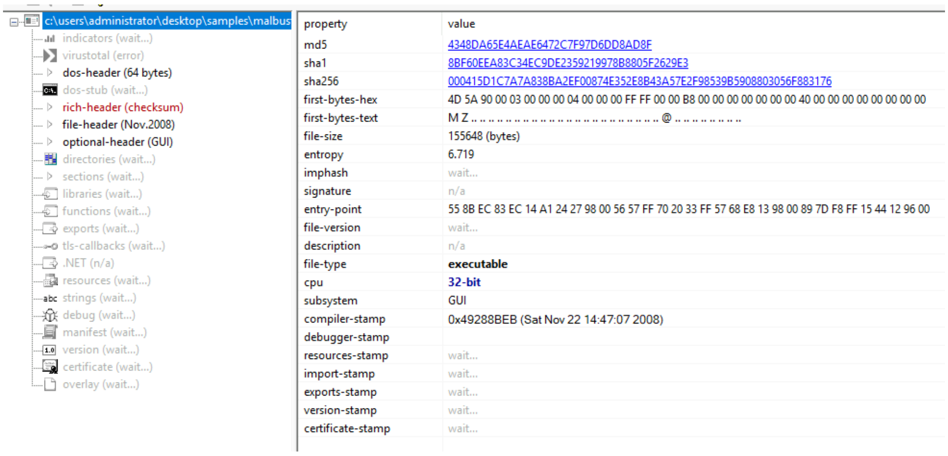
TryHackMe: MalBuster

The following writeup covers the MalBuster room on TryHackMe. This room involves analysing an unknown malware sample using static analysis techniques. It is aimed towards those new to malware analysis (like myself).

Scenario: You are currently working as a Malware Reverse Engineer for your organisation. Your team acts as a support for the SOC team when detections of unknown binaries occur. One of the SOC analysts triaged an alert triggered by binaries with unusual behaviour. Your task is to analyse the binaries detected by your SOC team and provide enough information to assist them in remediating the treat.

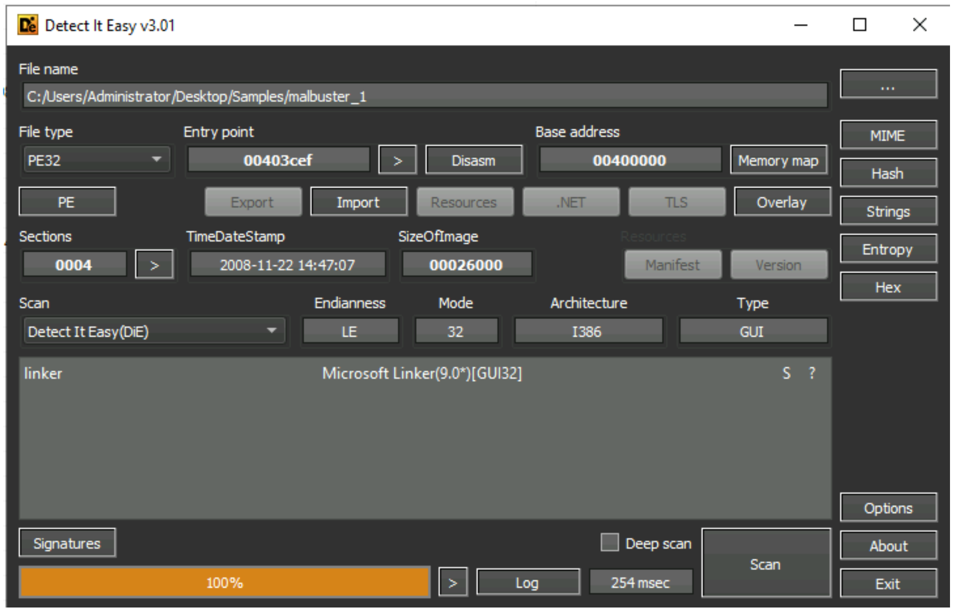
Based on the ARCHITECTURE of the binary, is malbuster_1 a 32-bit or a 64-bit application?

