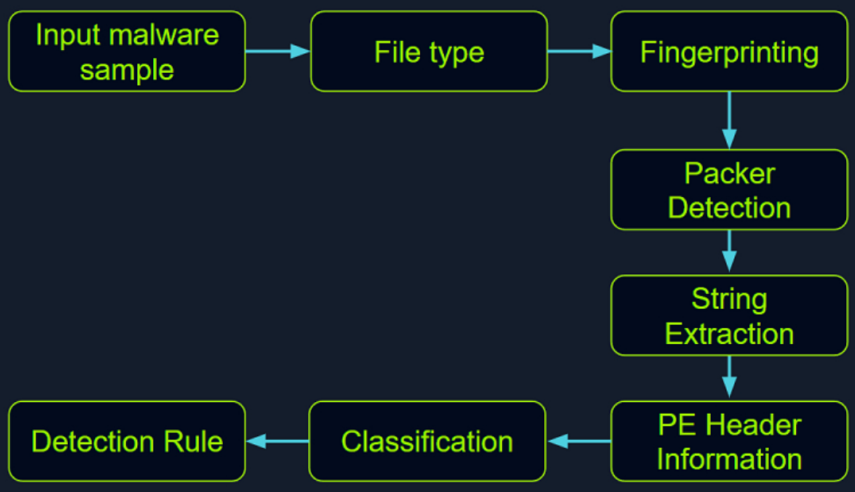
Static Analysis On Linux

In the realm of malware analysis, we exercise a method called static analysis to scrutinize malware without necessitating its execution. This involves the meticulous investigation of malware's code, data, and structural components, serving as a vital precursor for further, more detailed analysis.

Through static analysis, we endeavor to extract pivotal information which includes:

- File type
- File hash
- Strings
- Embedded elements
- Packer information
- Imports
- Exports
- Assembly code



Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, let's SSH into the Target IP using the provided credentials. The vast majority of the actions/commands covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Identifying The File Type

Our first port of call in this stage is to ascertain the rudimentary information about the malware specimen to lay the groundwork for our investigation. Given that file extensions can be manipulated and changed, our task is to devise a method to identify the actual file type we are encountering. Establishing the file type plays an integral role in static analysis, ensuring that the procedures we apply are appropriate and the results obtained are accurate.

Let's use a Windows-based malware named `Ransomware.wannacry.exe` residing in the `/home/htb-student/Samples/MalwareAnalysis` directory of this section's target as an illustration.

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

The command for checking the file type of this malware would be the following.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ file /home/htb-student/Samples/MalwareAnalysis/Ransomware.wannacry.exe
/home/htb-student/Samples/MalwareAnalysis/Ransomware.wannacry.exe: PE32 executable (GUI) Intel 8038
```

We can also do the same by manually checking the header with the help of the `hexdump` command as follows.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ hexdump -C /home/htb-student/Samples/MalwareAnalysis/Ransomware.wannacry.ex
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 f8 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  55 3c 53 90 11 5d 3d c3  11 5d 3d c3 11 5d 3d c3  |U<S..]=..]=..]=.|
00000090  6a 41 31 c3 10 5d 3d c3  92 41 33 c3 15 5d 3d c3  |jA1..]=..A3..]=.|
000000a0  7e 42 37 c3 1a 5d 3d c3  7e 42 36 c3 10 5d 3d c3  |~B7..]=.~B6..]=.|
000000b0  7e 42 39 c3 15 5d 3d c3  d2 52 60 c3 1a 5d 3d c3  |~B9..]=..R`..]=.|
000000c0  11 5d 3c c3 4a 5d 3d c3  27 7b 36 c3 10 5d 3d c3  |.]<.J]=.'{6..]=.|
000000d0  d6 5b 3b c3 10 5d 3d c3  52 69 63 68 11 5d 3d c3  |.[;..]=.Rich.]=.|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
00000100  cc 8e e7 4c 00 00 00 00  00 00 00 00 e0 00 0f 01  |...L............|
00000110  0b 01 06 00 00 90 00 00  00 30 38 00 00 00 00 00  |.........08.....|
00000120  16 9a 00 00 00 10 00 00  00 a0 00 00 00 00 40 00  |..............@.|
00000130  00 10 00 00 00 10 00 00  04 00 00 00 00 00 00 00  |................|
00000140  04 00 00 00 00 00 00 00  00 b0 66 00 00 10 00 00  |..........f.....|
00000150  00 00 00 00 02 00 00 00  00 00 10 00 00 10 00 00  |................|
00000160  00 00 10 00 00 10 00 00  00 00 00 00 10 00 00 00  |................|
00000170  00 00 00 00 00 00 00 00  e0 a1 00 00 a0 00 00 00  |................|
00000180  00 00 31 00 54 a4 35 00  00 00 00 00 00 00 00 00  |..1.T.5.........|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

On a Windows system, the presence of the ASCII string `MZ` (in hexadecimal: `4D 5A`) at the start of a file (known as the "magic number") denotes an executable file. `MZ` stands for Mark Zbikowski, a key architect of MS-DOS.

Malware Fingerprinting

In this stage, our mission is to create a unique identifier for the malware sample. This typically takes the form of a cryptographic hash - MD5, SHA1, or SHA256.

Fingerprinting is employed for numerous purposes, encompassing:

- Identification and tracking of malware samples
- Scanning an entire system for the presence of identical malware
- Confirmation of previous encounters and analyses of the same malware
- Sharing with stakeholders as IoC (Indicators of Compromise) or as part of threat intelligence reports

As an illustration, to check the MD5 file hash of the abovementioned malware the command would be the following.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ md5sum /home/htb-student/Samples/MalwareAnalysis/Ransomware.wannacry.exe
db349b97c37d22f5ea1d1841e3c89eb4  /home/htb-student/Samples/MalwareAnalysis/Ransomware.wannacry.exe
```

