

Fully automated analysis can be done using tools like Cuckoo Sandbox, an open-source automated malware analysis platform that can be tweaked to run custom scripts and generate comprehensive reports. There are several other alternative tools, both commercial and free, that are available in the market.

b. Stage Two: Static Properties Analysis / static Analysis

Static properties analysis involves looking at a file's metadata without executing the malware. This process is typically something you do within an isolated environment — such as a virtual machine — that's disconnected from the internet.

One of the free tools that you may find useful for this purpose is PeStudio. This tool flags suspicious artifacts within executable files and is designed for automated static properties analysis. PeStudio presents the file hashes that can be used to search VirusTotal, TotalHash, or other malware repositories to see if the file has previously been analyzed. Moreover, it can be used to examine the embedded strings, libraries, imports, and other indicators of compromise (IOCs) and compare any unusual values that differ from those typically seen in regular executable files.

Conducting static property analysis should ideally leave a malware analyst with a fair idea of whether to continue pursuing or cease the investigation.

c. Stage Three: Interactive Behavior Analysis / dynamic Analysis

In the next phase, behavior analysis, the malware sample is executed in isolation as the analyst observes how it interacts with the system and the changes it makes. Often, a piece of malware might refuse to execute if it detects a virtual environment or might be designed to avoid execution without manual interaction (i.e., in an automated environment).

There are several types of actions that should immediately raise a red flag, including:

- Adding or modifying new or existing files,
- Installing new services or processes, and
- Modifying the registry or changing system settings.

Some types of malware might try to connect to suspicious host IPs that don't belong to the environments. Others might also try to create mutex objects to avoid infecting the same host multiple times (to preserve operational stability). These findings are relevant indicators of compromise.

Some of the tools that you can use include:

- Wireshark for observing network packets,
- Process Hacker to observe the processes that are executing in memory,
- Process Monitor to observe real-time file system, registry, process activity for Windows, and
- ProcDot provides an interactive and graphical representation of all recorded activities.

Of course, you can conduct additional research on the new data points you gather by using any malware analysis database. Likewise, additional network analysis can disclose details about the command and control infrastructure of the malware specimen, the volume and kind of data it leaks, etc.

d. Stage Four: Manual Code Reversing

Reverse engineering the code of a sample malware can provide valuable insights. This process can:

- Shed some light on the logic and algorithms the malware uses,
- Expose hidden capabilities and exploitation techniques the malware uses, and
- Provide insights about the communication protocol between the client and the server on the command and control side.

Typically, to manually reverse the code, analysts make use of debuggers and disassemblers. Though code reversals are an extremely time-consuming process — and although the skills to perform them aren't particularly common — this step can provide plenty of important insights.

III. Phases of Malware Attack

To detail the phases of a malware attack we will be using the MITRE ATT&CK Matrix. MITRE ATT&CK is a knowledgebase of adversary tactics and techniques. It has become a useful tool across many cyber security use cases such as Threat Hunting, Red Teaming and Threat Intelligence Enrichment. The framework has been frequently discussed at cyber security conferences such as RSA, Black Hat and Gartner Security and Risk Management Summit.

The framework provides intelligence information based on real-world observation and therefore it is very useful for attacks investigations. Figure 3 illustrates the MITRE ATTA&CK components:

Figure 3: MITRE ATTA&CK components

1. Initial Access: The adversary is trying to get into your network.

Initial Access consists of techniques that use various entry vectors to gain their initial foothold within a network. Techniques used to gain a foothold include targeted spearphishing and exploiting weaknesses on public-facing web servers. Footholds gained through initial access may allow for continued access, like valid accounts and use of external remote services, or may be limited-use due to changing pswords.

2. Execution: The adversary is trying to run malicious code.

Execution consists of techniques that result in adversary-controlled code running on a local or remote system. Techniques that run malicious code are often paired with techniques from all other tactics to

achieve broader goals, like exploring a network or stealing data. For example, an adversary might use a remote access tool to run a PowerShell script that does Remote System Discovery.

3. Persistence : The adversary is trying to maintain their foothold.

Persistence consists of techniques that adversaries use to keep access to systems across restarts, changed credentials, and other interruptions that could cut off their access. Techniques used for persistence include any access, action, or configuration changes that let them maintain their foothold on systems, such as replacing or hijacking legitimate code or adding startup code.

4. Privilege Escalation : The adversary is trying to gain higher-level permissions.

Privilege Escalation consists of techniques that adversaries use to gain higher-level permissions on a system or network. Adversaries can often enter and explore a network with unprivileged access but require elevated permissions to follow through on their objectives. Common approaches are to take advantage of system weaknesses, misconfigurations, and vulnerabilities. Examples of elevated access include:

- SYSTEM/root level
- local administrator
- user account with admin-like access
- user accounts with access to specific system or perform specific function

These techniques often overlap with Persistence techniques, as OS features that let an adversary persist can execute in an elevated context.

5. Defense Evasion : The adversary is trying to avoid being detected.

Defense Evasion consists of techniques that adversaries use to avoid detection throughout their compromise. Techniques used for defense evasion include uninstalling/disabling security software or obfuscating/encrypting data and scripts. Adversaries also leverage and abuse trusted processes to hide and masquerade their malware. Other tactics' techniques are cross-listed here when those techniques include the added benefit of subverting defenses.

6. Credential Access : The adversary is trying to steal account names and passwords.

Credential Access consists of techniques for stealing credentials like account names and passwords. Techniques used to get credentials include keylogging or credential dumping. Using legitimate

credentials can give adversaries access to systems, make them harder to detect, and provide the opportunity to create more accounts to help achieve their goals

7. Discovery : The adversary is trying to figure out your environment.

Discovery consists of techniques an adversary may use to gain knowledge about the system and internal network. These techniques help adversaries observe the environment and orient themselves before deciding how to act. They also allow adversaries to explore what they can control and what's around their entry point in order to discover how it could benefit their current objective. Native operating system tools are often used toward this post-compromise information-gathering objective.

8. Lateral Movement : The adversary is trying to move through your environment.

Lateral Movement consists of techniques that adversaries use to enter and control remote systems on a network. Following through on their primary objective often requires exploring the network to find their target and subsequently gaining access to it. Reaching their objective often involves pivoting through multiple systems and accounts to gain. Adversaries might install their own remote access tools to accomplish Lateral Movement or use legitimate credentials with native network and operating system tools, which may be stealthier.

9. Collection : The adversary is trying to gather data of interest to their goal.

Collection consists of techniques adversaries may use to gather information and the sources information is collected from that are relevant to following through on the adversary's objectives. Frequently, the next goal after collecting data is to steal (exfiltrate) the data. Common target sources include various drive types, browsers, audio, video, and email. Common collection methods include capturing screenshots and keyboard input.

10. Exfiltration: The adversary is trying to steal data.

Exfiltration consists of techniques that adversaries may use to steal data from your network. Once they've collected data, adversaries often package it to avoid detection while removing it. This can include compression and encryption. Techniques for getting data out of a target network typically include transferring it over their command and control channel or an alternate channel and may also include putting size limits on the transmission.

11. Command and Control : The adversary is trying to communicate with compromised systems to control them.

Command and Control consists of techniques that adversaries may use to communicate with systems under their control within a victim network. Adversaries commonly attempt to mimic normal, expected traffic to avoid detection. There are many ways an adversary can establish command and control with various levels of stealth depending on the victim's network structure and defenses.

Chapter 2. Malware Analysis Stages (Vipasana Ransomware)

Introduction:

The objective of this chapter is to conduct a malware analysis over the Vipasana Ransomware (add some content here to describe the chapter)

I. Vipasana Ransomware

1. Definition

Vipasana ransomware (or Vipassana ransomware) another bird of ill omen. In its true sense, the word ‘‘Vipassana’’ refers to a meditation practice in the Buddhist tradition. A quite dissonance for ransomware threats to be called such names. Ransomware viruses bear the highest degree of danger. These nightmarish viruses block the user’s access to his/her data, especially the most valuable. Thus, they must be examined and preventative measures are to be taken against them. The presence of such threats as Vipasana ransomware is intolerable on one’s computer.

2. Ways of Spreading

Vipasana ransomware is spread via spam bots. These programs send spam e-mails to your e-mail account. The e-mails contain malicious links and/or malicious attachments. They can be disguised as links to must visit websites (relating your banking account, etc.) and/or the malicious files can pretend to be the important documents sent to you by official institutions (e.g. various invoices, etc.). You might also get e-mails from unknown senders which fall into the spam folder of your e-mail account. This Trojan Horse gets onto your computer once you follow the link and the malicious code is executed on your computer’s system. Exploit kits, such as Angler or Nuclear EK, may be involved in this. The same happens when you open the infected attachments and their malicious codes are started to be executed.

II. Stage One: Fully Automated Analysis

Before we get our hands dirty we are going to do a quick automated analysis to have a broad idea on how our malware is classified and how it behaves according to detection models formed by analyzing previously discovered malware samples in the wild. And for that we are going to use Any.Run Sandbox..

Any.Run SandBox is an interactive online malware analysis service for automatic dynamic and static research of most types of threats using any environments. The service can be used for a convenient in-depth analysis of new malicious objects, as well as for the investigation of cyber incidentals. Any.Run SandBox Provides us with a MITRE ATT&CK Matrix for the Phases of the analysis of the provided executable.

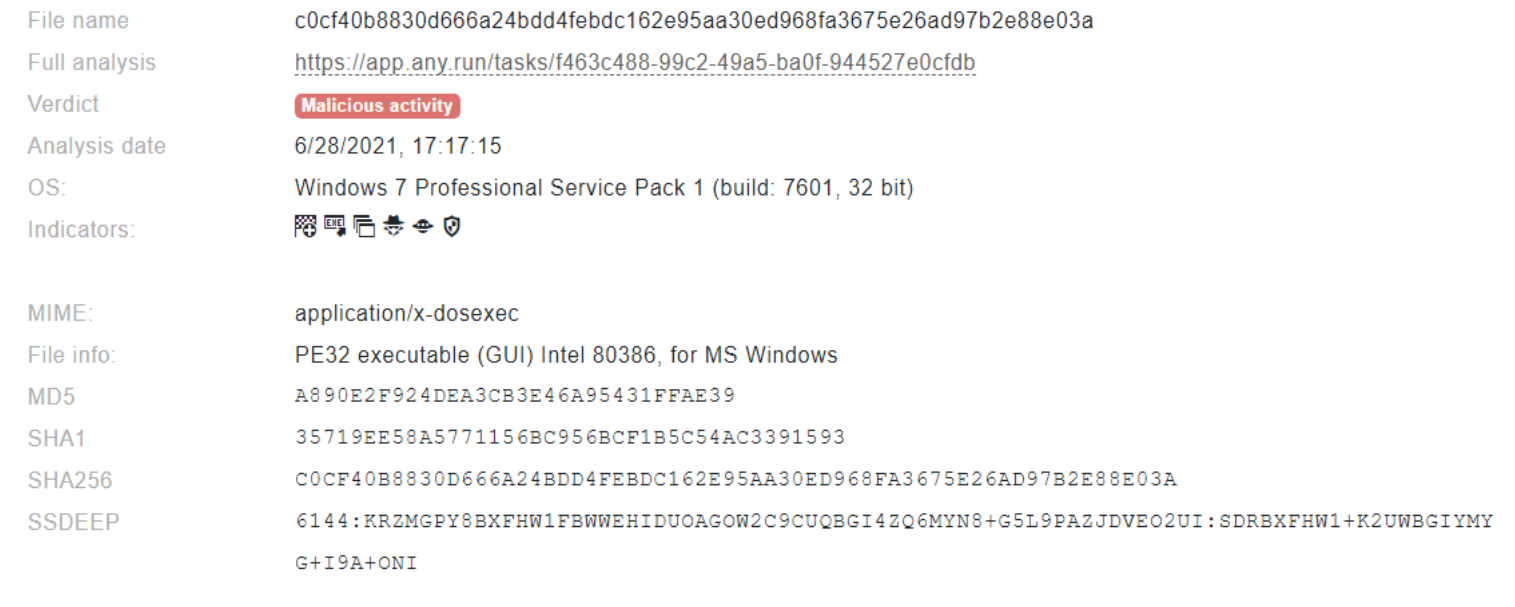


Figure 4: General informations

To Have more clearer idea of the phases this malware takes throughout its execution we are going to use the Mitre ATT@CK Matrix already discussed skipping some phases and adding others depending on how our ransomware behaves.

