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

We could open this up in Detect it easy and it will tell us but how can we do this without DiE?

If we take a look at the COFF header explanation we can see we can find if this is a 32bit or 64bit file.

COFF File Header (Object and Image)

At the beginning of an object file, or immediately after the signature of an image file, is a standard COFF file header in the following format. Note that the Windows loader limits the number of sections to 96.

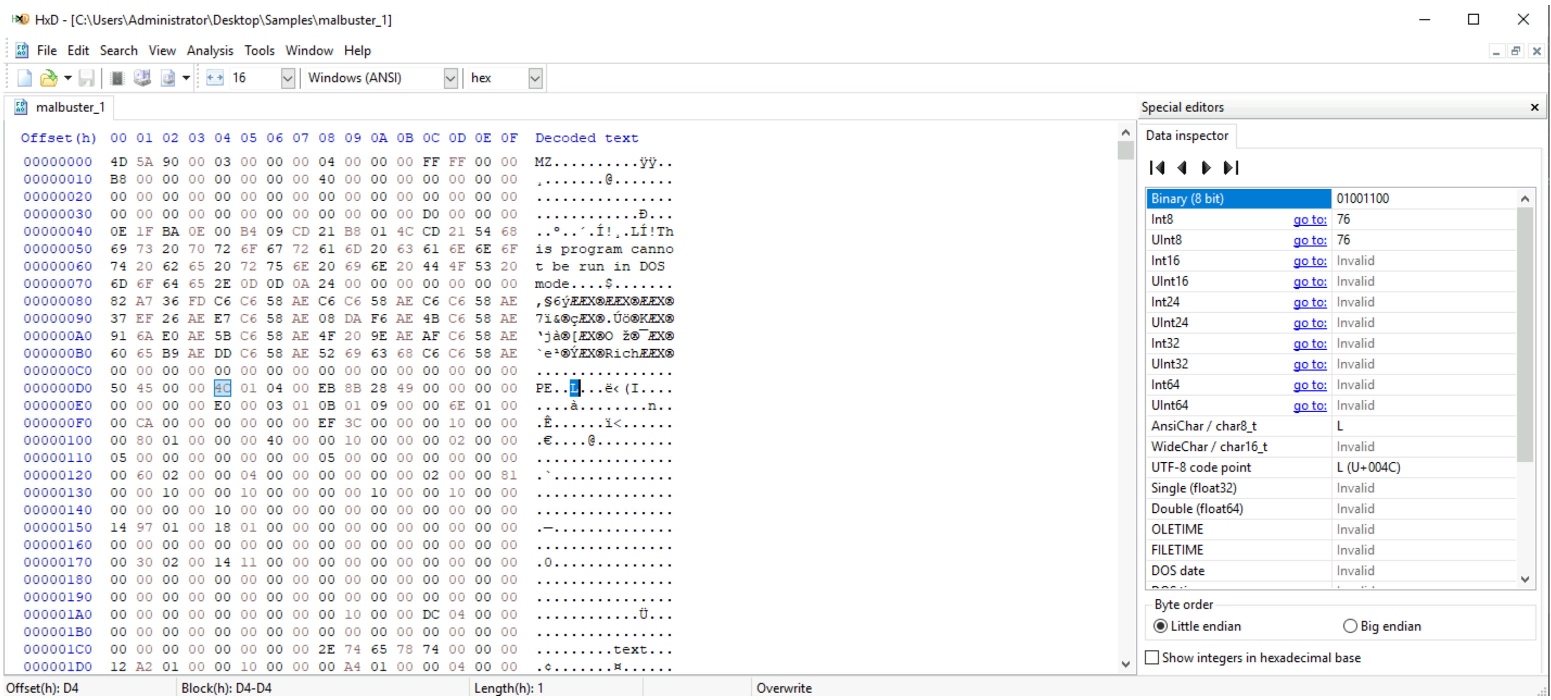
Offset	Size	Field	Description
0	2	Machine	The number that identifies the type of target machine. For more information, see Machine Types.
2	2	NumberOfSections	The number of sections. This indicates the size of the section table, which immediately follows the headers.
4	4	TimeDateStamp	The low 32 bits of the number of seconds since 00:00 January 1, 1970 (a C run-time time_t value), which indicates when the file was created.
8	4	PointerToSymbolTable	The file offset of the COFF symbol table, or zero if no COFF symbol table is present. This value should be zero for an image because COFF debugging information is deprecated.
12	4	NumberOfSymbols	The number of entries in the symbol table. This data can be used to locate the string table, which immediately follows the symbol table. This value should be zero for an image because COFF debugging information is deprecated.
16	2	SizeOfOptionalHeader	The size of the optional header, which is required for executable files but not for object files. This value should be zero for an object file. For a description of the header format, see Optional Header (Image Only).
18	2	Characteristics	The flags that indicate the attributes of the file. For specific flag values, see Characteristics.