To check the SHA256 file hash of the abovementioned malware the command would be the following.

Static Analysis On Linux

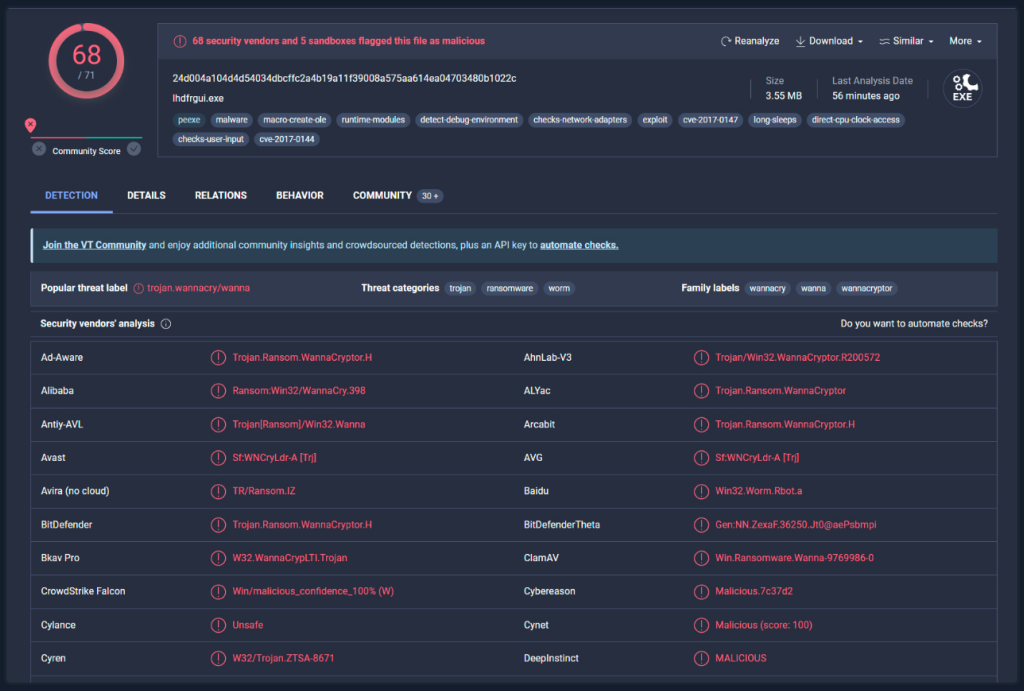
```
MisaelMacias@htb[/htb]$ sha256sum /home/htb-student/Samples/MalwareAnalysis/Ransomware.wannacry.exe
```

```
24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c  /home/htb-student/Samples/Malware
```

File Hash Lookup

The ensuing step involves checking the file hash produced in the prior step against online malware scanners and sandboxes such as Cuckoo sandbox. For instance, VirusTotal, an online malware scanning engine, which collaborates with various antivirus vendors, allows us to search for the file hash. This step aids us in comparing our results with existing knowledge about the malware sample.

