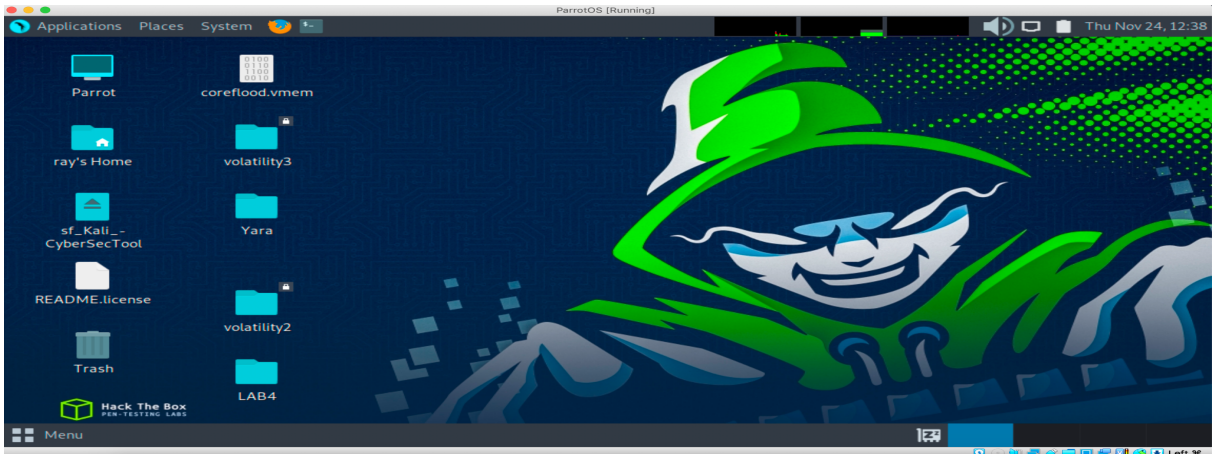
Introduction

In this lab, we are going to perform malware analysis with various detection techniques and find the intended behaviours in each one of the malware programs. Moreover, knowing how to write Yara rules and detect malware through Yara are vital to this lab's objective. And, this lab covers DNS, Hash, Windows API, Persistent maintenance and other malicious indicators.

As to Virtual Machines and Techniques, we use the ParrotOS HTB version, and SSH Port forwarding connects to the machine.




```
Ray@MacBook-Pro-2:~[$]sudo ssh -p 2221 ray@127.0.0.1
Password:
ray@127.0.0.1's password:
Linux parrot 6.0.0-2parrot1-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.0.2-
1parrot1 (2022-10-18) x86_64
 ____                                 _       ____
|  _ \ __ _ _ __ _ __ ___ | |_   / ___|   ___   ___
| |_) / _` | '__| '__/ _ \| __| \___ \ / _ \ / __|
|  __/ (_| | |  | | | (_) | |_   ___) |  __/ (__
|_|   \__,_|_|  |_|  \___/ \__| |____/ \___|\___|



The programs included with the Parrot GNU/Linux are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Parrot GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Nov 21 20:14:26 2022 from 10.0.2.2
┌──[ray@parrot]─[~]
└─── $_
```

Objective

Objective Malware Files:

1. njRAT-v0.6.4.zip
2. infected.7z
3. Group File **PWN Group**:

PWN	e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4.zip

4. e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4.zip

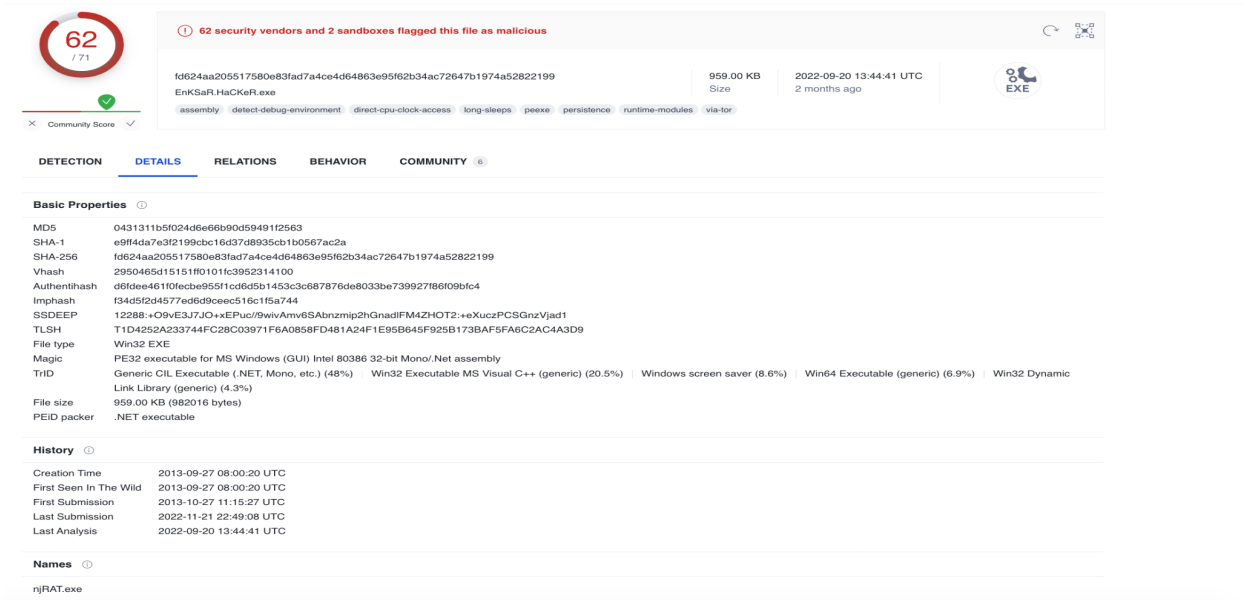
Extract Password: infected

The extracted files can be observed in the below screenshot.

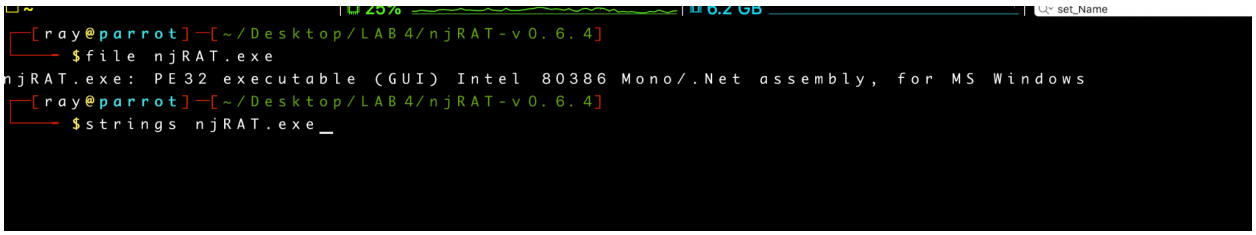
Task 1: Create a Yara rule for njRAT malware

Find the clue of Indicators of Compromise (IOC) through Yara rules.

The hash value of njRAT.exe is ***0431311b5f024d6e66b90d59491f2563,*** and the Virustotal indicates that the hash value has been recognized as a malicious file in 62 out of 71 vendors.