Machine Types

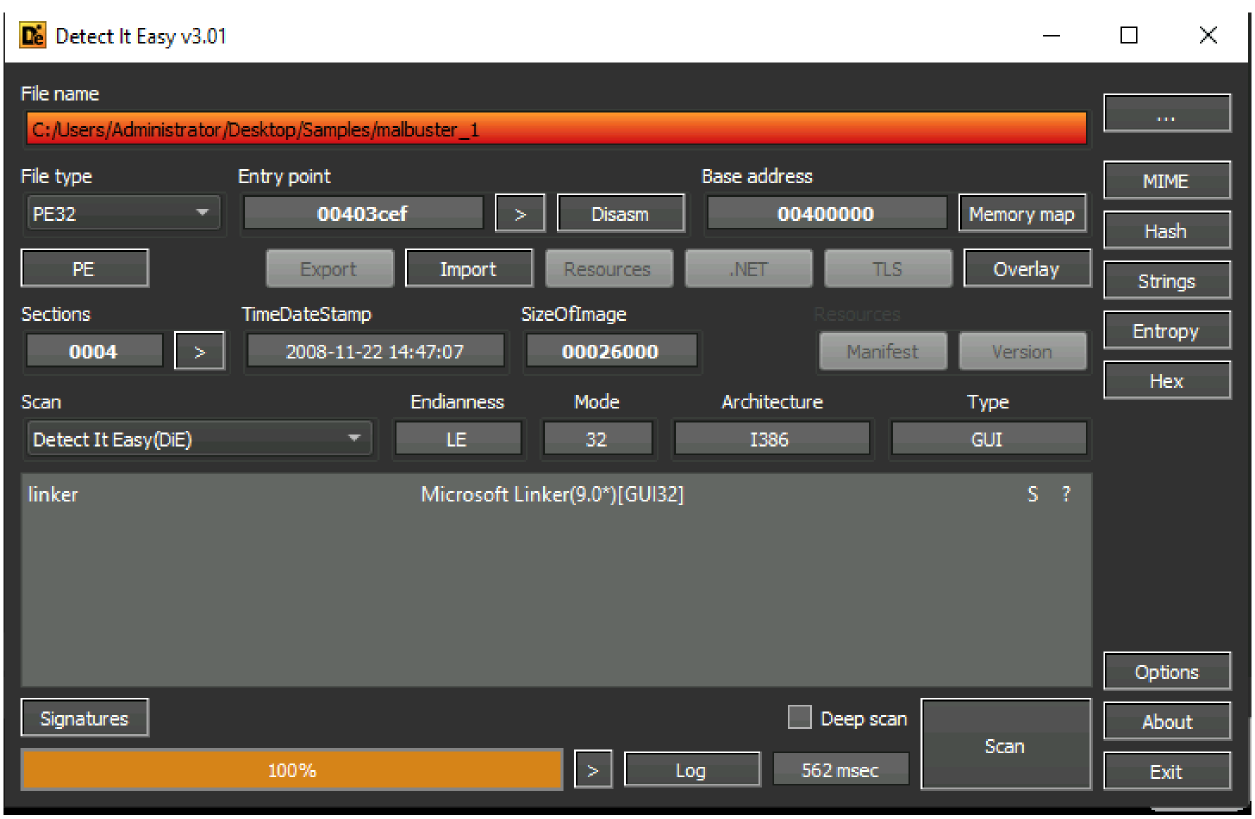
The Machine field has one of the following values, which specify the CPU type. An image file can be run only on the specified machine or on a system that emulates the specified machine.

Constant	Value	Description
IMAGE_FILE_MACHINE_UNKNOWN	0x0	The content of this field is assumed to be applicable to any machine type
IMAGE_FILE_MACHINE_ALPHA	0x184	Alpha AXP, 32-bit address space
IMAGE_FILE_MACHINE_ALPHA64	0x284	Alpha 64, 64-bit address space
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	0x8664	x64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian
IMAGE_FILE_MACHINE_ARM64	0xaa64	ARM64 little endian
IMAGE_FILE_MACHINE_ARMNT	0x1c4	ARM Thumb-2 little endian
IMAGE_FILE_MACHINE_AXP64	0x284	AXP 64 (Same as Alpha 64)
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 or later processors and compatible processors

So we take a look at the COFF header in HxD and can see 0x14c indicating this is a 32bit file.

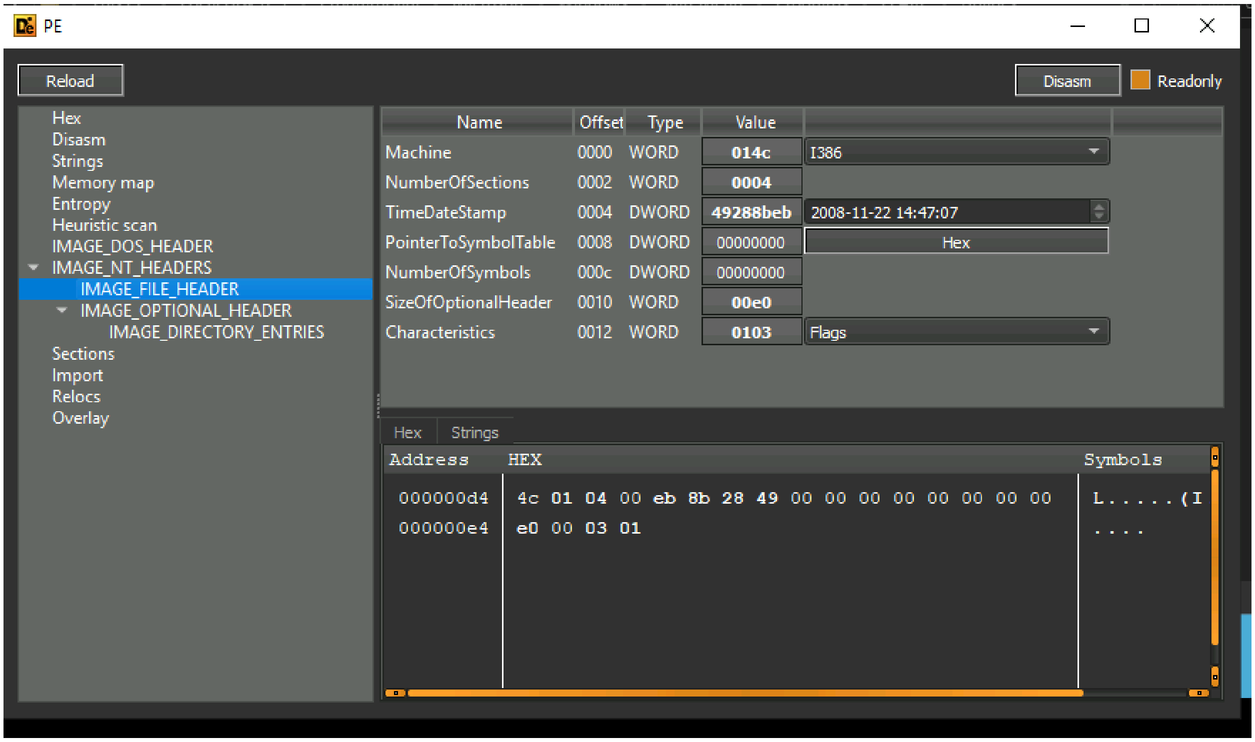


How to tell in detect it easy?