The following image displays the results from VirusTotal after the SHA256 file hash of the aforementioned malware was submitted.



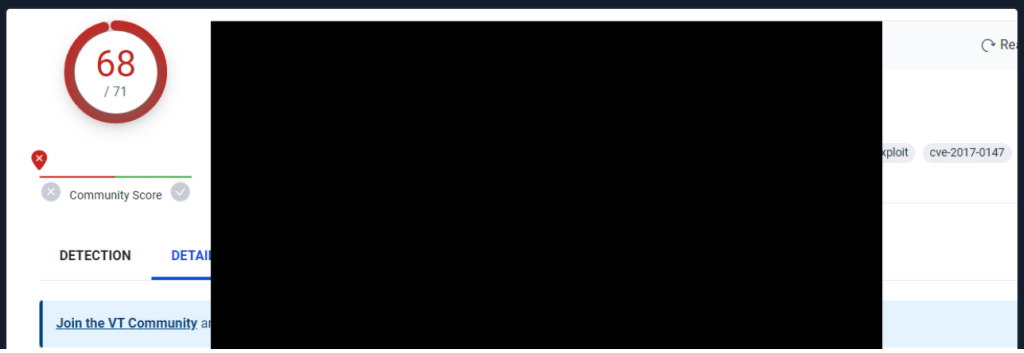
Even though a file hash like MD5, SHA1, or SHA256 is valuable for identifying identical samples with disparate names, it falls short when identifying similar malware samples. This is primarily because a malware author can alter the file hash value by making minor modifications to the code and recompiling it.

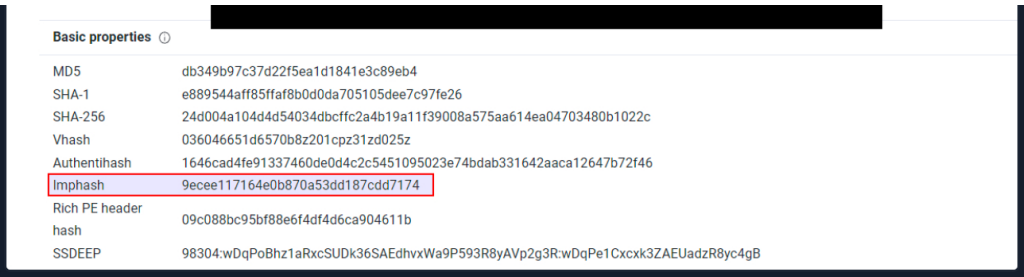
Nonetheless, there exist techniques that can aid in identifying similar samples:

Import Hashing (IMPHASH)

`IMPHASH`, an abbreviation for "Import Hash", is a cryptographic hash calculated from the import functions of a Windows Portable Executable (PE) file. Its algorithm functions by first converting all imported function names to lowercase. Following this, the DLL names and function names are fused together and arranged in alphabetical order. Finally, an MD5 hash is generated from the resulting string. Therefore, two PE files with identical import functions, in the same sequence, will share an `IMPHASH` value.

We can find the `IMPHASH` in the `Details` tab of the VirusTotal results.





Basic properties

MD5	db349b97c37d22f5ea1d1841e3c89eb4
SHA-1	e889544aff85ffaf8b0d0da705105dee7c97fe26
SHA-256	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
Vhash	036046651d6570b8z201cpz31zd025z
Authentihash	1646cad4fe91337460de0d4c2c5451095023e74bdab331642aaca12647b72f46
Imphash	9ecee117164e0b870a53dd187cdd7174
Rich PE header hash	09c088bc95bf88e6f4df4d6ca904611b
SSDEEP	98304:wDqPoBhz1aRxcSUDk36SAEdhvxWa9P593R8yAVp2g3R:wDqPe1Cxcxk3ZAEUadzR8yc4gB

Note that we can also use the pefile Python module to compute the IMPHASH of a file as follows.

