Basic Static Analysis

In this section, we are going to analyze two binaries provided in the PMAT-LAB.

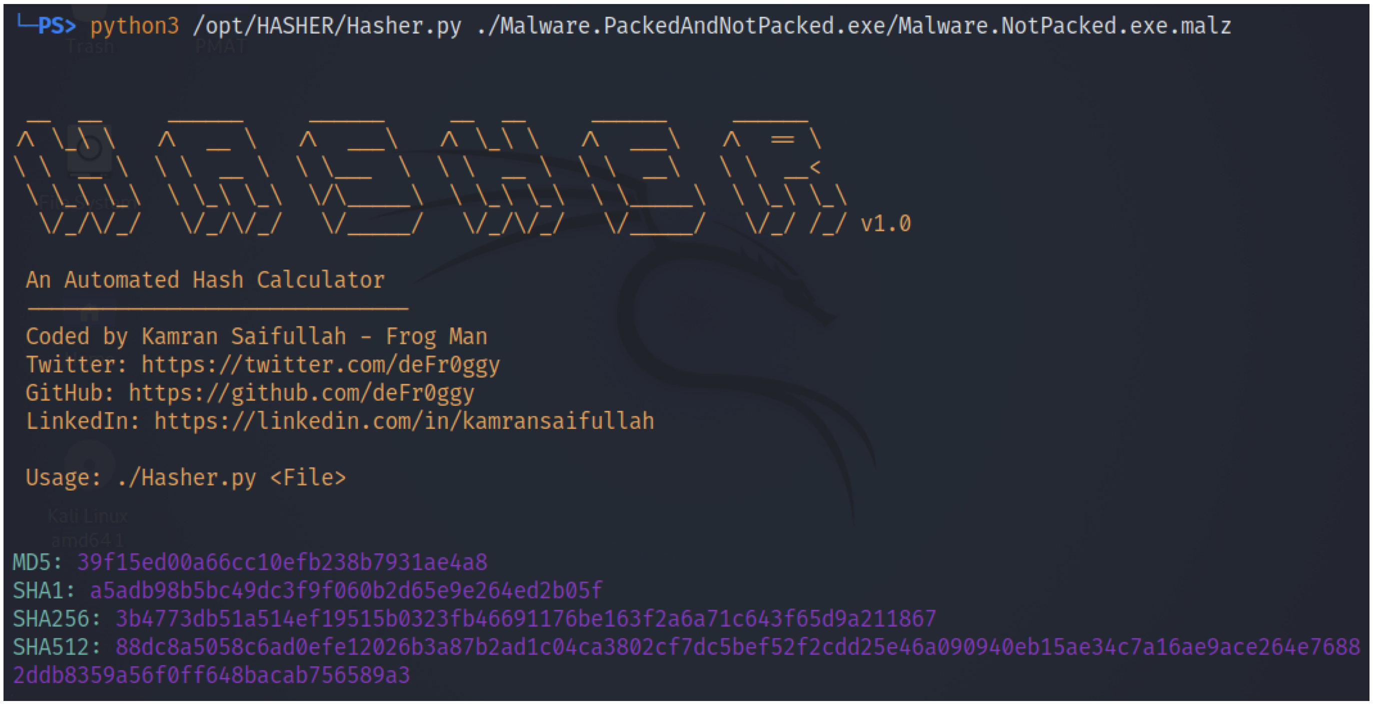
> https://github.com/HuskyHacks/PMAT-labs/tree/main/labs/1-1.BasicStaticAnalysis

Packed & Not Packed Executables

> https://github.com/HuskyHacks/PMAT-labs/tree/main/labs/1-1.BasicStaticAnalysis/Malware.PackedAndNotPacked.exe.malz

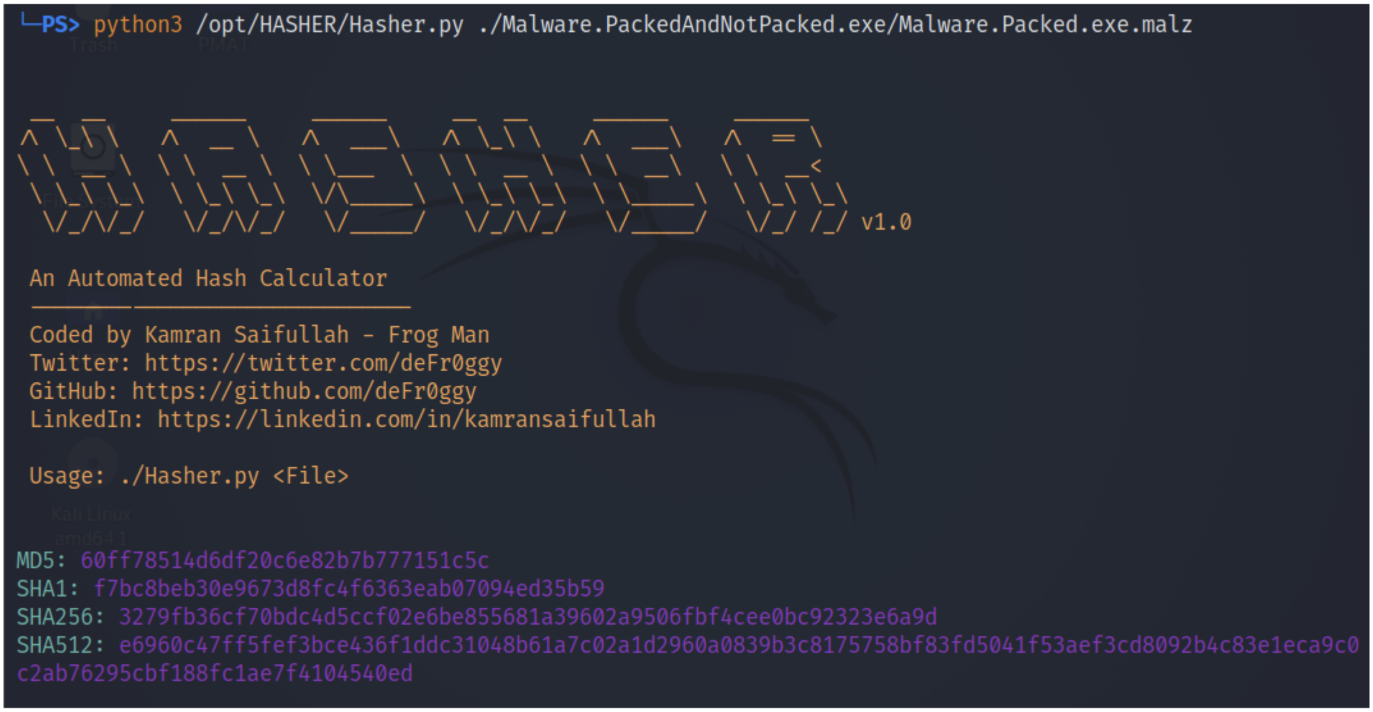
At first, we have a binary and we need to perform a basic static analysis. There are plenty of tools to get started with basic statis analysis but the first thing that is required to be done is to calculate the hashes of the sample.