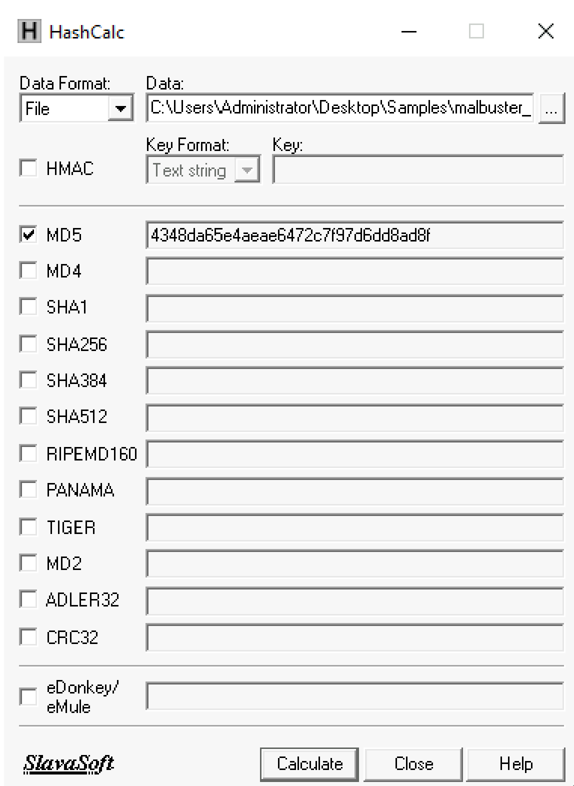
As we can see it is clearly defined in the “mode” section.

We can also check out the file header.



What is the MD5 hash of malbuster_1?

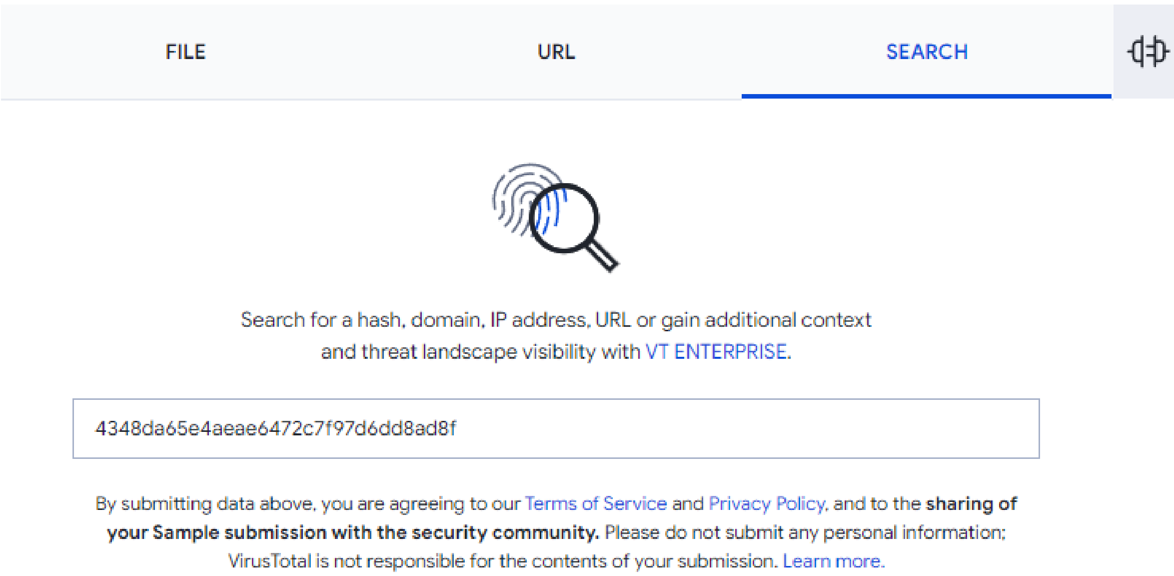
We can do this using HashCalc which comes as part of the flareVM.

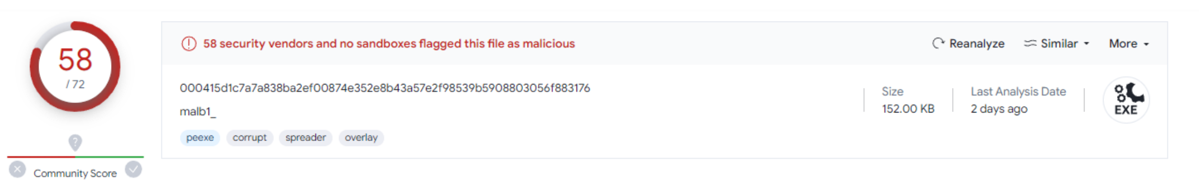


We can also do this from powershell using md5sum:

```
PS C:\Users\Administrator\Desktop\Samples > md5sum .\malbuster_1
\4348da65e4aeae6472c7f97d6dd8ad8f *.\\malbuster_1
```

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





We can clearly see that there is 58 detections.

However, this has updated since the activity was created so I had to bruteforce the answer by counting down from 58. (51 is the number).