There are several ways to determine if the binary malbuster_1 is a 32-bit or 64-bit application, you can use something like Detect It Easy (DIE) or pestudio:



```
c:\users\administrator\desktop\samples\malbus
  indicators (wait...)
  virustotal (error)
  dos-header (64 bytes)
  dos-stub (wait...)
  rich-header (checksum)
  file-header (Nov.2008)
  optional-header (GUI)
  directories (wait...)
  sections (wait...)
  libraries (wait...)
  functions (wait...)
  exports (wait...)
  tls-callbacks (wait...)
  .NET (n/a)
  resources (wait...)
  strings (wait...)
  debug (wait...)
  manifest (wait...)
  version (wait...)
  certificate (wait...)
  overlay (wait...)
```

property	value
md5	4348DA65E4AEAE6472C7F97D6DD8AD8F
sha1	8BF60EEA83C34EC9DE2359219978B8805F2629E3
sha256	000415D1C7A7A838BA2EF00874E352E8B43A57E2F98539B5908803056F883176
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
first-bytes-text	M Z .. .. @ .. ..
file-size	155648 (bytes)
entropy	6.719
imphash	wait...
signature	n/a
entry-point	55 8B EC 83 EC 14 A1 24 27 98 00 56 57 FF 70 20 33 FF 57 68 E8 13 98 00 89 7D F8 FF 15 44 12 96 00
file-version	wait...
description	n/a
file-type	**executable**
cpu	**32-bit**
subsystem	GUI
compiler-stamp	0x49288BEB (Sat Nov 22 14:47:07 2008)
debugger-stamp	
resources-stamp	wait...
import-stamp	wait...
exports-stamp	wait...
version-stamp	wait...
certificate-stamp	wait...



Detect It Easy v3.01

File name: C:/Users/Administrator/Desktop/Samples/malbuster_1

File type: PE32 | Entry point: 00403cef | Base address: 00400000

Sections: 0004 | TimeDateStamp: 2008-11-22 14:47:07 | SizeOfImage: 00026000

Scan: Detect It Easy(DiE) | Endianness: LE | Mode: 32 | Architecture: I386 | Type: GUI

linker Microsoft Linker(9.0*)[GUI32] S ?

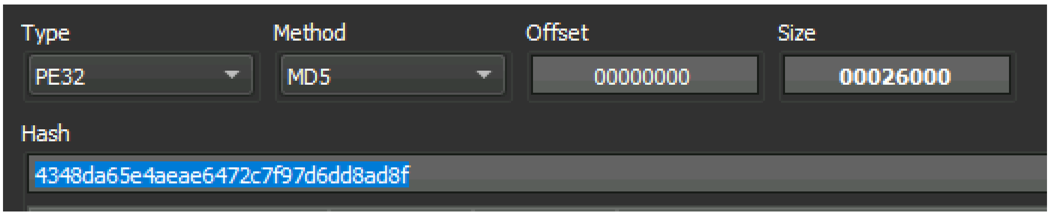
□ Deep scan

100% | 254 msec

As you can see, it is a 32-bit application.

What is the MD5 hash of malbuster_1?

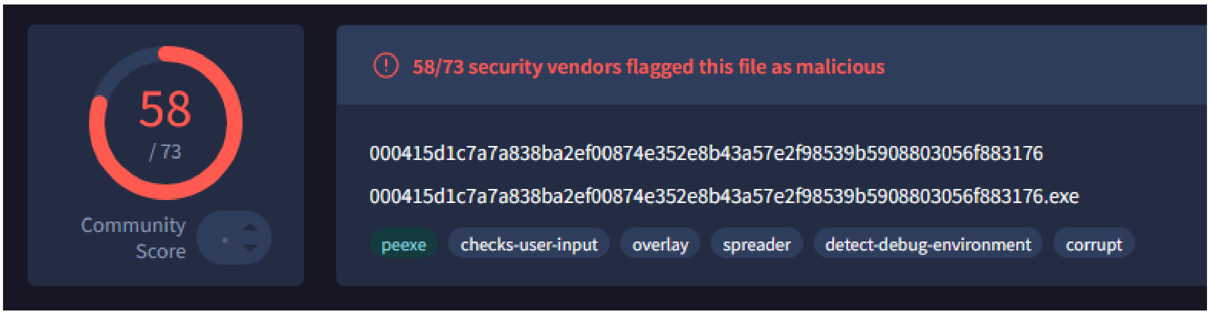
You can find the MD5 hash for the file in pestudio or DIE:



Type: PE32
Method: MD5
Offset: 00000000
Size: 00026000
Hash: 4348da65e4aeae6472c7f97d6dd8ad8f

Using the hash, what is the number of detections of malbuster_1 in VirusTotal?

