*** Before start, below are sample file we're going to analyze and I will refer those file by Sample 1, 2 and 3.**

Sample 1:

101ccbad7732fb185d51b91d31a67ff058cac3bc31ec36cec05094065a97d6fd.sample

Sample 2:

431d230862e958dd5d20ae221ce74aba07d40dde5fd8e45f2b164905e637b1c1.sample

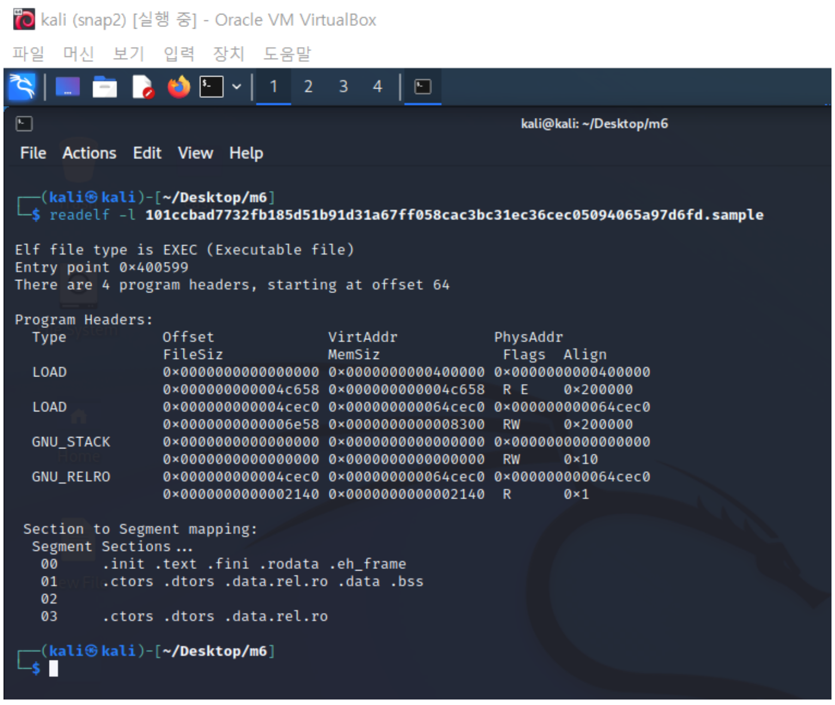
Sample 3:

f808a42b10cf55603389945a549ce45edc6a04562196d14f7489af04688f12bc.sample

1. Is the malware sample packed? What evidence do you have for your answer?

Sample1: It's packed.

Evidence: Segment types and count. The file contains only PT_LOAD segments (and PT_GNU_STACK).


kali (snap2) [실행 중] - Oracle VM VirtualBox
파일 머신 보기 입력 장치 도움말
1 2 3 4
kali@kali: ~/Desktop/m6
File Actions Edit View Help

```
┌──(kali㉿kali)-[~/Desktop/m6]
└─$ readelf -l 101ccbad7732fb185d51b91d31a67ff058cac3bc31ec36cec05094065a97d6fd.sample

Elf file type is EXEC (Executable file)
Entry point 0x400599
There are 4 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000004c658 0x000000000004c658  R E    0x200000
  LOAD           0x000000000004cec0 0x000000000064cec0 0x000000000064cec0
                 0x0000000000006e58 0x0000000000008300  RW     0x200000
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x000000000004cec0 0x000000000064cec0 0x000000000064cec0
                 0x0000000000002140 0x0000000000002140  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00     .init .text .fini .rodata .eh_frame
   01     .ctors .dtors .data.rel.ro .data .bss
   02
   03     .ctors .dtors .data.rel.ro

┌──(kali㉿kali)-[~/Desktop/m6]
└─$ 
```

Sample2: It's not packed.