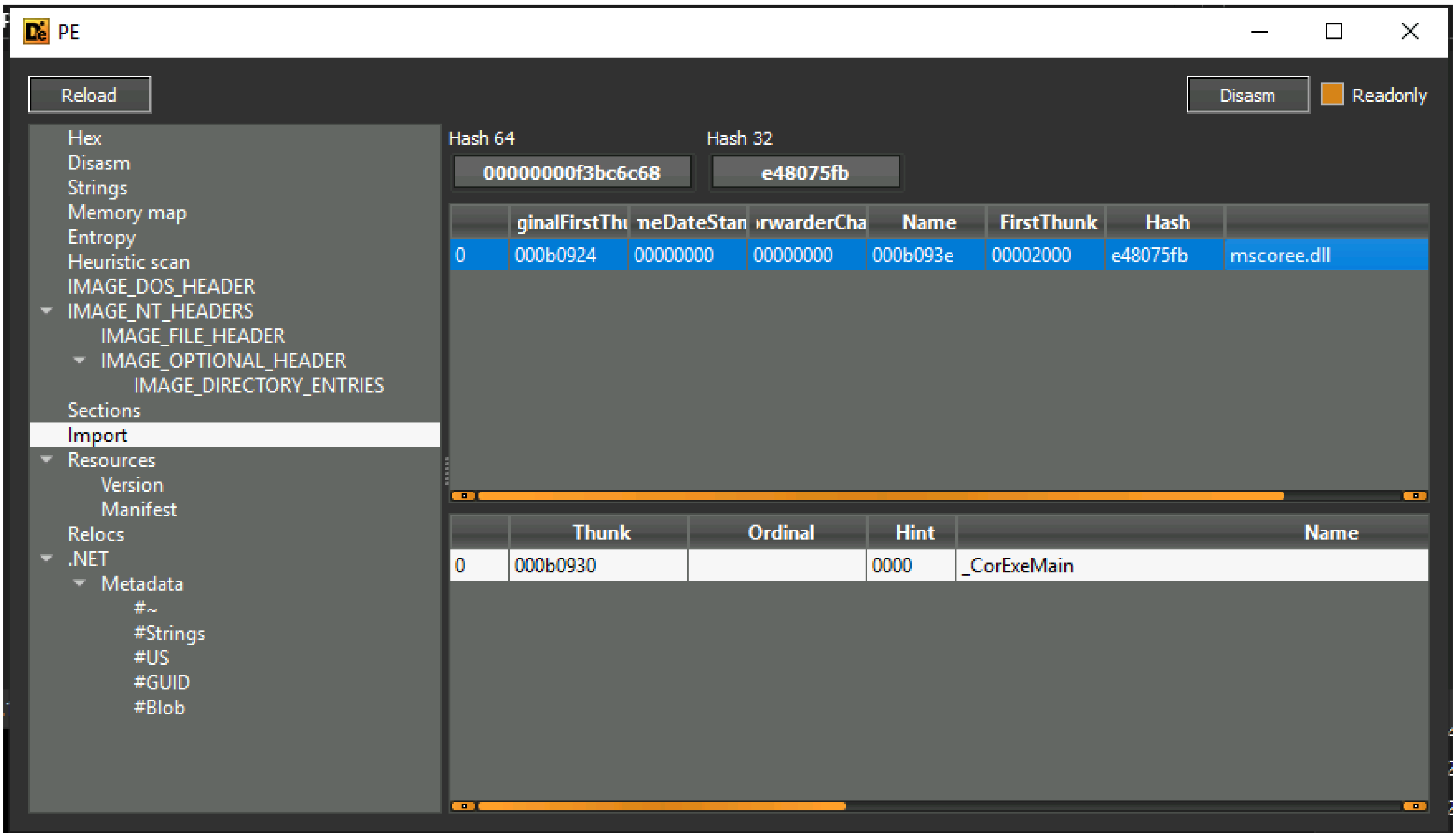
Based on VirusTotal detection, what is the malware signature of malbuster_2 according to Avira?

So we upload the file to virus total or just get the hash of it and search for it on virustotal.

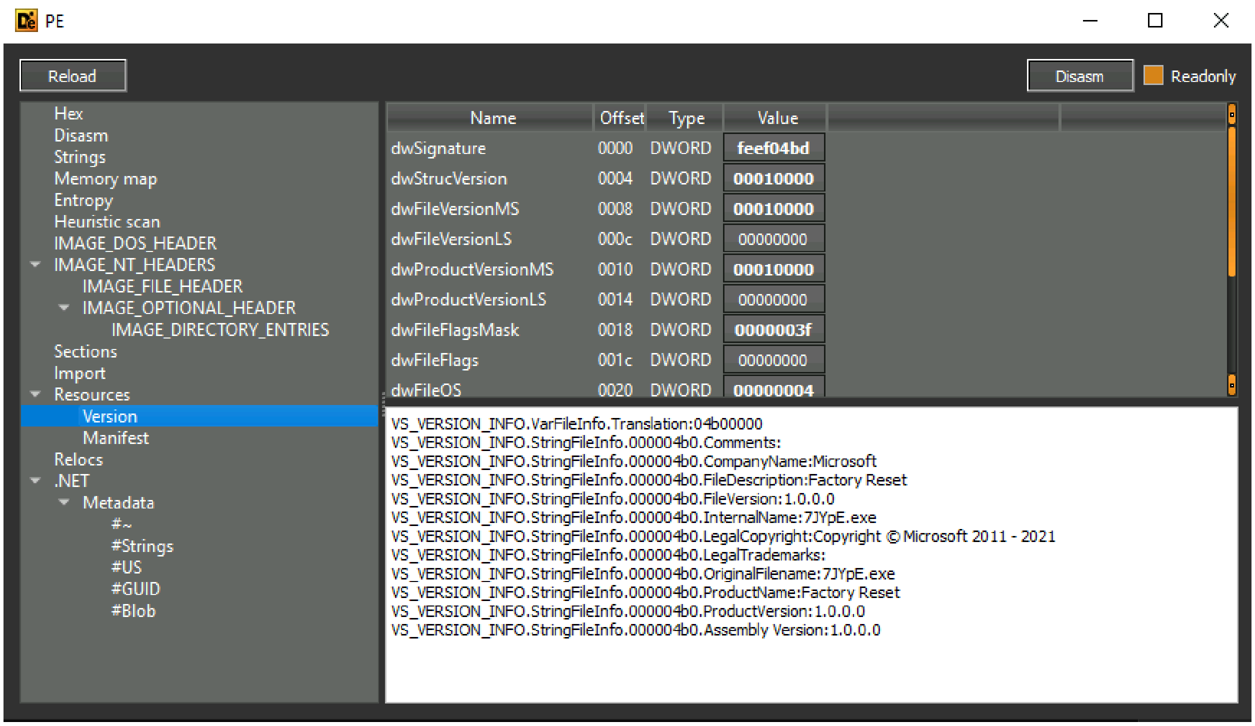
It is also the case that it has changed and none of the writeups contain the right answer.

