CDAC-TVM
DCSF

"INVESTIGATING MALWARE IN THE CONTEXT OF NETWORK FORENSICS"

MINI PROJECT

SUBMITTED BY:

SURAJ TALKATKAR
ATHARVA JOSHI
VIRAJ PADEKAR
PRISCILLA A.
NIKITA THORAT

MINI PROJECT REPORT

On

"Investigating Malware in the Context of Network Forensics"

Submitted to

For Partial Fulfilment

Of

Post Graduation Diploma In

CYBER SECURITY AND FORENSICS

(September 2023)

Submitted By:

NAMES	PRN
SURAJ TALKATKAR	230960940051
ATHARVA JOSHI	230960940008
VIRAJ PADEKAR	230960940058
PRISCILLA A	230960940041
NIKITA THORAT	230960940053

UNDER THE GUIDANCE OF

MR JAYARAM P.

(Project Guide & Centre Co-ordinator)

ACKNOWLEDGEMENT

We take the opportunity to express our gratitude to Prof. Jayaram P Who was the driving force during our project work. His guidance, encouragement and inspiration throughout the various phases of this project kept us on our toes; throughout the job without his advice, guidance and cooperation we would not have succeeded so far. He has helped us as a friend, philosopher and in all possible ways.

Suraj Talkatkar

Atharva Joshi

Priscilla A

Viraj Padekar

Nikita Thorat

INDEX

CHAPTER 1 - ABSTRACT

This is a small-scale project investigating malware in the context of Network Forensics. Most of the incidents requiring network forensics will be based on malware-oriented events, such as network breaches, financial crime, data theft, and command and control. Most attackers will deploy command and control malware to enslave the compromised machine and gain leverage over the internal network for lateral movement. Generally, network forensics and computer forensics go hand in hand in case of investigating malware. The computer forensics investigator will find all that has changed on the system and where the malware resides. Then, they will find the executable causing the issues and upload them to a site, as mentioned as follows https://www.virustotal.com or http://www.hybrid-analysis.com to find more about the malware and its behaviour on the system and the network.

CHAPTER 2 - OBJECTIVES

- Dissecting malware on the network
- PyLocky ransomware decryption using PCAP data
- Decrypting hidden tear ransomware
- Behavior patterns and analysis

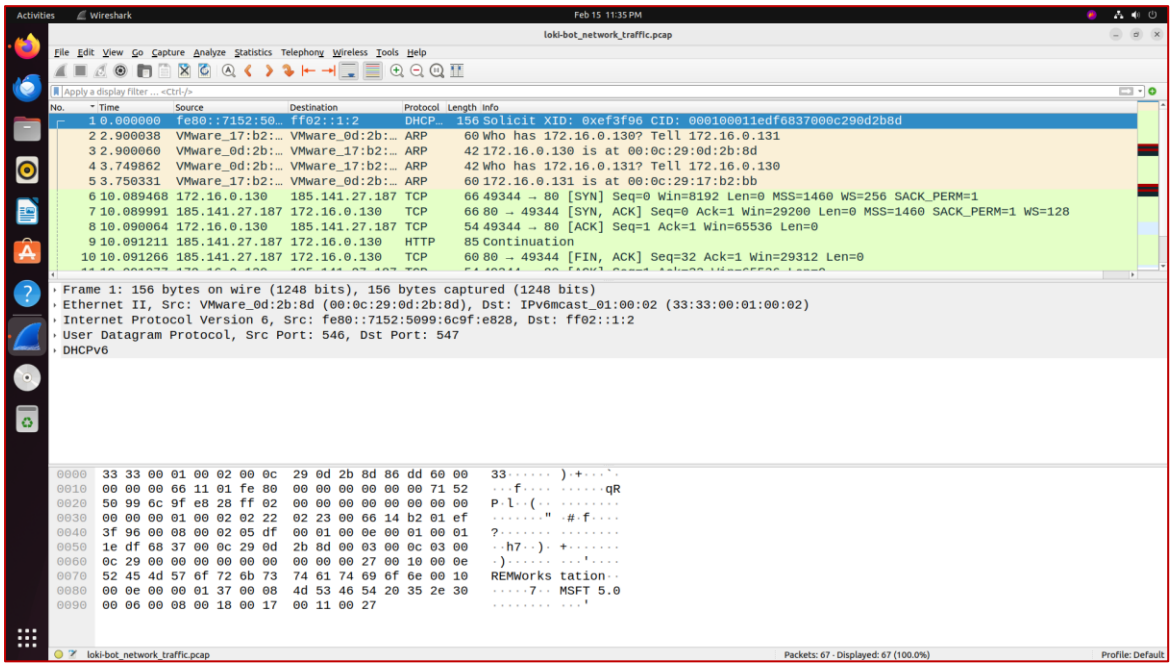
CHAPTER 3 - TECHNICAL REQUIREMENTS

- Wireshark v3.0.0 installed on
- Windows 10 OS and Ubuntu 14.04
- PCAP Files for the exercises
- NetworkMiner installed on Windows 10

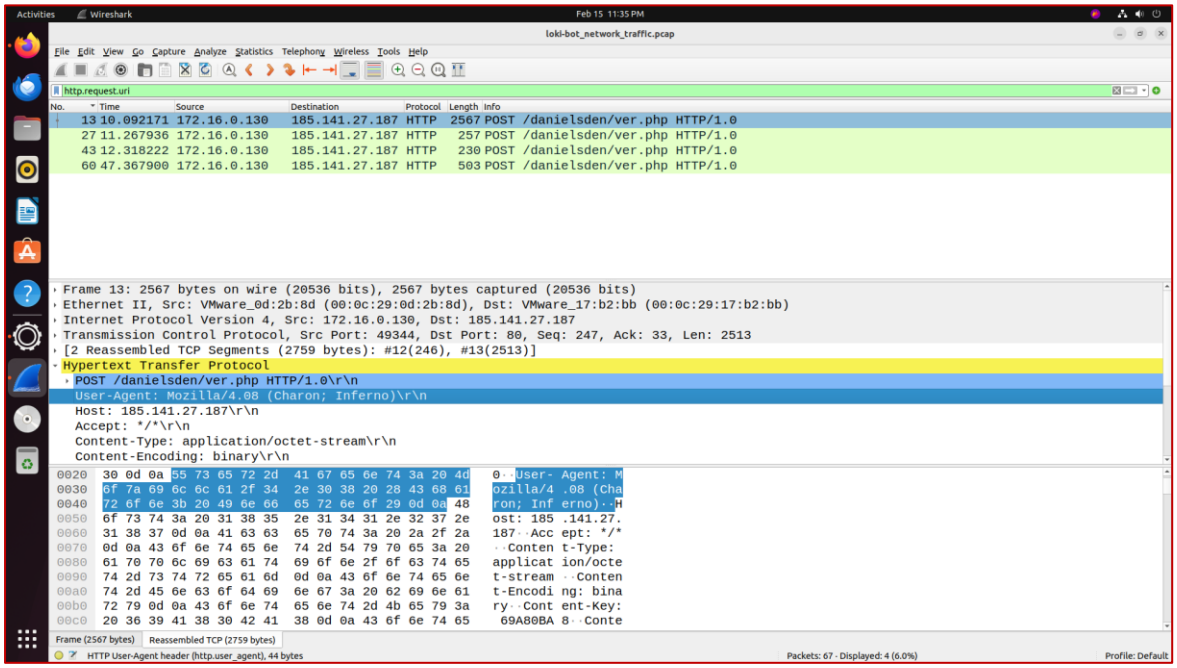
CHAPTER 4 - WORKING

Dissecting Malware on Network

Initially let's load the PCAP in Wireshark as follows:

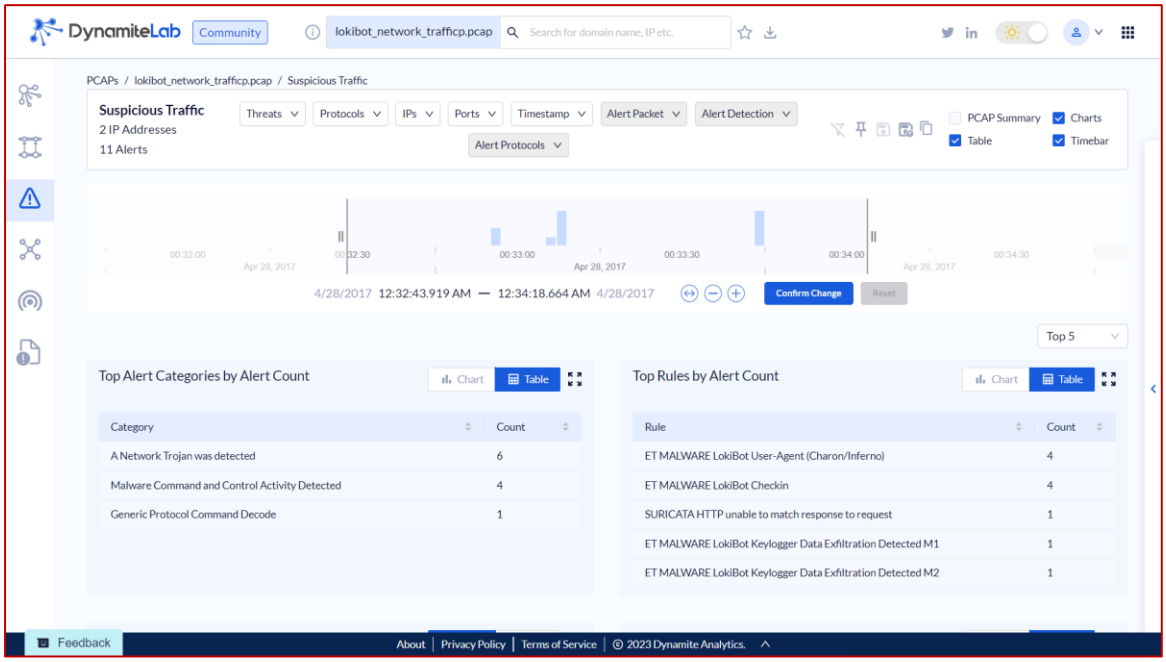


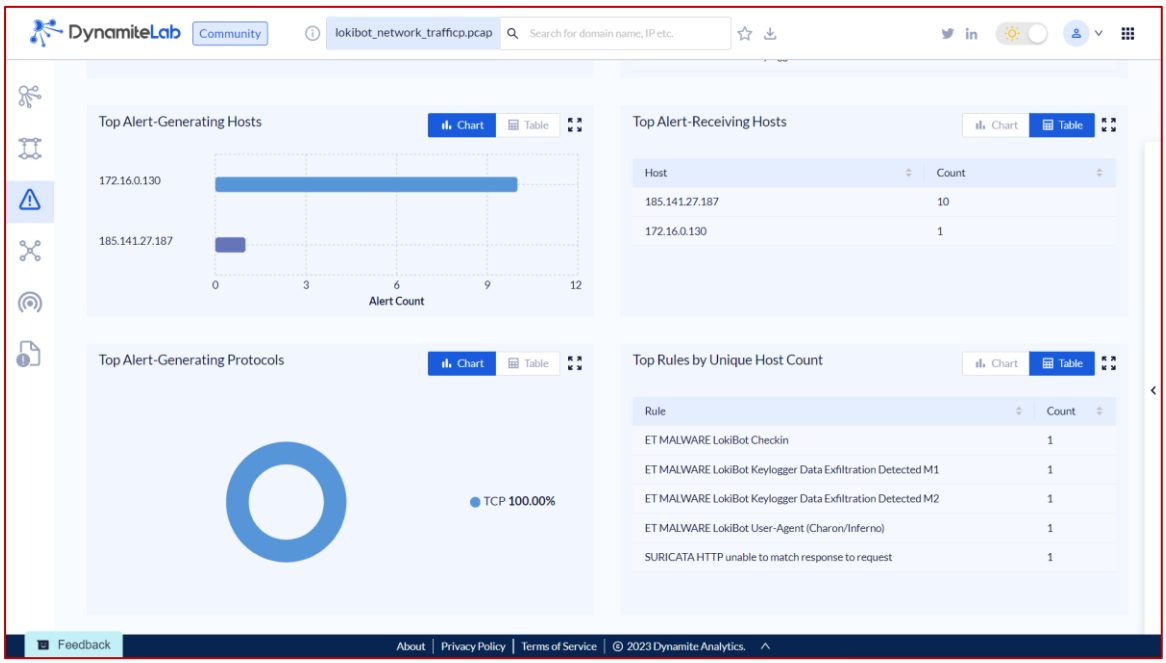
We can see in the above screenshot that there is a lot of HTTP data present in the We can see there is a lot of HTTP data present in the PCAP file. Let's add columns to display the full URI and User-Agent entries, and also filter the requests using the http.request.uri filter as follows:

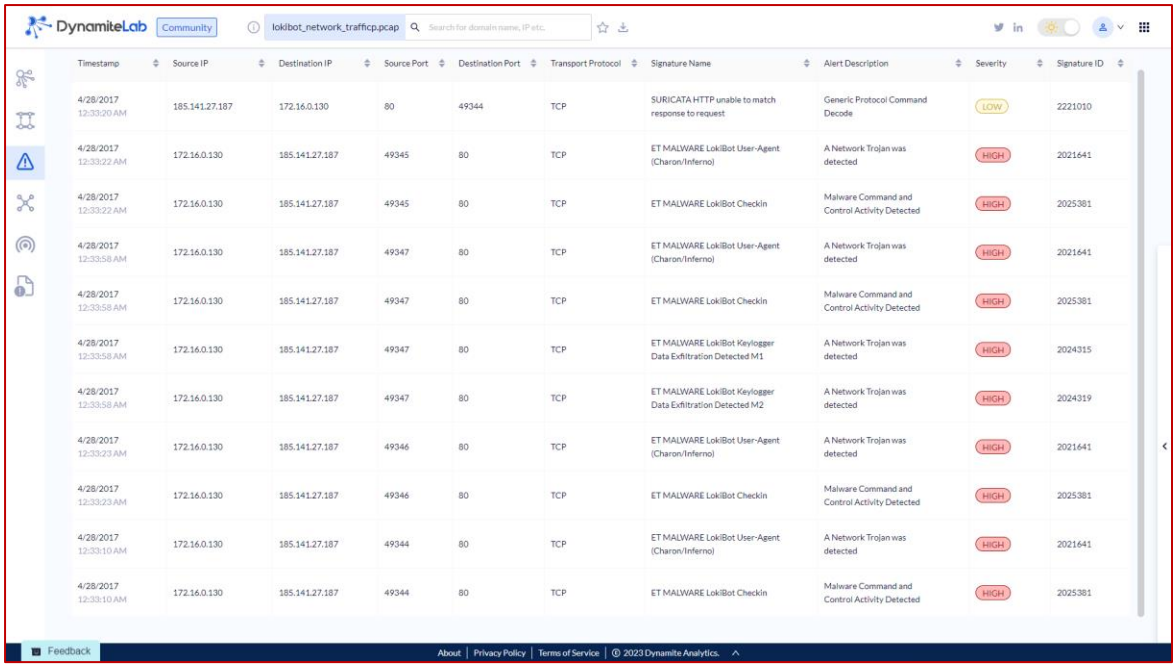


The user agent is quite important in malware communications since they might not be the standard user agents used by popular browsers. We can see we have Mozilla/4.08 (Charon; Inferno) as the user agent, and the URI contains a single user, as shown in the previous screenshot

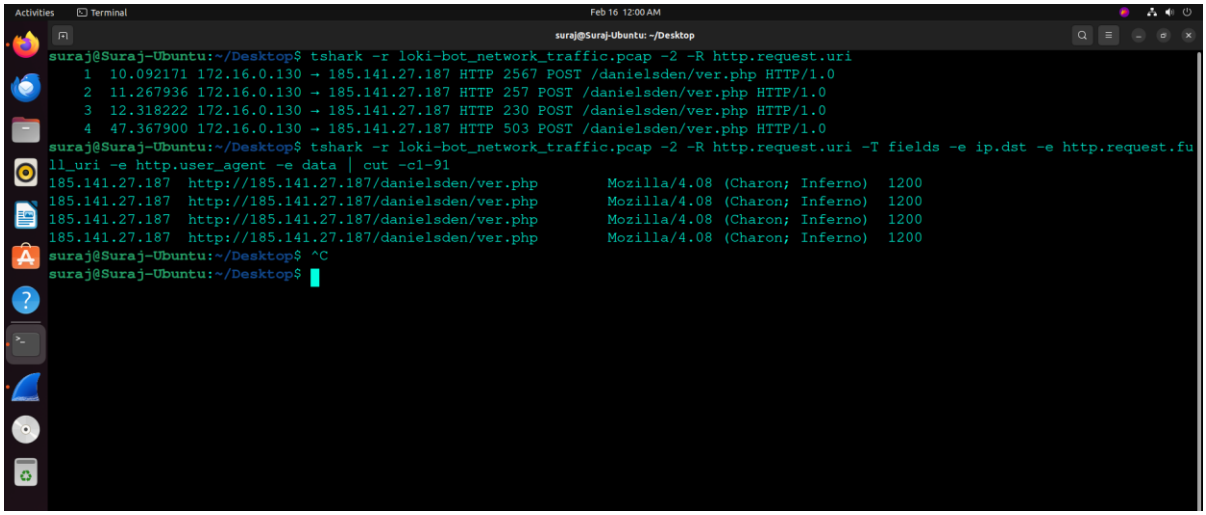
It seems that the HTTP requests are generated by the nefarious LokiBot. LokiBot is a popular malware that infiltrates data on infected systems. So as we can see in the recent result open the third link from the preceding results which is from https://lab.dynamite.ai/pcaps and analyze similar samples:







As we can see there have been numerous entries with similar behavior. The important items from the preceding list are the HTTP Method and the User-Agent columns. We can see that there is plenty of information about LokiBot analysis. The takeaway for us is the first-byte word of the HTTP payload is the LokiBot Version. Let's see what it is by making use of "tshark –r /home/deadlist/Desktop/loki bot_network_traffic.pcap -2 –R http.request.uri –Tfields –e ip.dst –e http.request.full_uri –e http.user_agent –e data –E separator=, | cut –c1-91" command. The command will read the PCAP file defined using the X switch and it will display all packets having the URI using http.request.uri filter. The command will print comma-separated values (-E separator=,) of fields like destination IP, full URI, User-Agent and Data (-Tfields). Since the last value is of the data field, the use of cut –c1-91 will print the first two bytes (Byte Word) of the data only as shown in the following screenshot:



We can see the first-byte word is 1200, which implies 00 12(18) being divided by 10, which means that we have the LokiBot version 1.8. We can see that, in the next word (the next two bytes), we have hexadecimal values of 27, 28, and 2b, and, according to the information that we have read, this value defines the functionality of the packet and a value 27 implies Exfiltrate Application/Credential Data, 28 implies Get C2 commands, and 2b implies Exfiltrate Keylogger Data.