The answer is 62, however, when I searched for the hash in VirusTotal only 58 vendors flagged the hash as malware:



58 / 73

Community Score

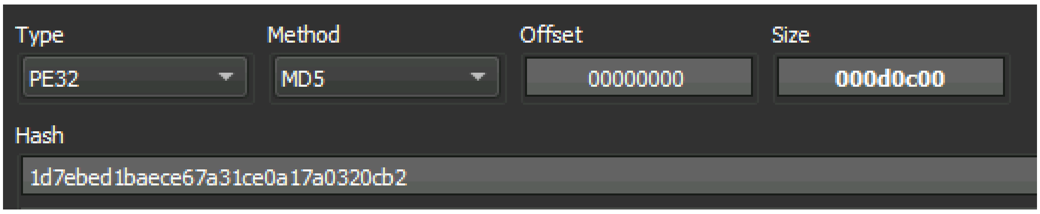
58/73 security vendors flagged this file as malicious

000415d1c7a7a838ba2ef00874e352e8b43a57e2f98539b5908803056f883176

000415d1c7a7a838ba2ef00874e352e8b43a57e2f98539b5908803056f883176.exe

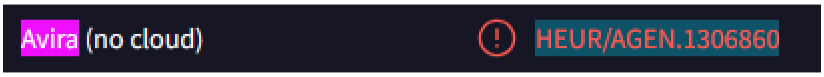
Based on VirusTotal detection, what is the malware signature of malbuster_2 according to Avira?

First, we need to generate the MD5 or SHA256 hash of the malbuster_2 binary:



Type: PE32
Method: MD5
Offset: 00000000
Size: 000d0c00
Hash: 1d7ebed1baece67a31ce0a17a0320cb2

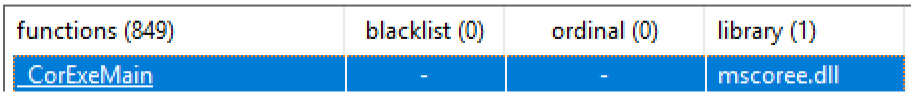
Once you enter this hash into VirusTotal, you can find the malware signature:



Avira (no cloud) ⓘ HEUR/AGEN.1306860

Malbuster_2 imports the function _CorExeMain. From which DLL file does it import this function?

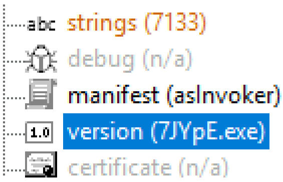
If you open up malbuster_2 in pestudio, and navigation to the function tab, you can determine that _CoreExeMain was imported from mscoree.dll:



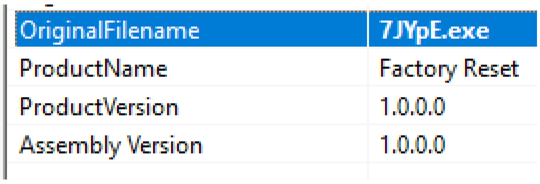
functions (849)	blacklist (0)	ordinal (0)	library (1)
_CorExeMain	-	-	mscoree.dll

Based on the VS_VERSION_INFO header, what is the original name of malbuster_2?

You can find the OriginalFilename in the version tab of pestudio:



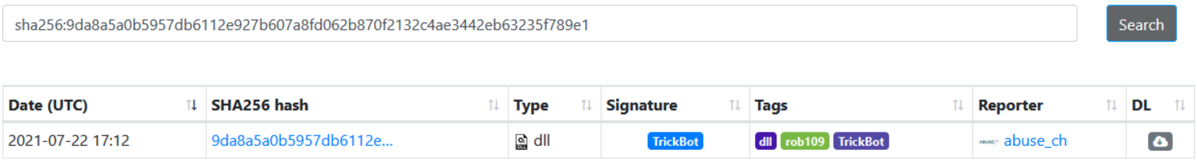
strings (7133)
debug (n/a)
manifest (asInvoker)
version (7JYpE.exe)
certificate (n/a)



OriginalFilename	7JYpE.exe
ProductName	Factory Reset
ProductVersion	1.0.0.0
Assembly Version	1.0.0.0

Using the hash of malbuster_3, what is its malware signature based on abuse.ch?