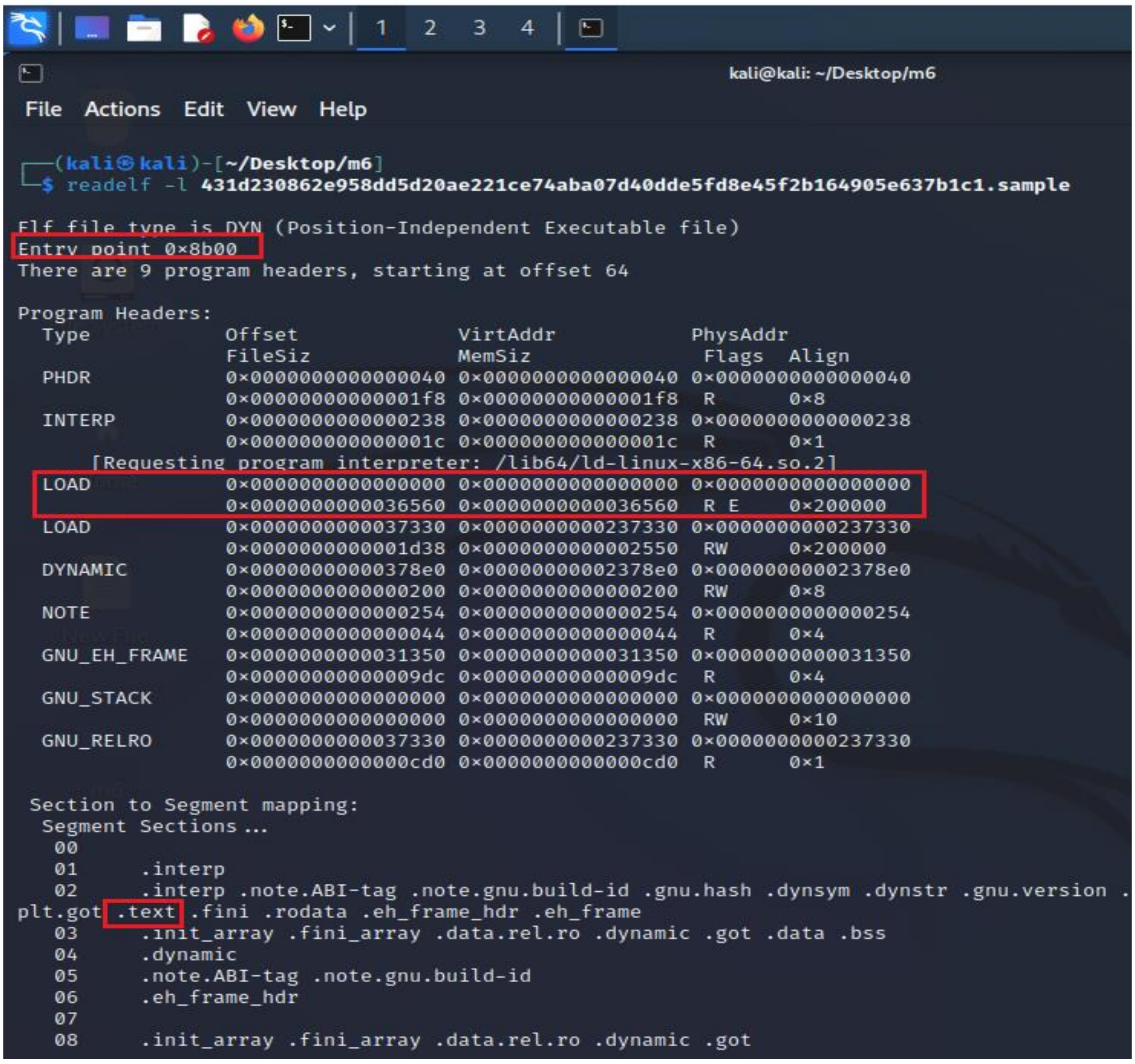
Evidence:

- readelf: Entry point resides in PT_LOAD which is designated with RE. Also, I can see '.text' in the Segment section and its location is in PT_LOAD. Looking at segment table but I couldn't find anything wrong.

- ELFparser: Didn't show anything about packer

- DIE(Detect it Easy): The program just died when chose a file. So I tried light version of it(the file also included in a deployment) and it showed nothing about packer.