This means that the LokiBot has done the following activities in order:

- Exfiltrated an application's credential data twice
- Made the new command, which was to exfiltrate keylogger data
- Sent keylogger data

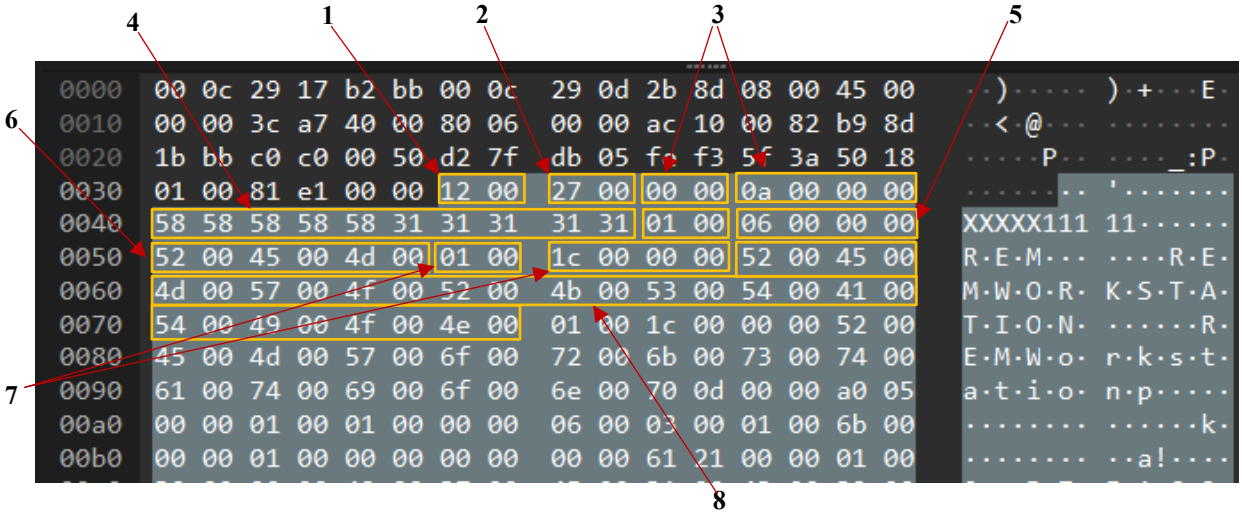
Finally, let's have a look at the data we have got so far:

- **The infected system**: 172.16.0.130
- **The command-and-control server**: 185.141.27.187
- **Malware used**: LokiBot
- **Malware detection**: User-Agent, HTTP Method (POST)
- **Malware activities**: Application data exfiltration and keylogging

Having basic information about the malware, let's dive deep into finding more information about the exfiltrated data by understanding its patterns in the next section.

CHAPTER 5 - FINDING NETWORK PATTERNS

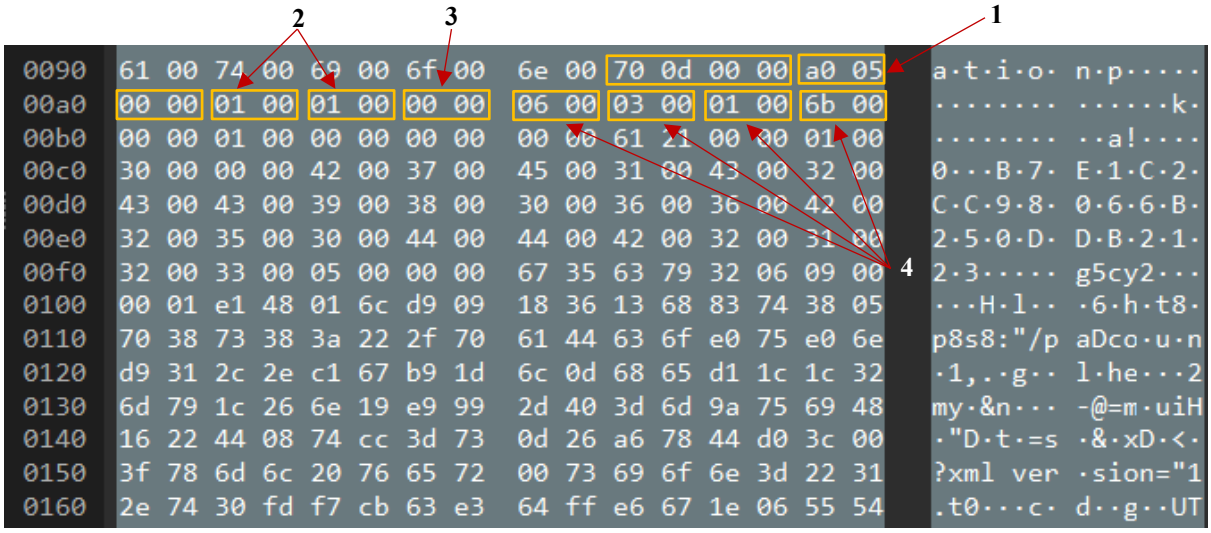
We know that the malware is stealing some application data, but we don't know which application it is and what data was stolen. To find this out by viewing the HTTP payload in the packet bytes (lowest pane) pane of the standard Wireshark display as follows:



1. LokiBot Version – 1.8
2. Exfiltrate Data: 27
3. Wide:0 Length:10
4. Binary ID: XXXXX11111
5. Wide:1 Length:6
6. Username: REM
7. Wide:1 Length: 1c = 28
8. Computer Name: REMWORKSTATION

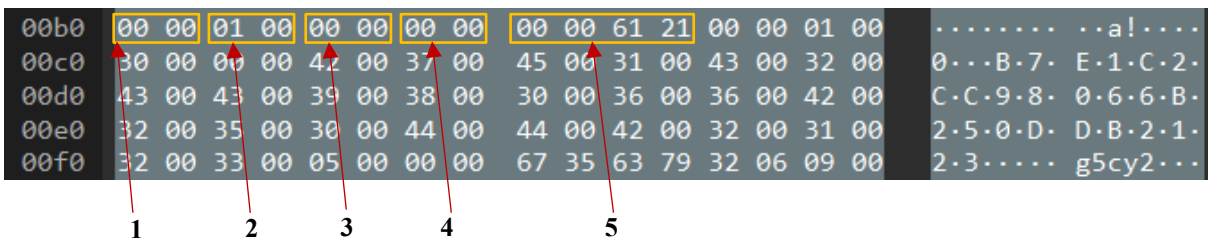
We can see from the preceding screenshot that the payload started with LokiBot version 18 in Decimal (12 in Hexadecimal), and we need to divide that by 10 to get the exact version. Also, we had 27 as the identifier for data exfiltration on application credentials. Next, the first word denotes a zero width, denoting that the payload value will be unpacked as a normal string. Next, we have a word value that denotes a length of 0a, 10 in decimal. We can see that we have a length of 10 bytes denoting the binary ID, which is XXXXX11111. Again, we have the next width and length, which will denote the system username; we can see we have a width of one and a length of six. Since we have a width of one, we will unpack this data as hex. Therefore, at two bytes each, we have the REM username. Next, we have the system name, and again

width is 1 and the length is 1c, denoting 28. The next 28 bytes indicate that the infected system name is REM WORKSTATION. Following the same notation for the values, the next value shows the domain, which is, again REM WORKSTATION. Below we can look at the next hex section as follows:



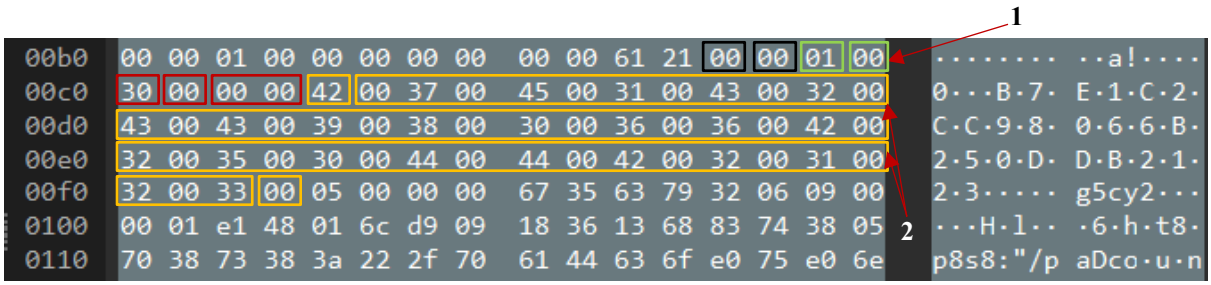
1. Screen Width: 07d0 (3440), Screen Height: 05a0 (1440)
2. IsLocalAdmin: 1(Yes), IsBuild In Admin (Yes)
3. is64Bit: No (0)
4. Major Version: 6, Minor Version: 3, Product Type: 1, OS_Bug Patch: 6b (107)