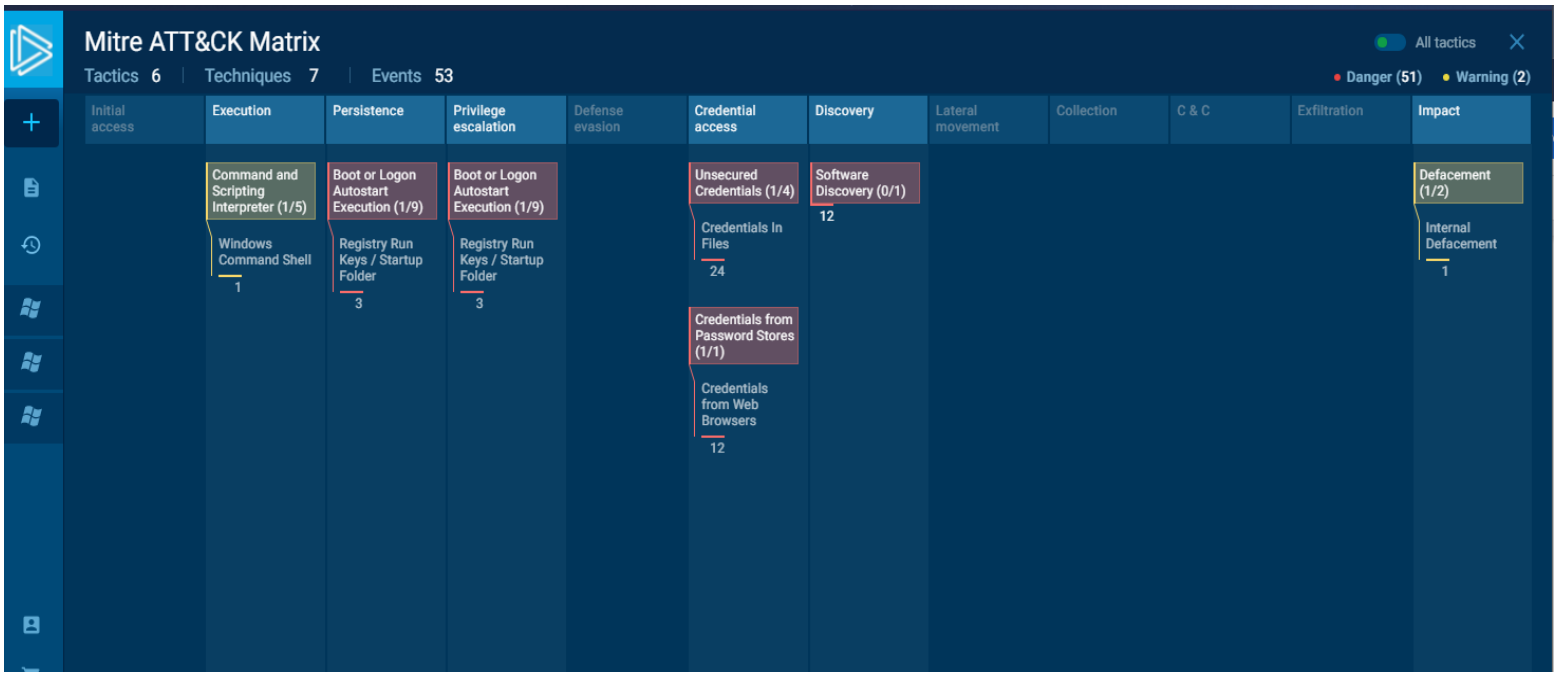


Figure 5: Mitre ATT&CK Matrix

Any.Run by default will only run the virtual machine for 60 seconds which is short but enough to make it to the persistence phase of our ransomware and we get a time extension for another 60 seconds to end our execution.

a. Execution

This Ransomware seems to abuse Windows Command shell to execute commands, scripts, or binaries.

«Windows Command Shell»

• Image	• C:\Windows\system32\cmd.exe
• Cmdline:	• ""C:\Users\admin\AppData\Local\Temp\XXXXX.bat"

Table 1 : start of CMD.exe

Figure 6: Execution on Any.Run

Process Graph

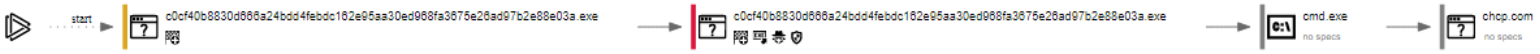


Figure 7: Process Graph

b. Persistence and privilege escalation

Registry Run Keys / Startup Folder

Adversaries may achieve persistence by adding a program to a startup folder or referencing it with a Registry run key. Adding an entry to the "run keys" in the Registry or startup folder will cause the program referenced to be executed when a user logs in. These programs will be executed under the context of the user and will have the account's associated permissions level.

• operation:	• WRITE
• Name:	• PR
• Value:	• C:\Program Files\c0cf40b8830d666a24bdd4febdc162e95aa30ed968fa3675e26ad97b2e88e03a.exe
• Key:	• HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\RUN
• TypeValue:	• REG_SZ

Table 2 :Changes the autorun value in the registry

c. Credential Access / Discovery

Credentials from Web Browsers

Adversaries may acquire credentials from web browsers by reading files specific to the target browser. Web browsers commonly save credentials such as website usernames and passwords so that they do not need to be entered manually in the future. Web browsers typically store the credentials in an encrypted format within a credential store; however, methods exist to extract plaintext credentials from web browsers.

• operation:	• READ
• device:	• DISK_FILE_SYSTEM
• object:	• FILE
• name:	• C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\qldyz51w.default\blocklist.xml • • C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\qldyz51w.default\gmp-widevinecdm\4.10.1440.18\LICENSE.txt

	• C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\qldyz51w.default\pkcs11.txt • C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\qldyz51w.default\gmp-widevinecdm\4.10.1440.18\LICENSE.txt • C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\qldyz51w.default\pkcs11.txt • C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\qldyz51w.default\SiteSecurityServiceState.txt
• status:	• 259
• created:	• NONE

Table 3 :Steals credentials from Web Browsers

d. Impact

An adversary may deface systems internal to an organization in an attempt to intimidate or mislead users. This may take the form of modifications to internal websites, or directly to user systems with the replacement of the desktop wallpaper.

• Operation:	• WRITE
• Name:	• WALLPAPER
• Value:	• C:\Users\admin\AppData\Local\Temp\desk.bmp
• Key:	• HKEY_CURRENT_USER\CONTROL PANEL\DESKTOP
• TypeValue:	• REG_SZ

Table 4 :Changes the desktop background image

Summary

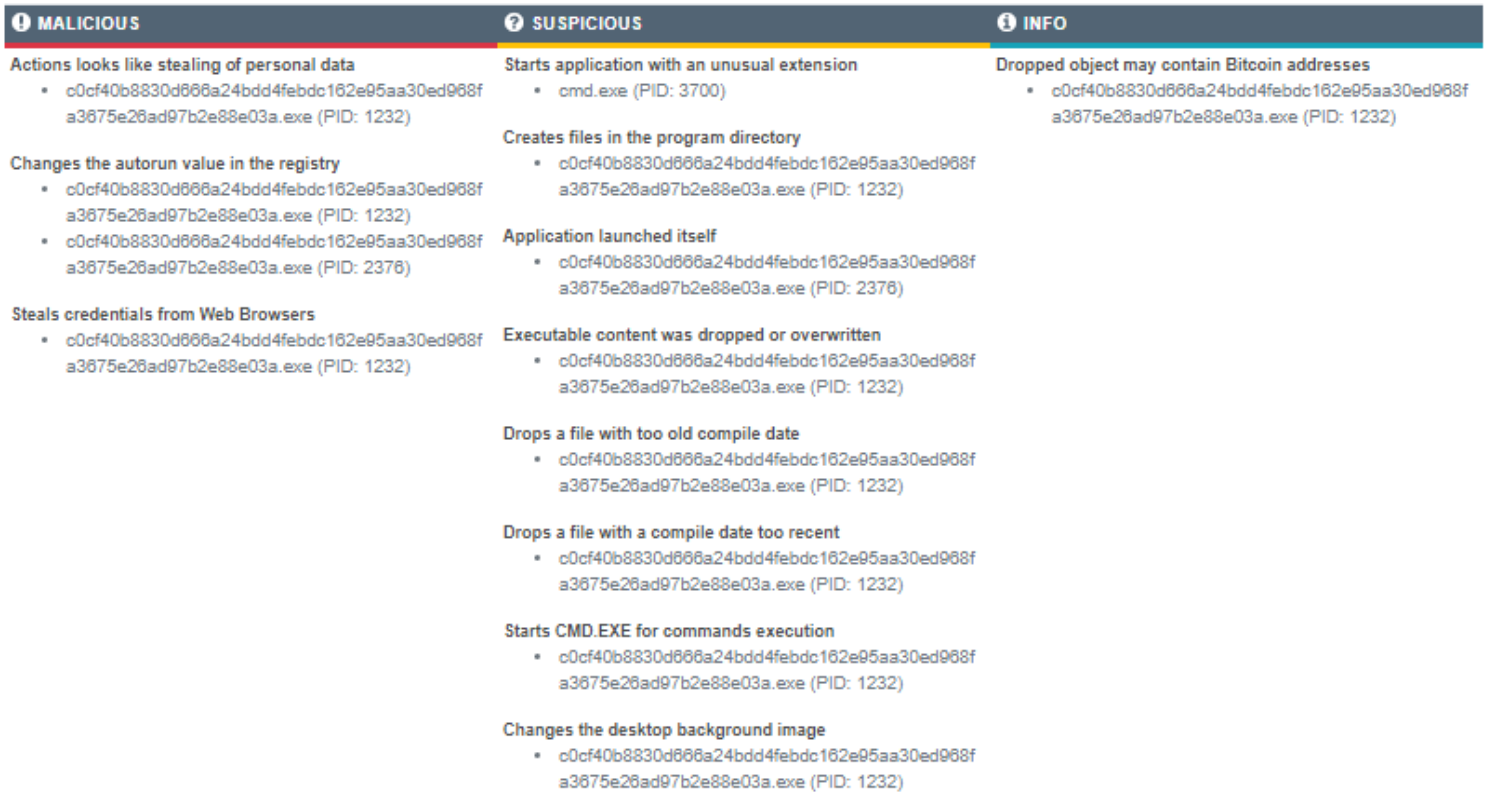


Figure 7: Behaviour Activities

III. Stage Two: Static Properties Analysis / static Analysis

1. FileType

Output file command :

c0cf40b8830d666a24bdd4febdc162e95aa30ed968fa3675e26ad97b2e88e03a: PE32 executable (GUI) Intel 80386, for MS Windows .