- strings command: No signs of 'UPX' or 'LSD'

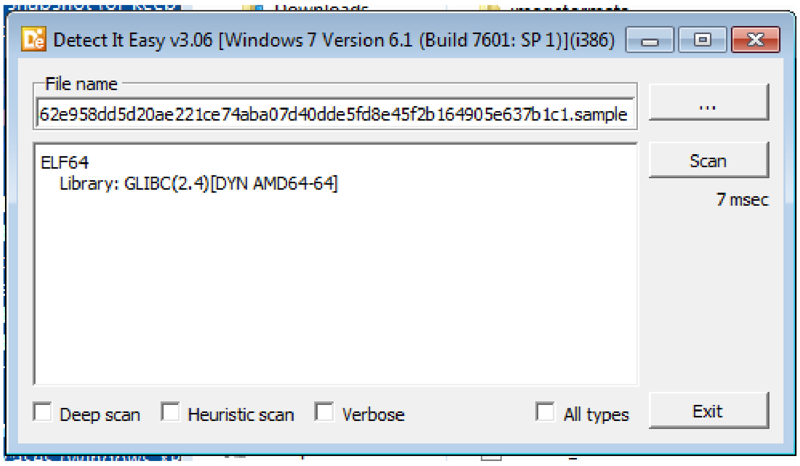


```
┌──(kali㉿kali)-[~/Desktop/m6]
└─$ readelf -l 431d230862e958dd5d20ae221ce74aba07d40dde5fd8e45f2b164905e637b1c1.sample

Elf file type is DYN (Position-Independent Executable file)
Entry point 0x8b00
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000001f8 0x00000000000001f8  R      0x8
  INTERP         0x0000000000000238 0x0000000000000238 0x0000000000000238
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000036560 0x0000000000036560  R E    0x200000
  LOAD           0x0000000000037330 0x0000000000237330 0x0000000000237330
                 0x0000000000001d38 0x0000000000002550  RW     0x200000
  DYNAMIC        0x00000000000378e0 0x00000000002378e0 0x00000000002378e0
                 0x0000000000000200 0x0000000000000200  RW     0x8
  NOTE           0x0000000000000254 0x0000000000000254 0x0000000000000254
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_EH_FRAME   0x0000000000031350 0x0000000000031350 0x0000000000031350
                 0x00000000000009dc 0x00000000000009dc  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000037330 0x0000000000237330 0x0000000000237330
                 0x0000000000000cd0 0x0000000000000cd0  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .
plt.got .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .data.rel.ro .dynamic .got .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .data.rel.ro .dynamic .got
```


Detect It Easy v3.06 [Windows 7 Version 6.1 (Build 7601: SP 1)](i386)

File name
62e958dd5d20ae221ce74aba07d40dde5fd8e45f2b164905e637b1c1.sample

```
ELF64
   Library: GLIBC(2.4)[DYN AMD64-64]
```

7 msec

☐ Deep scan ☐ Heuristic scan ☐ Verbose ☐ All types

Sample3: It’s not packed.

Evidence:

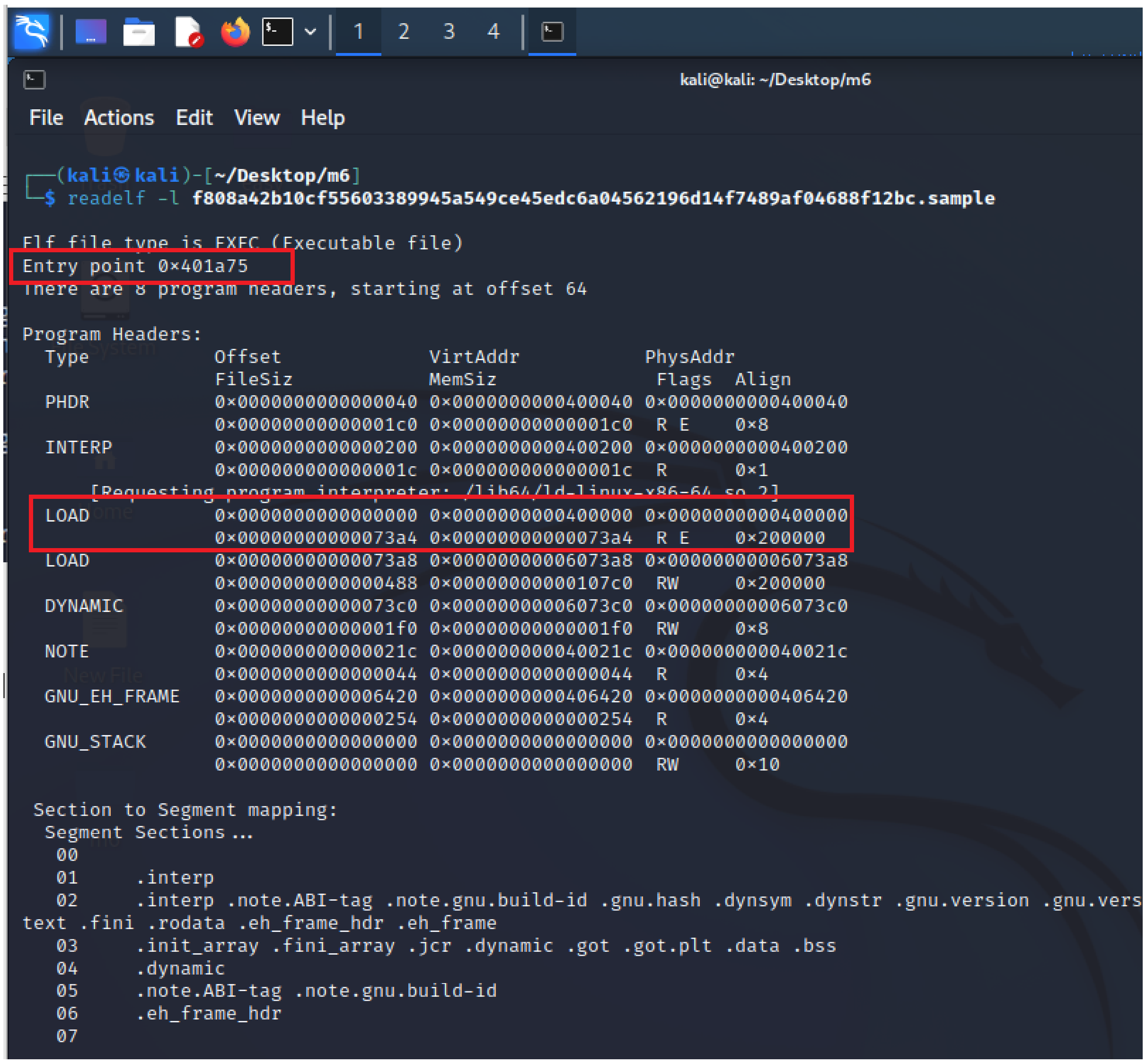
- readelf: Entry point resides in PT_LOAD which is designated with RE. Also, I can see ‘.text’ in the Segment section and its location is in PT_LOAD. Looking at segment table but I couldn’t find anything wrong.

- ELFparser: Didn't show anything about packer

- DIE(Detect it Easy): The program just died when chose a file. So I tried light version of it(the file also included in a deployment) and it showed nothing about packer.

- strings command: No signs of 'UPX' or 'LSD'

Entry point resides in PT_LOAD which is designated with RE. Also, I can see ‘.text’ in the Segment section and its location is in PT_LOAD.



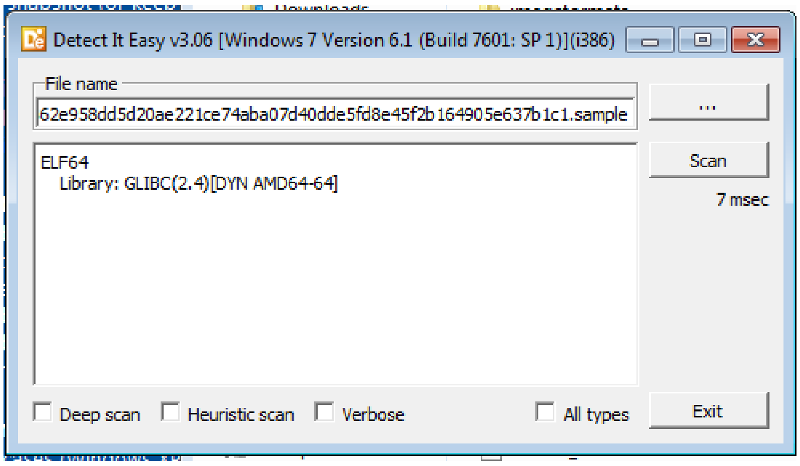
```
kali@kali: ~/Desktop/m6

┌──(kali㉿kali)-[~/Desktop/m6]
└─$ readelf -l f808a42b10cf55603389945a549ce45edc6a04562196d14f7489af04688f12bc.sample

Elf file type is EXEC (Executable file)
Entry point 0x401a75
There are 8 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001c0 0x00000000000001c0  R E    0x8
  INTERP         0x0000000000000200 0x0000000000400200 0x0000000000400200
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x00000000000073a4 0x00000000000073a4  R E    0x200000
  LOAD           0x00000000000073a8 0x00000000006073a8 0x00000000006073a8
                 0x0000000000000488 0x00000000000107c0  RW     0x200000
  DYNAMIC        0x00000000000073c0 0x00000000006073c0 0x00000000006073c0
                 0x00000000000001f0 0x00000000000001f0  RW     0x8
  NOTE           0x000000000000021c 0x000000000040021c 0x000000000040021c
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_EH_FRAME   0x0000000000006420 0x0000000000406420 0x0000000000406420
                 0x0000000000000254 0x0000000000000254  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.vers
text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
```