We have the next four bytes as the **Screen Width** and the following four as the **Screen Height**. We have a check on local admin and built-in admin, and the preceding screenshot shows that, in the next four bytes, both are showing a one, indicating a yes. The next two bytes are set to one if the OS is 64-bit, which is not the case, so it's set to zero. The next eight bytes define the OS major and OS minor products and the os_bug patch variables, which are 6,3,1,107 respectively. This means that we can denote the OS as 6.3.1.107, which is Windows 8. Additionally, the values stored here are in the little-endian format which means the last significant byte is the first. In the next section, we have the following:



1. Reported: 0
2. Compressed: 1
3. Encoded: 0
4. Encoding: 0
5. Original Stolen Data Length: 8545 (Hex:2161)

We can see the next two bytes as the value denoting the first-time connection as a zero. This means that the victim has connected for the first time. Next, two bytes denote that the data stolen is compressed, while the following two bytes define whether the stolen data is encoded or not, and following these two bytes are another two bytes defining the encoding type. The next four bytes denote the original stolen data's length, which is 8,545 bytes. A separator is in between, and we again have the width and length for the string:



1. Width:1 Length: 30 (48)
2. MUTEX

As shown in the preceding screenshot, we have a 48-byte-long mutex value used by the LokiBot. Next, LokiBot uses this mutex as follows:

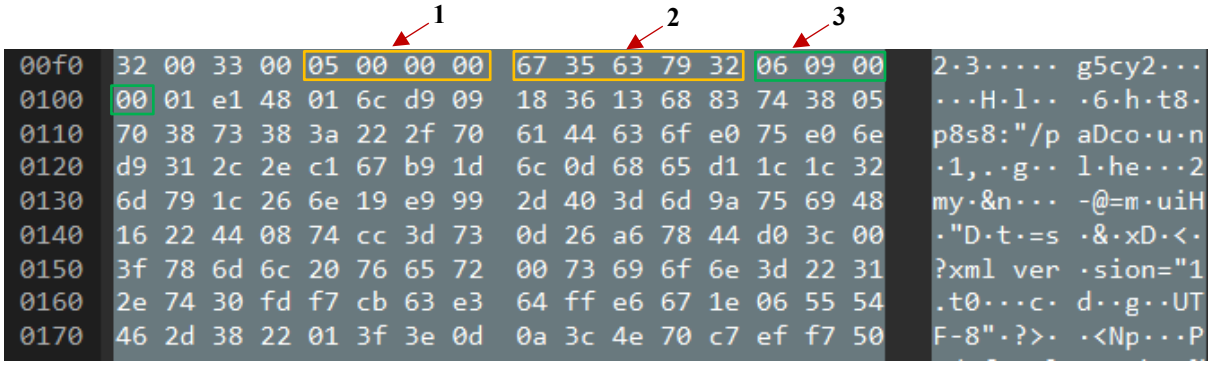
- Mutex: B7E1C2CC98066B250DDB2123

Based on this value, the LokiBot's files will be located in the following locations:

- Hash Database: "%APPDATA%\\C98066\\6B250D.hdb"
- Keylogger Database: "%APPDATA%\\C98066\\6B250D.kdb"
- Lock File: "%APPDATA%\\C98066\\6B250D.lck"
- Malware Exe: "%APPDATA%\\C98066\\6B250D.exe"

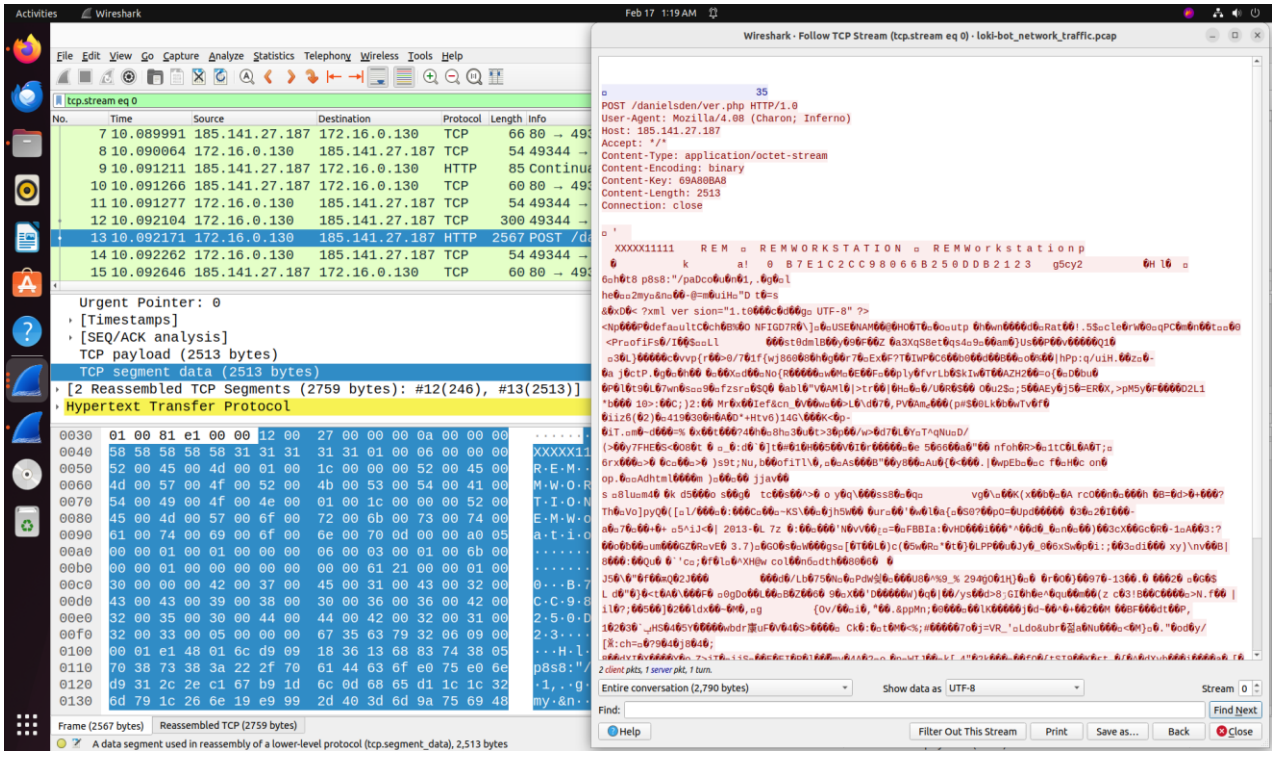
If we observe closely, we can see that the directory name starts from the 8th character to the 13th character of the Mutex while the file name starts from the 13th character to the 18th character.

Well! That was too much information traveling on the network. Let's see what's next:



1. Key Length: 5
2. Key
3. Compressed Data: 2310

We have the key length, the key itself, and the length of compressed data. We now know that the length of the compressed data is 2,310 bytes, which looks like this:



We can see some of the values as XML and HTML. But, we still need to decompress this data. On researching the malware executable file (Run strings command on the executable), we will discover that one of the strings in the binary executable contains LZSS, which is a popular data-compression encoding scheme. Using the library, we can copy the bytes from Wireshark

capture and feed it as input to the decompress function defined in the library. Let's decompress the data as follows:

```
□□□ <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<FileZilla3>
    <Settings>
        <Setting name="Use Pasv mode">1</Setting>
        <Setting name="Limit local ports">0</Setting>
        <Setting name="Limit ports low">6000</Setting>
        <Setting name="Limit ports high">7000</Setting>
        <Setting name="External IP mode">0</Setting>
        <Setting name="External IP"></Setting>
        <Setting name="External address resolver">http://ip.filezilla-project
 .org/ip.php</Setting>
        <Setting name="Last resolved IP"></Setting>
        <Setting name="No external ip on local conn">1</Setting>
        <Setting name="Pasv reply fallback mode">0</Setting>
        <Setting name="Timeout">20</Setting>
        <Setting name="Logging Debug Level">0</Setting>
        <Setting name="Logging Raw Listing">0</Setting>
        <Setting name="fzsftp executable"></Setting>
        <Setting name="Allow transfermode fallback">1</Setting>
```

The stolen data is from FileZilla, and it looks like a config file. On repeating the analysis for other packets, such as one with the value 2B (keylogger) type, we will have similar data, and on decompression, it will look similar to the following:

```
n

Window: Search Pane

otepad

Window: new  1 - Notepad++

i

Window: *new  1 - Notepad++

thdshfhasdlf jas jdflahslfdh ashflhsklf asjf lahshl ashflahsflhhfl ashasdl
 fhlshdf hasklfhls hfahflasf

 s

fas fashfdl ahshglhas lkjaslkhf lahsghalsjlasdflhalshf hasglha sldfhlhaslhg as
```

We have successfully gathered the following **Indicators of Compromise** (**IOC**) details by working on the preceding sample:

- **The infected system:** 172.16.0.130
- **The infected user**: REM
- **The infected system hostname**: REM WORKSTATION
- **Domain infected**: REMWorkstation
- **OS architecture**: 32 Bit
- **Screen resolution**: 3440 x 1440
- **Windows OS NT version**: 6.3.1 (Windows 8)
- **The command and control server**: 185.141.27.187
- **Malware used**: LokiBot
- **Malware detection**: User-Agent, HTTP method (POST)
- **Malware activities**: Application Data Exfiltration on FileZilla, Keylogging
- **Malware version**: 1.8
- **Malware compression**: LZSS
- **Malware encoding**: None
- **Malware files names**: %APPDATA%\\C98066\\6B250D.*