Code: python

```python
import sys
import pefile
import peutils

pe_file = sys.argv[1]
pe = pefile.PE(pe_file)
imphash = pe.get_imphash()

print(imphash)
```

To check the IMPHASH of the abovementioned WannaCry malware the command would be the following.

`imphash_calc.py` (available at `/home/htb-student`) contains the Python code above.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ python3 imphash_calc.py /home/htb-student/Samples/MalwareAnalysis/Ransomwar
9ecee117164e0b870a53dd187cdd7174
```

Fuzzy Hashing (SSDEEP)

`Fuzzy Hashing (SSDEEP)`, also referred to as context-triggered piecewise hashing (CTPH), is a hashing technique designed to compute a hash value indicative of content similarity between two files. This technique dissects a file into smaller, fixed-size blocks and calculates a hash for each block. The resulting hash values are then consolidated to generate the final fuzzy hash.

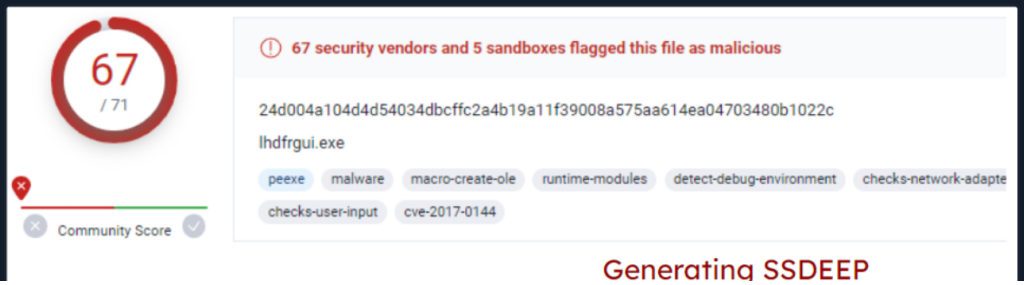
The `SSDEEP` algorithm allocates more weight to longer sequences of common blocks, making it highly effective in identifying files that have undergone minor modifications, or are similar but not identical, such as different variations of a malicious sample.

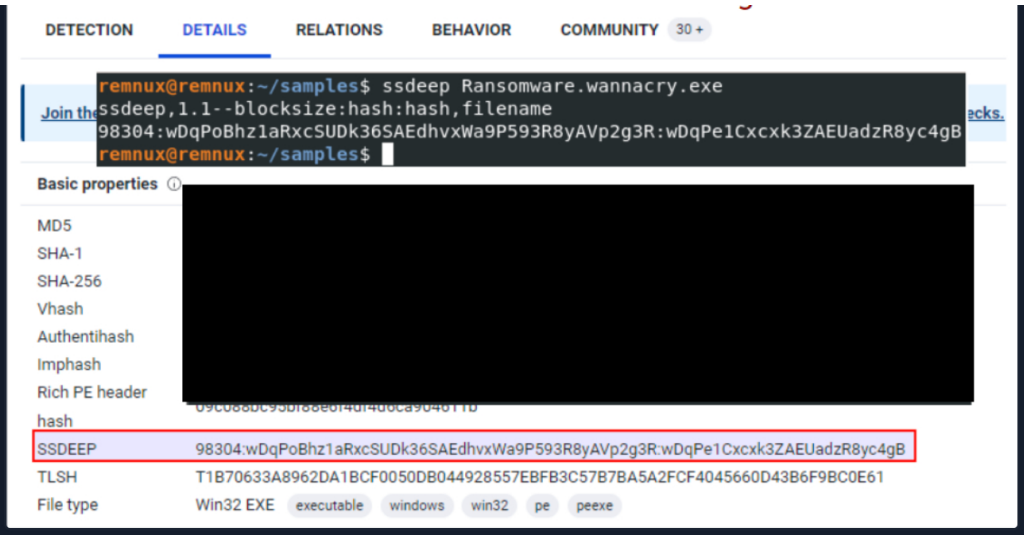
We can find the `SSDEEP` hash of a malware in the `Details` tab of the VirusTotal results.

We can also use the `ssdeep` command to calculate the `SSDEEP` hash of a file. To check the `SSDEEP` hash of the abovementioned WannaCry malware the command would be the following.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ ssdeep /home/htb-student/Samples/MalwareAnalysis/Ransomware.wannacry.exe
ssdeep,1.1--blocksize:hash:hash,filename
98304:wDqPoBhz1aRxcSUDk36SAEdhvxWa9P593R8yAVp2g3R:wDqPe1Cxcxk3ZAEUadzR8yc4gB,"/home/htb-student/Sam
```





The command line arguments `-pb` can be used to initiate matching mode in `SSDEEP` (while we are on the directory where the malware samples are stored - `/home/htb-student/Samples/MalwareAnalysis` in our case).