Detect It Easy v3.06 [Windows 7 Version 6.1 (Build 7601: SP 1)](i386)

File name: 62e958dd5d20ae221ce74aba07d40dde5fd8e45f2b164905e637b1c1.sample

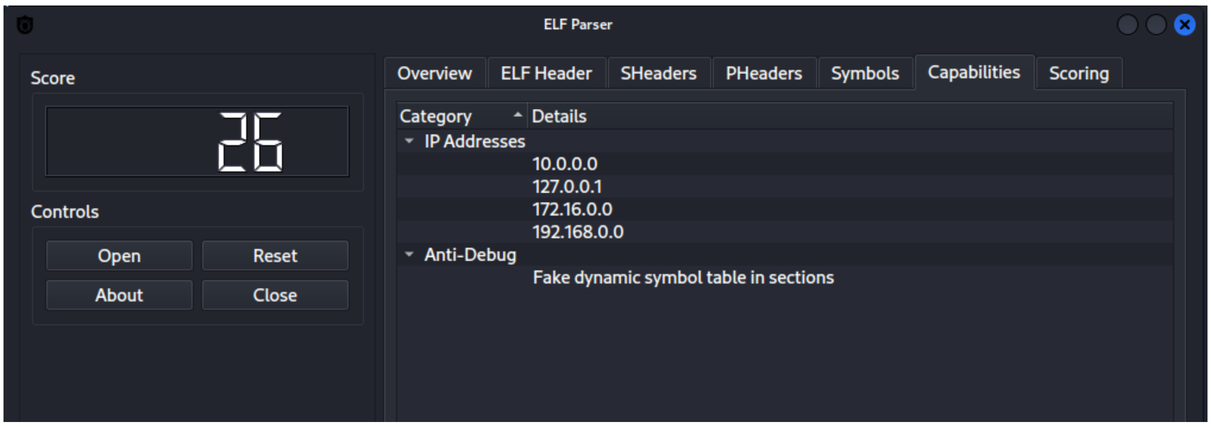
```
ELF64
   Library: GLIBC(2.4)[DYN AMD64-64]
```

7 msec

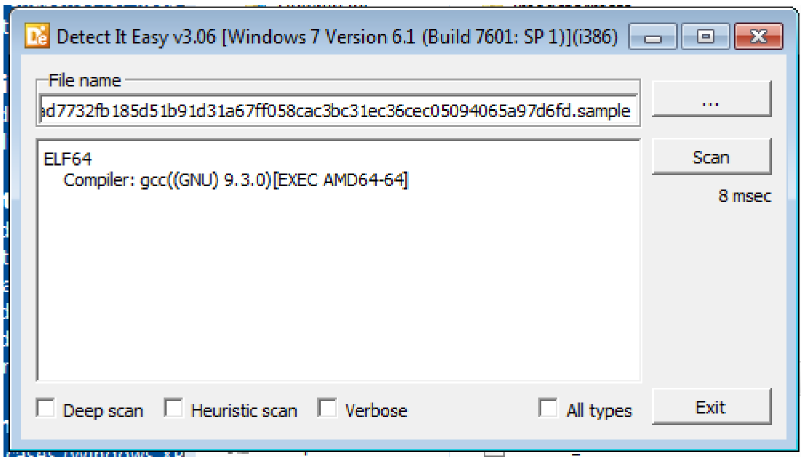
☐ Deep scan ☐ Heuristic scan ☐ Verbose ☐ All types

2. If so, can you unpack it easily? If so, what program/command can you use to unpack it? If not, why not? Please provide evidence either way.

Sample1: I could not get any information about packer from ELFparser and DetectItEasy. And I just tried to unpack it using UPX but it didn’t work.