CHAPTER 6 - PYLOCKY RANSOMWARE DECRYPTION USING PCAP DATA

Cisco has launched the PyLocky decryptor (https://github.com/Cisco-Talos/pylocky_decryptor), which searches through the PCAP to decrypt files on the system.PyLocky sends a single POST request to the control server containing the following parameters:

```
PCNAME=NAME&IV=KXyiJnifKQQ%3D%0A&GC=VGA+3D&PASSWORD=CVxAfel9ojCYJ9So&CPU=Intel%28R%29+Xeon%28R%29+CPU+E5-
1660+v4+%40+3.20GHz&LANG=en_US&INSERT=1&UID=XXXXXXXXXXXXXXXX&RAM=4&OSV=10.0.16299+16299&MAC=00%3A00%3A00%3A00%3A45%3A6B& OS=Microsoft+Windows+10+Pro
```

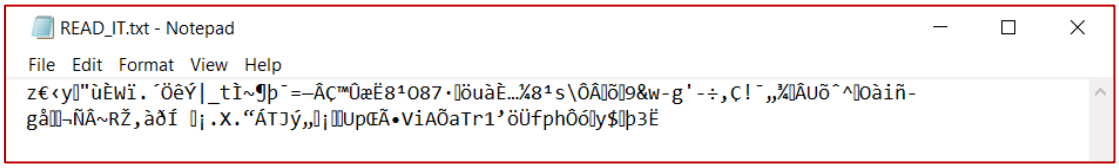
We can see that we have iv, the initialization vector, and the password as the parameters. In case the network was being logged at the time of the system infection, we could use this information to decrypt the files with ease. Let's look at PyLocky's code for decryption, as follows:

```
if "lockedfile" in fname:
    global counter
    fname_w_e = os.path.splitext(fname)[0]
    if debug:
        print("Opening fname: "+fname)
    fd = open(fname, "rb")
    data = fd.read()
    fd.close()
    if debug:
        print("Closed fname: "+fname)
    ddata = des3_decrypt(password, iv, data, debug)
    rdata = ddata.decode("base64")
    if debug:
        print("Opening fname_w_e: "+fname_w_e)
    fd = open(fname_w_e, "wb")
    fd.write(rdata)
    fd.close()
    if debug:
        print("Closed fname_w_e: "+fname_w_e)
    if debug:
        print("File processed correctly: "+fname)
    if remove:
        os.remove(fname)
        if debug:
            print("File removed correctly: "+fname)
    counter += 1
```

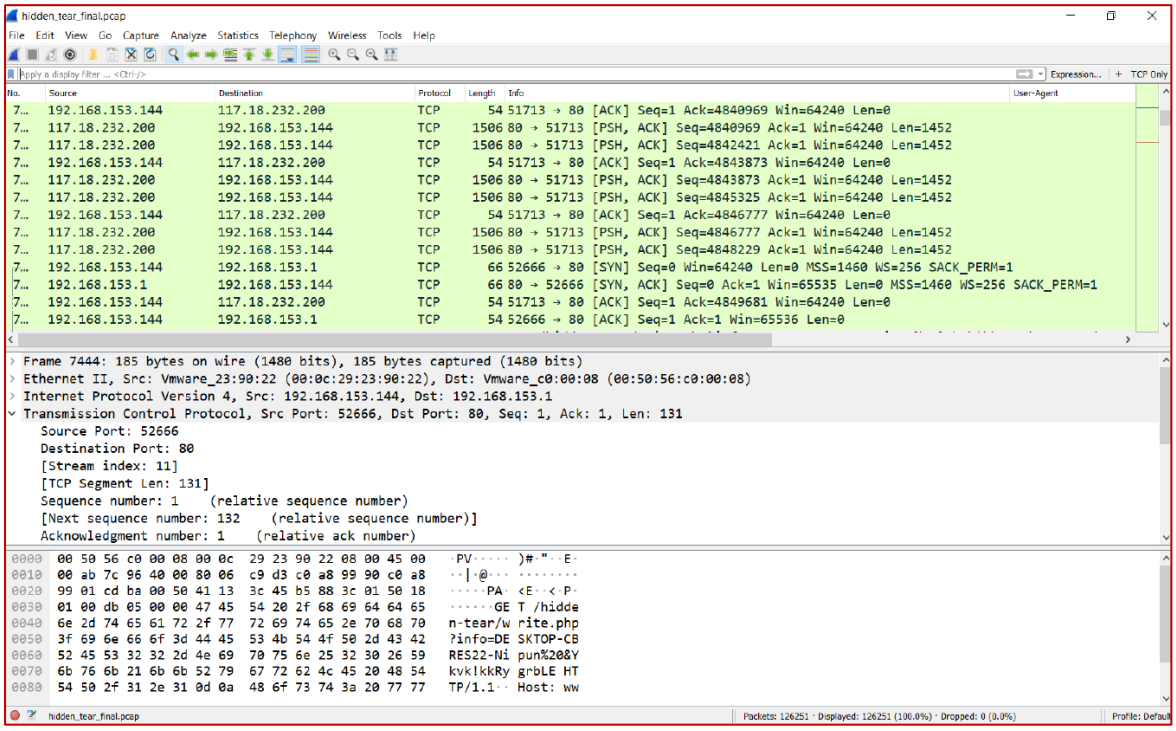
We can see that PyLocky decrypter makes use of IV and passwords to decrypt the files encrypted with the PyLocky ransomware, and generally, this way works for several ransomware types out there. PyLocky makes use of DES3 to encrypt the files that can be decrypted back.

CHAPTER 7 - DECRYPTING HIDDEN TEAR RANSOMWARE

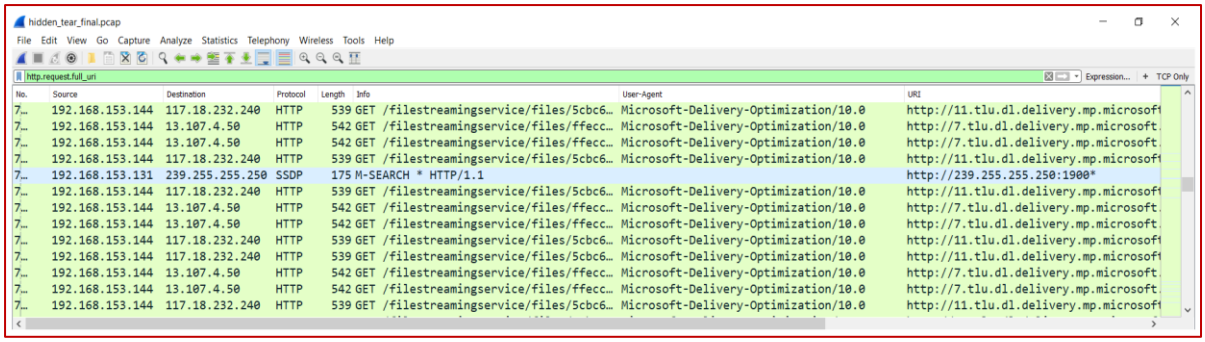
Let's see another example with hidden tear ransomware. Consider a scenario where hidden tear ransomware has locked files on a Windows 10 system, and the situation is pretty bad. It looks like the files are encrypted.



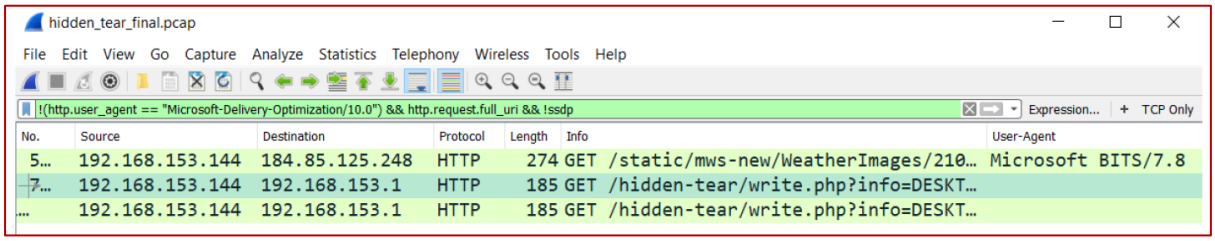
Yes—the contents of the file are encrypted. Luckily for us, we have a PCAP of the fully captured data with us. Let's start our analysis:



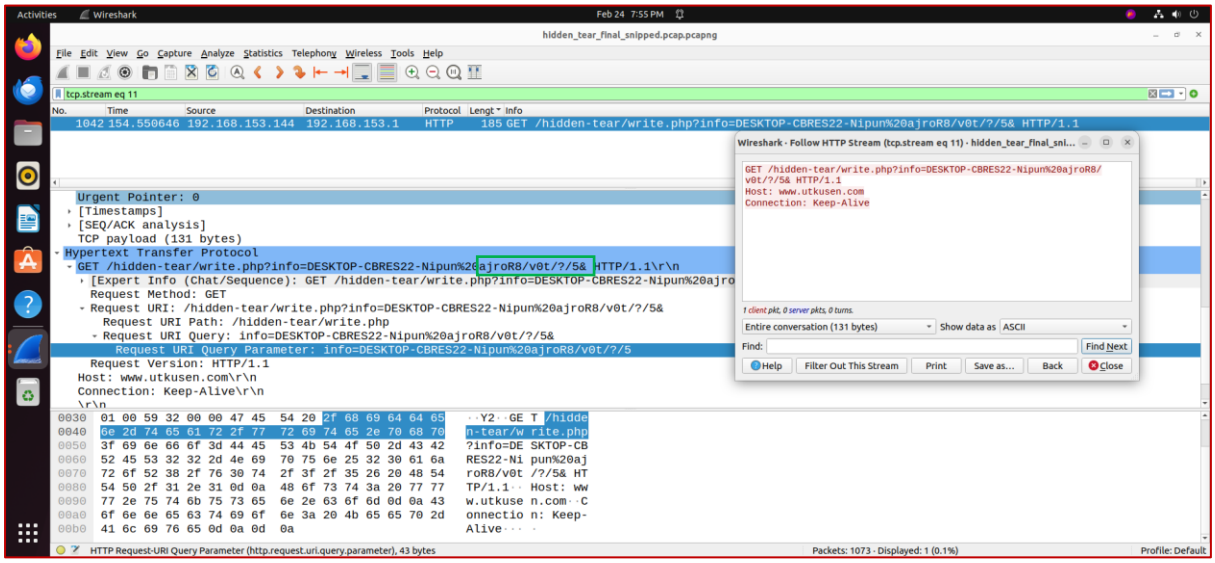
We can see we have a fairly large PCAP file, containing a good amount of HTTP data. Since we know that malware have issues with user-agents, display the full user-agent and URI data in Wireshark as we did in the earlier examples:



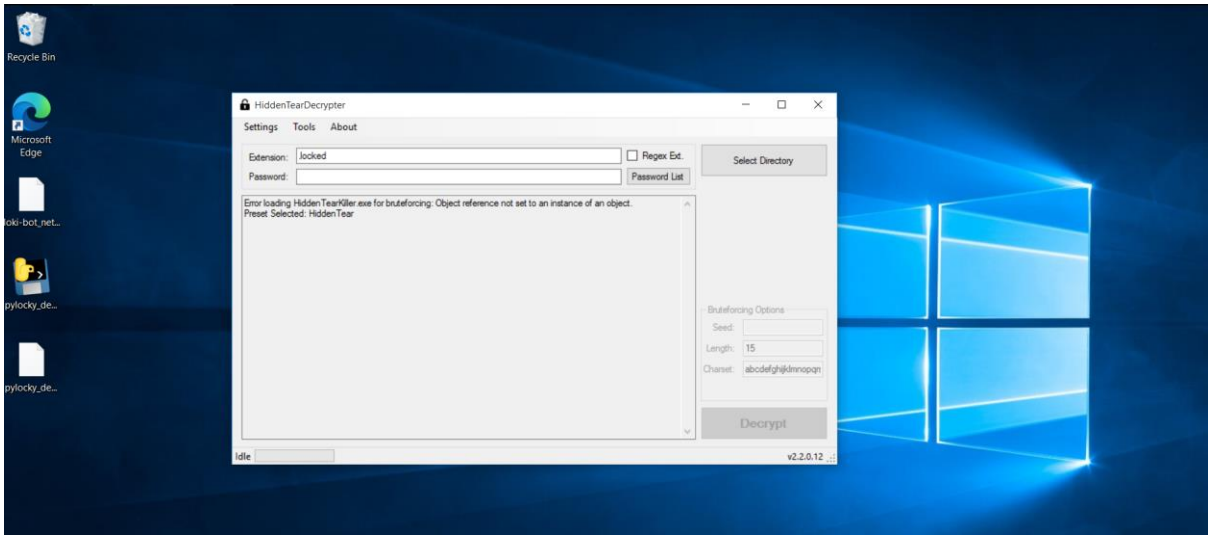
We can see that most of the data *is* being fetched from Microsoft domains, and probably looks like it is used by Windows update. Let's unselect this user-agent and see what we are left with:



We can see that by using the `!(http.user_agent=="Microsoft-Delivery-Optimization/10.0")&&http.request.full_uri&&ssdp` filter, we are left with only a few packets. Let's investigate the packets as follows:



We can see that a GET request containing our machine name and some string is sent to a domain. Could this be the password? We'll have to check. Let's download the decrypter from `https://github.com/goliate/hidden-tear`:

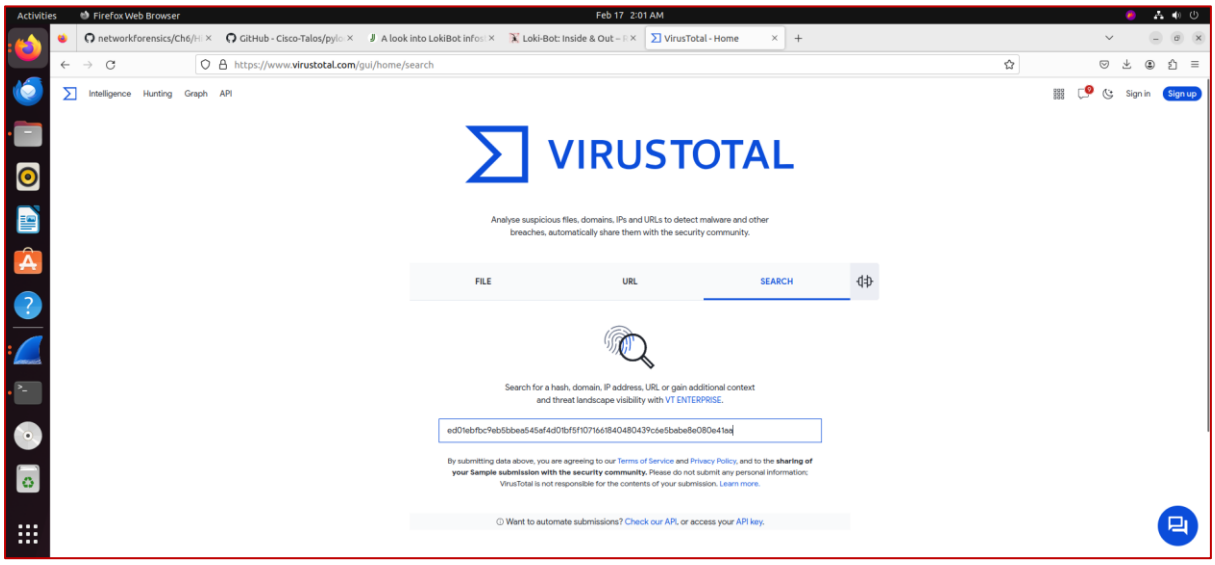


CHAPTER 8 - BEHAVIOUR PATTERNS AND ANALYSIS

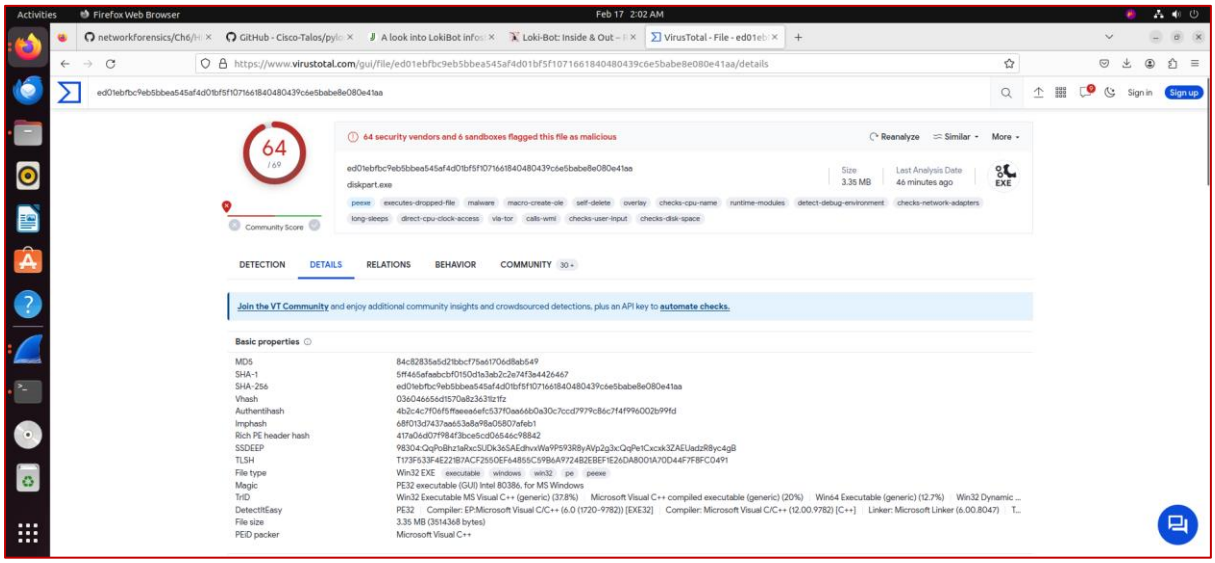
For a forensic network investigator, it is important to find the behaviour and network patterns of malware. Consider that you have received a few binaries (executable) and their hashes (signature) from the incident response team that are likely to be carrying malware. However, the analysis of PE/COFF executables is generally done by malware analysts and reverse engineers. What can you do with the PE executable? You don't have to study reverse engineering and malware analysis overnight to analyze the sample. Consider that you have received the file hash as