a. NotPacked Executable



```
└─PS> python3 /opt/HASHER/Hasher.py ./Malware.PackedAndNotPacked.exe/Malware.NotPacked.exe.malz


 __  __     ______     ______     __  __     ______     ______
/\ \_\ \   /\  __ \   /\  ___\   /\ \_\ \   /\  ___\   /\  == \
\ \  __ \  \ \  __ \  \ \___  \  \ \  __ \  \ \  __\   \ \  __<
 \ \_\ \_\  \ \_\ \_\  \/\_____\  \ \_\ \_\  \ \_____\  \ \_\ \_\
  \/_/\/_/   \/_/\/_/   \/_____/   \/_/\/_/   \/_____/   \/_/ /_/ v1.0

 An Automated Hash Calculator
 ______________________________
 Coded by Kamran Saifullah - Frog Man
 Twitter: https://twitter.com/deFr0ggy
 GitHub: https://github.com/deFr0ggy
 LinkedIn: https://linkedin.com/in/kamransaifullah

 Usage: ./Hasher.py <File>

MD5: 39f15ed00a66cc10efb238b7931ae4a8
SHA1: a5adb98b5bc49dc3f9f060b2d65e9e264ed2b05f
SHA256: 3b4773db51a514ef19515b0323fb46691176be163f2a6a71c643f65d9a211867
SHA512: 88dc8a5058c6ad0efe12026b3a87b2ad1c04ca3802cf7dc5bef52f2cdd25e46a090940eb15ae34c7a16ae9ace264e7688
2ddb8359a56f0ff648bacab756589a3
```

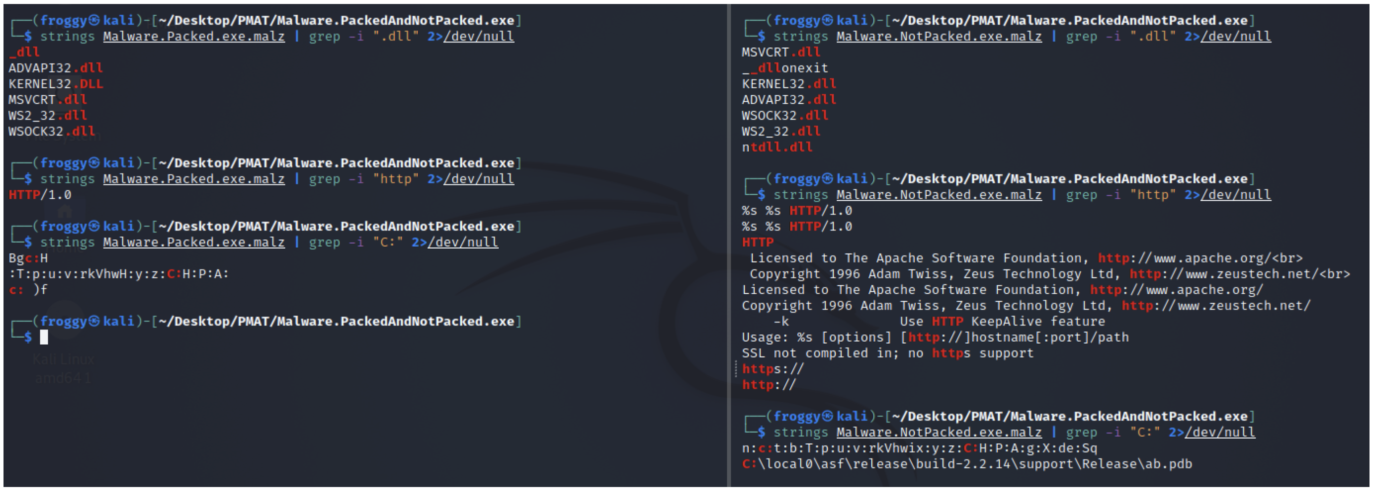
b. Packed Executable



```
└─PS> python3 /opt/HASHER/Hasher.py ./Malware.PackedAndNotPacked.exe/Malware.Packed.exe.malz


 __  __     ______     ______     __  __     ______     ______
/\ \_\ \   /\  __ \   /\  ___\   /\ \_\ \   /\  ___\   /\  == \
\ \  __ \  \ \  __ \  \ \___  \  \ \  __ \  \ \  __\   \ \  __<
 \ \_\ \_\  \ \_\ \_\  \/\_____\  \ \_\ \_\  \ \_____\  \ \_\ \_\
  \/_/\/_/   \/_/\/_/   \/_____/   \/_/\/_/   \/_____/   \/_/ /_/ v1.0

 An Automated Hash Calculator
 ______________________________
 Coded by Kamran Saifullah - Frog Man
 Twitter: https://twitter.com/deFr0ggy
 GitHub: https://github.com/deFr0ggy
 LinkedIn: https://linkedin.com/in/kamransaifullah

 Usage: ./Hasher.py <File>

MD5: 60ff78514d6df20c6e82b7b777151c5c
SHA1: f7bc8beb30e9673d8fc4f6363eab07094ed35b59
SHA256: 3279fb36cf70bdc4d5ccf02e6be855681a39602a9506fbf4cee0bc92323e6a9d
SHA512: e6960c47ff5fef3bce436f1ddc31048b61a7c02a1d2960a0839b3c8175758bf83fd5041f53aef3cd8092b4c83e1eca9c0
c2ab76295cbf188fc1ae7f4104540ed
```

Analysis

If we try strings on both executables, we can see the difference i.e. we will be able to see more data in the unpacked version of the executable than the packed version.