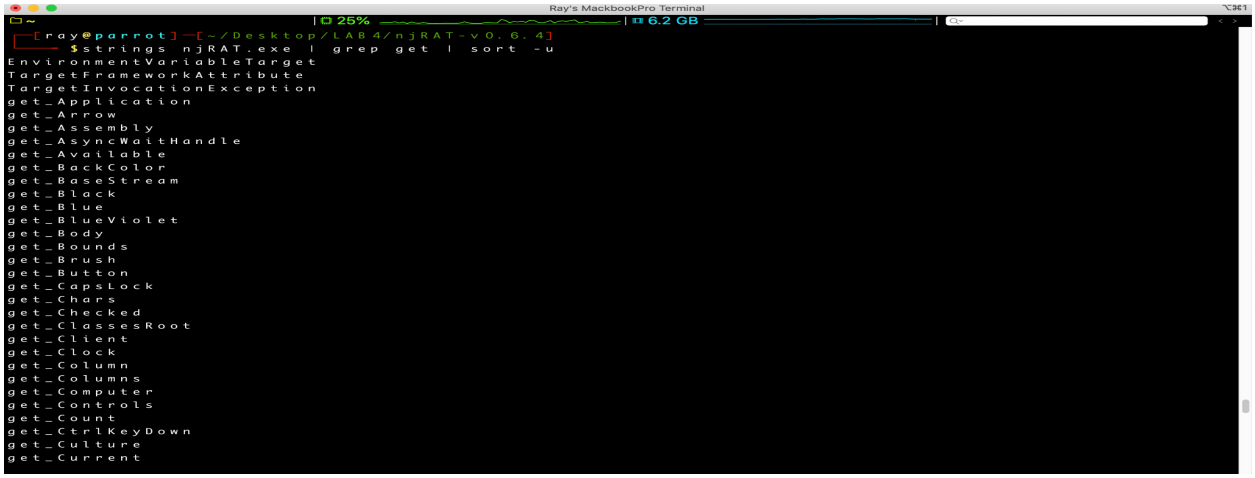


Firstly, we need to know what system of the executable it is via file command.

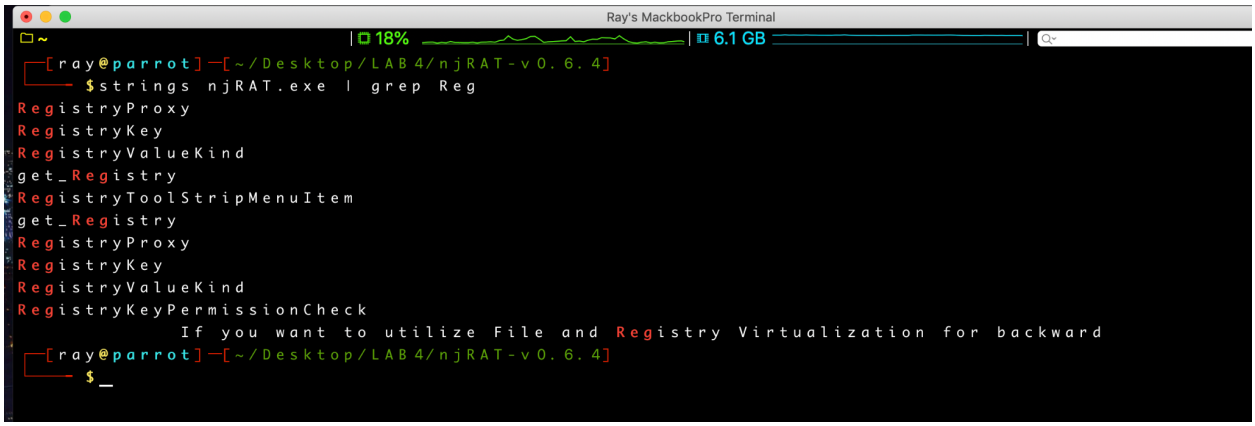


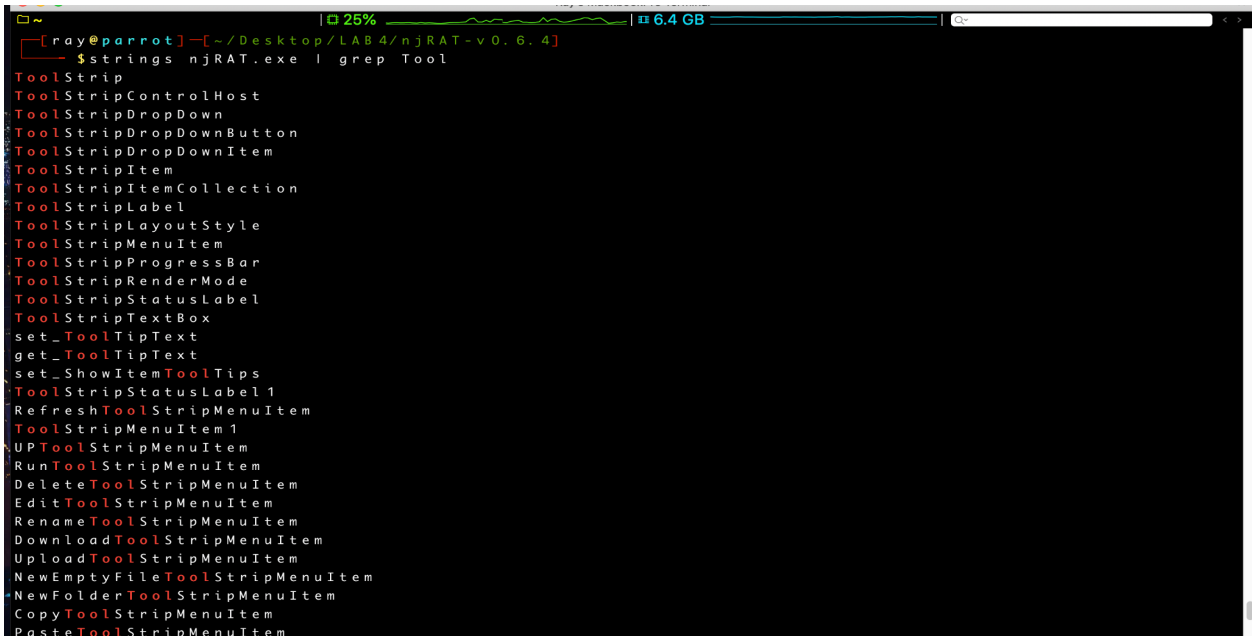
Utilize strings to extract information from the njRAT executable file, and this will show what Windows commands are used and malicious behaviour.

As we can see in the below screenshots, there are many Windows APIs the njRAT executable used [1] [2] [3] [4].



Windows API Set, Get, Registry File and Tool functions indicate that the attacker might run this executable to connect back to C2 Server.