malbuster_2 imports the function _CorExeMain. From which DLL file does it import this function?

Taking a look at the import table we can see that it comes from the mscoree.dll.

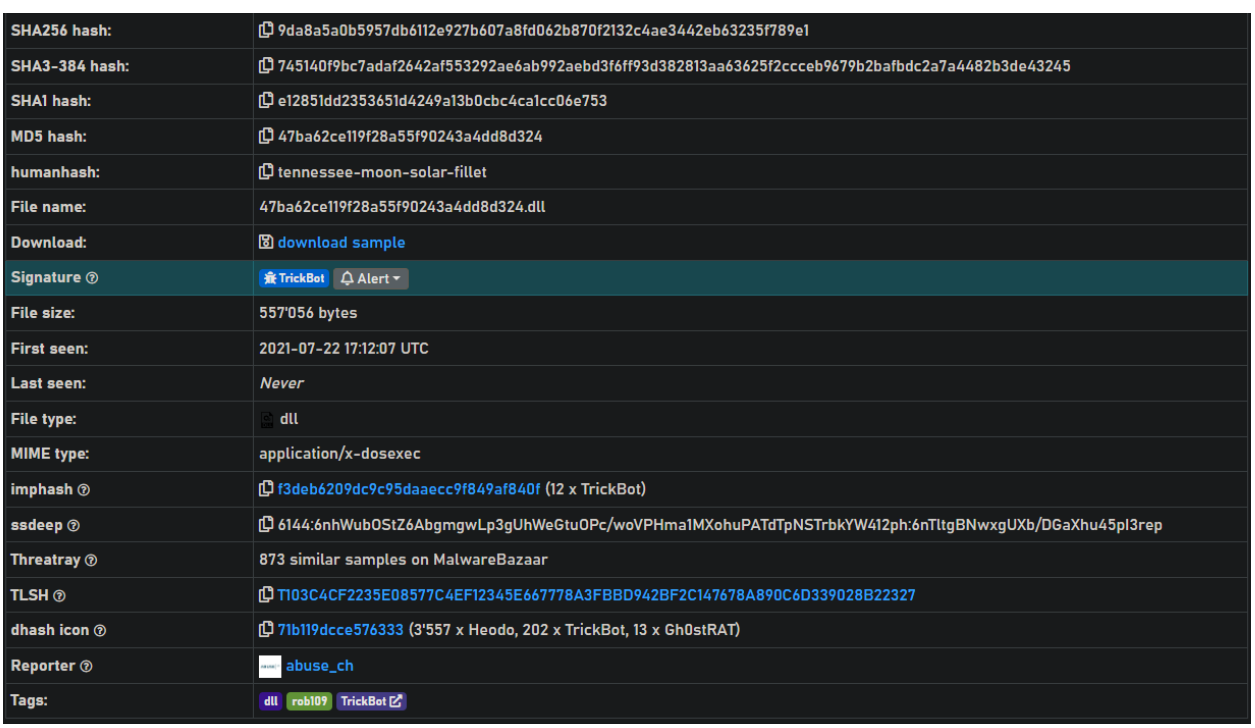


Based on the VS_VERSION_INFO header, what is the original name of malbuster_2?



We can see the original file name is 7JYpE.exe.

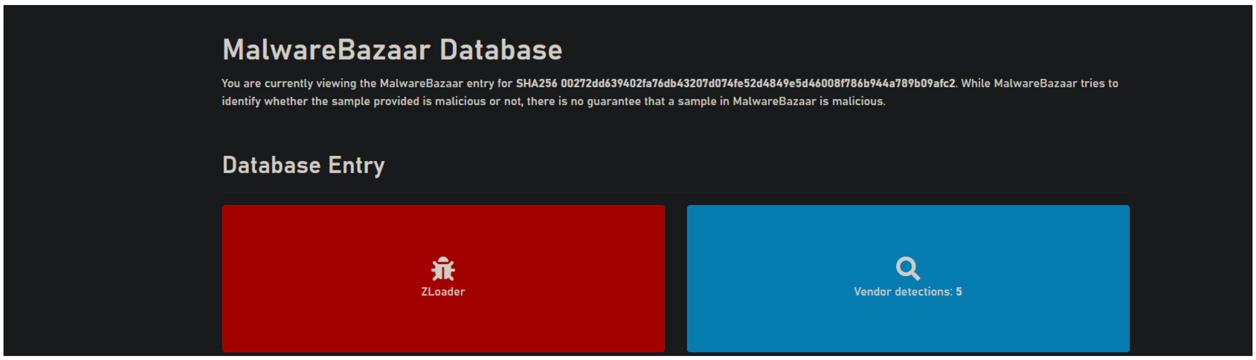
Using the hash of malbuster_3, what is its malware signature based on abuse.ch?