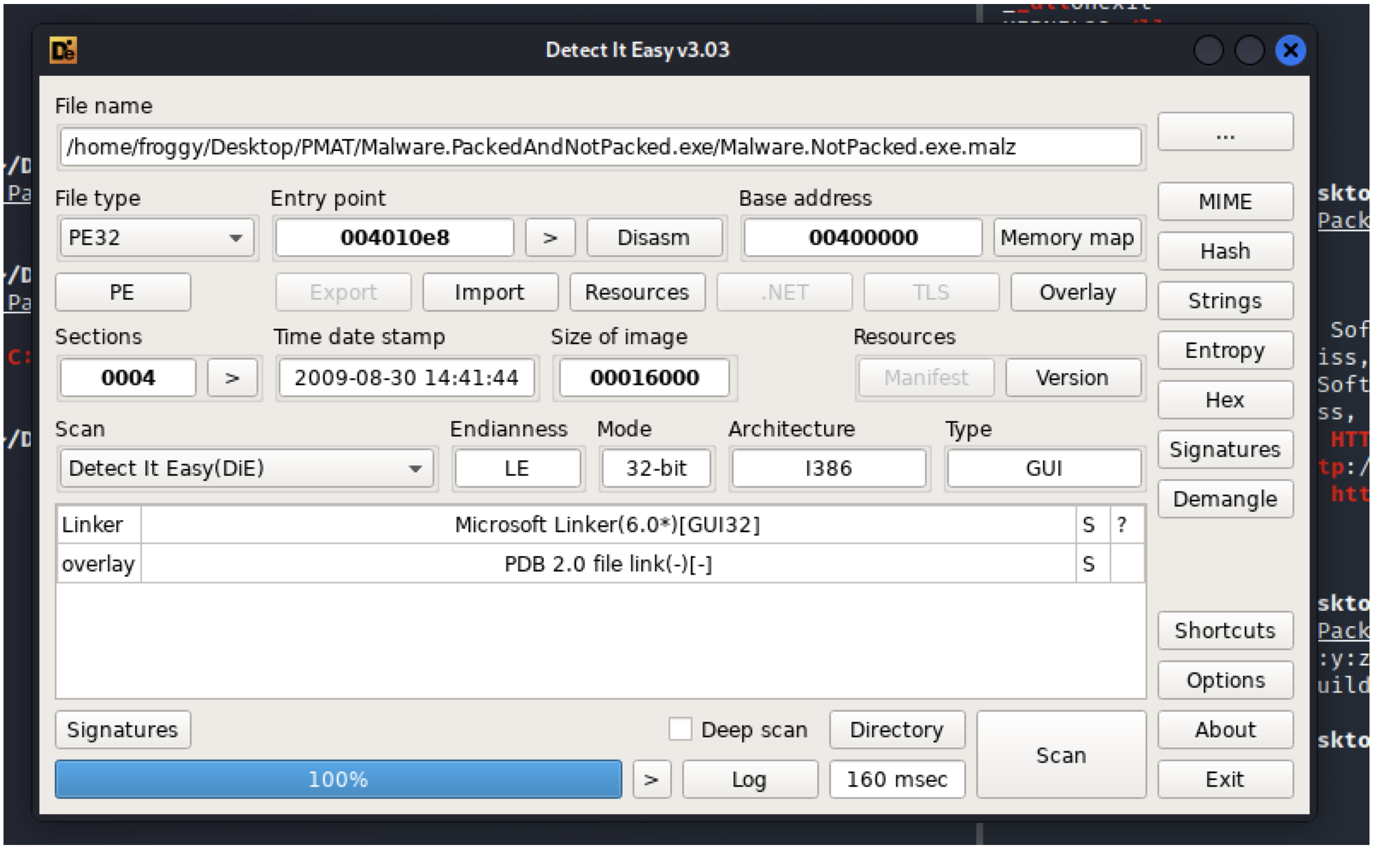
```
┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ strings Malware.Packed.exe.malz | grep -i ".dll" 2>/dev/null
_dll
ADVAPI32.dll
KERNEL32.DLL
MSVCRT.dll
WS2_32.dll
WSOCK32.dll

┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ strings Malware.Packed.exe.malz | grep -i "http" 2>/dev/null
HTTP/1.0

┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ strings Malware.Packed.exe.malz | grep -i "C:" 2>/dev/null
Bgc:H
:T:p:u:v:rkVhwH:y:z:C:H:P:A:
c: )f

┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ 
```

```
┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ strings Malware.NotPacked.exe.malz | grep -i ".dll" 2>/dev/null
MSVCRT.dll
__dllonexit
KERNEL32.dll
ADVAPI32.dll
WSOCK32.dll
WS2_32.dll
ntdll.dll

┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ strings Malware.NotPacked.exe.malz | grep -i "http" 2>/dev/null
%s %s HTTP/1.0
%s %s HTTP/1.0
HTTP
 Licensed to The Apache Software Foundation, http://www.apache.org/<br>
 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/<br>
Licensed to The Apache Software Foundation, http://www.apache.org/
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
    -k              Use HTTP KeepAlive feature
Usage: %s [options] [http://]hostname[:port]/path
SSL not compiled in; no https support
https://
http://

┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ strings Malware.NotPacked.exe.malz | grep -i "C:" 2>/dev/null
n:c:t:b:T:p:u:v:rkVhwix:y:z:C:H:P:A:g:X:de:Sq
C:\local0\asf\release\build-2.2.14\support\Release\ab.pdb
```

Although, it's always better to analyze the binaries using PEID, PEStudio, DIE etc.

> Detect It Easy - https://github.com/horsicq/DIE-engine/releases/download/3.03/Detect_It_Easy-3.03-x86_64.AppImage

In the below screenshot, we can see that executable NotPacked is actually not packed.



File name: /home/froggy/Desktop/PMAT/Malware.PackedAndNotPacked.exe/Malware.NotPacked.exe.malz

File type: PE32
Entry point: 004010e8
Base address: 00400000

Sections: 0004
Time date stamp: 2009-08-30 14:41:44
Size of image: 00016000

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

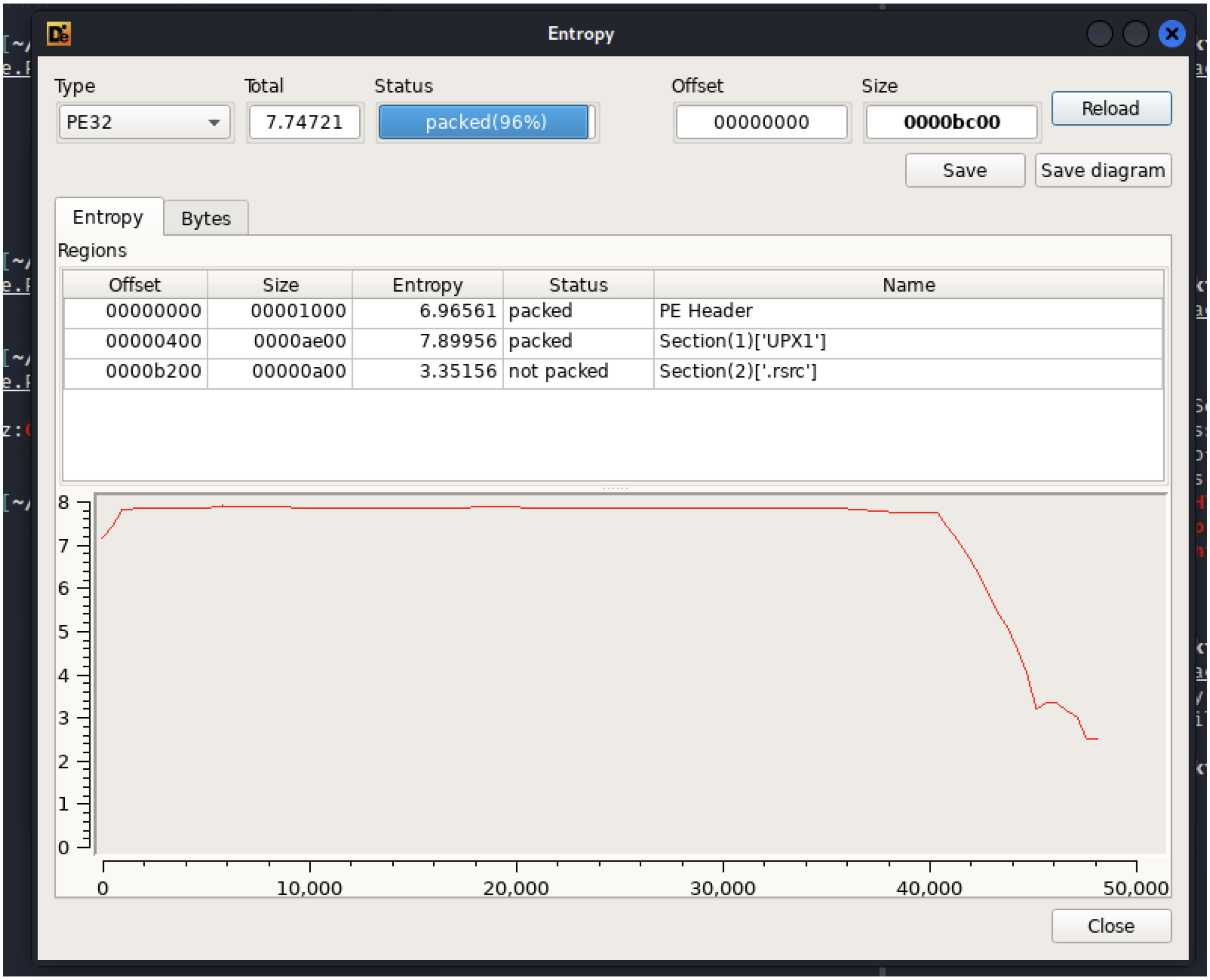
Linker	Microsoft Linker(6.0*)[GUI32]	S	?
overlay	PDB 2.0 file link(-)[-]	S	

100%
160 msec

In the below screenshot, we can that executable Packed is actually packed with UPX which is a compression software.