As we have the Windows API information, we can write the Yara rule for detecting the njRAT file. Yara rules have a ***meta*** header which is the owner of this file, a ***strings*** section that has the string variable for matching purposes in a malware file, and the last ***condition*** part is to trigger to detect based on the condition.

```
 GNU nano 5.4                          RAT_detection.yara                    1
rule RAT_Detection{
    meta:
        author = "Ray"
        descripion = "njRAT - Remote Access Trojan"
        comment = "Find Indictor of Compromise (IOC) for Windows API"

    strings:
        $stirng1 =  /ToolStrip/
        $stirng2 =  /ToolStripControlHost/
        $stirng3 =  /ToolStripDropDown/
        $stirng4 =  /ToolStripDropDownButton/
        $stirng5 =  /ToolStripDropDownItem/
        $stirng6 =  /ToolStripItem/
        $stirng7 =  /ToolStripItemCollection/
        $stirng8 =  /ToolStripLabel/
        $stirng9 =  /ToolStripLayoutStyle/
        $stirng10 = /ToolStripMenuItem/
        $stirng11 = /set_SendBufferSize/
        $stirng12 = /set_ReceiveBufferSize/
        $stirng13 = /RegistryProxy/
        $stirng14 = /RegistryToolStripMenuItem/
        $stirng15 = /RegistryKey/
        $stirng16 = /RegistryValueKind/
        $stirng17 = /RegistryProxy/
        $stirng18 = /get_Registry/
        $stirng19 = /RegistryValueKind/
        $stirng20 = /RegistryKeyPermissionCheck/
    condition:
            10 of them
}
```

The successful detection result after executing the Yara rule is in terms of the below screenshot. This shows ***RAT_Detection.***

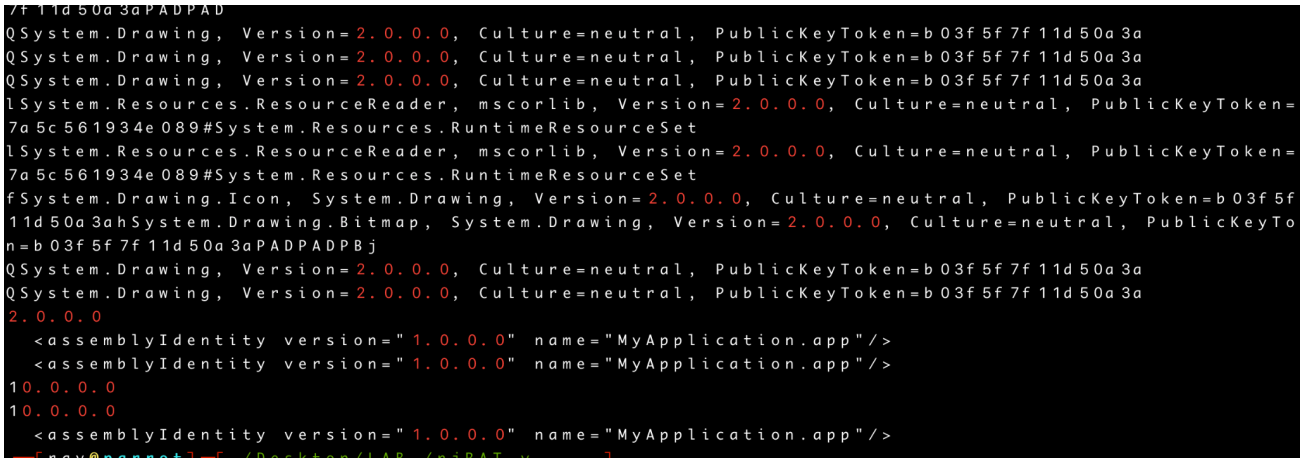
```
~                                  26%                    6.3 GB
┌──[ray@parrot]─[~/Desktop/LAB 4/njRAT-v0.6.4]
└─── $sudo nano RAT_detection.yara
┌──[ray@parrot]─[~/Desktop/LAB 4/njRAT-v0.6.4]
└─── $sudo yara RAT_detection.yara njRAT.exe
RAT_Detection njRAT.exe
┌──[ray@parrot]─[~/Desktop/LAB 4/njRAT-v0.6.4]
└─── $_
```

Find the C2 digital footprint:

We only find some numbers with ***grep "[0-255].[0-255].[0-255].[0-255]$"***

```
~                                  21%                    6.4 GB
┌──[ray@parrot]─[~/Desktop/LAB /njRAT-v . . ]
└─── $strings njRAT.exe  | grep [0-255].[0-255].[0-255].[0-255]$
2.0.0.0
10.0.0.0
10.0.0.0
┌──[ray@parrot]─[~/Desktop/LAB /njRAT-v . . ]
└─── $_
```

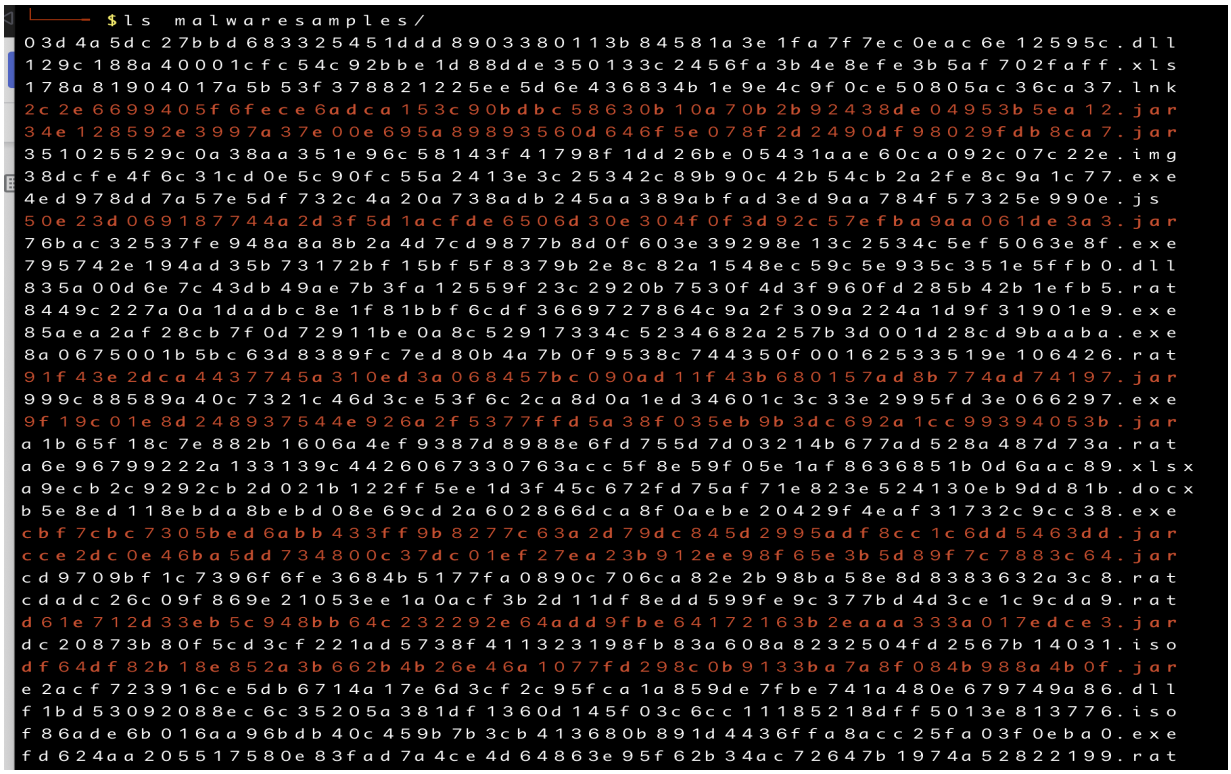
If we use ***grep "[0-255].[0-255].[0-255].[0-255]"*** without $ at the end, this shows the below result, which could be some infected machines' IP addresses or some windows service versions.