SHA256 hash:	9da8a5a0b5957db6112e927b607a8fd062b870f2132c4ae3442eb63235f789e1
SHA3-384 hash:	745140f9bc7adaf2642af553292ae6ab992aebd3f6ff93d382813aa63625f2ccceb9679b2bafbdc2a7a4482b3de43245
SHA1 hash:	e12851dd2353651d4249a13b0cbc4ca1cc06e753
MD5 hash:	47ba62ce119f28a55f90243a4dd8d324
humanhash:	tennessee-moon-solar-fillet
File name:	47ba62ce119f28a55f90243a4dd8d324.dll
Download:	download sample
Signature	TrickBot Alert
File size:	557'056 bytes
First seen:	2021-07-22 17:12:07 UTC
Last seen:	*Never*
File type:	dll
MIME type:	application/x-dosexec
imphash	f3deb6209dc9c95daaecc9f849af840f (12 x TrickBot)
ssdeep	6144:6nhWubOStZ6AbgmgwLp3gUhWeGtuOPc/woVPHma1MXohuPATdTpNSTrbkYW412ph:6nTltgBNwxgUXb/DGaXhu45pl3rep
Threatray	873 similar samples on MalwareBazaar
TLSH	T103C4CF2235E08577C4EF12345E667778A3FBBD942BF2C147678A890C6D339028B22327
dhash icon	71b119dcce576333 (3'557 x Heodo, 202 x TrickBot, 13 x Gh0stRAT)
Reporter	abuse_ch
Tags:	dll rob109 TrickBot

We can see it is trickbot.

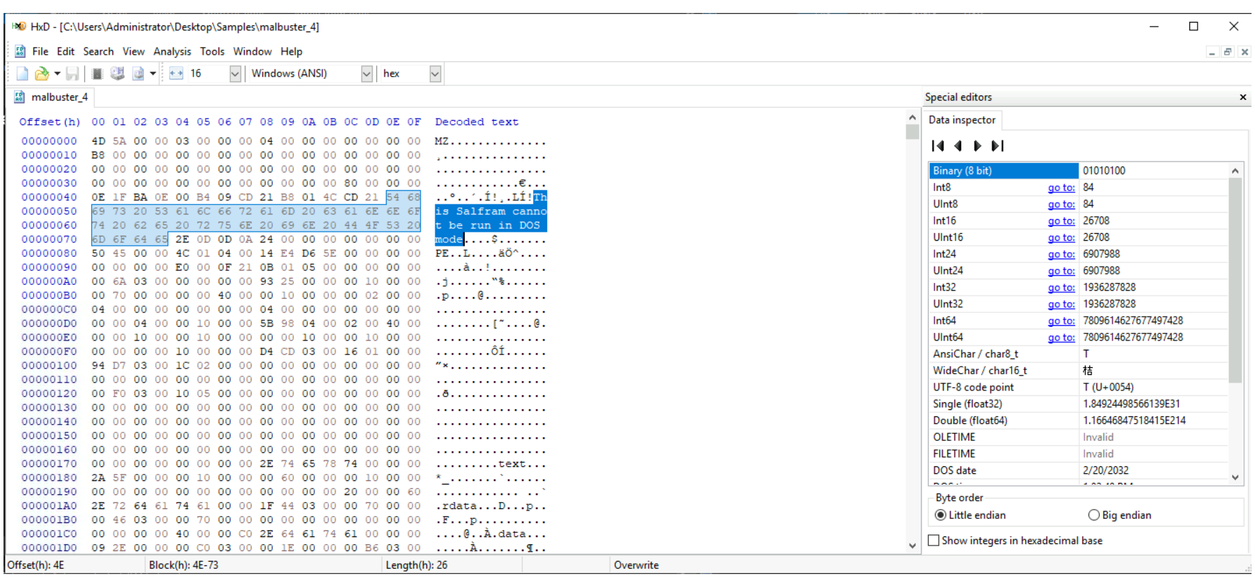
Using the hash of malbuster_4, what is its malware signature based on abuse.ch?



We generate the md5 hash as we have done in previous steps. Go to bazaar.abuse.ch. Type md5:the_md5_hash. Click on the hash and it tells us it is Zloader.

What is the message found in the DOS_STUB of malbuster_4?