Category	Details
IP Addresses	
	10.0.0.0
	127.0.0.1
	172.16.0.0
	192.168.0.0
Anti-Debug	
	Fake dynamic symbol table in sections


Detect It Easy v3.06 [Windows 7 Version 6.1 (Build 7601: SP 1)](i386)
File name
...
Scan
Deep scan
Heuristic scan
Verbose
All types
Exit

```
ad7732fb185d51b91d31a67ff058cac3bc31ec36cec05094065a97d6fd.sample

ELF64
    Compiler: gcc((GNU) 9.3.0)[EXEC AMD64-64]

8 msec
```

Sample2: It’s not unpacked.

Sample3: It’s not unpacked.

3. Are there any host- or network-based indicators that might help identify this malware if it were seen in the wild?

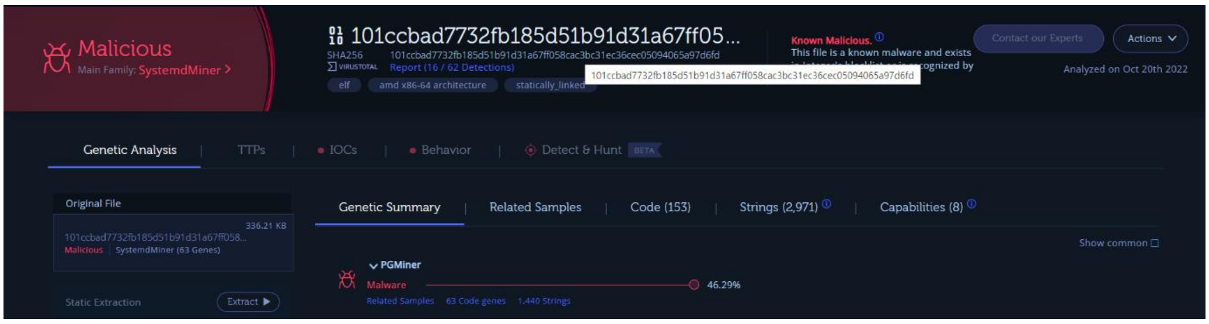
Sample1:

There are several IP address in Sample1 and I assume that 172.16.0.0 would be a host IP address and it could be an indicator of this malware.

4. Can you make an educated guess at any of the malware sample's functionality?

I think Sample1 and Sample3 are the malware program because when I ran the all 3 program using INTEZER ANALYZE and it showed Sample1 and 3 are malicious program.

Sample1: I couldn’t unpack the Sample1 so there’s not many thing to analyze. But from the ‘strings’ command I can see several IP addresses and it will try to connect a host when run this program.



Malicious
Main Family: SystemdMiner >

101ccbad7732fb185d51b91d31a67ff05...
SHA256 101ccbad7732fb185d51b91d31a67ff058cac3bc31ec36cec05094065a97d6fd
VIRUSTOTAL Report (16 / 62 Detections)
elf | amd x86-64 architecture | statically_linked

Known Malicious.
This file is a known malware and exists … cognized by

101ccbad7732fb185d51b91d31a67ff058cac3bc31ec36cec05094065a97d6fd

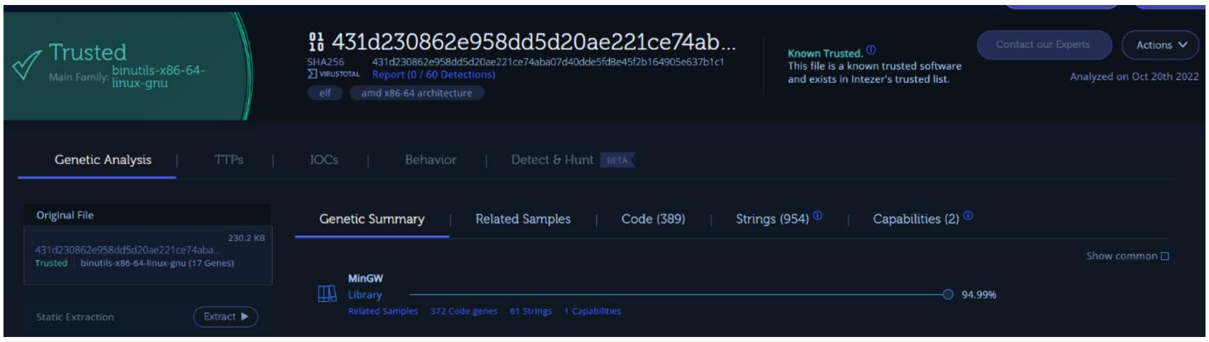
Analyzed on Oct 20th 2022

Original File
336.21 KB
101ccbad7732fb185d51b91d31a67ff058...
Malicious | SystemdMiner (63 Genes)

Genetic Summary | Related Samples | Code (153) | Strings (2,971) | Capabilities (8)

PGMiner
Malware 46.29%
Related Samples 63 Code genes 1,440 Strings

Sample2: It’s not a malicious program. Anyway, it is related with file operation from the information of ELFparser. And there are also chmod and chwon string found, so it is also trying to change the permission of the file.