2. Hashes

- Md5sum : A890E2F924DEA3CB3E46A95431FFAE39
- Sha1sum : 35719EE58A5771156BC956BCF1B5C54AC3391593
- Sha256sum : C0CF40B8830D666A24BDD4FEBDC162E95AA30ED96

8FA3675E26AD97B2E88E03A

3. More information with Pestudio :

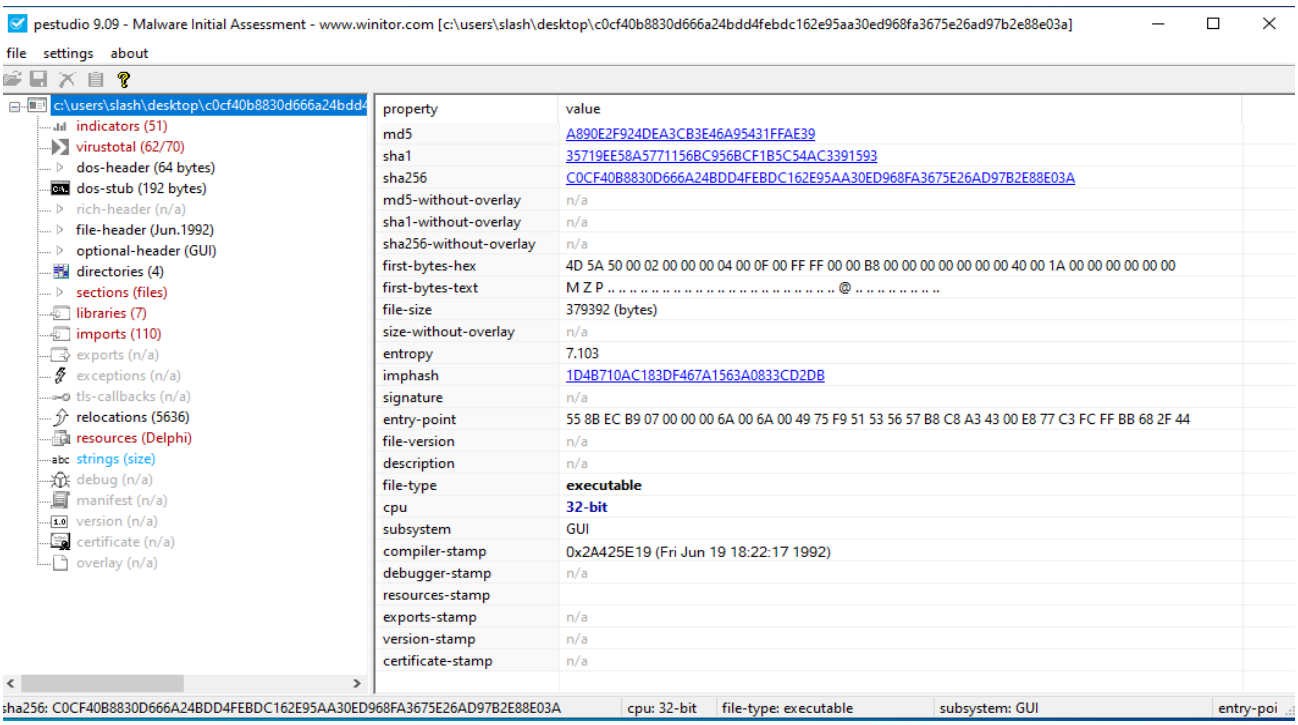


Figure 9: Malware Initial Assessment

There is some indicators that this file was created with delphi :

- the first bytes of the executable : “MZP” .
- Some strings in the resource section generated by the compiler (*ActiveX , Sysconst …*)

Virustotal , 62/70 antiviruses class it as general malware , only some of them detect it as ransomware .

Entropy : 7.103 (High) probably packed .

4. Sections

● CODE	0x3d50c Virtual Size 0x3d600 Size Of Raw Data
● DATA	0x25c4 Virtual Size 0x2600 Size Of Raw Data
● BSS	0x1395 Virtual Size 0x0 Size Of Raw Data
● .idata	0x13f0 Virtual Size 0x1400 Size Of Raw Data
● .tls	0xc Virtual Size 0x0 Size Of Raw Data
● .rdata	0x18 Virtual Size

	0x200 Size Of Raw Data
• .reloc	
• .rsrc	0x19000 Virtual Size 0x18600 Size Of Raw Data

Table 5 : Sections

In the code segment the virtual size and raw size are approximately the same which is a good sign that the malware is not packed or encrypted .

5. Imports :

DLL	Functions
• user32.dll	• GetKeyboardType • LoadStringA • MessageBoxA • CharNextA
• advapi32.dll	• RegQueryValueExA • RegOpenKeyExA • RegCloseKey

• oleaut32.dll	• SysFreeString • SysReAllocStringLen • SysAllocStringLen
• advapi32.dll	• RegSetValueExA • RegOpenKeyExA • RegFlushKey • RegCreateKeyExA • RegCloseKey • OpenThreadToken • OpenProcessToken • GetTokenInformation • FreeSid • EqualSid • AllocateAndInitializeSize
• kernel32.dll	• WriteFile • WaitForSingleObject • VirtualQuery • TerminateProcess • SizeofResource • SetFilePointer • SetEvent • SetErrorMode • SetEndOfFile • ResetEvent • ReadFile • OpenProcess • MulDiv

	• MoveFileA • LockResource • LoadResource • LeaveCriticalSection • InitializeCriticalSection • GlobalUnlock • GlobalReAlloc • GlobalHandle • GlobalLock • GlobalFree • GlobalAlloc • GetWindowsDirectoryA • GetVersionExA • GetTickCount • GetThreadLocale • GetTempPathA • GetSystemInfo • GetStringTypeExA • GetStdHandle • GetProcAddress • GetModuleHandleA • GetModuleFileNameA • GetLocaleInfoA • GetLocalTime • GetLastError • GetFullPathNameA • GetFileAttributesA • GetDiskFreeSpaceA • GetDateFormatA

	• GetCurrentThreadId • GetCurrentThread • GetCurrentProcessId • GetCurrentProcess • GetCPInfo • GetACP • FreeResource • FormatMessageA • FindResourceA • FindNextFileA • FindFirstFileA
	• Gdi32.dll • UnrealizeObject • StretchBlt • SetWinMetaFileBits • SetTextColor • SetStretchBltMode • SetROP2 • SetEnhMetaFileBits • SetDIBColorTable • SetBrushOrgEx • SetBkMode • SetBkColor • SelectPalette • SelectObject • RealizePalette • PlayEnhMetaFile

• User32.dll	• SystemParametersInfoA • ReleaseDC • MessageBoxA • LoadStringA • LoadIconA • GetSystemMetrics • GetSysColor • GetIconInfo • GetFocus • GetDC • GetClipboardData • FillRect • DrawIconEx • DestroyIcon • CreateIcon • CharNextA • CharLowerBuffA • CharToOemA
• kernel32.dll	• Sleep
• oleaut32.dll	• SafeArrayPtrOfIndex • SafeArrayGetUBound • SafeArrayGetLBound • SafeArrayCreate • VariantChangeType • VariantCopy • VariantClear • VariantInit

• shell32.dll	• ShellExecuteExA • ShellExecuteA
• wininet.dll	• InternetReadFile • InternetOpenUrlA • InternetOpenA • InternetCloseHandle

Table 6 : Imports

Thoughts about the most interesting imports :

- the wininet functions maybe the malware will try to query something from the internet.
- The registry functions , maybe it'll try to set up the malware as a service …
- Importing gdi32.dll probably because it's a GUI .
- Shellexecute it'll try to run another process .
- FindFirstFile , writefile … , this is probably the encryption part .

IV. Stage Three: Interactive Behavior Analysis /Dynamic Analysis

In contrast to static techniques, dynamic techniques analyze the code during run-time. While these techniques are non-exhaustive, they have the significant advantage that only those instructions are analyzed that the code actually executes. Thus, dynamic analysis is immune to obfuscation attempts and has no problems with self-modifying programs. When using dynamic analysis techniques, the question arises in which environment the sample should be executed. Of course, running malware directly on the analyst's computer, which is probably connected to the Internet, could be disastrous as the malicious code could easily escape and infect other machines. Furthermore, the use of a dedicated standalone machine that is reinstalled after each dynamic test run is not an efficient solution because of the overhead that is involved

Environment Setup

For the Dynamic analysis we use a windows 7 on a virtual machine for the purpose of providing an isolated environment for the malware execution.

We use

- Windows 7 on a VirtualBox
- Tools Used
- ProcMon + ProcDot
- RegShot
- ApateDNS
- Wireshark

After execution we dump the memory into a .dump file and we use Volatility to Analyse the memory Dump looking for any traces left

1. Process Behavioral

ProcMon + ProcDot

- ProcMon

Process monitor has the capability of monitoring, capturing and filtering all the artifacts.

We start the Capturing for the particular malware using its pid Directly after starting the malware

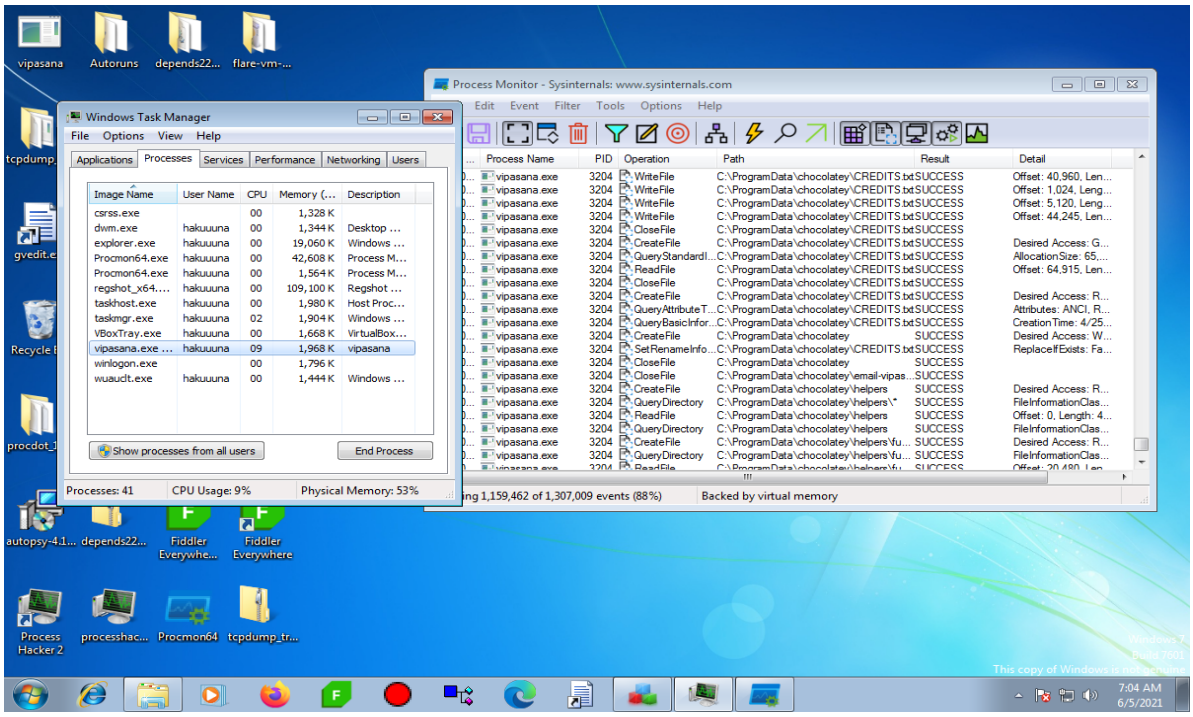


Figure 10: Desktop while malware execution

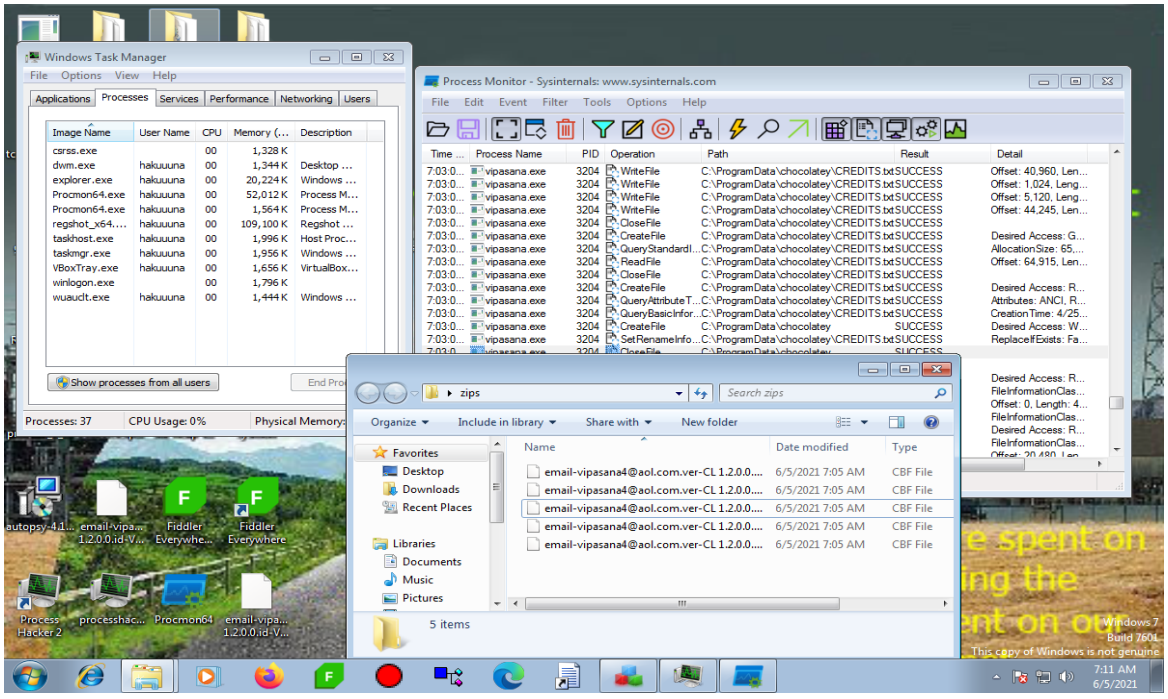


Figure 11: Desktop After Execution

- ProcDot

ProcDot allows a malware analyst to ingest the output from ProcMon and automatically generate a graphical representation of the captured data. Simply upload the csv into ProcDot and select the process name of the malware. Rather than creating filters and navigating hundreds of thousands of events you are now able to navigate a visual diagram of what recorded malware activity.