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ ssdeep -pb *
potato.exe matches svchost.exe (99)

svchost.exe matches potato.exe (99)
```

`-p` denotes Pretty matching mode, and `-b` is used to display only the file names, excluding full paths.

In the example above, a 99% similarity was observed between two malware samples (`svchost.exe` and `potato.exe`) using `SSDEEP`.

Section Hashing (Hashing PE Sections)

`Section hashing`, (hashing PE sections) is a powerful technique that allows analysts to identify sections of a Portable Executable (PE) file that have been modified. This can be particularly useful for identifying minor variations in malware samples, a common tactic employed by attackers to evade detection.

The `Section Hashing` technique works by calculating the cryptographic hash of each of these sections. When comparing two PE files, if the hash of corresponding sections in the two files matches, it suggests that the particular section has not been modified between the two versions of the file.

By applying `section hashing`, security analysts can identify parts of a PE file that have been tampered with or altered. This can help identify similar malware samples, even if they have been slightly modified to evade traditional signature-based detection methods.

Tools such as `pefile` in Python can be used to perform `section hashing`. In Python, for example, you can use the pefile module to access and hash the data in individual sections of a PE file as follows.

Code: python

```python
import sys
import pefile
pe_file = sys.argv[1]
pe = pefile.PE(pe_file)
for section in pe.sections:
    print (section.Name, "MD5 hash:", section.get_hash_md5())
    print (section.Name, "SHA256 hash:", section.get_hash_sha256())
```

Remember that while `section hashing` is a powerful technique, it is not foolproof. Malware authors might employ tactics like section name obfuscation or dynamically generating section names to try and bypass this kind of analysis.

As an illustration, to check the MD5 and SHA256 PE section hashes of a Wannacry executable stored in the `/home/htb-student/Samples/MalwareAnalysis` directory, the command would be the following. `section_hashing.py` (available at `/home/htb-student`) contains the Python code above.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ python3 section_hashing.py /home/htb-student/Samples/MalwareAnalysis/Ransom
b'.text\x00\x00\x00' MD5 hash: c7613102e2ecec5dcefc144f83189153
b'.text\x00\x00\x00' SHA256 hash: 7609ecc798a357dd1a2f0134f9a6ea06511a8885ec322c7acd0d84c569398678
b'.rdata\x00\x00' MD5 hash: d8037d744b539326c06e897625751cc9
b'.rdata\x00\x00' SHA256 hash: 532e9419f23eaf5eb0e8828b211a7164cbf80ad54461bc748c1ec2349552e6a2
b'.data\x00\x00\x00' MD5 hash: 22a8598dc29cad7078c291e94612ce26
b'.data\x00\x00\x00' SHA256 hash: 6f93fb1b241a990ecc281f9c782f0da471628f6068925aaf580c1b1de86bce8a
b'.rsrc\x00\x00\x00' MD5 hash: 12e1bd7375d82cca3a51ca48fe22d1a9
b'.rsrc\x00\x00\x00' SHA256 hash: 1efe677209c1284357ef0c7996a1318b7de3836dfb11f97d85335d6d3b8a8e42
```

String Analysis

In this phase, our objective is to extract strings (ASCII & Unicode) from a binary. Strings can furnish clues and valuable insight into the functionality of the malware. Occasionally, we can unearth unique embedded strings in a malware sample, such as:

- `Embedded filenames (e.g., dropped files)`
- `IP addresses or domain names`
- `Registry paths or keys`
- `Windows API functions`
- `Command-line arguments`
- `Unique information that might hint at a particular threat actor`