Start by generating the hash for malbuster_3, in this case I used DIE to generate the sha256 hash. Then all you need to do is visit bazaar.abuse.ch/browser and enter sha256: 9da8a5a0b5957db6112e927b607a8fd062b870f2132c4ae3442eb63235f789e1



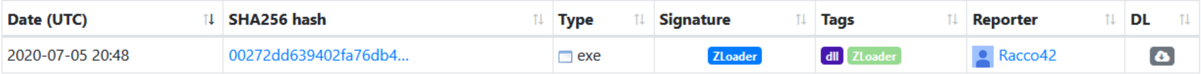
sha256:9da8a5a0b5957db6112e927b607a8fd062b870f2132c4ae3442eb63235f789e1

Date (UTC)	SHA256 hash	Type	Signature	Tags	Reporter	DL
2021-07-22 17:12	9da8a5a0b5957db6112e...	dll	TrickBot	dll rob109 TrickBot	abuse_ch	

As you can see, the malware signature is TrickBot, an infamous banking Trojan.

Using the hash of malbuster_4, what is its malware signature based on abuse.ch?

Follow the same process as the previous question but replace malbuster_3 with malbuster_4:

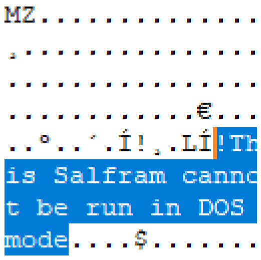


Date (UTC)	SHA256 hash	Type	Signature	Tags	Reporter	DL
2020-07-05 20:48	00272dd639402fa76db4...	exe	ZLoader	dll ZLoader	Racco42	

What is the message found in the DOS_STUB of malbuster_4?

Opening up the file using HxD (a hex editor), you can see that the message found in the DOS_STUB is:

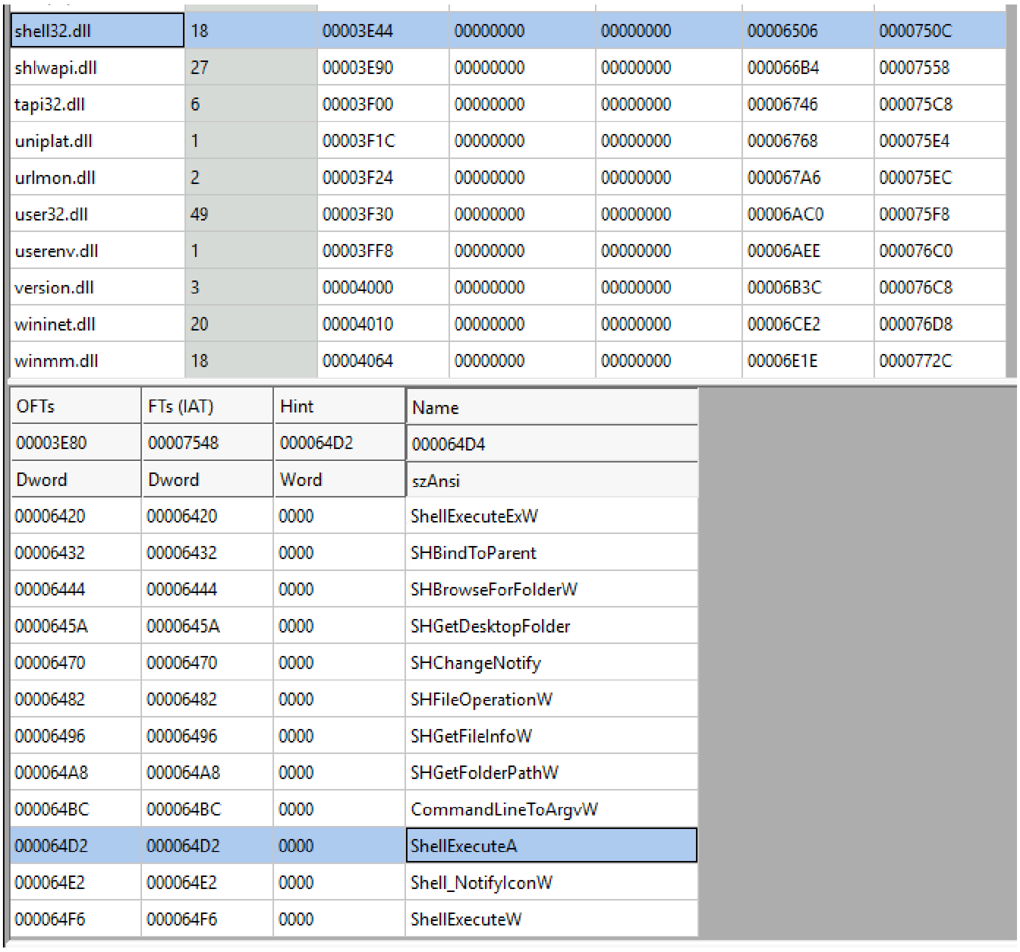
- !This Salfram cannot be run in DOS mode



```
MZ..............
¸...............
................
............€...
..º..´.Í!¸.LÍ!Th
is Salfram canno
t be run in DOS 
mode....$.......
```