The Following Architecture represents an approximate representation of ProcMon output for the time of 2 minutes execution based on the graphical representation generated by procDot filtering only operations attached to our main ransomware process.

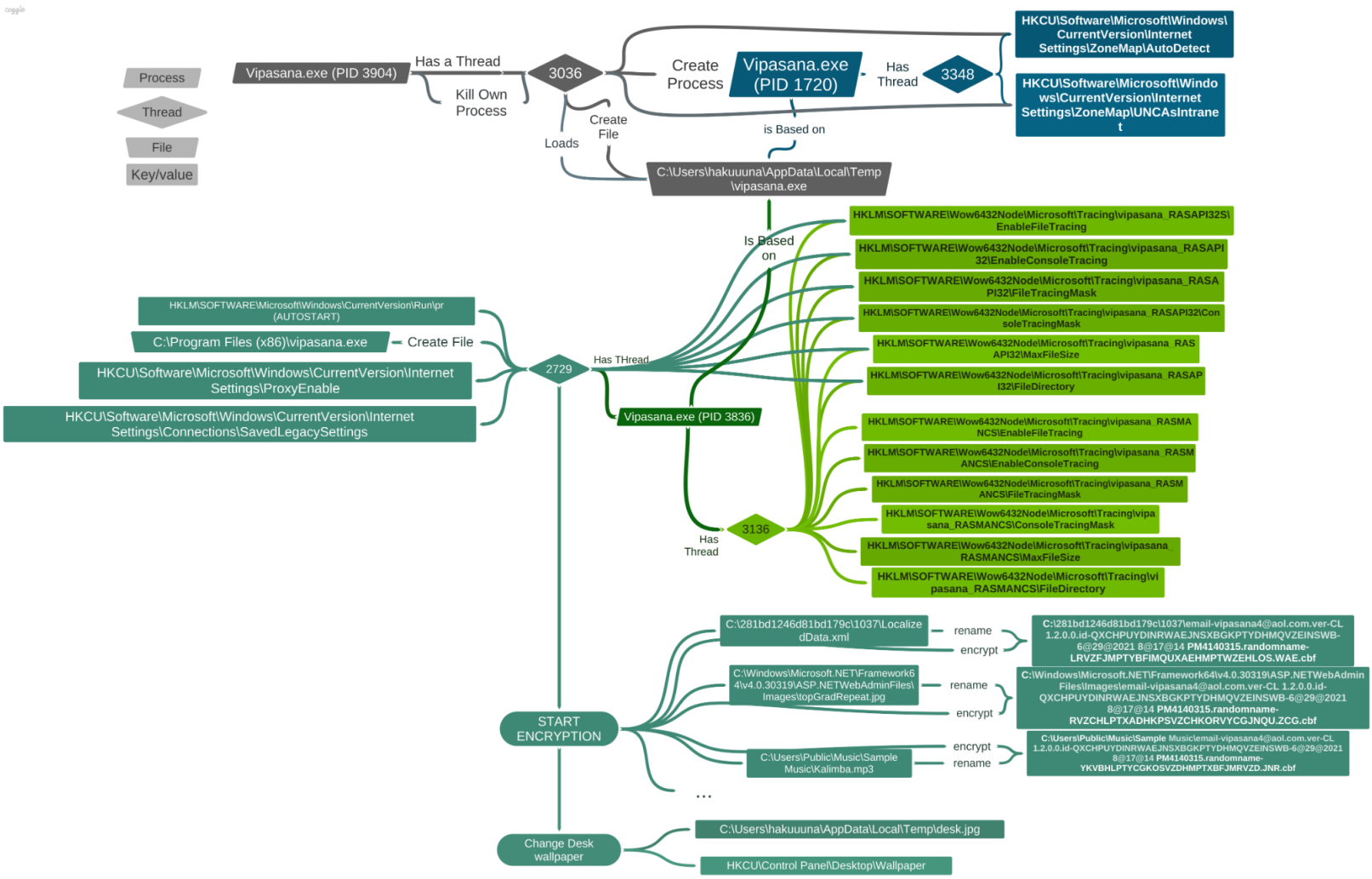


Figure 12: Graphical Representation of malware execution

- Interpretation

Our initial process creates 1 main thread, this thread responsible for creating a temporary file to hold our malware. Based on this file our malware gets executed another time creating another process and killing the initial one. Both threads of both processes tried to interact with the ZoneMap of internet Privacy settings which was added to Internet Explorer to give users more control over cookies.

We suspect the process is trying to get access to sites credentials and user information through detecting domains and sites.

This previous section gets repeated until a process is given higher privileges. This process creates 3 threads

First thread is responsible for maintaining the persistence part of our ransomware by adding the program to the Program files folder and referencing it with the autostart run key causing our ransomware to be executed when the user logs in.

We have both this thread and the second thread interacts with vipasana_RASAPI32 and vipasana_RARMANCS32. These registry keys get created the first time an application interacts with the Remote Access API, "rasapi32.dll", and the Remote Access Connection Manager, "rasman.dll". Since these Dynamic-Link Libraries (DLLs) are related to RAS, it indicates that applications that have "RASAPI32" and "RASMANCS" registry keys attempted network connections.

The third process is responsible for encryption. First it renames the file and then encrypts it.

Finally this same thread changes the Desktop background completing this ransomware execution

2. Registry analysis

The registry or Windows Registry contains information, settings, options, and other values for programs and hardware installed on all versions of Microsoft Windows operating systems. For example, when a program is installed, a new subkey containing settings such as a program's location, its version, and how to start the program, are all added to the Windows Registry.

There are common ways of achieving Persistence used by Malware like Modifying registry keys. Regshot seems a decent choice to analyze that.

- Regshot

Regshot is a dynamic malware analysis tool that allows an analyst to perform before and after snapshots of the Windows Registry. Typically, this is used to capture a snapshot of the system prior to executing malware and then immediately afterwards.

The goal is to identify any changes to the registry that the malware made. This may give more indication as to what the malware is capable of, if any additional files are dropped, or any other Indicators of Compromise ("IOCs"). In many cases, including my own, Regshot lives within its own Virtual Machine that is reserved for dynamic analysis.

Reshot tool The first shot was taken before running the sample on the target machine and the second shot was taken after the sample was executed.

- Taking Advantage of the Comparison of the registry before and after execution done with RegShot, we can see all the keys manipulations for the purpose of identifying any particular changes for the ransomware to achieve Persistence

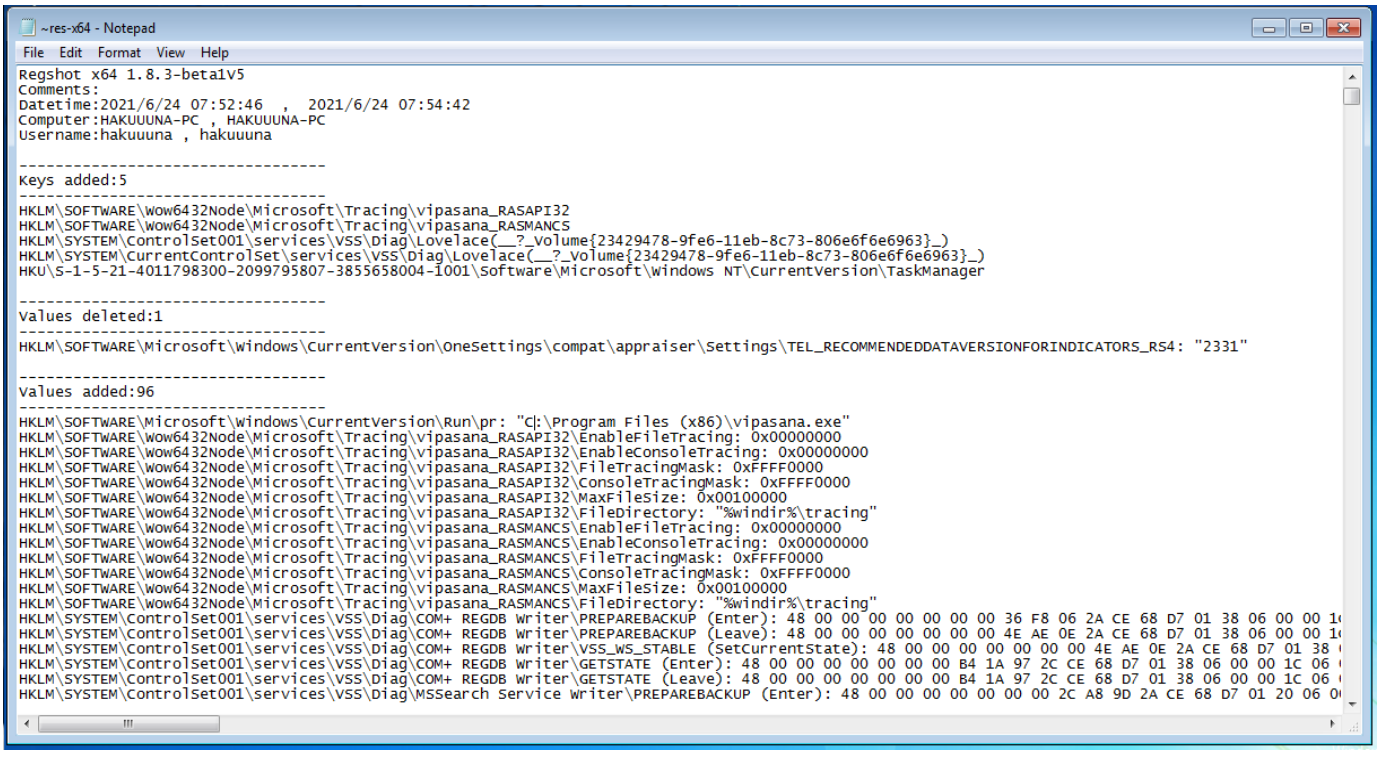
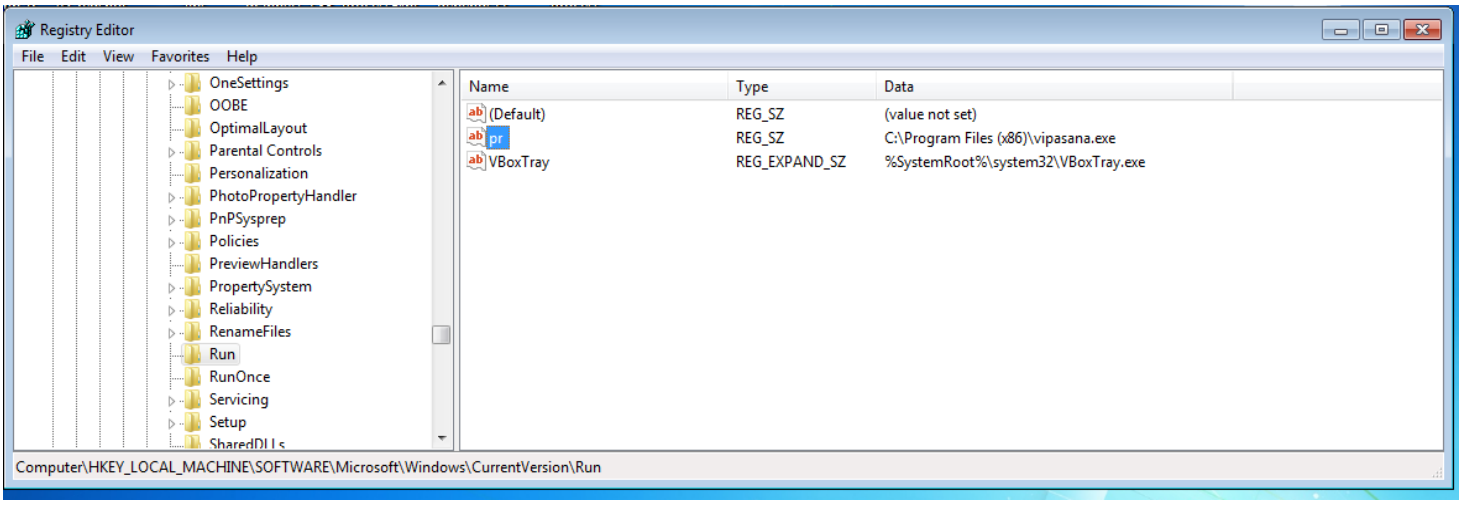