The Linux `strings` command can be deployed to display the strings contained within a malware. For instance, the command below will reveal strings for a ransomware sample named `dharma_sample.exe` residing in the `/home/htb-student/Samples/MalwareAnalysis` directory of this section's target.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ strings -n 15 /home/htb-student/Samples/MalwareAnalysis/dharma_sample.exe
!This program cannot be run in DOS mode.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@>@@@?456789:;<=@@@@@@@
!"#$%&'()*+,-./0123@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
WaitForSingleObject
InitializeCriticalSectionAndSpinCount
LeaveCriticalSection
EnterCriticalSection
C:\crysis\Release\PDB\payload.pdb
0123456789ABCDEF
```

`-n` specifies to print a sequence of at least the number specified - in our case, `15`.

Occasionally, string analysis can facilitate the linkage of a malware sample to a specific threat group if significant similarities are identified. For example, in the link provided, a string containing a PDB path was used to link the malware sample to the Dharma/Crysis family of ransomware.

Strings

- 0xc814:$s1: C:\crysis\Release\PDB\payload.pdb

It should be noted that another string analysis solution exists called `FLOSS`. `FLOSS`, short for "FireEye Labs Obfuscated String Solver", is a tool developed by FireEye's FLARE team to automatically deobfuscate strings in malware. It's designed to supplement the use of traditional string tools, like the strings command in Unix-based systems, which can

miss obfuscated strings that are commonly used by malware to evade detection.

For instance, the command below will reveal strings for a ransomware sample named `dharma_sample.exe` residing in the `/home/htb-student/Samples/MalwareAnalysis` directory of this section's target.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ floss /home/htb-student/Samples/MalwareAnalysis/dharma_sample.exe
INFO: floss: extracting static strings...
finding decoding function features: 100%|| 238/238 [00:00<00
INFO: floss.stackstrings: extracting stackstrings from 223 functions
INFO: floss.results: %sh(
extracting stackstrings: 100%|
INFO: floss.tightstrings: extracting tightstrings from 10 functions...
extracting tightstrings from function 0x4065e0: 100%|
INFO: floss.string_decoder: decoding strings
INFO: floss.results: EEED
INFO: floss.results: EEEDnnn
INFO: floss.results: uOKm
INFO: floss.results: %sh(
INFO: floss.results: uBIA
INFO: floss.results: uBIA
INFO: floss.results: \t\t\t\t\t\t\t\t
emulating function 0x405840 (call 4/9): 100%|
INFO: floss: finished execution after 23.56 seconds

FLARE FLOSS RESULTS (version v2.0.0-0-gdd9bea8)
+-----------------------+-------------------------------------------------------------------------------
| file path             | /home/htb-student/Samples/MalwareAnalysis/dharma_sample.exe
| extracted strings     |
|  static strings       | 720
|  stack strings        | 1
|  tight strings        | 0
|  decoded strings      | 7
+-----------------------+-------------------------------------------------------------------------------

-----------------------------
| FLOSS STATIC STRINGS (720) |
-----------------------------
-----------------------------
| FLOSS ASCII STRINGS (716) |
-----------------------------
!This program cannot be run in DOS mode.
Rich
.text
`.rdata
@.data
9A s
9A$v
A +B$
---SNIP---
+o*7
0123456789ABCDEF

-----------------------------
| FLOSS UTF-16LE STRINGS (4) |
-----------------------------
jjjj
%sh(
ssbss
0123456789ABCDEF

--------------------------
| FLOSS STACK STRINGS (1) |
--------------------------
%sh(

--------------------------
| FLOSS TIGHT STRINGS (0) |
--------------------------

-----------------------------
| FLOSS DECODED STRINGS (7) |
-----------------------------
EEED
EEEDnnn
uOKm
%sh(
uBIA
uBIA
\t\t\t\t\t\t\t\t
```

Unpacking UPX-packed Malware

In our static analysis, we might stumble upon a malware sample that's been compressed or obfuscated using a technique referred to as packing. Packing serves several purposes:

- It obfuscates the code, making it more challenging to discern its structure or functionality.
- It reduces the size of the executable, making it quicker to transfer or less conspicuous.
- It confounds security researchers by hindering traditional reverse engineering attempts.

This can impair string analysis because the references to strings are typically obscured or eliminated. It also replaces or camouflages conventional PE sections with a compact loader stub, which retrieves the original code from a compressed data section. As a result, the malware file becomes both smaller and more difficult to analyze, as the original code isn't directly observable.

A popular packer used in many malware variants is the `Ultimate Packer for Executables (UPX)`.