`ed01ebfbc9eb5bbea545af4d01bf5f10716618404804 39c6e5babe8e080e41aa.`

You can use websites such as `https://www.virustotal.com/gui/home/upload` and `https://www.hybrid-analysis.com/` to analyze your sample without analyzing it on your system. The following screenshot shows the VirusTotal website:

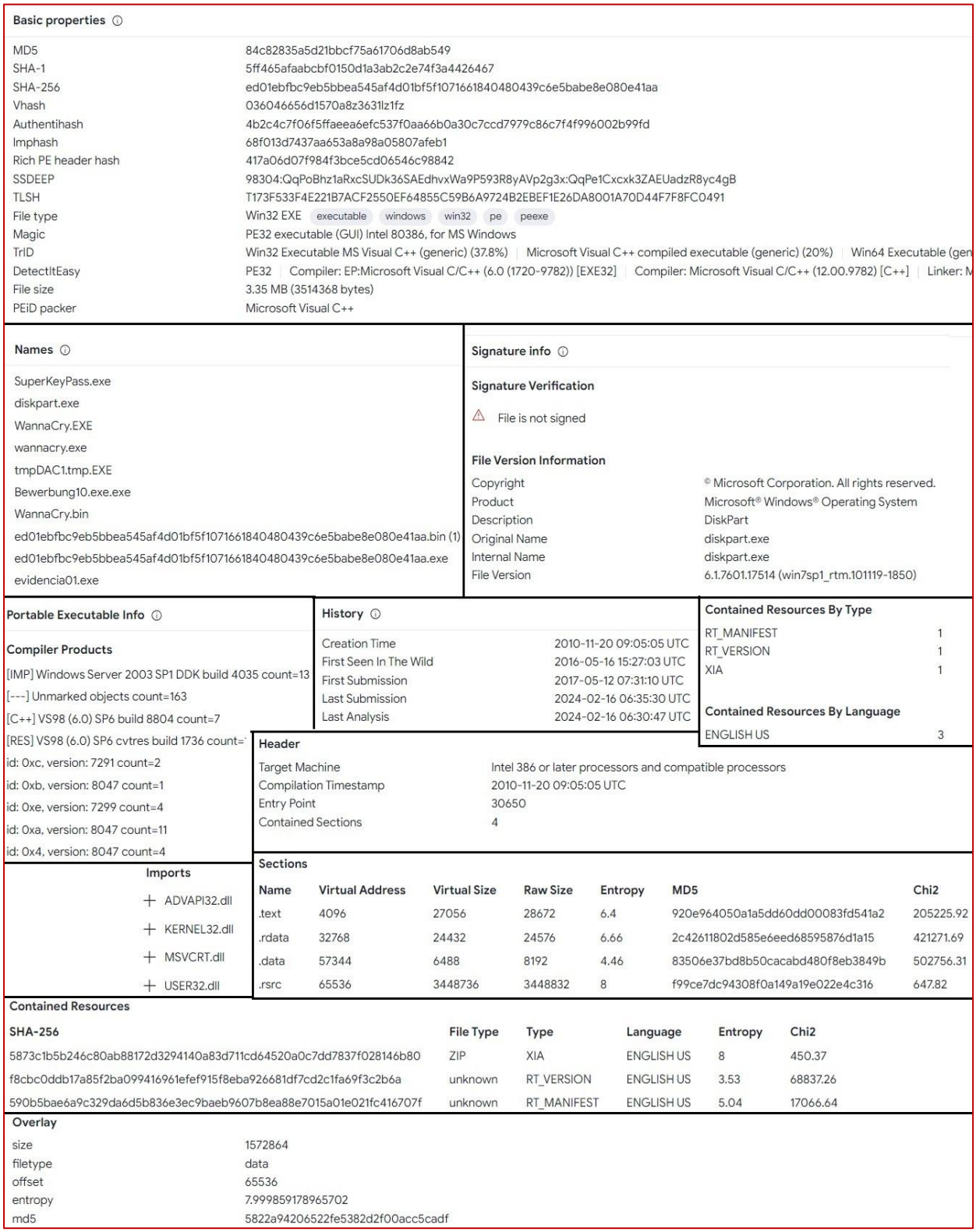


Let's search the hash of the file at VirusTotal. The results should show up if the file has previously been analyzed:



Here, 64/70 antivirus engines detect the file as malicious, and it may be a WannaCry ransomware sample.

Let's see the details from the Details tab as follows:



Basic properties

MD5	84c82835a5d21bbcf75a61706d8ab549
SHA-1	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
SHA-256	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
Vhash	036046656d1570a8z3631lz1fz
Authentihash	4b2c4c7f06f5ffaeea6efc537f0aa66b0a30c7ccd7979c86c7f4f996002b99fd
Imphash	68f013d7437aa653a8a98a05807afeb1
Rich PE header hash	417a06d07f984f3bce5cd06546c98842
SSDEEP	98304:QqPoBhz1aRxcSUDk36SAEdhvxWa9P593R8yAVp2g3x:QqPe1Cxcxk3ZAEUadzR8yc4gB
TLSH	T173F533F4E221B7ACF2550EF64855C59B6A9724B2EBEF1E26DA8001A70D44F7F8FC0491
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TrID	Win32 Executable MS Visual C++ (generic) (37.8%) | Microsoft Visual C++ compiled executable (generic) (20%) | Win64 Executable (gen
DetectItEasy	PE32 | Compiler: EP:Microsoft Visual C/C++ (6.0 (1720-9782)) [EXE32] | Compiler: Microsoft Visual C/C++ (12.00.9782) [C++] | Linker: M
File size	3.35 MB (3514368 bytes)
PEiD packer	Microsoft Visual C++

Names

SuperKeyPass.exe
diskpart.exe
WannaCry.EXE
wannacry.exe
tmpDAC1.tmp.EXE
Bewerbung10.exe.exe
WannaCry.bin
ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.bin (1)
ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe
evidencia01.exe

Signature info

Signature Verification

⚠ File is not signed

File Version Information

Copyright	© Microsoft Corporation. All rights reserved.
Product	Microsoft® Windows® Operating System
Description	DiskPart
Original Name	diskpart.exe
Internal Name	diskpart.exe
File Version	6.1.7601.17514 (win7sp1_rtm.101119-1850)

Portable Executable Info

Compiler Products

[IMP] Windows Server 2003 SP1 DDK build 4035 count=13
[---] Unmarked objects count=163
[C++] VS98 (6.0) SP6 build 8804 count=7
[RES] VS98 (6.0) SP6 cvtres build 1736 count=
id: 0xc, version: 7291 count=2
id: 0xb, version: 8047 count=1
id: 0xe, version: 7299 count=4
id: 0xa, version: 8047 count=11
id: 0x4, version: 8047 count=4

History

Creation Time	2010-11-20 09:05:05 UTC
First Seen In The Wild	2016-05-16 15:27:03 UTC
First Submission	2017-05-12 07:31:10 UTC
Last Submission	2024-02-16 06:35:30 UTC
Last Analysis	2024-02-16 06:30:47 UTC

Contained Resources By Type

RT_MANIFEST	1
RT_VERSION	1
XIA	1

Contained Resources By Language

ENGLISH US	3

Header

Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2010-11-20 09:05:05 UTC
Entry Point	30650
Contained Sections	4

Imports

+ ADVAPI32.dll
+ KERNEL32.dll
+ MSVCRT.dll
+ USER32.dll

Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	27056	28672	6.4	920e964050a1a5dd60dd00083fd541a2	205225.92
.rdata	32768	24432	24576	6.66	2c42611802d585e6eed68595876d1a15	421271.69
.data	57344	6488	8192	4.46	83506e37bd8b50cacabd480f8eb3849b	502756.31
.rsrc	65536	3448736	3448832	8	f99ce7dc94308f0a149a19e022e4c316	647.82