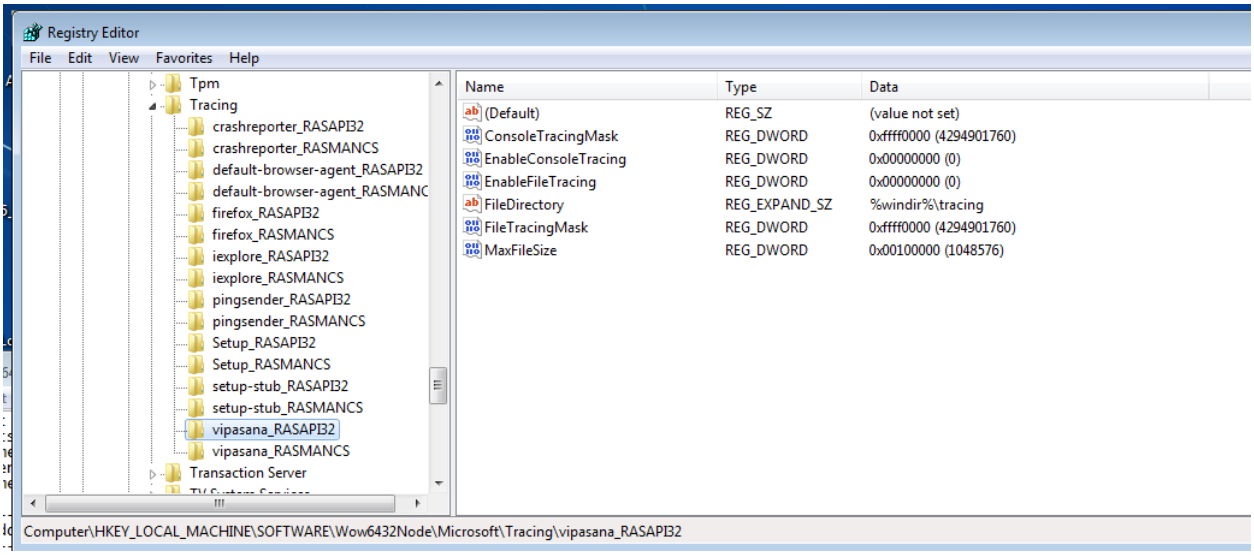
Figure 13: Regshot Result

The result showed that a malicious Run key to the registry was added by the sample.

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Run

Which is often used by malware analysis to achieve persistence to exploit the system level privileges.





We Also notice that our ransomware interacts with Vss writers which is interesting to analyze.

- In-Box VSS Writers

Shadow Copy (also known as Volume Snapshot Service, Volume Shadow Copy Service or VSS) is a technology included in Microsoft Windows that allows taking manual or automatic backup copies or snapshots of computer files or volumes, even when they are in use.

- Vssadmin

Vssadmin is a default Windows process that manipulates volume shadow copies of the files on a given computer. These shadow copies are often used as backups, and they can be used to restore or revert files back to a previous state if they are corrupted or lost for some reason. Vssadmin is commonly used by backup utilities and systems administrators. Also adversaries reliably use it to delete backups during ransomware infections.

- OM+ Class Registration Database Writer

This writer is responsible for the contents of the %SystemRoot%\Registration directory.

- MSSearch Service Writer

This writer exists to delete search index files from shadow copies after creation. This is done to minimize the impact of Copy-on-Write I/O during regular I/O on these files on the shadow-copied volume.

- Registry Writer

The Windows Registry Service supports a VSS writer, called the registry writer, which allows requesters to back up a system registry using data stored on a shadow copied volume.

- Shadow Copy Optimization Writer

This writer deletes certain files from volume shadow copies. This is done to minimize the impact of Copy-on-Write I/O during regular I/O on these files on the shadow-copied volume. The files that are deleted are typically temporary files or files that do not contain user or system state

Vss Writer (Keys)	Values
OM+ Class Registration Database Writer	GetState (Enter/Leave) PrepareBuckup(Enter/Leave)

	Vss_ws_stable(SetCurrentState)
MSSearch Service Writer	GetState (Enter/Leave) PrepareBuckup(Enter/Leave) Vss_ws_stable(SetCurrentState)
Registry Writer	GetState (Enter/Leave) PrepareBuckup(Enter/Leave) Vss_ws_stable(SetCurrentState)
Shadow Copy Optimization Writer	GetState (Enter/Leave) PrepareBuckup(Enter/Leave) Vss_ws_stable(SetCurrentState)
SwProvider	PROVIDER_BEGINPREPARE(Enter/Leave) PrOVIDER_ENDPREPARE(Enter)

Table 6: Vss Writers (Key/Value)

- All interactions with Vss by ransomware are suspicious to delete Shadow copies so that we can't restore file access by reverting to the shadow copies. As a note, interacting with vssadmin *should* require administrative privileges.

3. Network Analysis

- ApateDNS

Another great tool for performing Dynamic Malware Analysis is ApateDNS. ApateDNS is a tool for controlling DNS responses and acts as a DNS server on your local system. ApateDNS will spoof DNS responses to DNS requests generated by the malware to a specified IP address on UDP port 53. The IP address or hostname is often retrieved from the malware by performing static malware analysis, for example by examining the resources sections, or by using sandboxes.

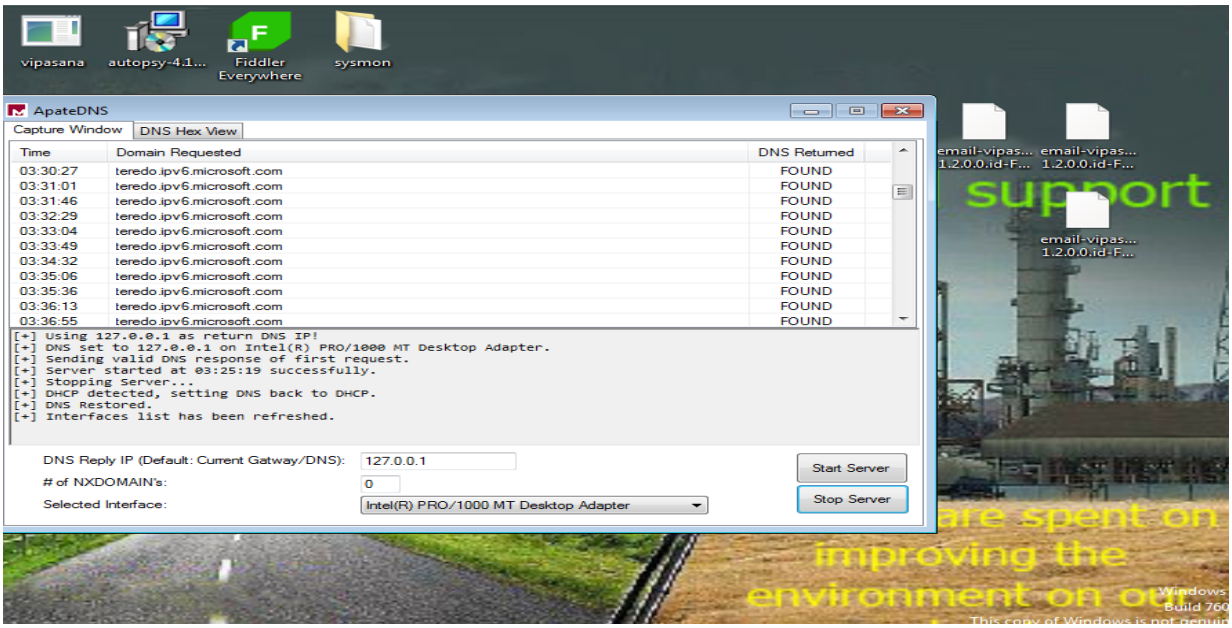


Figure 13: ApateDNS during Execution

The sample made no relevant DNS request. Implying that it is an offline Ransomware; sample does not require internet in order to carry out its malicious activities on the victim's machine

- Wireshark Tool

Wireshark is used to analyse a network to the greatest detail to see what is currently happening and capture packets to files. Wireshark can be used for live packet capturing, deep inspection of hundreds of protocols,

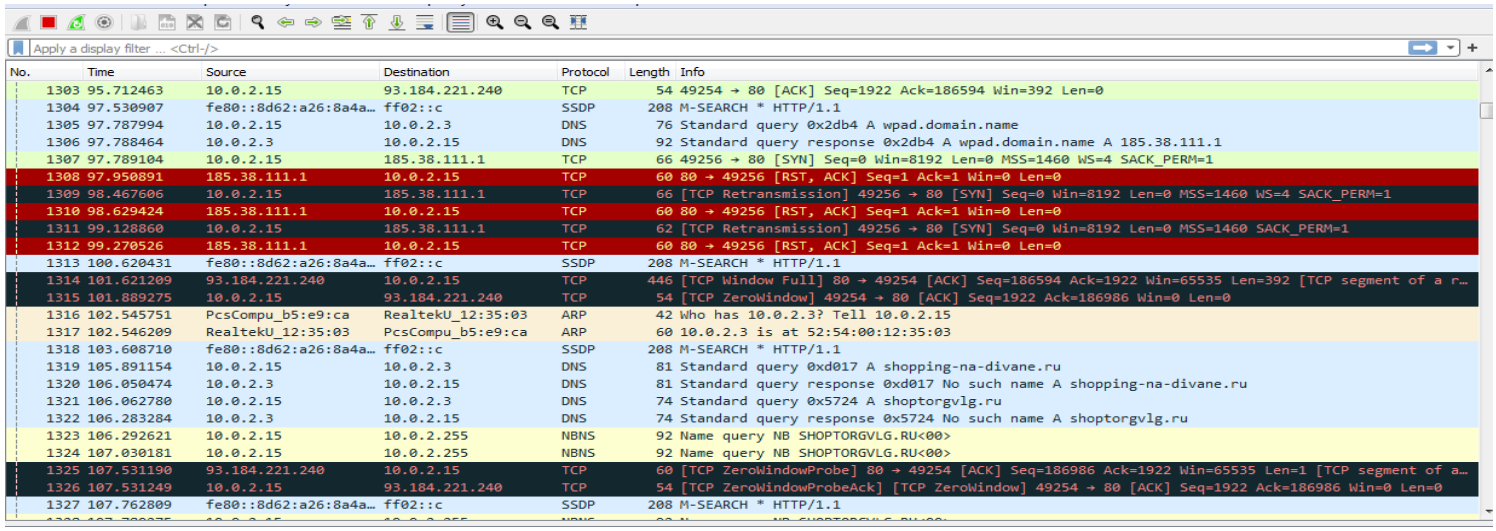


Figure 12: Wireshark during Execution

No relevant hosts were contacted; no relevant HTTP requests were made. No communication between the sample and any external server like command & control were requested.