Let's first see what happens when we run the `strings` command on a UPX-packed malware sample named `credential_stealer.exe` residing in the `/home/htb-student/Samples/MalwareAnalysis/packed` directory of this section's target.

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ strings /home/htb-student/Samples/MalwareAnalysis/packed/credential_stealer
!This program cannot be run in DOS mode.
UPX0
UPX1
UPX2
3.96
UPX!
8MZu
HcP<H
VDgxt
$ /uX
OAUATUWVSH
%0rv
o?H9
c`fG
[^_]A\A]
> -P
        fo{Wnl
c9"^$!=
v/7>
07ZC
_L$AAl
mug.%(
#8%,X
e]'^
---SNIP---
```

Observe the strings that include `UPX`, and take note that the remainder of the output doesn't yield any valuable information regarding the functionality of the malware.

We can unpack the malware using the `UPX` tool with the following command (while we are on the directory where the packed malware samples are stored - `/home/htb-student/Samples/MalwareAnalysis/packed` in our case).

Static Analysis On Linux

```
MisaelMacias@htb[/htb]$ upx -d -o unpacked_credential_stealer.exe credential_stealer.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     16896 <-      8704   51.52%    win64/pe     unpacked_credential_stealer.exe

Unpacked 1 file.
```

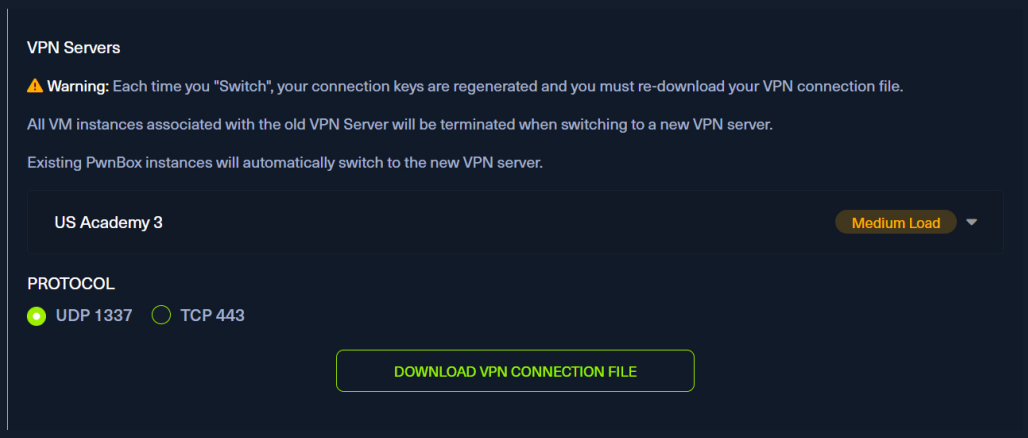
Let's now run the `strings` command on the unpacked sample.

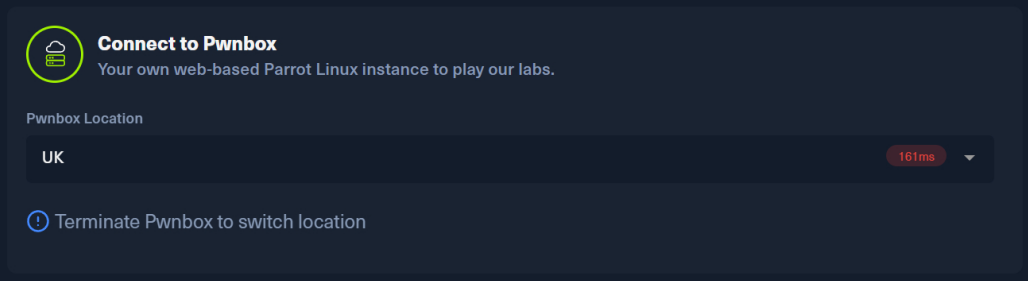
Static Analysis On Linux

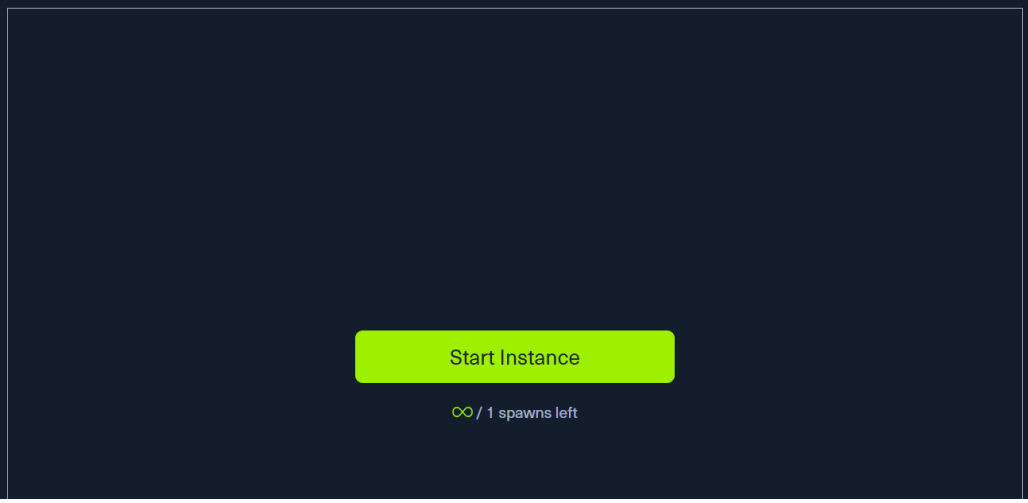
```
MisaelMacias@htb[/htb]$ strings unpacked_credential_stealer.exe
!This program cannot be run in DOS mode.
.text
P`.data
.rdata
`@.pdata
0@.xdata
0@.bss
.idata
.CRT
.tls
---SNIP---
AVAUATH
@A\A]A^
SeDebugPrivilege
SE Debug Privilege is adjusted
lsass.exe
Searching lsass PID
Lsass PID is: %lu
Error is - %lu
lsassmem.dmp
LSASS Memory is dumped successfully
Err 2: %lu
Unknown error
Argument domain error (DOMAIN)
Overflow range error (OVERFLOW)
Partial loss of significance (PLOSS)