Malbuster_4 imports the function ShellExecuteA. From which DLL file does it import this function?

If you open up the binary in CFF Explorer and navigate to the Import Directory tab, you can see that ShellExecuteA was imported from shell32.dll:

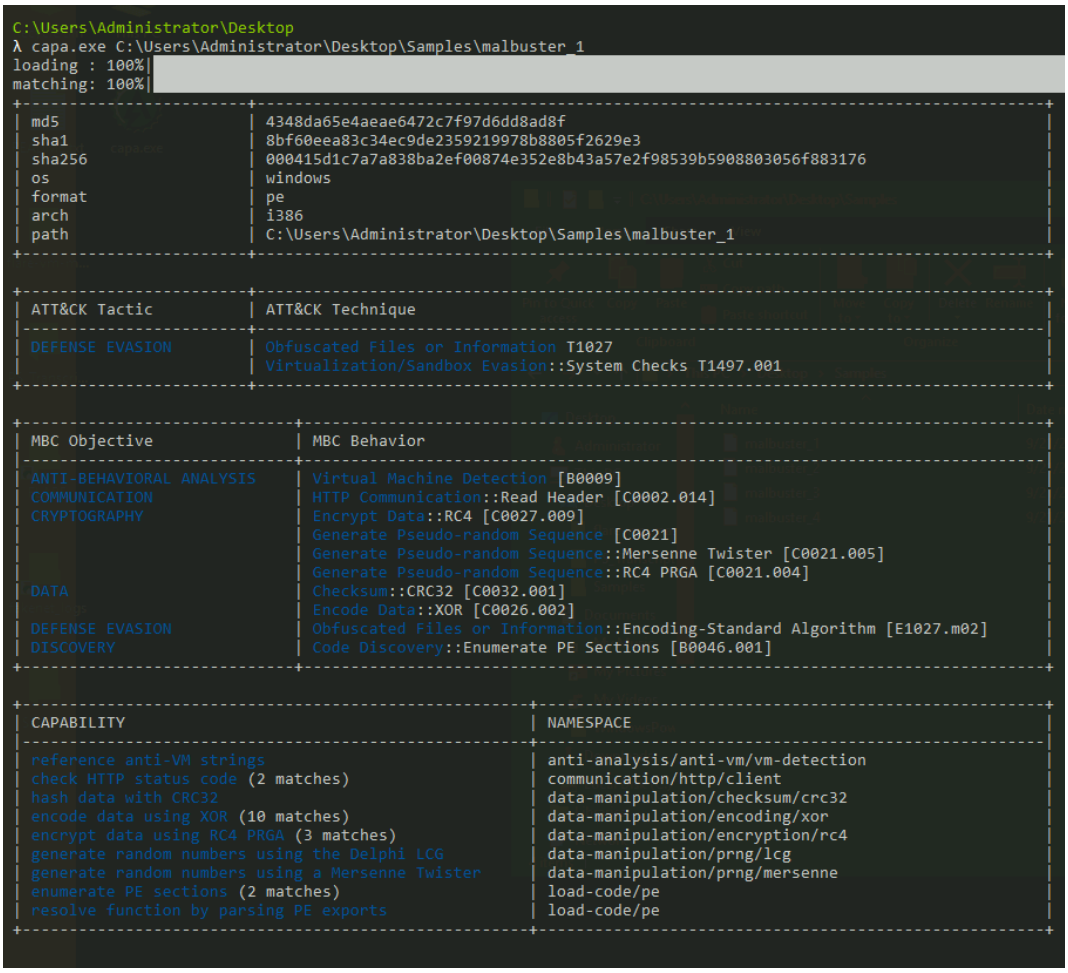


shell32.dll	18	00003E44	00000000	00000000	00006506	0000750C
shlwapi.dll	27	00003E90	00000000	00000000	000066B4	00007558
tapi32.dll	6	00003F00	00000000	00000000	00006746	000075C8
uniplat.dll	1	00003F1C	00000000	00000000	00006768	000075E4
urlmon.dll	2	00003F24	00000000	00000000	000067A6	000075EC
user32.dll	49	00003F30	00000000	00000000	00006AC0	000075F8
userenv.dll	1	00003FF8	00000000	00000000	00006AEE	000076C0
version.dll	3	00004000	00000000	00000000	00006B3C	000076C8
wininet.dll	20	00004010	00000000	00000000	00006CE2	000076D8
winmm.dll	18	00004064	00000000	00000000	00006E1E	0000772C

OFTs	FTs (IAT)	Hint	Name
00003E80	00007548	000064D2	000064D4
Dword	Dword	Word	szAnsi
00006420	00006420	0000	ShellExecuteExW
00006432	00006432	0000	SHBindToParent
00006444	00006444	0000	SHBrowseForFolderW
0000645A	0000645A	0000	SHGetDesktopFolder
00006470	00006470	0000	SHChangeNotify
00006482	00006482	0000	SHFileOperationW
00006496	00006496	0000	SHGetFileInfoW
000064A8	000064A8	0000	SHGetFolderPathW
000064BC	000064BC	0000	CommandLineToArgvW
000064D2	000064D2	0000	ShellExecuteA
000064E2	000064E2	0000	Shell_NotifyIconW
000064F6	000064F6	0000	ShellExecuteW

Using capa, how man anti-VM instructions were identified in malbuster_1?

Capa is a tool that analyses a binary and recognises its behaviours:

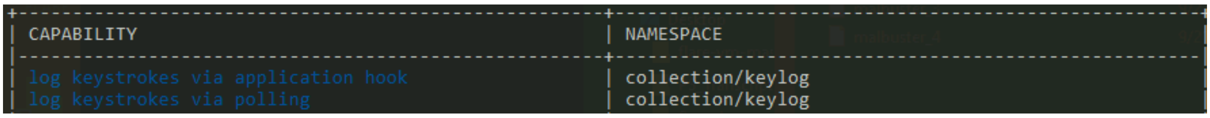


```
C:\Users\Administrator\Desktop
λ capa.exe C:\Users\Administrator\Desktop\Samples\malbuster_1
loading : 100%|
matching: 100%|
+-------------------------+-----------------------------------------------------------------------------------+
| md5                     | 4348da65e4aeae6472c7f97d6dd8ad8f                                                  |
| sha1                    | 8bf60eea83c34ec9de2359219978b8805f2629e3                                          |
| sha256                  | 000415d1c7a7a838ba2ef00874e352e8b43a57e2f98539b5908803056f883176                  |
| os                      | windows                                                                           |
| format                  | pe                                                                                |
| arch                    | i386                                                                              |
| path                    | C:\Users\Administrator\Desktop\Samples\malbuster_1                                |
+-------------------------+-----------------------------------------------------------------------------------+

+-------------------------+-----------------------------------------------------------------------------------+
| ATT&CK Tactic           | ATT&CK Technique                                                                  |
|-------------------------+-----------------------------------------------------------------------------------|
| DEFENSE EVASION         | Obfuscated Files or Information T1027                                             |