Meaning that Vipasana does not require an internet connection to do its misdeeds which is an innovation, since previous ransomware strategies pretty much did communicate over the network > generally they creates an RSA key pair, encrypt your files with the public key

– generally done in a hybrid approach by mixing symmetric and asymmetric encryption to send the private key required for decryption to the server of the attacker > and gives it back after ransom has been paid .. or not ;)

4. Memory analysis using Volatility

dumping the memory after the ransomware had run. And just in case we needed to use more tools we gonna copy the image dump into a .raw file.

skipping the initial profile identification to our particular malware process

```
> volatility -f vipasana.raw --profile=Win7SP1x64 pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                         Exit
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
***
0xfffffa8004192600 VBoxTray.exe          2384   2232     15      146      1      0 2021-06-04 09:46:52 UTC+0000
0xfffffa8003f58b00 vipasana.exe          2396   2232      1       92      1      1 2021-06-04 09:46:53 UTC+0000
0xfffffa80041b0060 dllhost.exe           2488    584      6       91      1      0 2021-06-04 09:46:53 UTC+0000
```

output 1: memory dump processes list

Since we already identified dlls and other properties regarding our ransomware. In this analysis we are more interested in handles which includes files, registry keys, mutexes, named pipes, events, window stations, desktops, threads, and all other types of securable executive objects.

- Keys

```
volatility -f vipasana.raw --profile=Win7SP1x64 handles -p 2396 -t Key
Volatility Foundation Volatility Framework 2.6
Offset(V)          Pid      Handle     Access Type     Details
------------------ ------ ---------- ---------- ---------- -------
0xfffff8a0021ca860   2396        0x4        0x9 Key          MACHINE\SOFTWARE\MICROSOFT\WINDOWS
NT\CURRENTVERSION\IMAGE FILE EXECUTION OPTIONS
0xfffff8a001df5b40   2396       0x14        0x9 Key          MACHINE\SOFTWARE\MICROSOFT\WINDOWS
NT\CURRENTVERSION\IMAGE FILE EXECUTION OPTIONS
0xfffff8a001dc36a0         2396                0x20             0x20019   Key
MACHINE\SYSTEM\CONTROLSET001\CONTROL\NLS\SORTING\VERSIONS
0xfffff8a002e4bb90         2396                0x24                0x1   Key
MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER
0xfffff8a0017829e0   2396        0x3c     0x20019 Key          MACHINE
0xfffff8a001dad470         2396                0x98             0xf003f   Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001
0xfffff8a0041fd5c0         2396                0xa4             0x20019   Key
MACHINE\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\RUN
0xfffff8a001df58d0         2396                0xd0                0x1   Key
MACHINE\SYSTEM\CONTROLSET001\CONTROL\NLS\CUSTOMLOCALE
0xfffff8a002fff5b0         2396                0xe0                0x1   Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVER
SION\EXPLORER
0xfffff8a0021cc660         2396                0xe4             0xf003f   Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001_CLASSES
0xfffff8a00a26b4f0         2396                0xe8             0xf003f   Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001_CLASSES
0xfffff8a0091db320         2396                0xec             0x20019   Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVER
SION\EXPLORER\FILEEXTS
0xfffff8a0091d84c0         2396                0xf4             0x20019   Key
MACHINE\SOFTWARE\WOW6432NODE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER\FOLDERDESCRIP
TIONS\{B4BFCC3A-DB2C-424C-B029-7FE99A87C641}\PROPERTYBAG
0xfffff8a0021e57c0   2396       0x144     0x20019 Key          MACHINE\SOFTWARE\CLASSES\.EXE
0xfffff8a0092a5150         2396                0x148             0x20019   Key
MACHINE\SOFTWARE\CLASSES\EXEFILE\SHELL\RUNAS
0xfffff8a002531260   2396       0x14c     0x20019 Key          MACHINE\SOFTWARE\CLASSES\EXEFILE
```

```
0xfffff8a0024e5060          2396                                0x150                          0x20019    Key
MACHINE\SOFTWARE\WOW6432NODE\MICROSOFT\INTERNET
EXPLORER\MAIN\FEATURECONTROL\FEATURE_UNC_SAVEDFILECHECK
0xfffff8a0025137f0          2396                                0x154                          0x20019    Key
MACHINE\SOFTWARE\CLASSES\EXEFILE\SHELL\RUNAS
0xfffff8a0025d23a0          2396                                0x15c                          0x20019    Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVER
SION\INTERNET SETTINGS\ZONEMAP
0xfffff8a002596060  2396          0x160          0x20019 Key          MACHINE\SOFTWARE\POLICIES
0xfffff8a0024e5610          2396                                0x164                          0x20019    Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001\SOFTWARE\POLICIES
0xfffff8a0025b2060          2396                                0x168                          0x20019    Key
USER\S-1-5-21-4011798300-2099795807-3855658004-1001\SOFTWARE
0xfffff8a00939b9d0  2396          0x16c          0x20019 Key          MACHINE\SOFTWARE\WOW6432NODE
0xfffff8a0024e5550          2396                                0x170                          0x1    Key
MACHINE\SOFTWARE\POLICIES\MICROSOFT\WINDOWS\CURRENTVERSION\INTERNET SETTINGS
```

Output 2: Vipasana Keys list

- Importance of Mutex

It is used by malware to make sure once infected computer doesn't get infected again to reduce the noise and footprint.

```
volatility -f vipasana.raw --profile=Win7SP1x64 handles -p 2396 -t Mutant
Volatility Foundation Volatility Framework 2.6
Offset(V)          Pid          Handle          Access Type          Details
------------------ ------ ------------------ ------------------ ---------------- -------
0xfffffa80040ef520  2396           0x34         0x1f0001 Mutant
0xfffffa80041be9e0  2396           0x38         0x1f0001 Mutant
0xfffffa8004109ce0  2396          0x158         0x1f0001 Mutant         ZonesCounterMutex
```

Output 2: Vipasana Mutexes list

V. Stage Four: Manual Code Reversing

1. The Main Function

start with copying 3 RSA public keys into 3 seperate variables and obfuscating them with concatenating something like 13G13G....

There is 3 blocks of code like this , each one for an RSA key

```
copy(
        (volatile signed __int32 *)&RSA_KEY_1,
        (signed
__int32)"821857355842386857676328616241772748872935655387414443624355838508201329201831426999905132114312183
5"
"151586955166758552592175126886015333262208526809701515868703069177388605678085226984126750655824441 1");
 copy_to_address(
        (char **)&RSA_KEY_1,
"034841179011923620441073737062996478327605039893796109496209032899610769220472357095694421483517763907710 4
5819406380" "727053579573045429371687056037657759416374532634792426441798055715624726414878656648");
 copy_to_address(
        (char **)&RSA_KEY_1,
"979320572088638040973173801825576261200923711609713660251520171228818322245914565214120414363524247155454 0
8606242333" "615724802894715060374899930748348693104568782083465816005355018530243731351021765414");
 copy_to_address(
        (char **)&RSA_KEY_1,
"1746100718072807:65537|||13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G
13G13G13G13G13G1"
"3G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G13G1");
 copy_to_address(
        (char **)&RSA_KEY_1,
"G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G1
4G14G14G14G14G1"
"4G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G14G1");
 copy_to_address(
        (char **)&RSA_KEY_1,
"G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G1
5G15G15G15G15G1"
"5G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G15G1");
 copy_to_address(
        (char                                                                                **)&RSA_KEY_1,
"G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G1
6G16G16G16G16G1"
"6G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G16G1");
 copy_to_address(
        (char **)&RSA_KEY_1,
"G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G1
7G17G17G17G17G1"
"7G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G17G1");
 copy_to_address(
```

```
            (char **)&RSA_KEY_1,
"G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G1
8G18G18G18G18G18G1"
"8G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G18G1");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G1
9G19G19G19G19G19G1"
"9G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G19G1");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G
110G110G110G110"
"G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110G110");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111
G111G111G111G111"
"G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111G111");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112
G112G112G112G112"
"G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112G112");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113
G113G113G113G113"
"G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113G113");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114
G114G114G114G114"
"G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114G114");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115
G115G115G115G115"
"G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115G115");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116
G116G116G116G116"
"G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116G116");
 copy_to_address(
            (char **)&RSA_KEY_1,
"G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117
G117G117G117G117"
"G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117G117");
 copy_to_address(
```

```
            (char **)&RSA_KEY_1,
"G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118
G118G118G118G118"
"G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118G118");
  copy_to_address(
            (char **)&RSA_KEY_1,

"G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119
G119G119G119G119"
"G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119G119");
  copy_to_address(
            (char **)&RSA_KEY_1,
"G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120
G120G120G120G120"
"G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120G120");
```

Code 1: RSA Keys Obfuscation

After This the malware will call 2 functions to remove the concatenated random string

```
v2 = length_of_good_data((_BYTE *)def, (_BYTE *)RSA_KEY_1); // look where ||| start
  keep_good_data(RSA_KEY_1, 1, (int)(v2 - 1), (char **)&RSA_KEY_1); // removing everything from
||| to the end
```

Code 2: Removal of concatenated random String

It will do this 2 more times

Public keys :

key1 :

```
821857355842386857676328616241772748872935655387414443624355838508201329201831426999905132114312183515158695516675855259217512688601533326220852680970151586870306917738860567808522698412675065582444110348411790119236204410737370629964783276050398937961094962090328996107692204723570956944214835177639077104581940638072705357957304542937168705603765775941637453263479242644179805571562472641487865664897932057208863804097317380182557626120092371160971366025152017122881832224591456521412041436352424715545408606242333615724802894715060374899930748348693104568782083465816005355018530243731351021765414174610071807280 7:65537
```

key2 :

```
4372050014365306731665318876644049576237295145980325470433244938064878163614460 2
6577236003064166731819186331585637124165446185782493072948952026240795801918744 6
0024499933472381883027218220456554995247651582812147457184212659659894425045623 9
4836294548216033955590900254939392369170497228950606445248805833582659669548453 9
0249653878952051623577989835005175557711226524736345836930175859781073506130465 07
4628669730692231071425576803186344618387042261673228591278428278650137710547002 2
8739253573019680614943070394172993588023726914179370105897964778879852347053961 8
808018157421725342172583159698752661298411915027215343 3:65537
```

Key3 :

```
1126048008758310315572083729673247664750217706440310384210425629270517516681654 1
1940182144165267285922508856933464056319020782787993221703588259052568140752260 0
4951445616760965330764671293592279229722221601967433056071066493177969864547534 5
9601233200635881921575473627110125132446168309888011818714113680056574527396080 0
3735953272315257595705653934528792786750521804080914049076565678631053401774277 9
3199394758642966771898522304566530909595729469230977110341062913201724128815529 6
5151218490667137298567296227170148 43
9504013389642532700732026101084373980605809835866817301737873032183002270768131 0
903828104524796878143:65537
```

We can notice that it's using a 65537 as exponent

- This is the part where the malware will Create the data that will replace the name of the files after encryption

```
copy(
          (volatile signed __int32 *)&email,
          (signed
__int32)"vipasana4@aol.com|||KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
KKKKKKKKKKKKKKKKKKKKKKKKKKKK"

"KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
KKKKKKKKKKKKKKKKKKKKKKK");//
  //
  //
  //
  v5 = length_of_good_data((_BYTE *)def, (_BYTE *)email);
```

```
keep_good_data(email, 1, (int)(v5 - 1), (char **)&email);
*(double *)&time_date = sub_409BB0();
sub_40A7E8(v7, (char **)&v65, time_date, time_date >> 32);
copy((volatile signed __int32 *)&::time_date, v65);
copy((volatile signed __int32 *)&CL_str, (signed __int32)"CL 1.2.0.0");//
```

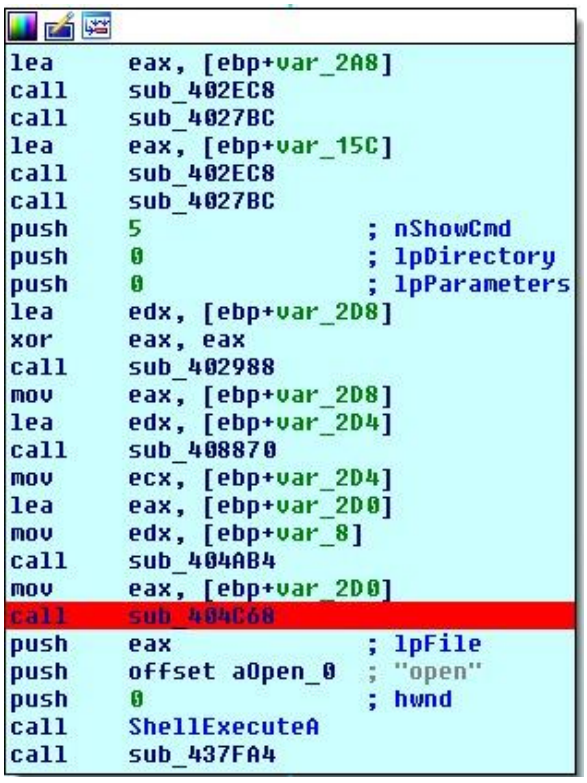
Code 3: file name after encryption

the name of files after encryption will contain email "vipasana4@aol.com" , the Time and date when the file were crypted and the string "CL 1.2.0.0" and (another random 2 generated string we will see them later)

2. Persistence

Now to the persistence part , The malware will try to copy itself to C:\Program Files (x86) and rerun the malware from there and kill the current process

it's using the API shellexecuteA to Create the new process then killing the current one with OpenProcess ,TerminateProcess in function sub_437FA4.