Task 2: Run Yara on malware samples

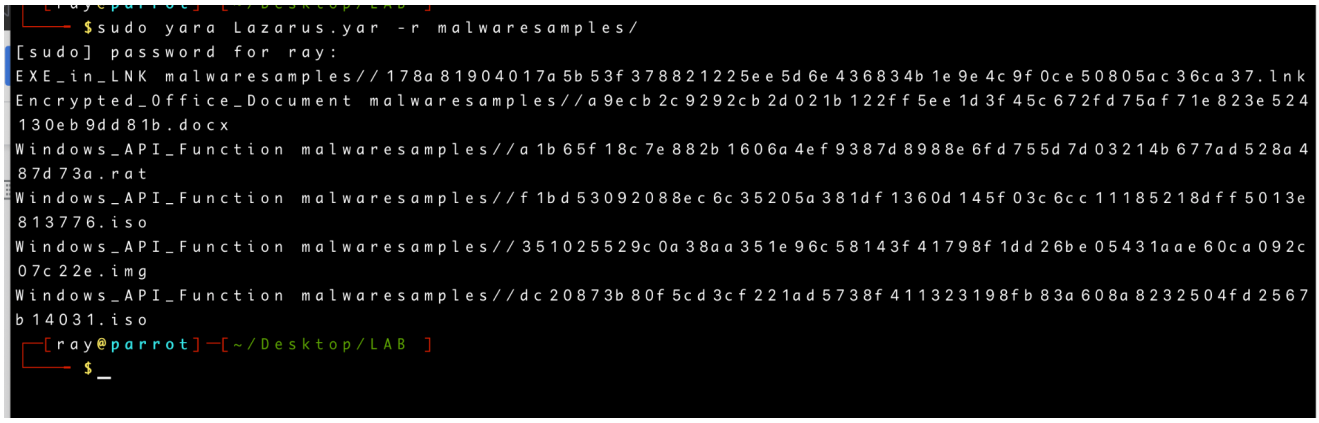
This Task 2 is to utilize the provided Yara rules to detect malware examples.

Malware Examples:



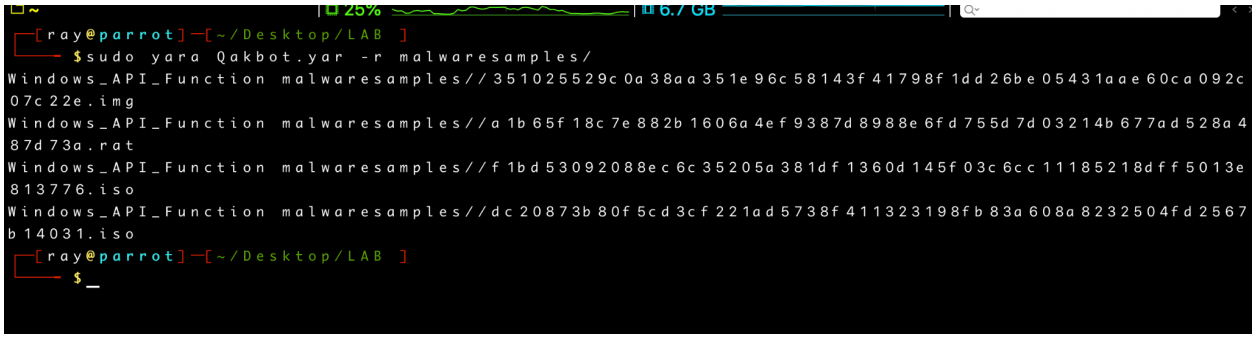
Lazarus.yar:

The result of using Lazarus.yar rule with "-r" recursively detects each file of malware example directories.



Qakbot.yar:

The result of using the Qakbot.yar rule with "-r" recursively detects each file of malware example directories.



RAT_Njrat.yar:

The result of using the RAT_Njrat.yar rule with "-r" recursively detects each file of malware example directories.

```
┌──[ray@parrot]─[~/Desktop/LAB ]
└─── $sudo yara RAT_Njrat.yar -r malwaresamples/
Njrat malwaresamples//a1b65f18c7e882b1606a4ef9387d8988e6fd755d7d03214b677ad528a487d73a.rat
Njrat malwaresamples//fd624aa205517580e83fad7a4ce4d64863e95f62b34ac72647b1974a52822199.rat
njrat1 malwaresamples//fd624aa205517580e83fad7a4ce4d64863e95f62b34ac72647b1974a52822199.rat
┌──[ray@parrot]─[~/Desktop/LAB ]
└─── $_
```

crime_wannacry.yar:

The result of using crime_wannacry.yar rule with "-r" recursively detects each file of malware example directories.

```
┌──[ray@parrot]─[~/Desktop/LAB ]
└─── $sudo yara crime_wannacry.yar -r malwaresamples/
WannaCry_Ransomware malwaresamples//795742e194ad35b73172bf15bf5f8379b2e8c82a1548ec59c5e935c351e5ffb0.dll
WannaCry_Ransomware_Gen malwaresamples//795742e194ad35b73172bf15bf5f8379b2e8c82a1548ec59c5e935c351e5ffb0.dll
WannaCry_Ransomware malwaresamples//8449c227a0a1dadbc8e1f81bbf6cdf3669727864c9a2f309a224a1d9f31901e9.exe
WannaCry_Ransomware_Gen malwaresamples//8449c227a0a1dadbc8e1f81bbf6cdf3669727864c9a2f309a224a1d9f31901e9.exe
WannaCry_Ransomware malwaresamples//85aea2af28cb7f0d72911be0a8c52917334c5234682a257b3d001d28cd9baaba.exe
WannaCry_Ransomware_Gen malwaresamples//85aea2af28cb7f0d72911be0a8c52917334c5234682a257b3d001d28cd9baaba.exe
WannaCry_Ransomware malwaresamples//76bac32537fe948a8a8b2a4d7cd9877b8d0f603e39298e13c2534c5ef5063e8f.exe
WannaCry_Ransomware malwaresamples//b5e8ed118ebda8bebd08e69cd2a602866dca8f0aebe20429f4eaf31732c9cc38.exe
WannaCry_Ransomware malwaresamples//999c88589a40c7321c46d3ce53f6c2ca8d0a1ed34601c3c33e2995fd3e066297.exe
WannaCry_Ransomware_Gen malwaresamples//999c88589a40c7321c46d3ce53f6c2ca8d0a1ed34601c3c33e2995fd3e066297.exe
WannaCry_Ransomware malwaresamples//03d4a5dc27bbd683325451ddd8903380113b84581a3e1fa7f7ec0eac6e12595c.dll
┌──[ray@parrot]─[~/Desktop/LAB ]
└─── $_
```

redline_stealer.yar:

The result of using the redline_stealer.yar rule with "-r" recursively detects each file of malware example directories.

```
┌──[ray@parrot]─[~/Desktop/LAB ]
└─── $sudo yara redline_stealer.yar -r malwaresamples/
MALWARE_Win_zgRAT malwaresamples//e2acf723916ce5db6714a17e6d3cf2c95fca1a859de7fbe741a480e679749a86.dll
MALWARE_Win_RedLine malwaresamples//38dcfe4f6c31cd0e5c90fc55a2413e3c25342c89b90c42b54cb2a2fe8c9a1c77.exe
MALWARE_Win_NjRAT malwaresamples//fd624aa205517580e83fad7a4ce4d64863e95f62b34ac72647b1974a52822199.rat
INDICATOR_EXE_Packed_Themida malwaresamples//f86ade6b016aa96bdb40c459b7b3cb413680b891d4436ffa8acc25fa03f0eba0.exe
┌──[ray@parrot]─[~/Desktop/LAB ]
└─── $_
```

general_rats_malwareconfig.yar:

The result of using general_rats_malwareconfig.yar rule with "-r" recursively detects each file of malware example directories.

```
┌──[ray@parrot]─[~/Desktop/LAB ]
└─────$sudo yara general_rats_malwareconfig.yar -r malwaresamples/
MAL_JRAT_Oct 18_1 malwaresamples//d61e712d33eb5c948bb64c232292e64add9fbe64172163b2eaaa333a017edc
e3.jar
RAT_njRat malwaresamples//fd624aa205517580e83fad7a4ce4d64863e95f62b34ac72647b1974a52822199.rat
```

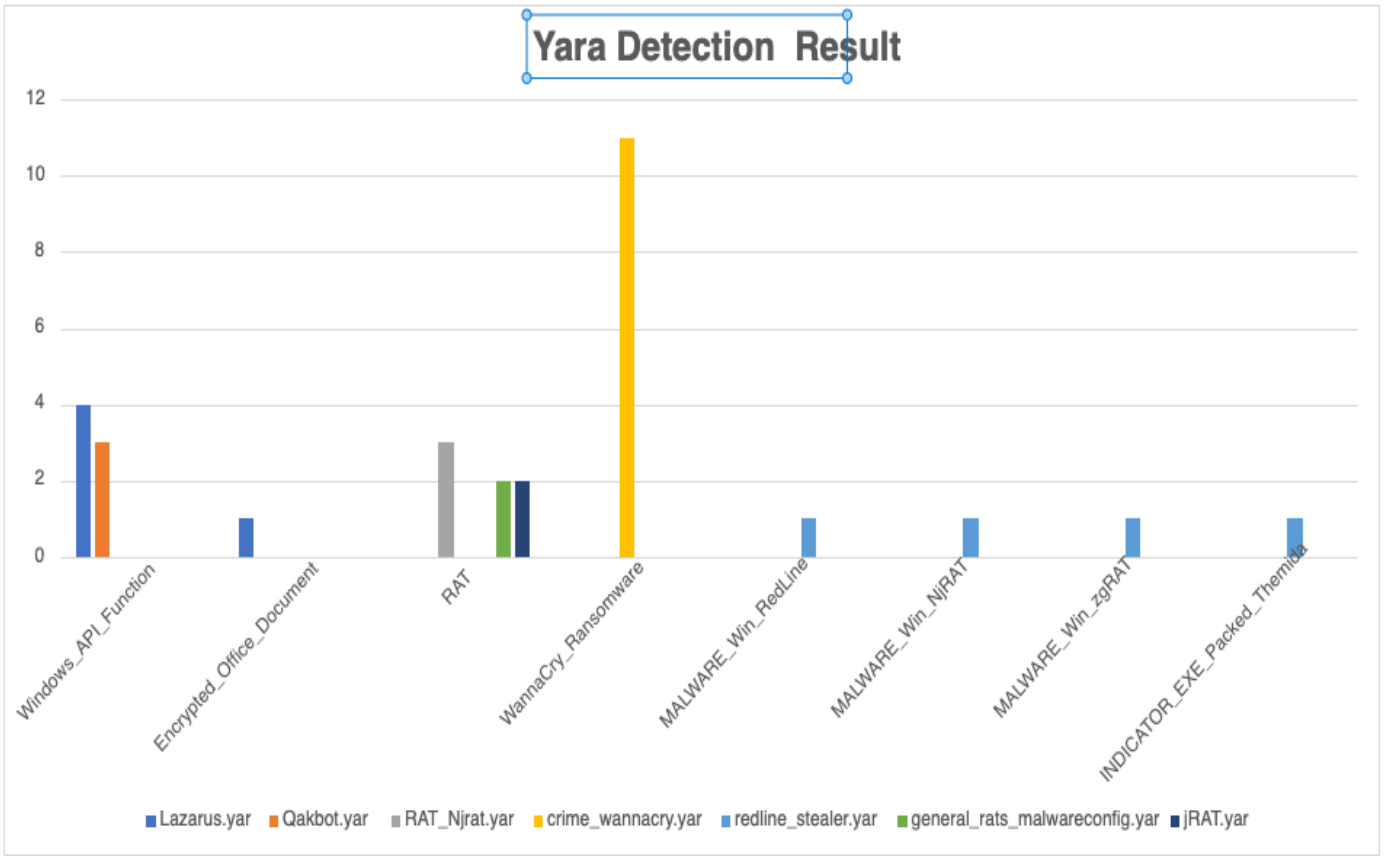
jRAT.yar:

The result of using the jRAT.yar rule with "-r" recursively detects each file of malware example directories.

```
┌──[ray@parrot]─[~/Desktop/LAB ]
└─────$sudo yara jRAT.yar -r malwaresamples/
jRat malwaresamples//df64df82b18e852a3b662b4b26e46a1077fd298c0b9133ba7a8f084b988a4b0f.jar
jRat malwaresamples//2c2e6699405f6fece6adca153c90bdbc58630b10a70b2b92438de04953b5ea12.jar
┌──[ray@parrot]─[~/Desktop/LAB ]
└─────$_
```

The bar chart:

If we put the Yara result below, this indicates most malware belongs to WannaCry, which was a well-known ransomware in 2017 [5].

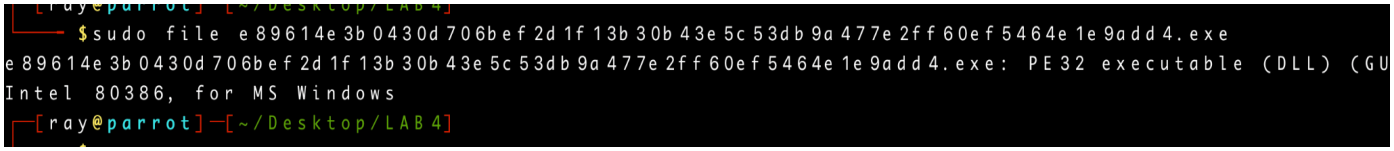


Task 3: Analyse the malware sample

Given the malware and extract the zip file with password:

e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4.exe

Firstly, we have to confirm what file it is and what version of this executable, which is referred to below screenshot.