|                         | Virtualization/Sandbox Evasion::System Checks T1497.001                           |
+-------------------------+-----------------------------------------------------------------------------------+

+------------------------------+------------------------------------------------------------------------------+
| MBC Objective                | MBC Behavior                                                                 |
|------------------------------+------------------------------------------------------------------------------|
| ANTI-BEHAVIORAL ANALYSIS     | Virtual Machine Detection [B0009]                                            |
| COMMUNICATION                | HTTP Communication::Read Header [C0002.014]                                  |
| CRYPTOGRAPHY                 | Encrypt Data::RC4 [C0027.009]                                                |
|                              | Generate Pseudo-random Sequence [C0021]                                      |
|                              | Generate Pseudo-random Sequence::Mersenne Twister [C0021.005]                |
|                              | Generate Pseudo-random Sequence::RC4 PRGA [C0021.004]                        |
| DATA                         | Checksum::CRC32 [C0032.001]                                                  |
|                              | Encode Data::XOR [C0026.002]                                                 |
| DEFENSE EVASION              | Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]     |
| DISCOVERY                    | Code Discovery::Enumerate PE Sections [B0046.001]                            |
+------------------------------+------------------------------------------------------------------------------+

+------------------------------------------------------+------------------------------------------------------+
| CAPABILITY                                           | NAMESPACE                                            |
|------------------------------------------------------+------------------------------------------------------|
| reference anti-VM strings                            | anti-analysis/anti-vm/vm-detection                   |
| check HTTP status code (2 matches)                   | communication/http/client                            |
| hash data with CRC32                                 | data-manipulation/checksum/crc32                     |
| encode data using XOR (10 matches)                   | data-manipulation/encoding/xor                       |
| encrypt data using RC4 PRGA (3 matches)              | data-manipulation/encryption/rc4                     |
| generate random numbers using the Delphi LCG         | data-manipulation/prng/lcg                           |
| generate random numbers using a Mersenne Twister     | data-manipulation/prng/mersenne                      |
| enumerate PE sections (2 matches)                    | load-code/pe                                         |
| resolve function by parsing PE exports               | load-code/pe                                         |
+------------------------------------------------------+------------------------------------------------------+
```