Also, we can check the entropy for the both executables which tells more about the sections of the executable whether they are packed or not.

For the Packed executable we can observe that Offset and Size are almost equivalent to 0 because the data is not in those offsets/locations.



Entropy

Type: PE32

Total: 7.74721

Status: packed(96%)

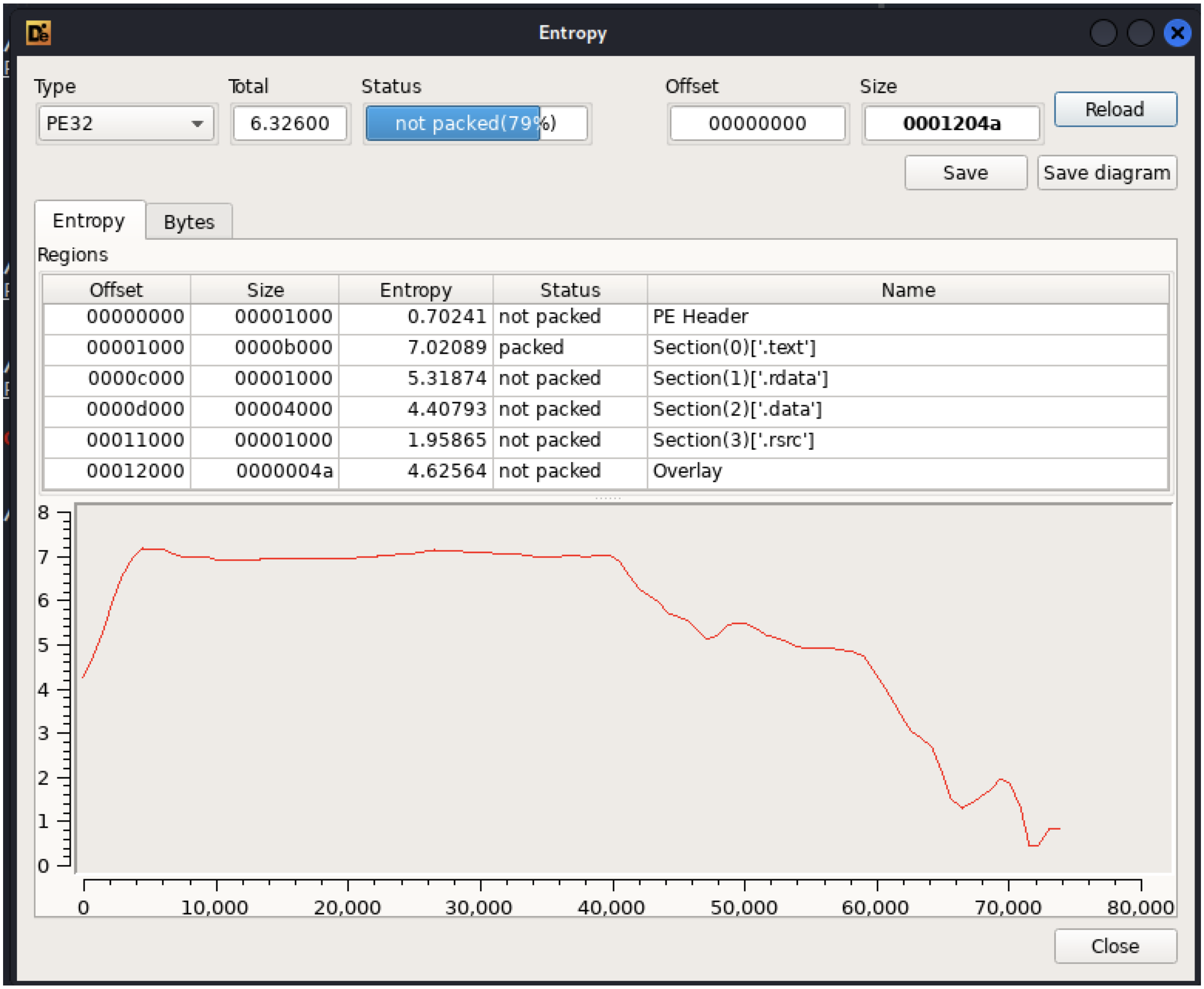
Offset: 00000000

Size: 0000bc00

Regions

Offset	Size	Entropy	Status	Name
00000000	00001000	6.96561	packed	PE Header
00000400	0000ae00	7.89956	packed	Section(1)['UPX1']
0000b200	00000a00	3.35156	not packed	Section(2)['.rsrc']

But, for the NotPacked executable we can observe tht graph is actually varying and the Offset and Sizes are close to each other thus giving us the hint that the executable is actually not packed.



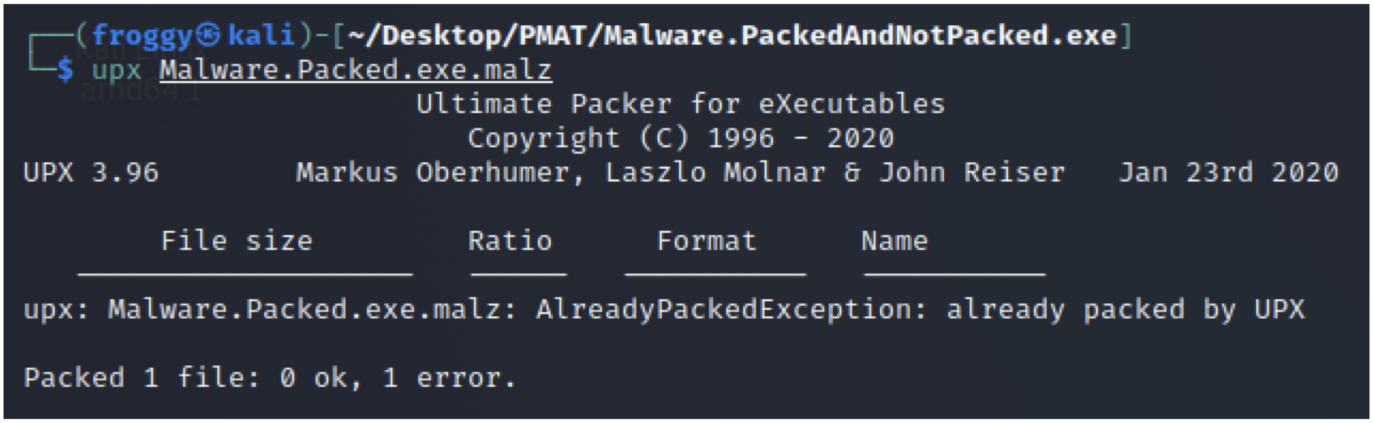
Entropy

Type: PE32
Total: 6.32600
Status: not packed(79%)
Offset: 00000000
Size: 0001204a

Regions

Offset	Size	Entropy	Status	Name
00000000	00001000	0.70241	not packed	PE Header
00001000	0000b000	7.02089	packed	Section(0)['.text']
0000c000	00001000	5.31874	not packed	Section(1)['.rdata']
0000d000	00004000	4.40793	not packed	Section(2)['.data']
00011000	00001000	1.95865	not packed	Section(3)['.rsrc']
00012000	0000004a	4.62564	not packed	Overlay