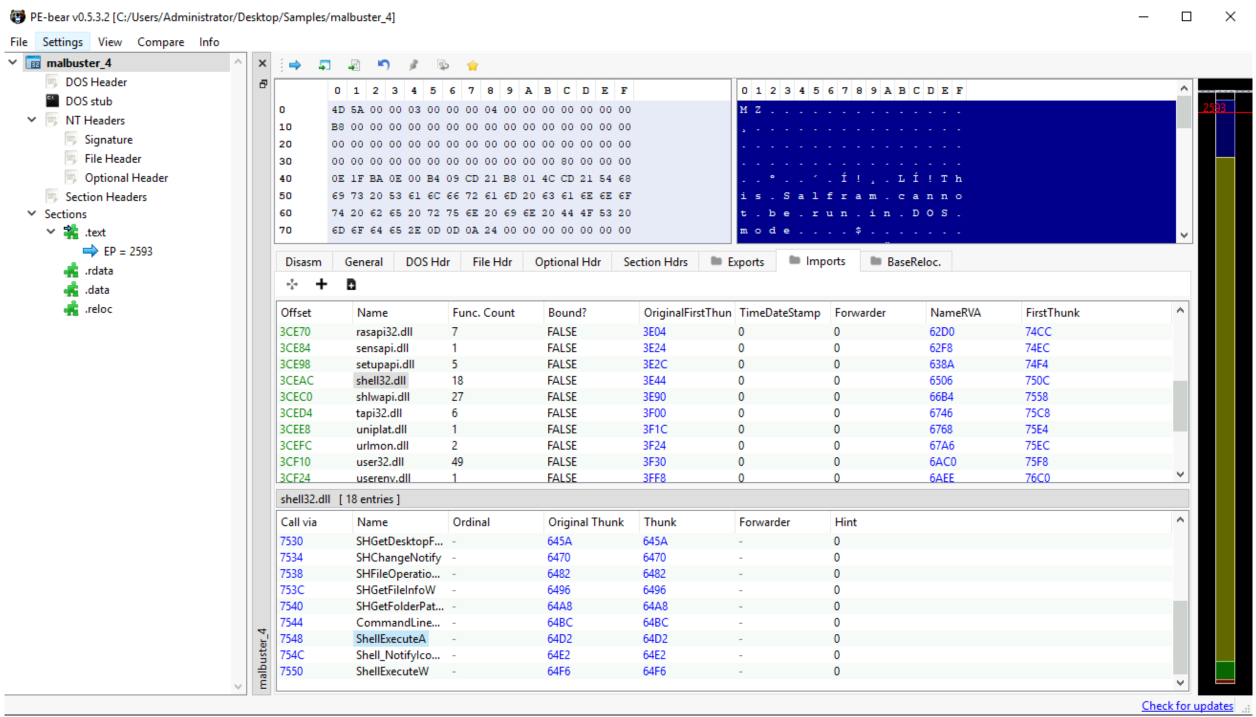
We could look at this in detect it easy or any hex editor. I chose to look at it in a hex editor.



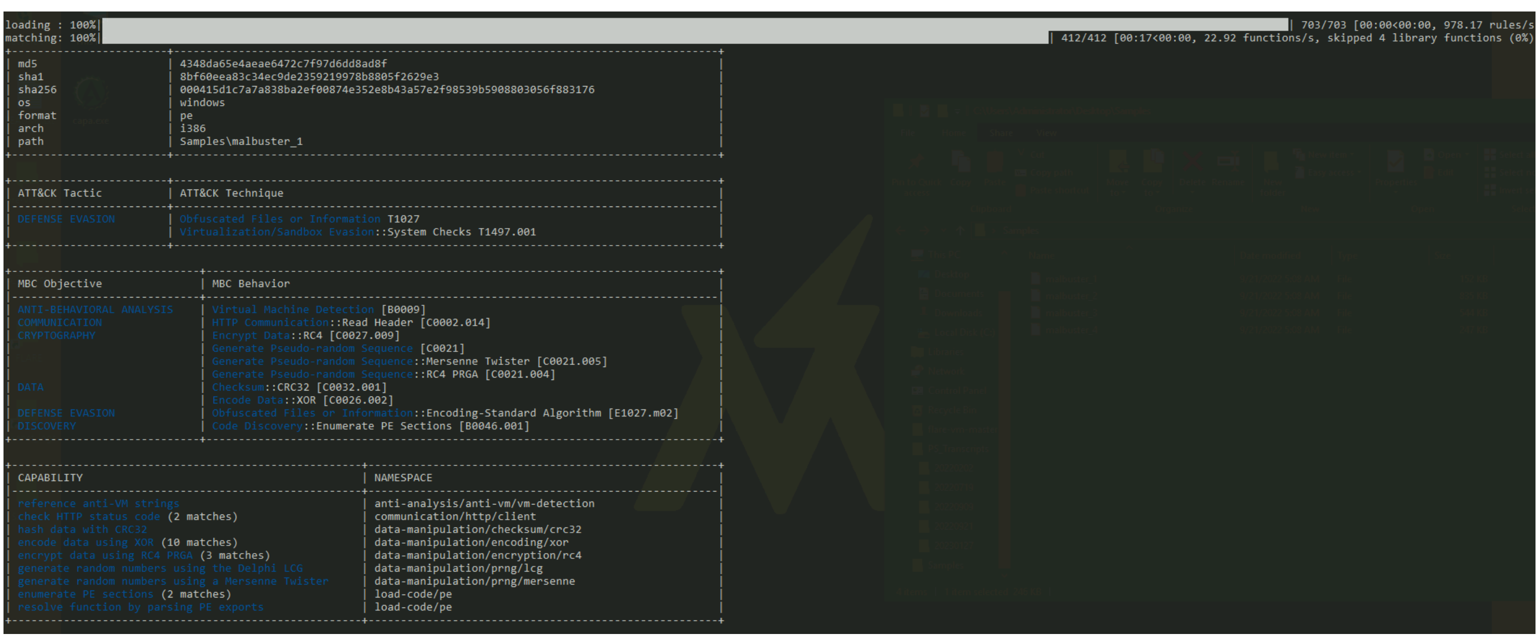
We see the string “This Salfram cannot be run in DOS mode.”

malbuster_4 imports the function ShellExecuteA. From which DLL file does it import this function?

I loaded this in pe bear as I think it’s useful to get familiar with a multitude of tools. As we have previously used die I chose pe bear. Going into the Imports tab and browsing through each dll we can see that the function ShellExecuteA is imported from the shell32.dll.



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



The three Generate Pseudo-random Sequence are used as an anti-vm technique. This technique involves exploiting how a virtual machine generates random numbers compared to a non virtualized machine. I'm sure chatGPT can explain it much better than me:

Using a pseudo-random sequence as an anti-VM (Virtual Machine) technique is a way to detect or evade virtualized environments, such as those used by malware analysts and security researchers, to analyze the behavior of malicious software. The idea is to leverage the differences in how virtual machines and physical machines handle random number generation. In the context of anti-VM techniques, you are trying to distinguish between a real physical machine and a virtualized one. Here's how you could use the mentioned pseudo-random sequence generators for this purpose:

Mersenne Twister:

- The Mersenne Twister is a widely used pseudo-random number generator. To use it as an anti-VM technique, you can generate a sequence of random numbers on the host system and compare it to the sequence generated within the potentially virtualized environment.
- If the two sequences are significantly different, it might indicate that the application is running in a virtualized environment. VMs often have predictable patterns or less randomness in their random number generation.

