

Digital Forensics (01)

Assignment2

2171056 강승연

분석 환경 : Window 10 명령 프롬프트

추가 설치 프로그램 :

Strings(<https://learn.microsoft.com/ko-kr/sysinternals/downloads/strings>)

Task 1

윈도우 환경에서 분석을 진행해 strings를 추가로 다운받아 실행시켰다.

각 단어에 대해 strings를 실행하고 단어를 검출하기 위해서 | find "{원하는 단어}" 명령어를 추가로 사용했다.

결과적으로 세 단어 모두 메모리에서 검출되었으며, 메모리가 Stuxnet을 포함하고 있다는 사실이 자명해졌다.

사용 명령어 : strings {stuxnet.vmem 파일 경로} | find "{원하는 단어}"

a. ".stub"

```
>strings D:\Workplace\Study\2024-1\DigitalForensics\assignment\HW2\stuxnet.vmem | find ".stub"
.stub
B.stub
.stub
B.stub
.stub
.stub
.stub
.stub
```

b. "mrxnet.sys"

```
>strings D:\Workplace\Study\2024-1\DigitalForensics\assignment\HW2\stuxnet.vmem | find "mrxnet.sys"
mrxnet.sys
\\WINDOWS\system32\Drivers\mrxnet.sys
WINDOWS\system32\drivers\mrxnet.sys
C:\\WINDOWS\system32\drivers\mrxnet.sysV
C:\\WINDOWS\system32\drivers\mrxnet.sys[
C:\\WINDOWS\system32\drivers\mrxnet.sysT
C:\\WINDOWS\system32\drivers\mrxnet.sysT
mrxnet.sys$Temp#
mrxnet.sys$
```

c. "verisign"

```
D:\Workplace\Study\2024-1\DigitalForensics\assignment\HW2>strings D:\Workplace\Study\2024-1\DigitalForensics\assignment\HW2\stuxnet.vmem | find "verisign"
=www.verisign.com/repository/RPA-Incorp. By Ref.,LIAB.LTD(c)981H0
F
$http://crl.verisign.com/pca1.1.1.crl0G
www.verisign.com/repository/RPA0
```

Task 2

먼저 stuxnet.vmem 파일에 대하여 각 옵션을 적절히 이용해 output을 {옵션}_result.txt 파일로 추출했다. 모든 근거 캡처본에 대해서는 어떤 옵션을 사용했는지 명시했다.

a. Memory dump 는 2011-06-03 04:31:36 에 생성되었다.

[ANSWER]

2011-06-03 04:31:36

사용 옵션 : windows.info

windows.info의 SystemTime은 해당 Memory dump 가 언제 생성되었는지를 기록한다.



b. 의심스러운 프로세스는 아래와 같다.

[ANSWER]

csrss.exe
services.exe
lsass.exe
svchost.exe
exploar.exe

사용 옵션 : windows.malfind

악성코드와 관련된 메모리 할당을 확인하는 windows.malfind 옵션을 사용해 프로세스들을 검출했다.

csrss.exe
services.exe
lsass.exe
svchost.exe
exploar.exe



아래는 그 중 pstree를 살펴보면 특히 의심스러웠던 프로세스다.

lsass.exe - 일반적으로 한번만 실행되는 프로세스가 여러번 실행되었다.

cmd.exe - 프로세스 아래에서 cmd가 실행되었다는 건 악성코드가 cmd를 통해 명령어를 입

c. 위 프로세스 중 cmd는 cmd 자체가 악성코드가 아니라 악성코드가 cmd를 이용해 악성 행위를 했을 것으로 추정하는 것이다. 따라서 cmd의 dll 파일은 확인하지 않았다.

```
lsass.exe
services.exe
svchost.exe
```

 psscan_result.txt - Windows 메모장
  psscan_result.txt - Windows 메모장

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
1928	668	lsass.exe		
868	668	lsass.exe		

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
680	624	lsass.exe	0x2070020	

PID	Process	Base	Size	Name	Path	LoadTime
680	lsass.exe	0x1000000			0x6000	lsass.exe
680	lsass.exe	0x7c900000			0xaf000	-
680	lsass.exe	0x7c800000			0xf6000	kernel32.dll
PID	Process	Base	Size	Name	Path	LoadTime
1928	lsass.exe	0x1000000			0x6000	lsass.exe
1928	lsass.exe	0x7c900000			0xaf000	ntdll.dll
1928	lsass.exe	0x7c800000			0xf6000	kernel32.dll
PID	Process	Base	Size	Name	Path	LoadTime
868	lsass.exe	0x1000000			0x6000	lsass.exe
868	lsass.exe	0x7c900000			0xaf000	ntdll.dll
868	lsass.exe	0x7c800000			0xf6000	kernel32.dll

사용 옵션 : windows.dlllist

위와 같은 dll 파일이 services.exe와 svchost.exe에서도 실행됨을 확인하였다.