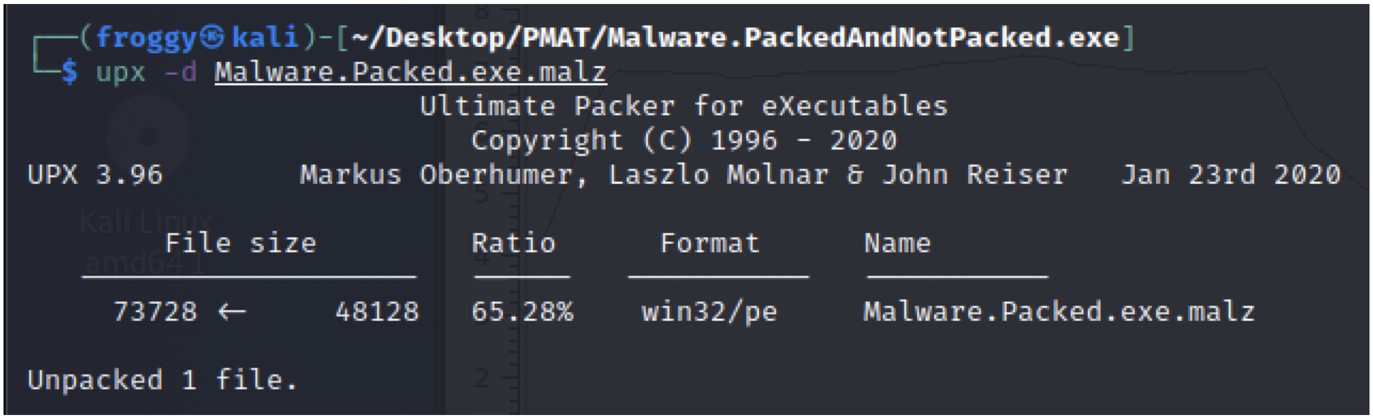
To confirm, we can actually use UPX which in the below screenshot we can confirm that executable is already packed using UPX.



```
┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ upx Malware.Packed.exe.malz
                       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
   --------------------   ------   -----------   -----------
upx: Malware.Packed.exe.malz: AlreadyPackedException: already packed by UPX

Packed 1 file: 0 ok, 1 error.
```

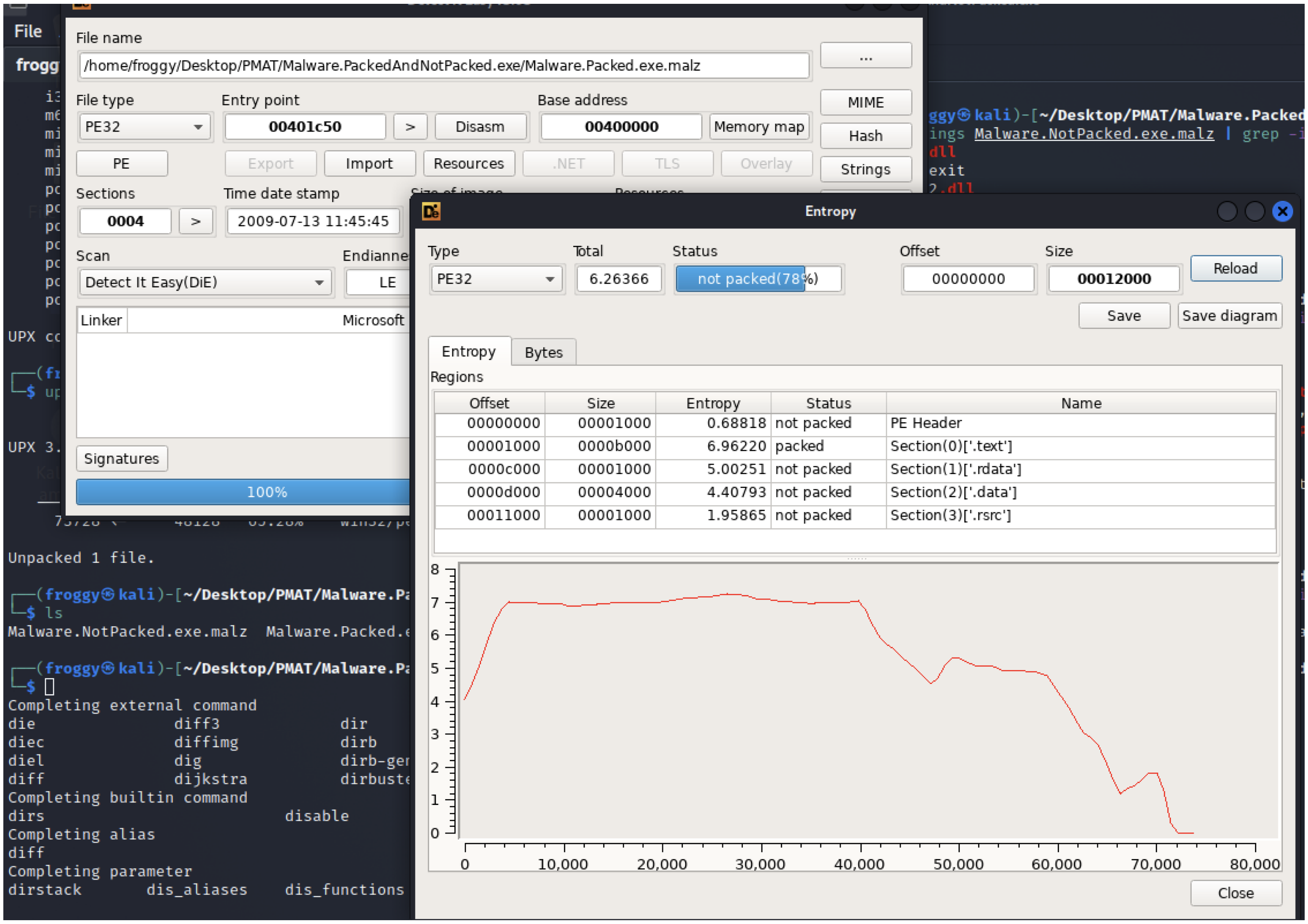
We can unpack the executable using UPX.

```
upx -d Malware.Packed.exe.malz
```



```
┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.PackedAndNotPacked.exe]
└─$ upx -d Malware.Packed.exe.malz
                       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
   --------------------   ------   -----------   -----------
     73728 <-     48128   65.28%    win32/pe     Malware.Packed.exe.malz

Unpacked 1 file.
```

Now, we can load the exe back into the DIE to observe the Entropy.



File name: /home/froggy/Desktop/PMAT/Malware.PackedAndNotPacked.exe/Malware.Packed.exe.malz

File type: PE32
Entry point: 00401c50
Base address: 00400000
Sections: 0004
Time date stamp: 2009-07-13 11:45:45
Scan: Detect It Easy(DiE)
Endianness: LE
Linker: Microsoft

Entropy

Type: PE32
Total: 6.26366
Status: not packed(78%)
Offset: 00000000
Size: 00012000

Offset	Size	Entropy	Status	Name
00000000	00001000	0.68818	not packed	PE Header
00001000	0000b000	6.96220	packed	Section(0)['.text']
0000c000	00001000	5.00251	not packed	Section(1)['.rdata']
0000d000	00004000	4.40793	not packed	Section(2)['.data']
00011000	00001000	1.95865	not packed	Section(3)['.rsrc']

```
Unpacked 1 file.

┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.Pa
└─$ ls
Malware.NotPacked.exe.malz  Malware.Packed.e

┌──(froggy㉿kali)-[~/Desktop/PMAT/Malware.Pa
└─$ 
Completing external command
die                 diff3               dir
diec                diffimg             dirb
diel                dig                 dirb-ger
diff                dijkstra            dirbuste
Completing builtin command
dirs                                    disable
Completing alias
diff
Completing parameter
dirstack        dis_aliases     dis_functions
```

We can observe in the above screenshot, that now we have the PE sections in place.