Generating a poc from chatGPT resulted in:

```
import random
import time

def generate_reference_sequence(length):
    random.seed(time.time())  # Seed with current time
    return [random.randint(0, 1) for _ in range(length)]

def detect_virtualization():
    sequence_length = 1000  # Length of the reference and generated
sequences
    reference_sequence = generate_reference_sequence(sequence_length)
    generated_sequence = [random.randint(0, 1) for _ in
range(sequence_length)]
```

```python
    # Compare the generated sequence with the reference sequence
    similarity_score = sum(a == b for a, b in zip(reference_sequence,
generated_sequence))

    # You can adjust the threshold to define what similarity score
indicates a virtualized environment
    threshold = 500  # Adjust this value as needed

    if similarity_score < threshold:
        return "Likely running in a virtualized environment"
    else:
        return "Likely running on a physical machine"

if __name__ == "__main__":
    result = detect_virtualization()
    print(result)
```

This code generates two sequences of random binary values, one using the system time as a seed (reference sequence) and the other using the Mersenne Twister (generated sequence). It then compares the two sequences and calculates a similarity score. If the similarity score is below a certain threshold, it suggests that the environment may be virtualized.

There are many other ways to do this. However, let's answer the question. Which the answer is 3.

Using capa, which binary can log keystrokes?

Essentially we just have to launch capa for each binary.
Malbuster_2 output:

```
λ capa Samples\malbuster_2
loading : 100%|
matching: 100%|                                                                 | 534/534 [00:06<00:00,
+------------------------+------------------------------------------------------------------------+
| md5                    | 1d7ebed1baece67a31ce0a17a0320cb2                                       |
| sha1                   | f0b75348be8941ee8b1ce41bfa70dbec406b5cd4                               |
| sha256                 | ace3a5e5849c1c00760dfe67add397775f5946333357f5f8dee25cd4363e36b6       |
| os                     | windows                                                                |
| format                 | pe                                                                     |
| arch                   | i386                                                                   |
| path                   | Samples\malbuster_2                                                    |
+------------------------+------------------------------------------------------------------------+

+------------------------+------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                       |
|------------------------+------------------------------------------------------------------------|
| DEFENSE EVASION        | Reflective Code Loading T1620                                          |
+------------------------+------------------------------------------------------------------------+

+-----------------------------+-------------------------------------------------------------------+
| MBC Objective               | MBC Behavior                                                      |
|-----------------------------+-------------------------------------------------------------------|
| CRYPTOGRAPHY                | Generate Pseudo-random Sequence::Use API [C0021.003]              |
+-----------------------------+-------------------------------------------------------------------+

+--------------------------------------------------+----------------------------------------------+
| CAPABILITY                                       | NAMESPACE                                    |
|--------------------------------------------------+----------------------------------------------|
| generate random numbers in .NET (16 matches)     | data-manipulation/prng                       |
| load .NET assembly                               | load-code/dotnet                             |
| compiled to the .NET platform                    | runtime/dotnet                               |
+--------------------------------------------------+----------------------------------------------+
```

As we can see there is no output for logging keystrokes. However, for malbuster_3 we can see

```
+-------------------------------------------------------+-------------------------------------------------------+
| CAPABILITY                                            | NAMESPACE                                             |
|-------------------------------------------------------+-------------------------------------------------------|
| log keystrokes via application hook                   | collection/keylog                                     |
| log keystrokes via polling                            | collection/keylog                                     |
| encode data using XOR                                 | data-manipulation/encoding/xor                        |
| encrypt data using RC4 KSA                            | data-manipulation/encryption/rc4                      |
| encrypt data using RC4 PRGA                           | data-manipulation/encryption/rc4                      |
| contain a resource (.rsrc) section                    | executable/pe/section/rsrc                            |
| extract resource via kernel32 functions (5 matches)   | executable/resource                                   |
| get common file path                                  | host-interaction/file-system                          |
| delete file                                           | host-interaction/file-system/delete                   |
| move file                                             | host-interaction/file-system/move                     |
| read .ini file                                        | host-interaction/file-system/read                     |
| write file on Windows                                 | host-interaction/file-system/write                    |
| get graphical window text (4 matches)                 | host-interaction/gui/window/get-text                  |
| create process on Windows                             | host-interaction/process/create                       |
| create or open registry key                           | host-interaction/registry                             |
| query or enumerate registry value (2 matches)         | host-interaction/registry                             |
| link function at runtime on Windows                   | linking/runtime-linking                               |
| resolve function by parsing PE exports                | load-code/pe                                          |
+-------------------------------------------------------+-------------------------------------------------------+
```

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

```
+------------------------+--------------------------------------------
| ATT&CK Tactic          | ATT&CK Technique
|------------------------+--------------------------------------------
| DISCOVERY              | File and Directory Discovery T1083
+------------------------+--------------------------------------------
```

Running capa we can see T1083.

Which binary contains the string GodMode?

```
C:\Users\Administrator\Desktop\Samples
λ strings malbuster_1 | grep "God"

C:\Users\Administrator\Desktop\Samples
λ strings malbuster_2 | grep "God"
get_GodMode
set_GodMode
GodMode
```

Running strings on each binary and grepping for God tells us the answer is malbuster_2.

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

```
C:\Users\Administrator\Desktop\Samples
λ strings malbuster_1 | grep "Mozilla"
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```