Total loss of significance (TLOSS)
The result is too small to be represented (UNDERFLOW)
Argument singularity (SIGN)
_matherr(): %s in %s(%g, %g)  (retval=%g)
Mingw-w64 runtime failure:
Address %p has no image-section
  VirtualQuery failed for %d bytes at address %p
  VirtualProtect failed with code 0x%x
  Unknown pseudo relocation protocol version %d.
  Unknown pseudo relocation bit size %d.
.pdata
AdjustTokenPrivileges
LookupPrivilegeValueA
OpenProcessToken
MiniDumpWriteDump
CloseHandle
CreateFileA
CreateToolhelp32Snapshot
DeleteCriticalSection
EnterCriticalSection
GetCurrentProcess
GetCurrentProcessId
GetCurrentThreadId
GetLastError
GetStartupInfoA
GetSystemTimeAsFileTime
GetTickCount
InitializeCriticalSection
LeaveCriticalSection
OpenProcess
Process32First
Process32Next
QueryPerformanceCounter
RtlAddFunctionTable
RtlCaptureContext
RtlLookupFunctionEntry
RtlVirtualUnwind
SetUnhandledExceptionFilter
Sleep
TerminateProcess
TlsGetValue
UnhandledExceptionFilter
VirtualProtect
VirtualQuery
__C_specific_handler
__getmainargs
__initenv
__iob_func
__lconv_init
__set_app_type
```

```
__setusermatherr
_acmdln
_amsg_exit
_cexit
_fmode
_initterm
_onexit
abort
calloc
exit
fprintf
free
fwrite
malloc
memcpy
printf
puts
signal
strcmp
strlen
strncmp
vfprintf
ADVAPI32.dll
dbghelp.dll
KERNEL32.DLL
msvcrt.dll
```

Now, we observe a more comprehensible output that includes the actual strings present in the sample.







Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Download VPN Connection File

Target(s): Click here to spawn the target system!

SSH to with user "htb-student" and password "HTB_@cademy_stdnt!"

+ 1 In the /home/htb-student/Samples/MalwareAnalysis directory of this section's target, there is a file called potato.exe. Compute the IMPHASH of this file and submit it as your answer.

3399c4043c56fea40a8189de302fd889

Submit

Previous Next

Mark Complete & Next