Now, it's late but we should know what these sections already mean and what actual data they hold.

.text - This section contains the actual code of the program. The code which the programmer/coder has written.
.data - This section contains the initialized data of the program.

The variables which has been assigned values.

```
int a = 5;
char b = 'c';
```

.bss - This section contains the uninitialized data of the program. The variables whih has not been assigned the values.

```
int a;
char b;
float c;
```

.stack - The stack is allocated where the local variables and arguments of the program are loaded.
.heap - Heap is the extra memory allocated by the OS so that if program needs more memory it can utilize this memory.
.rsrc - Section which holds **information about various other resources needed by the executable**, such as the icon that is shown when looking at the executable file in explorer.
.rdata - Holds read-only initialized data.

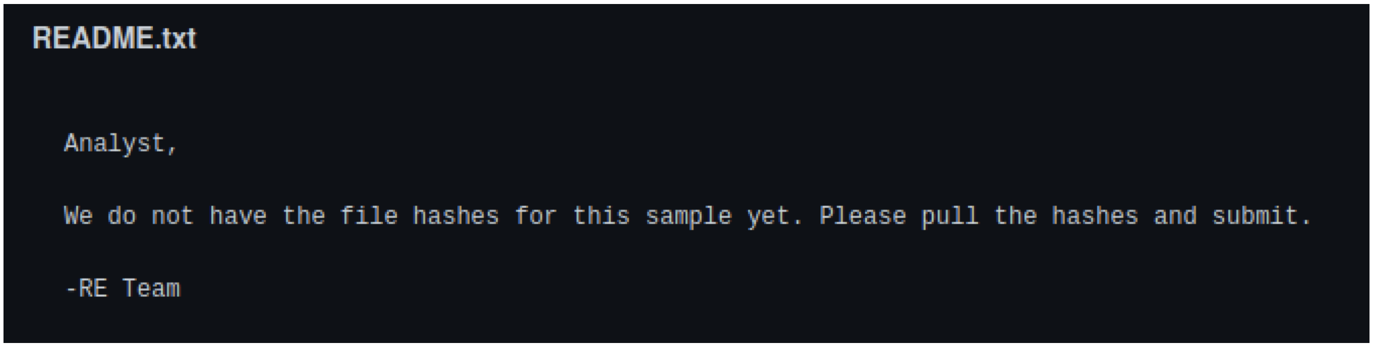
There are a lot more sections in PE (Portable Executables) which can be found on the below mentioned link.

> https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

Analysing the Unkown Executable

> https://github.com/HuskyHacks/PMAT-labs/tree/main/labs/1-1.BasicStaticAnalysis/Malware.Unknown.exe.malz

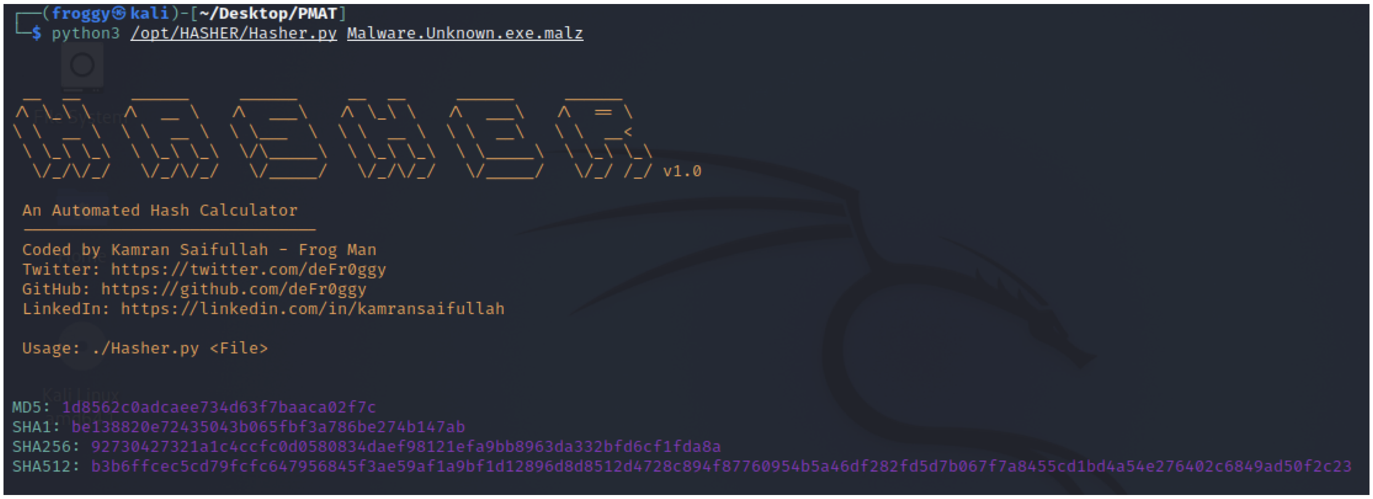
So, the hashes for this executable has not yet calculated. So, we will calculate the hashes using HASHER.



README.txt

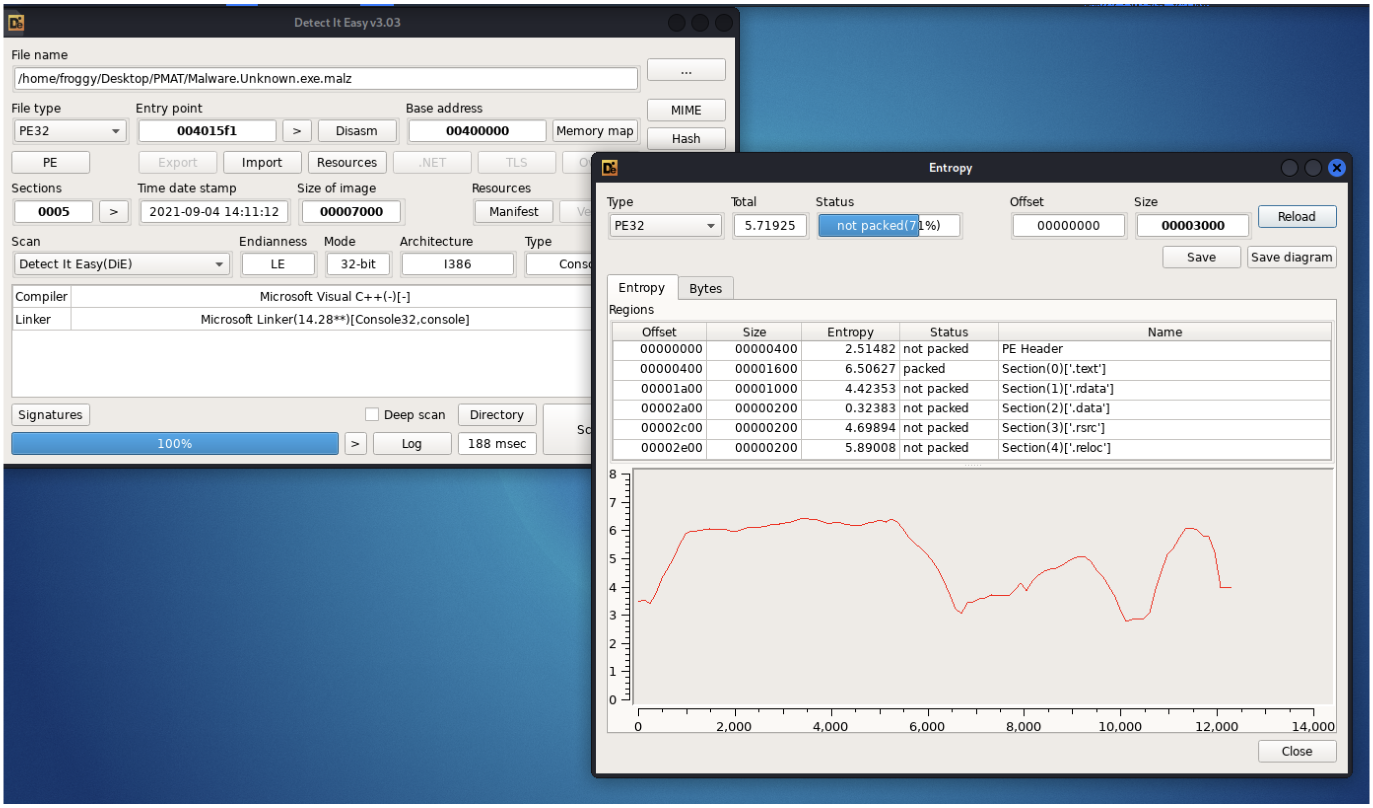
```
Analyst,

We do not have the file hashes for this sample yet. Please pull the hashes and submit.

-RE Team
```