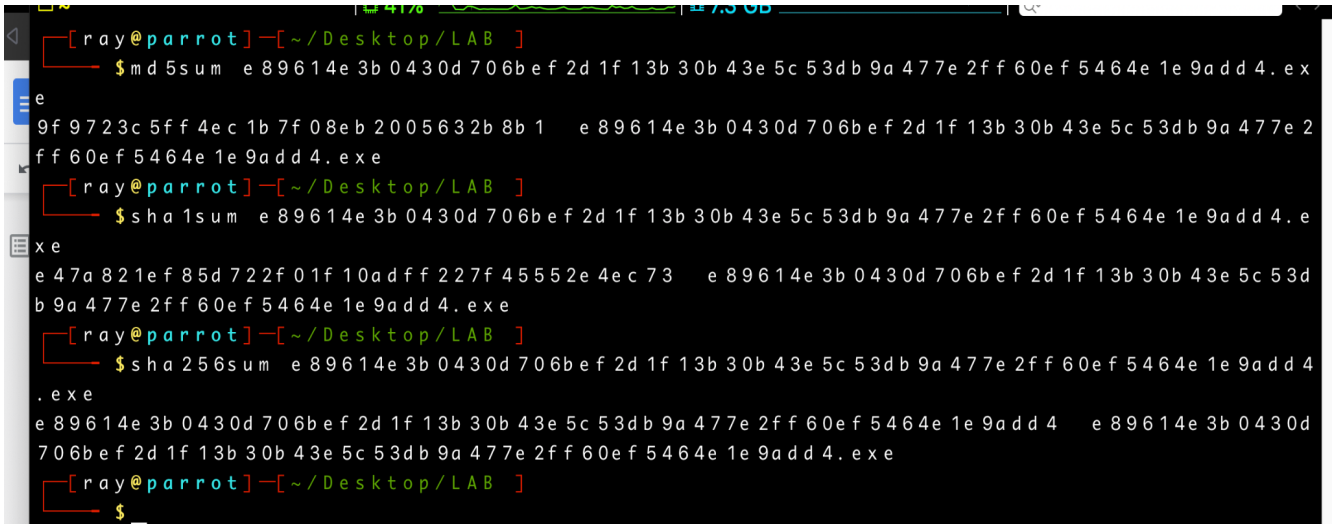
Then, we collect eahc of the following information.

1. ***Hash Value***

   md5: 9f9723c5ff4ec1b7f08eb2005632b8b1

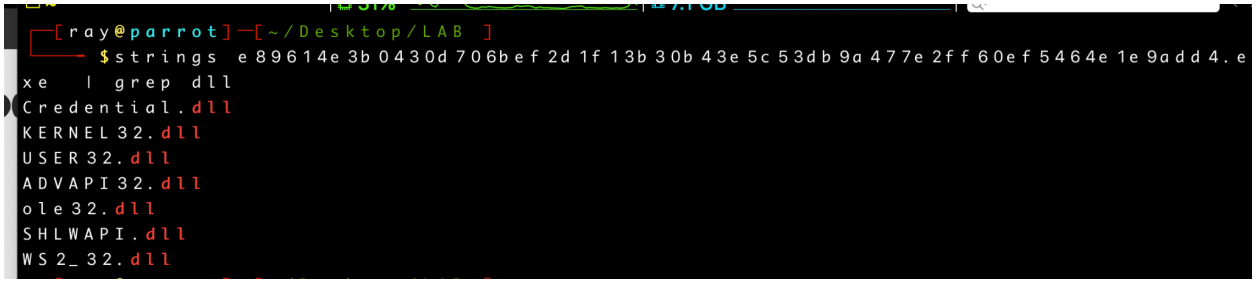
   sha1sum: e47a821ef85d722f01f10adff227f45552e4ec73

   sha256sum: e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4



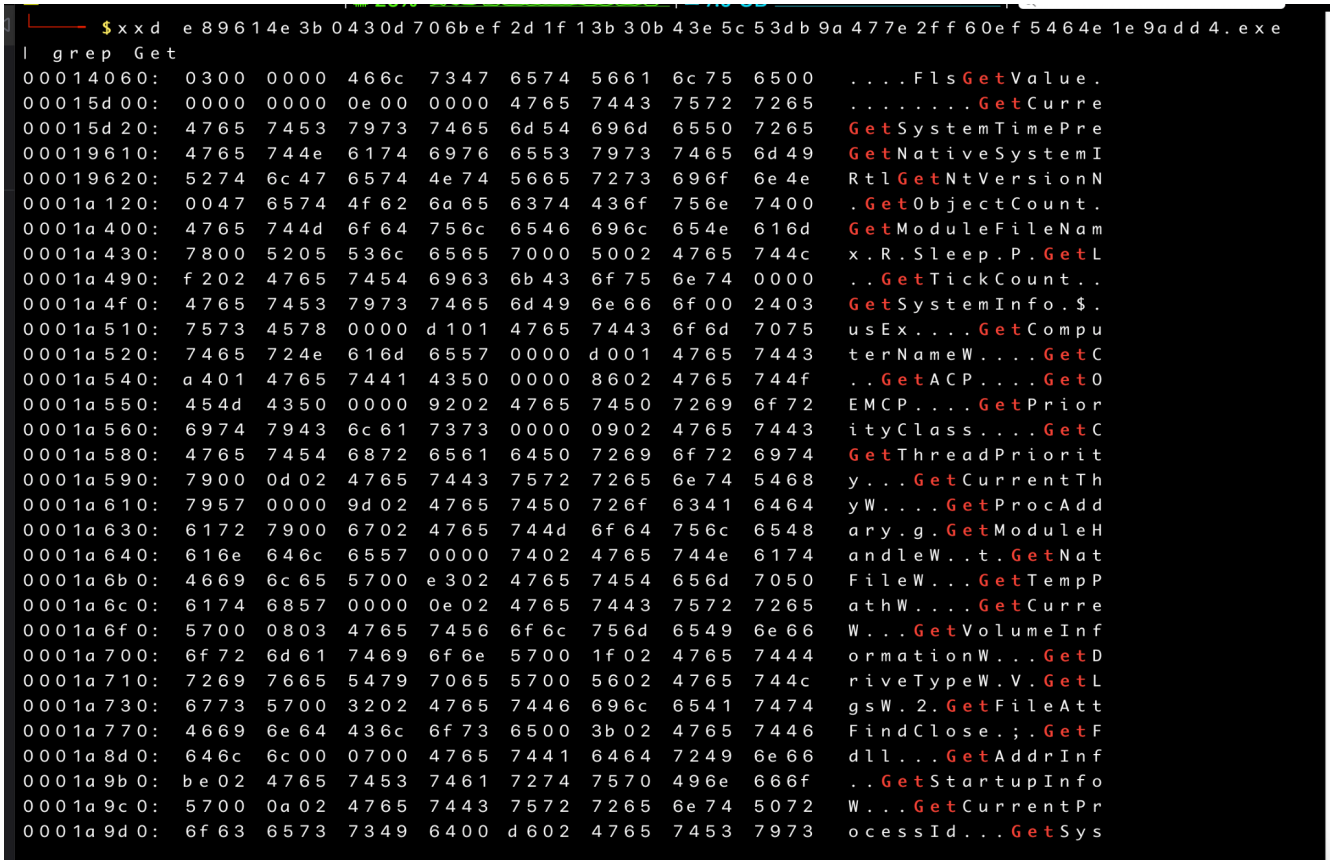
2. *Imported DLLs*

If we use string to extract the information from malware file, this will show this malware utilizes Credential.dll, KERNEL32.dll, USER32.dll, ADVAPI32.dll, ole32.dll, SHLWAPI.dll, WS2_32.dll. These dll indicates that this malware has some network activities in term os ws2_32.dll, and KernelL21.dll has a various functions call to critical service [6].