Code 4 : Assembly Code calling function used for malware persistence

the new process will redo all the previous operations but it will check its location , this time its in "C:\Program Files (x86)" so it will pass the creation of new process and try to add new Key in the registry in HKEY_LOCAL_MACHINE\software\microsoft\windows\currentversion\run using the APIs RegCreateKeyExA , RegSetValueExA, this tells us that anytime the computer starts it will start the malware on any user , so it will not just encrypt the current user files , All users will lose their files .

Generation of some keys that will be used in encryption

Detecting Encryption functions with PEID We see The RSA encryption and decryption functions , and after more digging into the code it looks like it makes sense that those are The Encryption and decryption Functions.

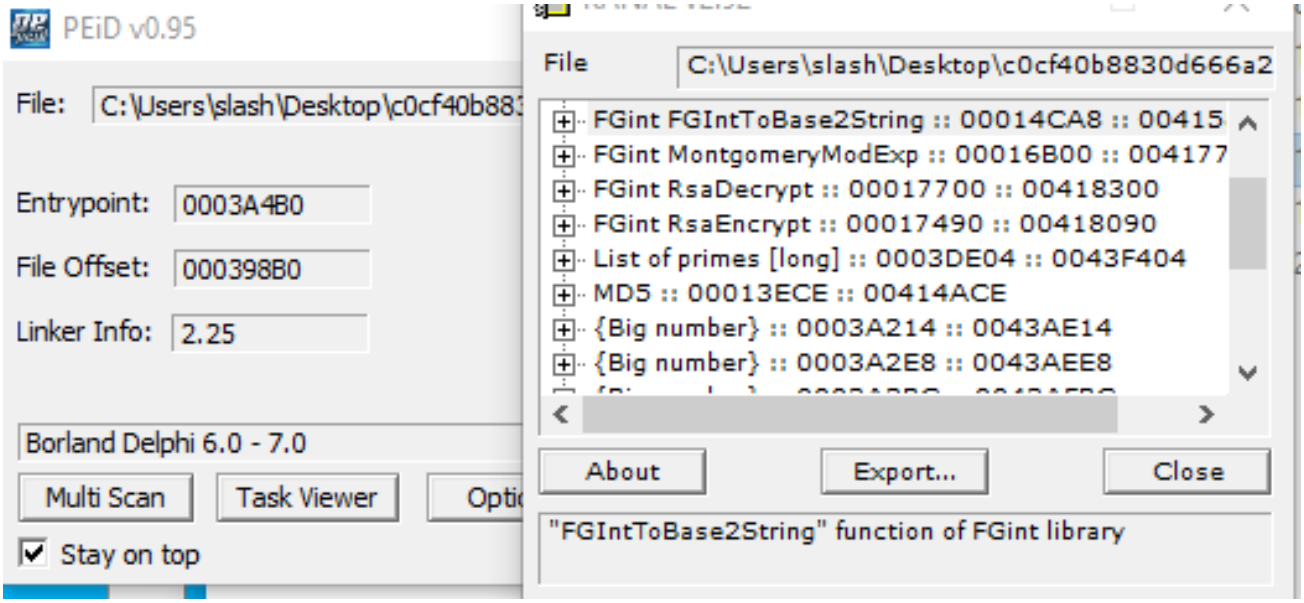


Figure 16 : Encryption functions detection using PEiD

First the malware will Encrypt and Decrypt the String "12345678900987654321" using RSA (with a generated key of course not the one will be used in files encryption)to test maybe if the encryption is working as expected , The malware will then generate random 2048 alphanum characters based on an internal state that gets modified by checking the time , looks like its creating some sort of custom pseudo random Number generator and (maybe defeating the debugger since program execution in debugger is slower !!) but there is ways to defeat that ,using maybe some Hooking technique

```
do
 {// generating 2048 caracter ,
 //
          checking_for_speed_of_program();
          if ( Generating(2u) )
          {
          if ( Generating(2u) )
          {
          checking_for_speed_of_program(); //
          v34 = Generating(0x1Au);
          LOBYTE(v34) = v34 + 65;
          copy_generated_caracter_0(v35, v34, (char **)&v45);
          sub_407EE4(v36, &v46, v45);
          copy_to_address((char **)&GENERATED_2048, v46);
          }
          else
          {
          checking_for_speed_of_program();
          v32 = Generating(0x1Au);
          LOBYTE(v32) = v32 + 65;
          copy_generated_caracter_0(v33, v32, &v47);
          copy_to_address((char **)&GENERATED_2048, v47);
          }
          }
          else
          {
          checking_for_speed_of_program();
          v31 = Generating(0xAu);
          copy_generated_caracter(v31, &v48);
          copy_to_address((char **)&GENERATED_2048, v48);
          }
          ++v30;
 }
 while ( v30 != 2049 );// // end generation
```

Code 5 : PRNG to generate 2048 characters

Something like :

```
234PrwA12345quy01344Psv01234NquY01234osw01234OrvZ02234pty01234Prv
A12345quY01234nrvZ1234NqUWz1234osWz1234oswa1234oTvz1234oswA1234Nq
uy01344PsWZ01234Prva1234osw01234Orvz1234oSuy1231234OQtY01234nrVz1
2344qUWz1234orWz1234nswa1234oSvz1234orwA12235pty01234ptx01234Orvz
1234nrVz0234nrwZ01235pux01234OrvA1234Nptx12235PsWY01234pTvz1234oS
Vy01245PrvA0234Nquy01234Psw01234OqtX01234orWY01234Prvz1234osw0123
4OrvZ01234otX00234nrvA11235ptX01234osWY01234PRuY01234osWz1234NqVX
z1234ptX00234nrWY01234Orvz1234osw01234OquY01234psXZ02334Qswa1244p
tx01234PrWY01234Orvz1234osWy12235pTXz1234osWy1234OquZ01234otWz123
4osv01234Nquy0124nrVy11234PSuz1234nRuy12345PTvZ01234ptx01234OrVx0
123NptX00234nrVy1124NpTw01134nSuy12235PSuZ02334qtXZ12345Qsx01234O
rvz1234nRUx01234Orwz1234otXZ02234PSuZ01234puYa1234PrvZ01344qtYa12
35PrVZ01234Orva1234osXz1234oswa1234oTVy01334PsvA12345quy02344QsWZ
01235PRuY01234osw01234OrvZ01234ptXz1234OrvZ02334qUWz1234osWz1234N
quY01234osWz1234oswA1124Npuy12234Psw01234Oquy1234nrVy01344qUw0123
4nSVx01234OrwA1134Nptx12235Psw01234OquY01234otx01234Oqvz12345QtXA
0234nsvZ11234ptX01234osvZ12235ptY01234orWY01234PRuZ01234osWz1234N
rvZ01234psXa1234osx01234Prvz1234osw01234Oquz1234nRvy01344Qsw01234
NQsX00234nrWy1134NptX01234osWY01234Prvz1234osWz1234Qsw01234Orvz12
34nsw01234Orvz1234osw01234OquY01234ptXZ02344QSvz1234oSva1234nrw01
234Oqvy1134nRUx01234OquY01234ptx01234PrvZ01344qtYa1235Prw01234Nqu
y1234oSUx01234Orvz1234osXZ02335QSuZ01244ptXz1234OrVy01235ptX01234
oRty1234OquY01234psXZ02234Pswz1234pTvz1234osvZ1234NquY01234nsWz12
34OrvZ01234otXZ12234Psw01234Oquy12345Qtx01234Orvz1234nrWz1234NquY
01234osx01234PrVx01234ptXa1234PrVZ01234Oqvz1234osw01234NquZ01234o
tx01234OrvZ12234ptY01234osvZ12335quY01234osw01234Orvz1234NQsWA113
4nswZ12234ptx01234ptx01234Oquz12345Qsx01234QSvz1234oSVx1123nrVx11
235ptX01234osWY01234Prwa1234otx01234Orvz1234nRvy0134nswZ01234ptx0
1234PsWY01234PRuY01234orWZ01234Pr
```

Then it will generate another 20 characters (which is actually a static global key stream)but this time just numbers using the same random generation technique.

Example :

```
8913789023456702
```

```
do
 {// generating 20 caracter
 //
        checking_for_speed_of_program();// changing an internal state for generating the next character
        v38 = Generating(0xAu);
        copy_generated_caracter(v38, &v44);
        copy_to_address((char **)&state_20_caracters, v44);
        ++v37;
 }
 while ( v37 != 21 );// End Generating
```

Code 6 : Generating static global key stream

2048 alphanum and the static global algo key stream will be encrypted with RSA with the 3 found first keys using this order key1 > key2 > key3 .

```
958086918083948811810444797761874776297322714107803427494 5
848488585714542348352528121515740438192995585409901276028264826402
833995791750115016371063002288075613954356459478752702903644415741
885949488286521129959697548034904382157080327448735755133943292081
652411557442026968782554777664924474472582723573913884340574079431
297941699874886786925529980524745584242406492091154204734351261821
568159528843754413914188077640909813818949284357216057170605005411
465347623635116645101466246053471827855740123350363177132310137691
888736429810688237873115665785805255148770188192416454329395688251
989649877839539215715234651863265564213824775018360674544668914 8
486876008624543165187912875281964117064691403264027945617474589361
434143206791599848132882608185503172228191641097269224426099586931
902496114073719312860919298241439757861431258716804726424302186121
438017847820422681440431654765260230563433992363334339072683834691
170478144892150454078624727841008609573055742403258059621304033631
```

```
89707251437139169517715688834978330606131237286576831852995073214
18971316974736748445861693435840267450645276465178811095334796100
10700915115701488130343161390600641290692691654996108644262879793
62915652191648111262710236801628481573501371339813098688399229152
```

This how the attacker can recover files since he has the private , he can recover the 2048 and the static global key