> https://github.com/deFr0ggy/HASHER



```
┌──(froggy㉿kali)-[~/Desktop/PMAT]
└─$ python3 /opt/HASHER/Hasher.py Malware.Unknown.exe.malz

 __  __     ______     ______     __  __     ______     ______
/\ \_\ \   /\  __ \   /\  ___\   /\ \_\ \   /\  ___\   /\  == \
\ \  __ \  \ \  __ \  \ \___  \  \ \  __ \  \ \  __\   \ \  __<
 \ \_\ \_\  \ \_\ \_\  \/\_____\  \ \_\ \_\  \ \_____\  \ \_\ \_\
  \/_/\/_/   \/_/\/_/   \/_____/   \/_/\/_/   \/_____/   \/_/ /_/ v1.0

 An Automated Hash Calculator
 ______________________________

 Coded by Kamran Saifullah - Frog Man
 Twitter: https://twitter.com/deFr0ggy
 GitHub: https://github.com/deFr0ggy
 LinkedIn: https://linkedin.com/in/kamransaifullah

 Usage: ./Hasher.py <File>


MD5: 1d8562c0adcaee734d63f7baaca02f7c
SHA1: be138820e72435043b065fbf3a786be274b147ab
SHA256: 92730427321a1c4ccfc0d0580834daef98121efa9bb8963da332bfd6cf1fda8a
SHA512: b3b6ffcec5cd79fcfc647956845f3ae59af1a9bf1d12896d8d8512d4728c894f87760954b5a46df282fd5d7b067f7a8455cd1bd4a54e276402c6849ad50f2c23
```

As we have now, calculated the hashes, we will move forward to load the binary in DIE to check whether it's packed or not packed.



Detect It Easy v3.03

File name: /home/froggy/Desktop/PMAT/Malware.Unknown.exe.malz

File type: PE32 | Entry point: 004015f1 | Base address: 00400000

Sections: 0005 | Time date stamp: 2021-09-04 14:11:12 | Size of image: 00007000

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

Compiler	Microsoft Visual C++(-)[-]
Linker	Microsoft Linker(14.28**)[Console32,console]

□ Deep scan | 100% | 188 msec

Entropy

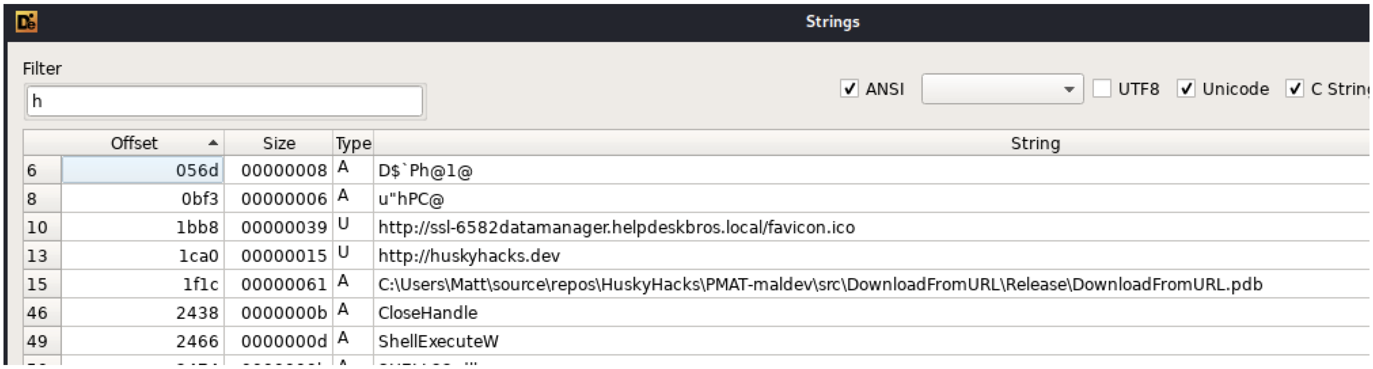
Type: PE32 | Total: 5.71925 | Status: not packed(71%) | Offset: 00000000 | Size: 00003000

Regions

Offset	Size	Entropy	Status	Name
00000000	00000400	2.51482	not packed	PE Header
00000400	00001600	6.50627	packed	Section(0)['.text']
00001a00	00001000	4.42353	not packed	Section(1)['.rdata']
00002a00	00000200	0.32383	not packed	Section(2)['.data']
00002c00	00000200	4.69894	not packed	Section(3)['.rsrc']
00002e00	00000200	5.89008	not packed	Section(4)['.reloc']

From the above screenshot, we can confirm that the binary is not packed.

DIE also finds out the strings and gives us a nice dashboard to filter through. In this, particular scenario we can find that there are URLs in the binary and the potential local GitHub repo where @HuskyHacks wrote this malware/binary.