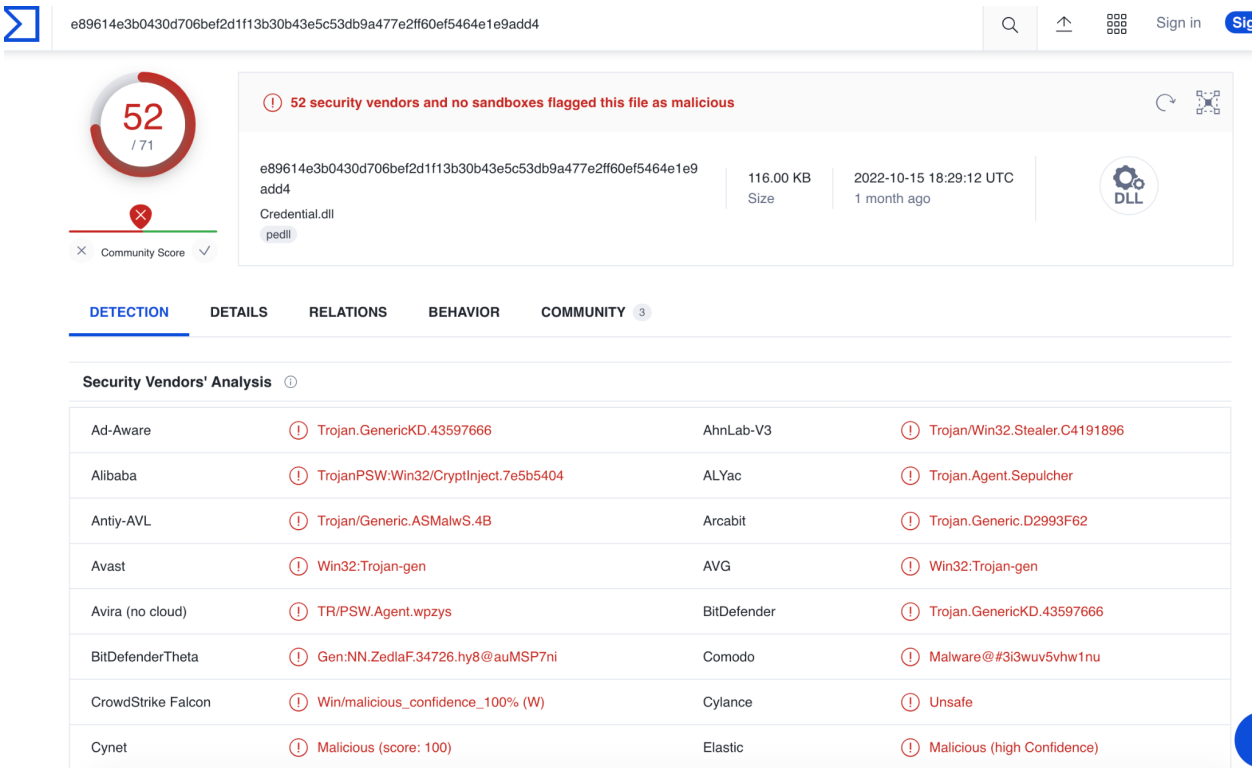
3. *Common Windows API*

We can use some dump commands to filter content and see the common windows API in terms of the below screenshot. The command we utilized is "***xxd***" command to dump the hexadecimal content from the malware file, then pipeline to the "***grep***" to filter Windows API information.



4. VirusTotal Result

Based on the hash file extracted from the malware file, the file is flagged as the malware.



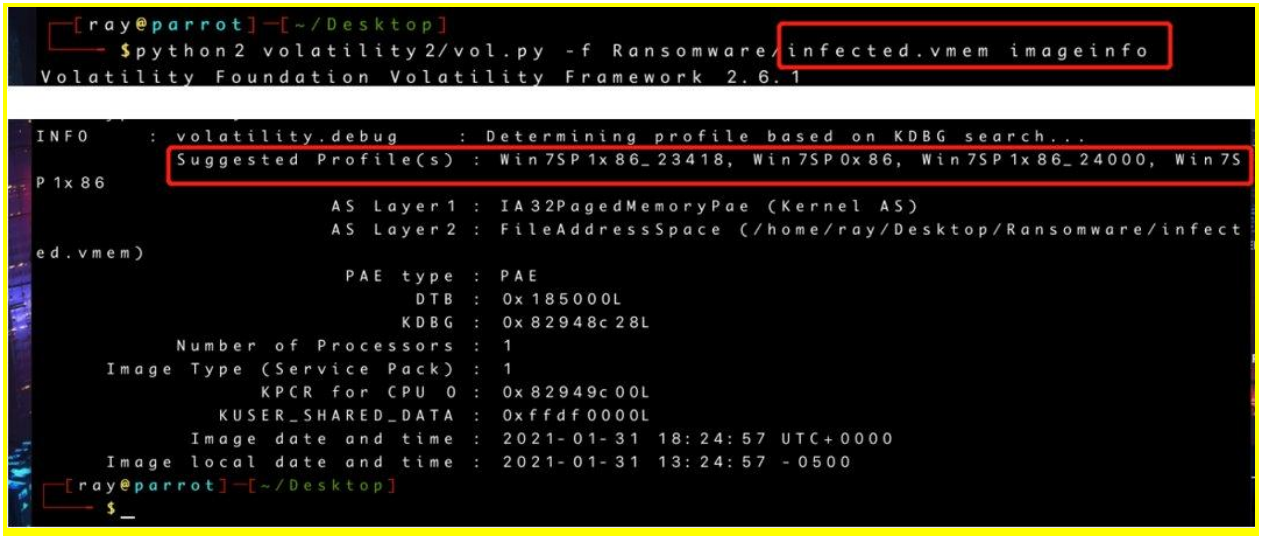
Considering that we can not executable the malware file this this task, and can not get much more information from the malware file. So, we decide to do the extra task.

Extra Task : Analyse the infected memory affecting from WannaCry via memory analysis tool - Volatility.

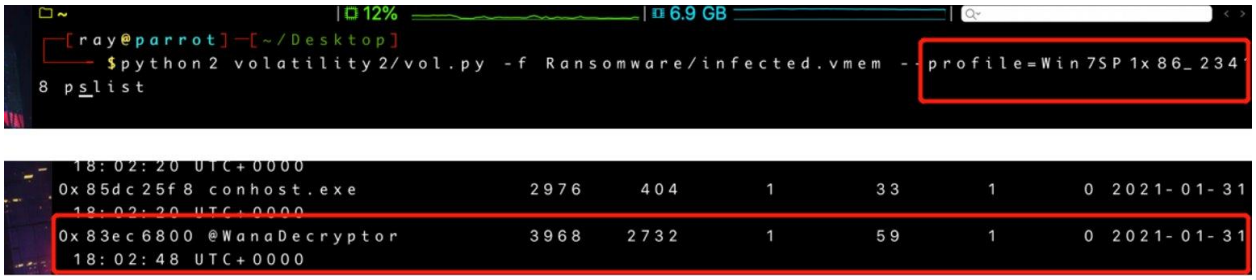
We downloaded an infected file from Blue Team training platform, which is https://blueteamlabs.online/home/challenge/memory-analysis-ransomware-7da6c9244d

From the SOC objective we have to analysis the infected backup memory after the cyberattack.

Confirm the infected machine which version it uses via ***imageinfo*** parameter.