668 services.exe 0x13f0000 0x138000 KERNEL32.DLL ASLR.0360c5e2 C:\WINDOWS\system32\KERNEL32.DLL ASLR.0360c5e2

dlllist_result.txt - Windows 메모장
 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
 940 svchost.exe 0xd000000x138000KERNEL32.DLL.ASLR.0360c8ee C:\WINDOWS\system32\KERNEL32.DLL.ASLR.0360c8ee

이로 미루어 보아 lsass.exe, services.exe, svchost.exe가 가장 의심스러운 프로세스로 간주할 수 있다.

d. 의심스러운 드라이버를 찾기 위해 vmem 파일에서 driverscan으로 드라이버를 추출했다.

[ANSWER]

RAW

사용 옵션 : windows.driverscan

추출된 파일에서 의심스러운 드라이브를 하나 발견했다.

*driverscan_result.txt - Windows 메모장
 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

0x25c8278	0xf869a000	0x9180	isapnp	N/A	WDriverWisapnp
0x25e3560	0x0	0x0		N/A	WFileSystemWRAW
0x25e8160	0x0	0x0	WDriverWACPI_HAL	N/A	WDriverWACPI_HAL
0x25eb2c8	0x0	0x0	WDriverWPnpManager	N/A	WDriverW

다른 드라이브는 모두 이름이 있는데 RAW에서 시작된 드라이브만 이름이 불분명하다. 그리고 0x0로 메모리 사이즈가 0이고 시작 주소도 0x0이다. (메모리 사이즈가 0인 드라이브는 총 4개 존재했다.)

e. 위에서 의심스럽다고 판단했던 1928, 668, 940 프로세스에 대해 프로세스 덤프를 진행했다. 그 결과를 {프로세스 PID}_procdump.txt로 추출했다. 이를 함께 첨부했다.

사용한 명령은 다음과 같다.

```
python vol.py -f {stuxnet.vmem경로}/stuxnet.vmem windows.memmap --pid {pid}
```

688_procdump.txt

940_procdump.txt

1928_procdump.txt

다음은 1928 프로세스의 덤프 파일 일부이다.

그리고 RAW 드라이버 또한 캡처했으나, 사이즈가 0x0이라 아무것도 캡처되지 않은 것과 동일한 결과가 나온다.

사용한 명령은 다음과 같다.

```
python vol.py -f {stuxnet.vmem경로}\stuxnet.vmem windows.dumpfiles --virtaddr
```

1928_procdump.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

Volatility 3 Framework 2.7.0

Virtual	Physical	Size	Offset in File		File output
0x10000	0x4208000		0x1000	0x0	Disabled
0x20000	0x8147000		0x1000	0x1000	Disabled
0x6e000	0x122b8000		0x1000	0x2000	Disabled

0x25e3560 > raw_driver_dump.txt

raw_driver_dump.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

Volatility 3 Framework 2.7.0

위 파일들을 .bin으로 추출한 뒤 파이썬 코드를 이용해 sha-256으로 해싱했다. 코드는 아래와 같다.

파일 경로 및 해시값 저장 파일 경로는 매번 바꿔서 진행했다.

결과는 다음과 같이 나왔으며, 이 파일들은 전부 함께 첨부했다.

raw_driver_sha256.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

SHA-256: e169a6b7501eedc58aeecc6c7c4865a86cafe7dd4f591c0a1bfe253d70fde41

```
1  import hashlib
2  |
3  # 덤프된 파일 경로
4  dump_file_path = './raw_driver_dump.bin'
5  # 해시 값을 저장할 파일 경로
6  hash_file_path = './raw_driver_sha256.txt'
7
8  # SHA-256 해시 계산 함수
9  def calculate_sha256(file_path):
10     sha256_hash = hashlib.sha256()
11     with open(file_path, 'rb') as f:
12         for byte_block in iter(lambda: f.read(4096), b''):
13             sha256_hash.update(byte_block)
14     return sha256_hash.hexdigest()
15
16 # 해시 값 계산
17 hash_value = calculate_sha256(dump_file_path)
18
19 # 해시 값을 파일에 저장
20 with open(hash_file_path, 'w') as hash_file:
21     hash_file.write(f'SHA-256: {hash_value}\n')
22
23 print(f'SHA-256 hash value saved to {hash_file_path}')
24
```