Trusted
Main Family: binutils-x86-64-linux-gnu

431d230862e958dd5d20ae221ce74ab...
SHA256 431d230862e958dd5d20ae221ce74aba07d40dde5fd8e45f2b164905e637b1c1
VIRUSTOTAL Report (0 / 60 Detections)
elf | amd x86-64 architecture

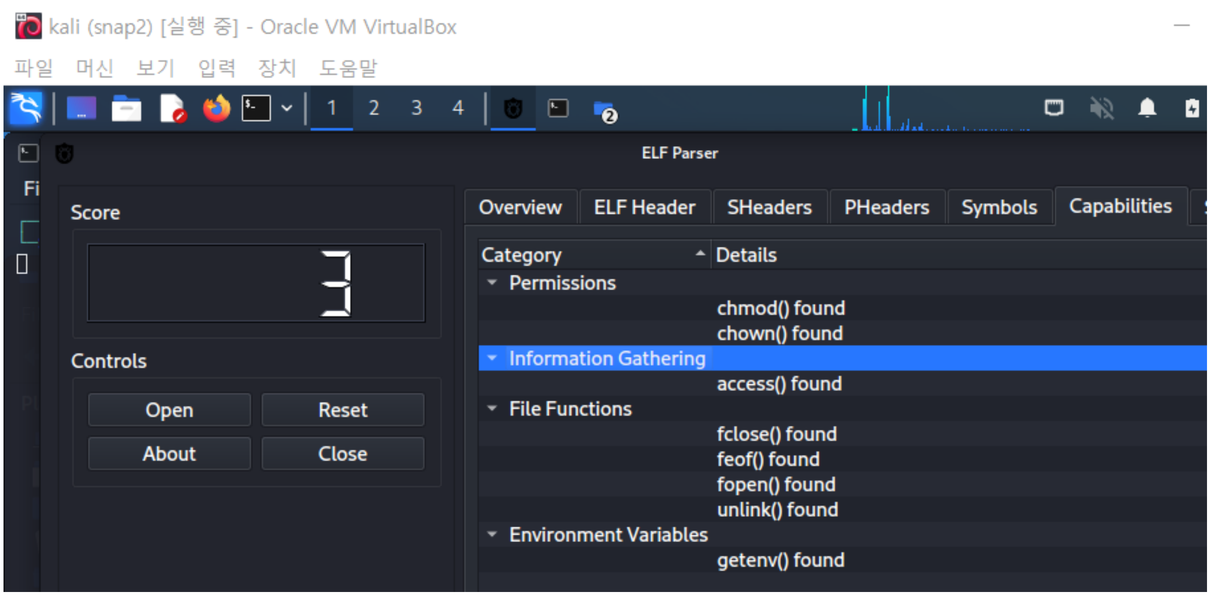
Known Trusted.
This file is a known trusted software and exists in Intezer's trusted list.

Analyzed on Oct 20th 2022

Original File
230.2 KB
431d230862e958dd5d20ae221ce74aba...
Trusted | binutils-x86-64-linux-gnu (17 Genes)

Genetic Summary | Related Samples | Code (389) | Strings (954) | Capabilities (2)