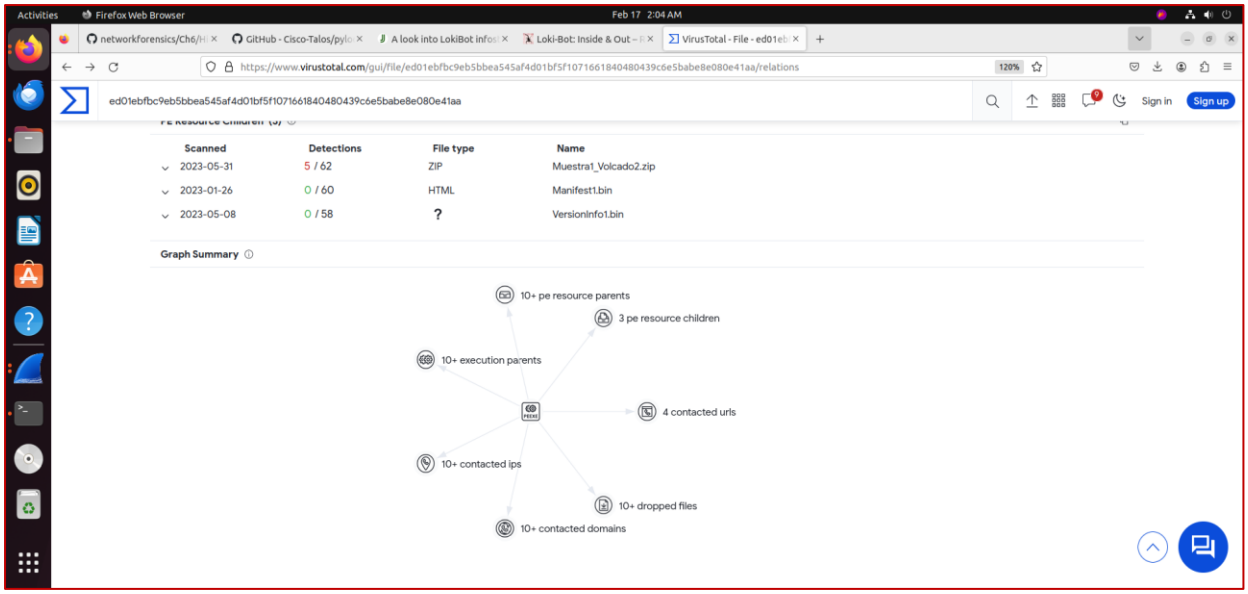
Contained Resources

SHA-256	File Type	Type	Language	Entropy	Chi2
5873c1b5b246c80ab88172d3294140a83d711cd64520a0c7dd7837f028146b80	ZIP	XIA	ENGLISH US	8	450.37
f8cbc0ddb17a85f2ba099416961efef915f8eba926681df7cd2c1fa69f3c2b6a	unknown	RT_VERSION	ENGLISH US	3.53	68837.26
590b5bae6a9c329da6d5b836e3ec9baeb9607b8ea88e7015a01e021fc416707f	unknown	RT_MANIFEST	ENGLISH US	5.04	17066.64

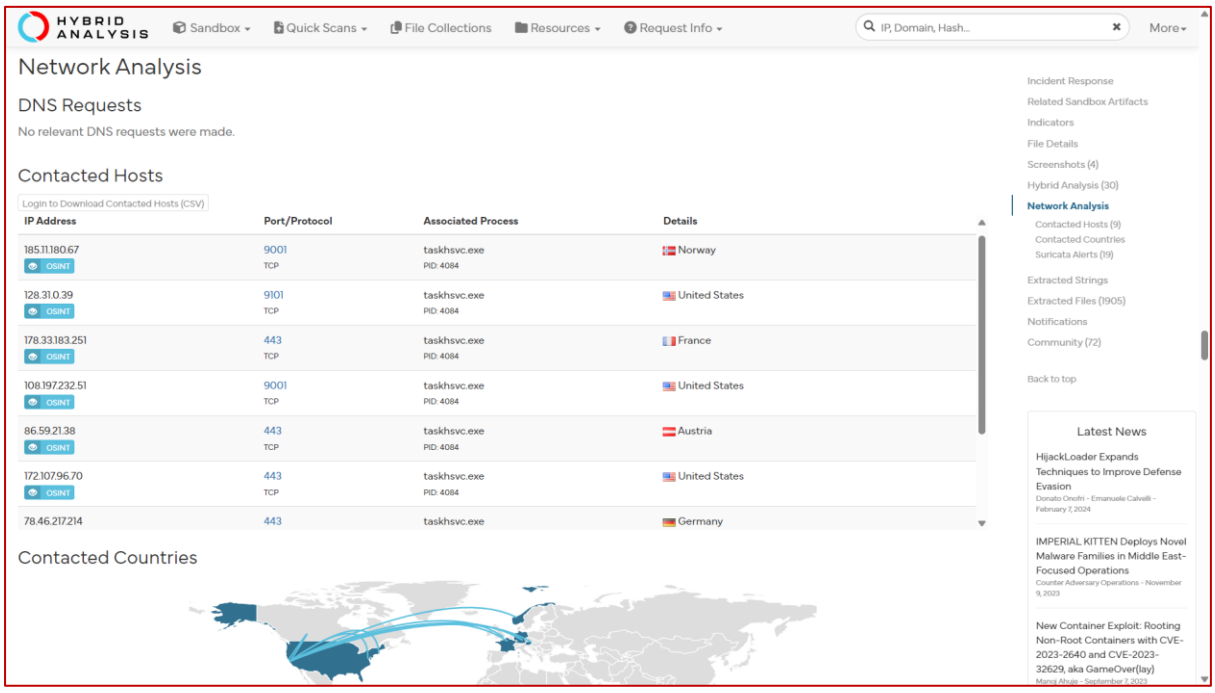
Overlay

size	1572864
filetype	data
offset	65536
entropy	7.999859178965702
md5	5822a94206522fe5382d2f00acc5cadf

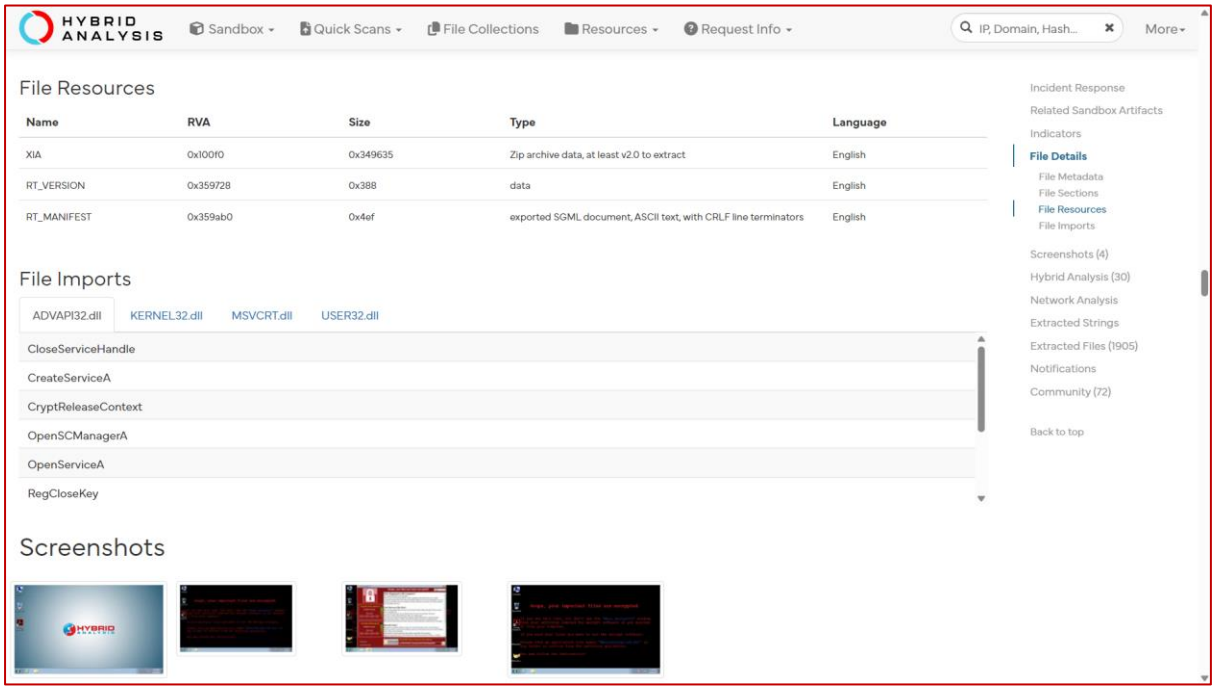
Plenty of detail can be seen on the **DETAILS** tab, especially the common names of the files causing this infection. We can also see that the file has been analyzed previously with a different name. Additionally, we have the following details:



We can see that there are five IP addresses contacted by the WannaCry executable. We can filter the network based on these details to check infections in the network and pinpoint the infected source. Let's also upload/search the sample on the Hybrid-Analysis website (`https://www.hybrid-analysis.com/`) as well:



On searching the sample on Hybrid Analysis, we can see that we have the list of connected IP addresses, and a list of ports as well. This information will help us to narrow the outbound connections down from the infected system. We can see that Hybrid-Analysis has gone ahead and executed the associated sample file of the hash we provided for analysis in a secured environment:



We can see the state of the system before and after the execution of the malware, where we can see that the system got infected with WannaCry ransomware.

CHAPTER 9 - CONCLUSION

We have executed an in-depth exploration of malware, with a focus on dissecting the LokiBot on the packet level and decrypting ransomware variants such as PyLocky and Hidden Tear Employed automated techniques utilizing platforms like VirusTotal, Hybrid-Analysis to enhance Investigative processes.

CHAPTER 10 - REFERENCES

- https://community.juniper.net/blogs/elevate-member/2020/12/22/a-look-into-lokibot-infostealer
- https://r3mrum.wordpress.com/2017/07/13/loki-bot-inside-out/
- https://r3mrum.files.wordpress.com/2017/07/loki_bot-grem_gold.pdf
- https://sensorstechforum.com/use-wireshark-decrypt-ransomware-files/
- https://paper.bobylive.com/Security/Hands-On-Network-Forensics.pdf
- https://lynnseygrahamnovak.medium.com/network-forensic-investigation-identifying-malware-in-network-traffic-9a6bc32116dc
- https://www.dionach.com/behavioural-analysis-of-malware-via-network-forensics/