Filter: h

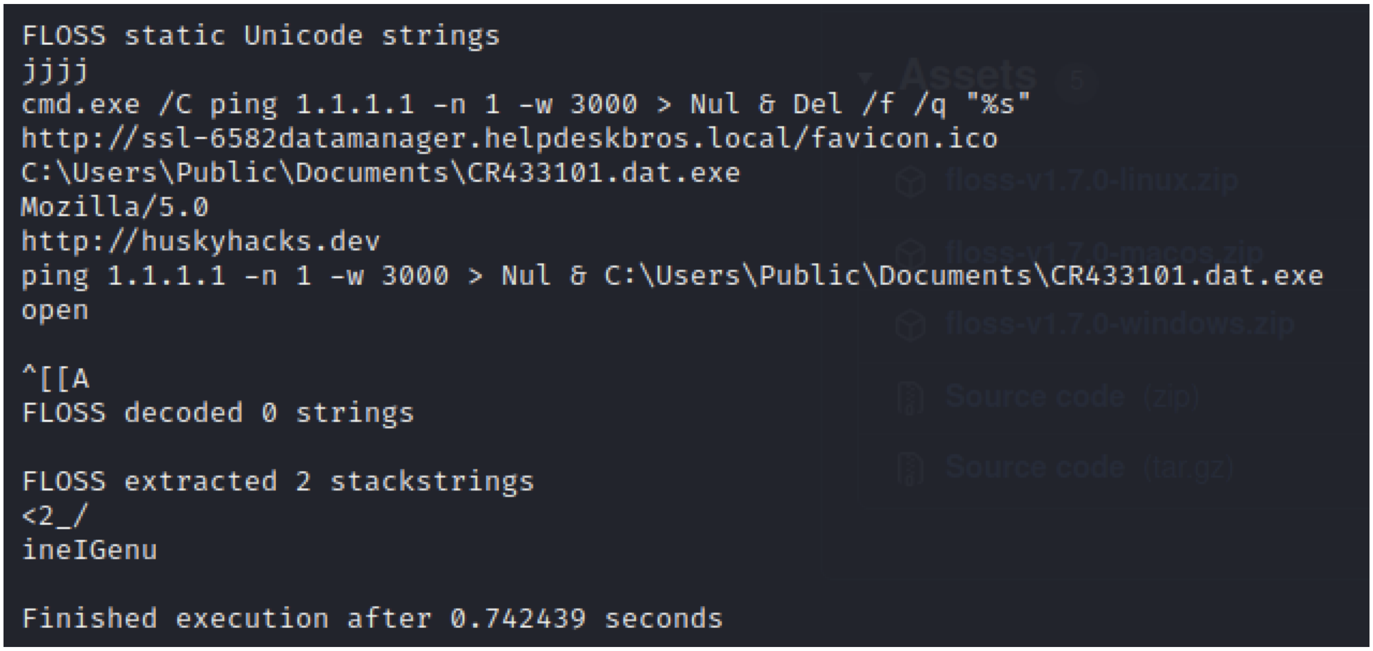
☑ ANSI ☐ UTF8 ☑ Unicode ☑ C Strin

	Offset	Size	Type	String
6	056d	00000008	A	D$`Ph@1@
8	0bf3	00000006	A	u"hPC@
10	1bb8	00000039	U	http://ssl-6582datamanager.helpdeskbros.local/favicon.ico
13	1ca0	00000015	U	http://huskyhacks.dev
15	1f1c	00000061	A	C:\Users\Matt\source\repos\HuskyHacks\PMAT-maldev\src\DownloadFromURL\Release\DownloadFromURL.pdb
46	2438	0000000b	A	CloseHandle
49	2466	0000000d	A	ShellExecuteW

- http://ssl-6582datamanager.helpdeskbros.local/favicon.ico
- http://huskyhacks.dev
- C:\Users\Matt\source\repos\HuskyHacks\PMAT-maldev\src\DownloadFromURL\Release\DownloadFromURL.pdb

As these URLs are not being shown when we try to use the String utility, we will move forward to use FLOSS which is basically ***strings on steroids***.

> https://github.com/mandiant/flare-floss/releases



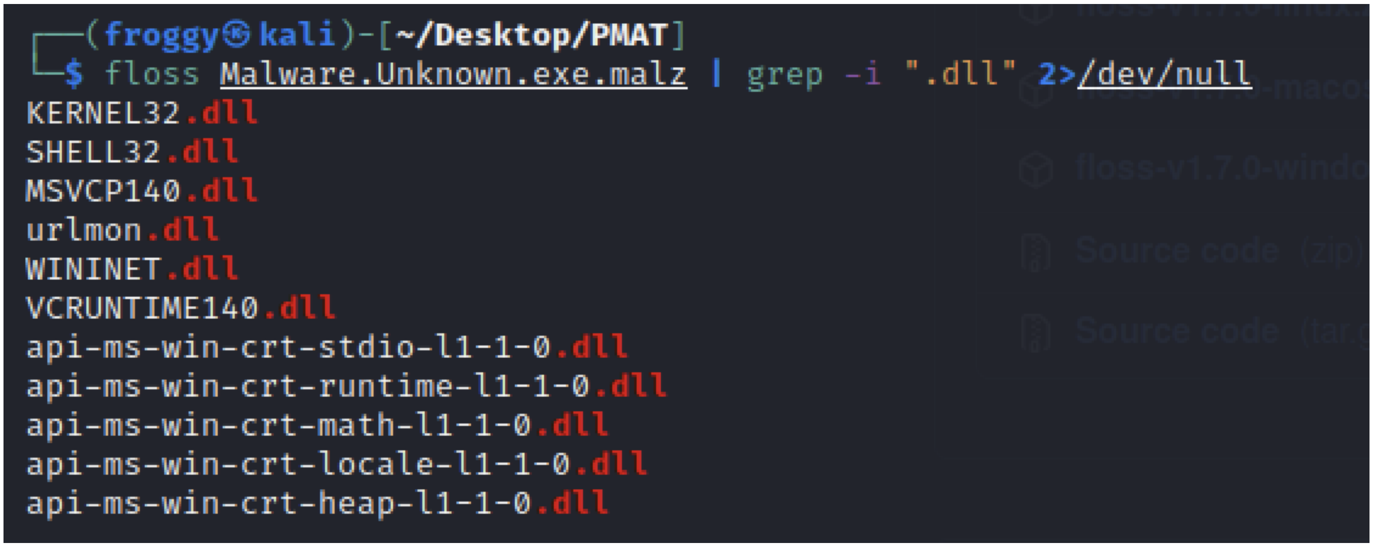
```
FLOSS static Unicode strings
jjjj
cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
http://ssl-6582datamanager.helpdeskbros.local/favicon.ico
C:\Users\Public\Documents\CR433101.dat.exe
Mozilla/5.0
http://huskyhacks.dev
ping 1.1.1.1 -n 1 -w 3000 > Nul & C:\Users\Public\Documents\CR433101.dat.exe
open

^[[A
FLOSS decoded 0 strings

FLOSS extracted 2 stackstrings
<2_/
ineIGenu

Finished execution after 0.742439 seconds
```

```
cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
http://ssl-6582datamanager.helpdeskbros.local/favicon.ico
C:\Users\Public\Documents\CR433101.dat.exe
http://huskyhacks.dev
ping 1.1.1.1 -n 1 -w 3000 > Nul &
C:\Users\Public\Documents\CR433101.dat.exe
```

So, in addition to the previous URLs we have found, using floss we have found some great information. We have found the executable which is normally not present on the user system.

Also, i like to grep for the DLLs which gives insights into what the executalbe might be capable of. In this scenario, we can see that SHELL32.dll is loaded and WININET.DLL is loaded.

So using the SHELL32.dll API Calls/Functions/Sub-Routines, the cmd.exe function is called from within the main executable and by using the WININET.dll API Calls/Functions/Sub-Routines internet functionality is used to query the two URLs which we have found previously.



```
┌──(froggy㉿kali)-[~/Desktop/PMAT]
└─$ floss Malware.Unknown.exe.malz | grep -i ".dll" 2>/dev/null
KERNEL32.dll
SHELL32.dll
MSVCP140.dll
urlmon.dll
WININET.dll
VCRUNTIME140.dll
api-ms-win-crt-stdio-l1-1-0.dll
api-ms-win-crt-runtime-l1-1-0.dll
api-ms-win-crt-math-l1-1-0.dll
api-ms-win-crt-locale-l1-1-0.dll
api-ms-win-crt-heap-l1-1-0.dll
```

WININET.DLL

```
┌──(froggy㉿kali)-[~/Desktop/PMAT]
└─$ floss Malware.Unknown.exe.malz | grep -i "Internet"
2>/dev/null
1 ×
InternetOpenUrlW
InternetOpenW
```

SHELL32.DLL

```
┌──(froggy㉿kali)-[~/Desktop/PMAT]
└─$ floss Malware.Unknown.exe.malz | grep -i "SHELL"
2>/dev/null
ShellExecuteW
```

```
SHELL32.dll
```

Follow Me

Twitter: https://twitter.com/deFr0ggy

GitHub: https://github.com/deFr0ggy

LinkedIn: https://linkedin.com/in/kamransaifullah