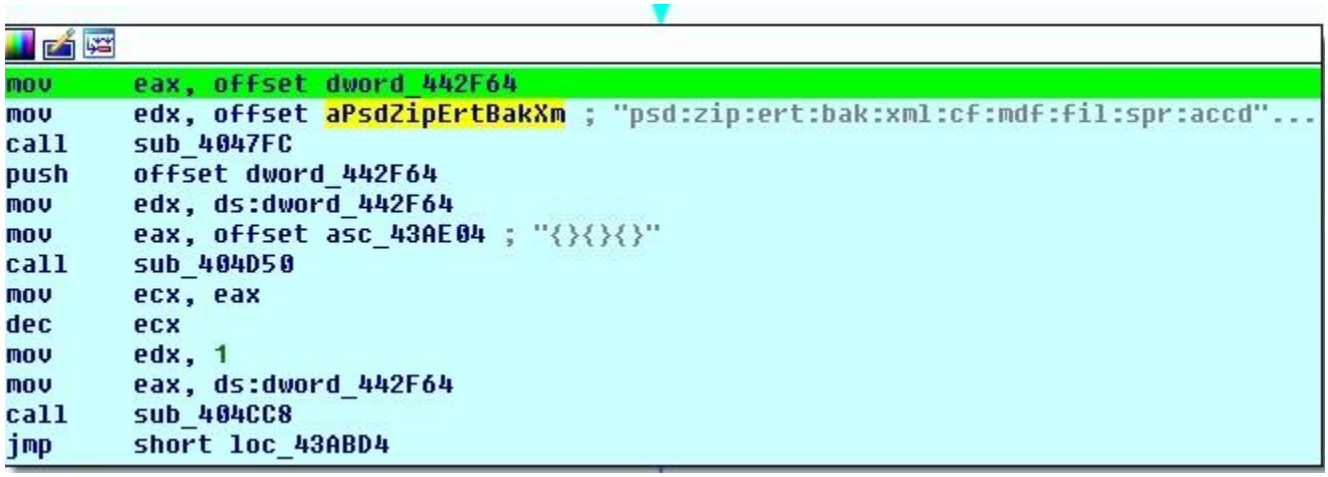
After That we find the Function that will generate The Random 2 strings that we talked about earlier , it will be something like this :

LTWZDGILNPRTWYACEHJLNQRUWYADEHJLNQRU-4@4@2021 7@22@11 AM369703

let's name it str1 OGOMATVCDJ.EAA0

3. Extensions

The malware will set up the extensions of course like the keys they are obfuscated with "{}{}{}" it will split them into different addresses , the splitting character is ":"



```
mov     eax, offset dword_442F64
mov     edx, offset aPsdZipErtBakXm ; "psd:zip:ert:bak:xml:cf:mdf:fil:spr:accd"...
call    sub_4047FC
push    offset dword_442F64
mov     edx, ds:dword_442F64
mov     eax, offset asc_43AE04 ; "{}{}{}"
call    sub_404D50
mov     ecx, eax
dec     ecx
mov     edx, 1
mov     eax, ds:dword_442F64
call    sub_404CC8
jmp     short loc_43ABD4
```

Code 7: Assembly Code setting up the extensions

Now every file with those extensions on all disks (We will see this part later) will get encrypted :

.r3d ,rwl ,rx2 ,p12 ,sbs ,sldasm ,wps ,sldprt ,odc ,odb ,old ,nbd ,nx1 ,nrw ,orf ,ppt ,mov ,mpeg ,csv ,mdb ,cer ,arj ,ods ,mkv ,avi ,odt ,pdf ,docx ,gzip ,m2v ,cpt ,raw ,cdr ,cdx ,1cd ,3gp ,7z ,rar ,db3 ,zip ,xlsx ,xls ,rtf ,doc ,jpeg ,jpg ,psd ,ert ,bak ,xml ,cf ,mdf ,fil ,spr ,accdb ,abf ,a3d ,asm ,fbx ,fbw ,fbk ,bdb ,fbf ,max

,m3d ,dbf ,ldf ,keystore ,iv2i ,gbk ,gho ,sn1 ,sna ,spf ,sr2 ,srf ,srw ,tis ,tbl ,x3f ,ods ,pef ,pptm ,txt ,pst ,ptx ,pz3 ,mp3 ,odp ,qic ,wps .

4. Encryption

First it will loop over All alphabet letters(Maj and min) Looking for available disks (Since in windows ,Disks are named by alphabet letters) then for each letter it checks if that disk exists , if true it will append

":\"

```
GetWindowsDirectoryA(&Buffer, 0xFFu);
  counter = 0;
  do
  {//  looping through Drives names
  //starts encrypting other drives
  //
    copy_generated_caracter_0(v30, counter + 65, (char **)&v60);
    sub_407EE4(v32, (LPSTR *)&v61, v60);
    v33 = v61;
    LOBYTE(v34) = Buffer;
    copy_generated_caracter_0(v35, v34, (char **)&v58);
    sub_407EE4(v36, (LPSTR *)&a2, v58);
    str_cmp(v33, a2);
    if ( !v15 )
    {
      copy_generated_caracter_0(v30, counter + 65, (char **)&v57);
      copy_to_address((char **)&v57, (char *)str_path_backslash);
      Looping_through_file_to_encrypt(v57);
    }
    ++counter;
  }
  while ( counter != 26 );
```

Code 8 : Looping over available drives to encrypt

then it will Call another function that will Loop over Files using APIs : FindFirstFileA and FindNextFileA. Those apis will return the handle to the file then it will use CreateFileA , ReadFileA to Read content from the file .

a. Encryption algorithm

first we have the 2048 alphanum already generated and encrypted , it will generate another 512 bytes (those are positions in the 2048 alphanum), so with the 512 positions , we choose 512 characters (lets call it an internal state) .the internal state is updated by this algorithm .

```
38
39  void updateState1(uint8_t* state) { // 0043547b
40      for (uint32_t i = 0; i < 0x200; i++) {
41          uint32_t currentValue = (uint32_t)state[i];
42          state[i] = (state[i] + 0x80 > 0x100) ? state[i] - 0x80 : state[i] + 0x80;
43      }
44  }
45
46  void updateState2(uint8_t* state) { // 004354f8
47      for (uint32_t i = 0; i < 0x1ff; i++) {
48          state[i] += state[i + 1];
49      }
50  }
51
52  void updateState3(uint8_t* state) { // 00435589
53      for (uint32_t i = 0; i < 0x200; i++) {
54          state[i] *= 2;
55      }
56  }
57
58  void scrambleValues(uint32_t* toUpdate, uint8_t* baseBytes) { // 00414a58
59      uint32_t oldValues[] = {
60          toUpdate[0],
61          toUpdate[1],
```

Code 9 : Update internal state

this state is put into a PRNG and return a 30000 bytes (this is the key cipher stream) , depending on the static global algo key stream we will choose an operation to do (there is 10 in total)

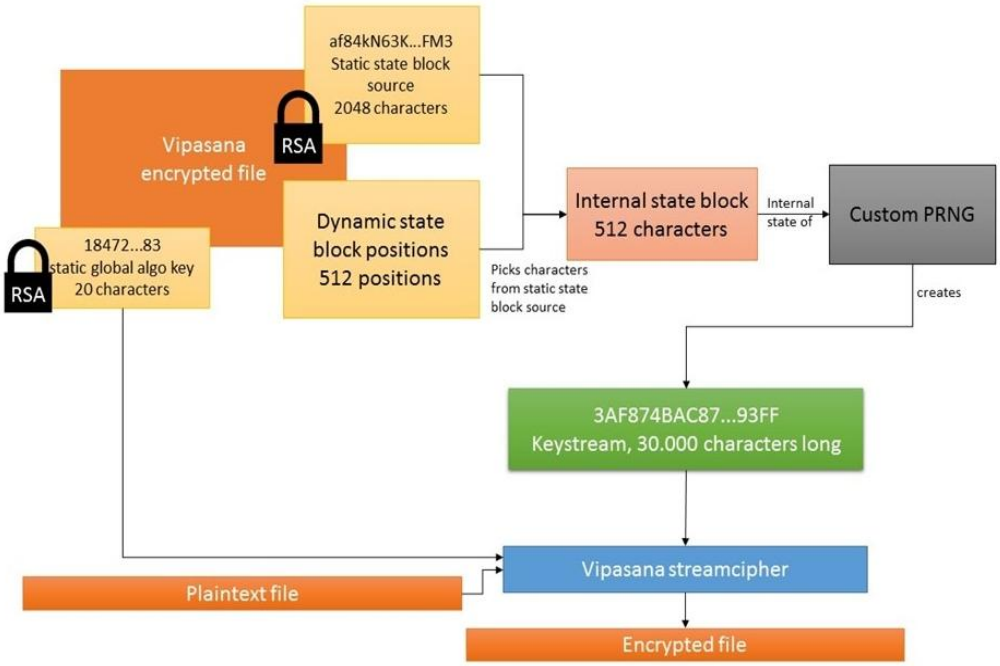


Figure 17 : Encryption Algorithm

the algorithm will be something like :

```
void encryptFileContent(uint8_t* fileContent, char* keystream, char *globalAlgoKey) // 004355f5
{
   for (int i = 0; i < 0x7530; i++) { // only the first 7530 bytes are actually encrypted!
          char algo = globalAlgoKey[i % 20];
          char keystreamByte = (uint8_t)keystream[i];
          switch (algo) {
          case '0':
                    fileContent[i] += keystreamByte;
                    break;
          case '1':
                    fileContent[i] -= keystreamByte;
                    break;
          case '2':
                    fileContent[i] += keystreamByte;
                    fileContent[i] += 0x32;
                    break;
```

```
        case '3':
                fileContent[i] += keystreamByte;
                fileContent[i] += 0x03;
                break;
        case '4':
                fileContent[i] -= keystreamByte;
                fileContent[i] -= 0x34;
                break;
        case '5':
                fileContent[i] -= keystreamByte;
                fileContent[i] -= 0x05;
                break;
        case '6':
                fileContent[i] += 0x06;
                break;
        case '7':
                fileContent[i] -= 0x07;
                break;
        case '8':
                fileContent[i] += 0x38;
                break;
        case '9':
                fileContent[i] -= 0x39;
                break;
        }
  }
}
```

Code 10 : Encryption Algorithm

Then at the end of the file , the malware will append the 512 positions , the global algo key encrypted with RSA , and the 2048 alphanum encrypted with RSA .

The malware will change the name of the file after encryption to something like : email-vipasana4@aol.com.ver-CL-1.2.0.id-OGOMATVCDJ.EAA0-time-date-of-the-day

5. Last part changing the Desktop image

in the the end the malware will load an embedded image from it's ressource section with The API LoadRessourceA named DESK and set it on the victim background

Figure 18 : Changing Desktop Image

Summary

- The malware copies itself to C:\Program Files (x86) , adds a new registry key in HKEY_LOCAL_MACHINE\software\microsoft\windows\currentversion\run for persistence .
- It encrypts the first 30 000 bytes of the file , if the file is large enough we can recover some informations .
- The times static global algo key contains 6 , 7 , 8 ,9 the key cipher will not even be used .
- Some windows files are encrypted , which means the possibility of plaintext attack on the static global algo key .
- Some researchers on boxcryptor found that the keystream is converging after the first 768 bytes because of how the the internal state of the PRNG change , since the update3 function will multiply the old state by 2 , after 8 times of calling update3 the internal state will always be 0 , since the call of the internal state will be like this : update1 -> update2 -> update3 -> update1 ….. , and we call each update function after generating 32 bytes of keystream , so update3 will be called after generating 96 bytes , 96 * 8 = 768 , so after 768 bytes the keystream is known .

Conclusion

The stages of our ransomware are in the form of a pyramid and as we go higher in the pyramid, the complexity of the analysis stage increases.

We began by a fully automated analysis with an online service ANY.RUN which gave us a broad overview of how our malware is behaving through using the Mitre Att@ck Matrix but did not go so far to identify anything about encryption.

Next to the static analysis we extracted the static properties of our executable identifying DLLs and Sections. Through knowing these DLLs functions we can identify what our ransomware might be doing.

Third we have the Dynamic analysis where we interact with the malware detecting all changes made while executing from process behavioral where we went through main operations done by our ransomware from persistence / credential access / encryption and finally changing the desktop wallpaper. Registry key manipulations where we identified ransomware interacting with Vssadmin leaving the victim with no backup files and finally network analysis verifying that our ransomware needs no internet access throughout its execution.

And finally the most difficult part is attempting to reverse engineer the ransomware where we identified the obfuscation and encryption functions used to encrypt the files.