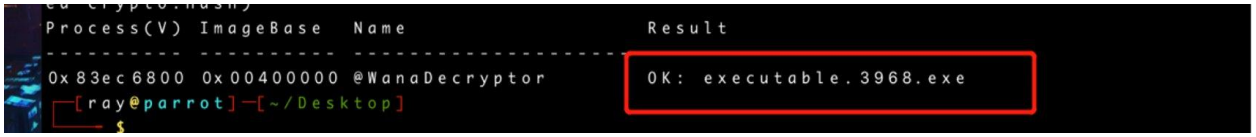


After confirming the system is infected, we can dump process list from the infected memory via ***pslist***

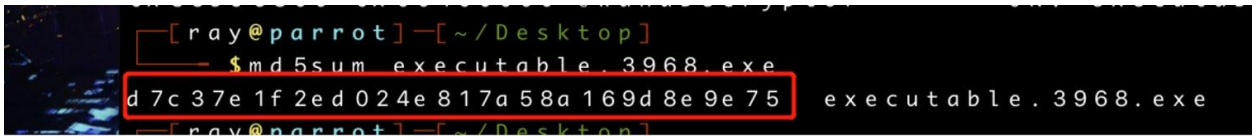


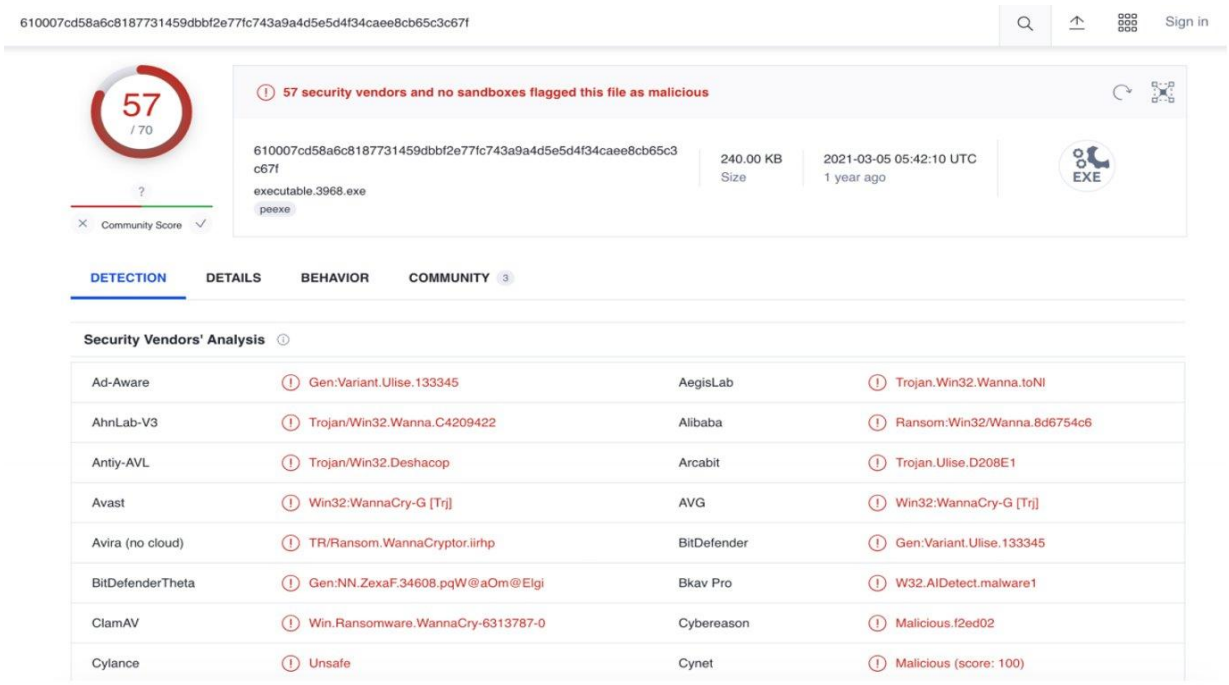
As to the below screenshot, it shows the process id 3968 is running WanaDecryptor process, and we can extract the file from this process with ***procdump***

The malware file is executable.3968.exe

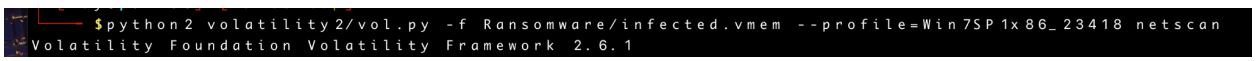


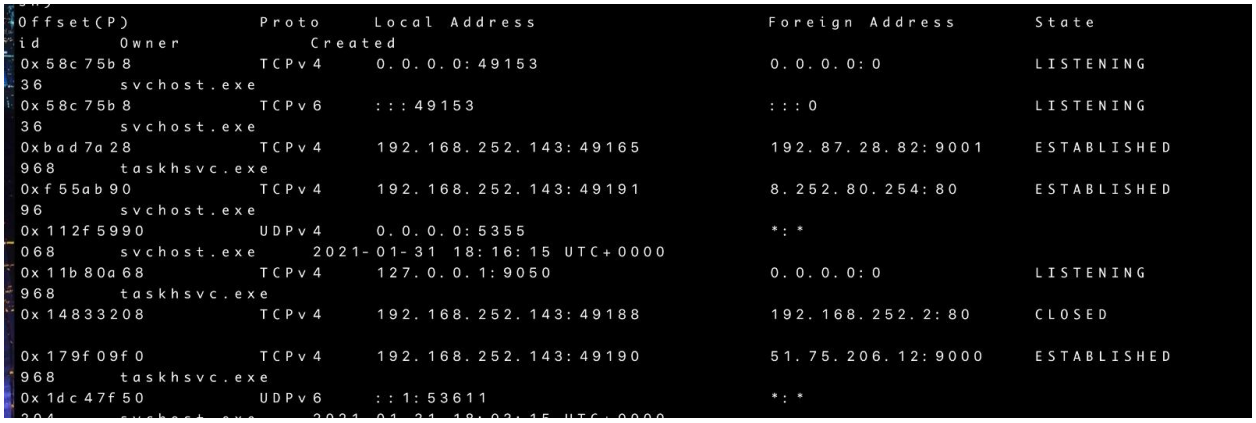
The md5 hash value of the malware is d7c37e1f2ed024e817a58a169d8e9e75, and this is flagged as WannaCry ransomware via VirusTotal





As for the network connection, we can find there is a suspicious connection after using ***netscan*** command. The suspicious connection establishing between ***192.168.252.143:49190*** and ***51.75.206.12:9000***





Conclusion

To summarize this malware detection lab, we have obtained the knowledge how to analyze malware file and how to use Yara rule to detect the file whether it is malware or not. The most important this is we do some extra tasks, which is to perform how to use memory analysis tool with dumping infected virtual memory, and see the supscipicious connection.

Reference

1. GrantMeStrength, “Windows API index - win32 apps,” *Win32 apps | Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list. [Accessed: 24-Nov-2022].

2. Stevewhims, “Registry functions - win32 apps,” *Win32 apps | Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry-functions. [Accessed: 24-Nov-2022].

3. Alvinashcraft, “SetFileTime function (fileapi.h) - win32 apps,” *Win32 apps | Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfiletime. [Accessed: 24-Nov-2022].

4. Dotnet-Bot, “Toolstriptextbox class (system.windows.forms),” *(System.Windows.Forms) | Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.toolstriptextbox?view=windowsdesktop-7.0. [Accessed: 24-Nov-2022].

5. “WannaCry ransomware attack,” *Wikipedia*, 17-Nov-2022. [Online]. Available: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack. [Accessed: 28-Nov-2022].

6. K. K. Reddy, “Basic static analysis of malware and common DLL,” *Medium*, 28-Dec-2019. [Online]. Available: https://medium.com/mrx-007/basic-static-analysis-of-malware-and-common-dll-ef9455d49968. [Accessed: 28-Nov-2022].