MinGW
Library 94.99%
Related Samples 372 Code genes 61 Strings 1 Capabilities



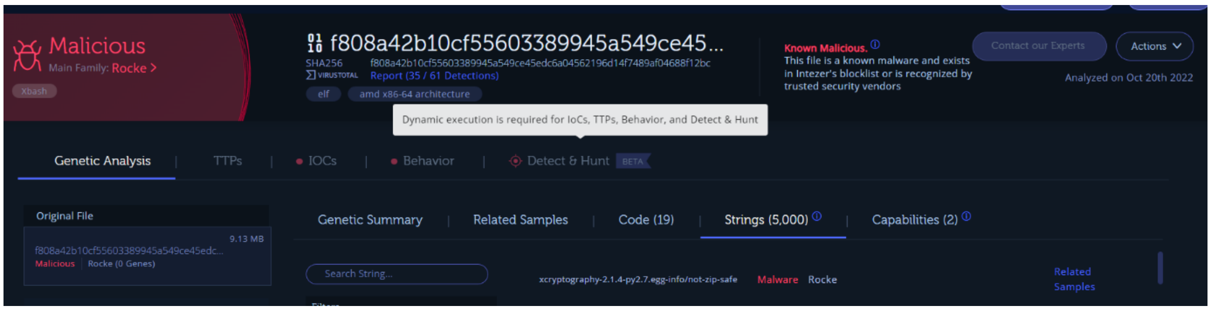
kali (snap2) [실행 중] - Oracle VM VirtualBox

ELF Parser

Score
3

Category	Details
Permissions	
	chmod() found
	chown() found
Information Gathering	
	access() found
File Functions	
	fclose() found
	feof() found
	fopen() found
	unlink() found
Environment Variables	
	getenv() found

Sample3: From the information from ELFparser, it will kill some process and will create another process. It also look up and set environment variable, I think it is trying to change process with another process.



Malicious
Main Family: Rocke
Xbash

f808a42b10cf55603389945a549ce45...
SHA256 f808a42b10cf55603389945a549ce45edc6a04562196d14f7489af04688f12bc
VIRUSTOTAL Report (35 / 61 Detections)
elf | amd x86-64 architecture

Known Malicious.
This file is a known malware and exists in Intezer's blocklist or is recognized by trusted security vendors

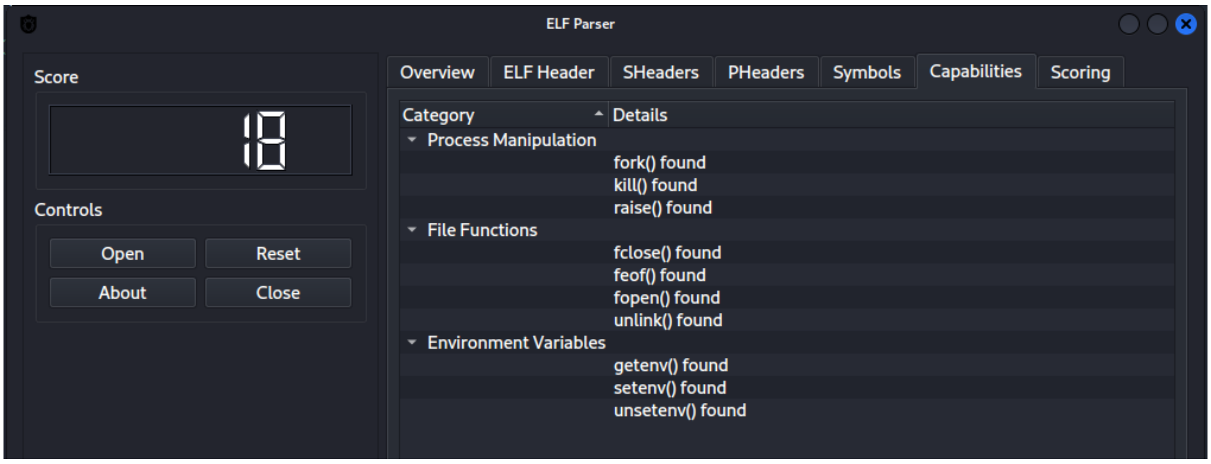
Analyzed on Oct 20th 2022

Dynamic execution is required for IoCs, TTPs, Behavior, and Detect & Hunt

Contact our Experts
Actions
Genetic Analysis | TTPs | IOCs | Behavior | Detect & Hunt BETA
Genetic Summary | Related Samples | Code (19) | Strings (5,000) | Capabilities (2)
Search String...
Related Samples

Original File
9.13 MB
f808a42b10cf55603389945a549ce45edc...
Malicious | Rocke (0 Genes)

xcryptography-2.1.4-py2.7.egg-info/not-zip-safe Malware Rocke



ELF Parser

Score
18

Category	Details
Process Manipulation	
	fork() found
	kill() found
	raise() found
File Functions	
	fclose() found
	feof() found
	fopen() found
	unlink() found
Environment Variables	
	getenv() found
	setenv() found
	unsetenv() found