The answer is 3.

Using capa, which binary can log keystrokes?

After running capa against all 4 files, you can determine that malbuster_3 is capable of logging keystrokes as seen here:



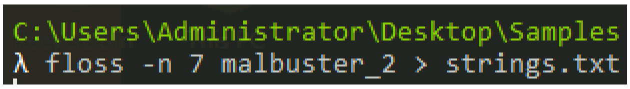
```
+-------------------------------------------------------+------------------------------------------------------+
| CAPABILITY                                            | NAMESPACE                                            |
|-------------------------------------------------------+------------------------------------------------------|
| log keystrokes via application hook                   | collection/keylog                                    |
| log keystrokes via polling                            | collection/keylog                                    |
```

Using capa, what is the MITRE ID of the discovery technique used by malbuster_4?

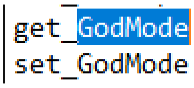
The technique is T1083.

Which binary contains the string GodMode?

We can use a tool called Floss (FLARE Obfuscated String Solver) which is essentially an advanced version of the strings command. In the following example I am setting the minimum length of the string to 7 using the -n switch and saving the output to strings.txt:

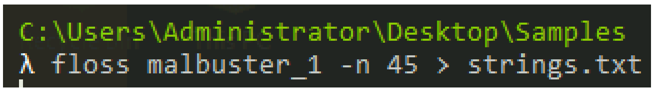


```
C:\Users\Administrator\Desktop\Samples
λ floss -n 7 malbuster_2 > strings.txt
```

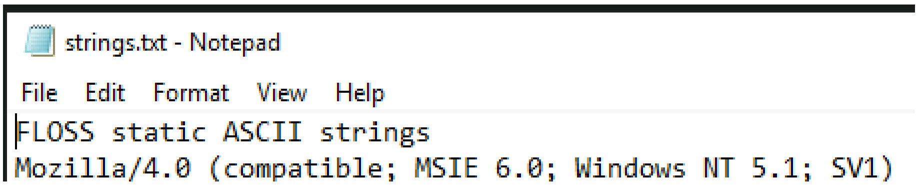


```
get_GodMode
set_GodMode
```

Which binary contains the string Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)?



```
C:\Users\Administrator\Desktop\Samples
λ floss malbuster_1 -n 45 > strings.txt
```



```
FLOSS static ASCII strings
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

This brings the room to a close, I really enjoyed it all and it stays true to being a room for beginners. If you need any help, feel free to contact me.