

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN



Môn học: Phân tích mã độc
Bài thực hành số 1

Họ và tên: Trần Thị Thu Phương

Mã sinh viên: B21DCAT151

Nhóm môn học: 03

Tổ thực hành: 01

Giảng viên: Đỗ Xuân Chợt

Hà Nội, 2024

MỤC LỤC

1. PTIT-VOLATILITY	1
1.1. Mục đích	1
1.2. Yêu cầu đối với sinh viên	1
1.3. Nội dung thực hành	1
2. PTIT – MALDET	14
2.1. Mục đích	14
2.2. Yêu cầu đối với sinh viên	14
2.3. Nội dung thực hành	14
3. PTIT-RADARE2	21
3.1. Mục đích	21
3.2. Yêu cầu đối với sinh viên	21
3.3. Nội dung thực hành	21
4. PTIT – STATIC- ANALYSIS	27
4.1. Mục đích	27
4.2. Yêu cầu đối với sinh viên	28
4.3. Nội dung thực hành	28
5. PTIT-STATIC-DANABOT	39
5.1. Mục đích	39
5.2. Yêu cầu đối với sinh viên	39
5.3. Nội dung thực hành	40

1. PTIT-VOLATILITY

1.1. Mục đích

Volatility là một khung phần mềm mã nguồn mở dành cho phân tích bộ nhớ và phản ứng sự cố phần mềm độc hại. Nó được viết bằng Python và hỗ trợ các hệ điều hành như Microsoft Windows, MacOS và Linux.

Sinh viên có thể sử dụng Volatility để phân tích và trích xuất thông tin quan trọng từ một bản sao bộ nhớ. Điều này cho phép người phản hồi sự cố xác định khi một cuộc tấn công xảy ra, từ đâu nó bắt nguồn, và nhiều hơn nữa.

Volatility là một công cụ dựa trên Python cho phép **trích xuất thông tin từ bộ nhớ đệm, như các quá trình, kết nối mạng, khóa registry, mật khẩu, khóa mã hóa**, và nhiều hơn nữa. Volatility cũng có thể thực hiện các nhiệm vụ nâng cao, như phát hiện phần mềm độc hại, phân tích dòng thời gian, khắc phục lỗi bộ nhớ, và tiêm mã. Volatility hoạt động bằng cách sử dụng các plugin cụ thể cho mỗi hệ điều hành và định dạng bộ nhớ. Bạn cũng có thể tạo ra các plugin của riêng mình hoặc sử dụng các plugin do cộng đồng đóng góp.

1.2. Yêu cầu đối với sinh viên

Sinh viên cần có kiến thức cơ bản về hệ điều hành Linux và các khái niệm cơ bản về phân tích malware để hiểu và áp dụng công cụ Volatility. Hiểu biết về các công cụ như VirusTotal có thể hữu ích để bổ sung kiến thức trong quá trình sử dụng Volatility.

1.3. Nội dung thực hành

Volatility có 2 phiên bản, cụ thể là Volatility 2 và Volatility 3. Trong bài lab này, sinh viên sẽ sử dụng Volatility 2 vì nó có một số lượng lớn các plugin hữu ích.

Khởi động bài lab:

- Vào terminal, gõ:

```
labtainer ptit-volatility
```

(chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Các nhiệm vụ:

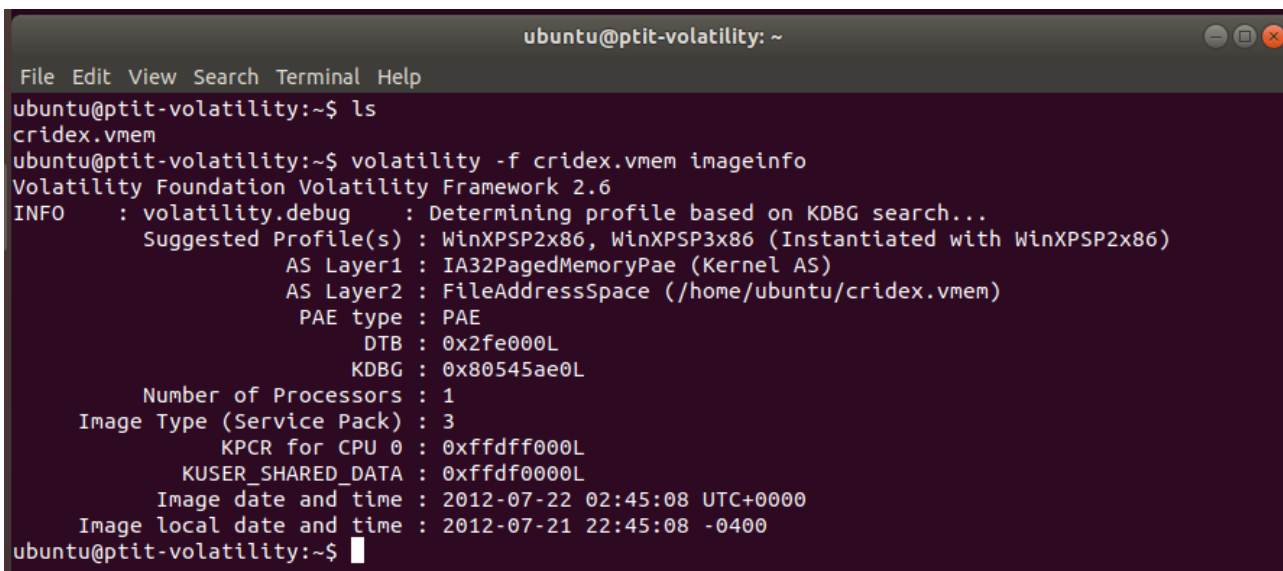
- Task 1: Xác định thông tin mẫu.

+ Sau khi khởi động xong 1 terminal ảo sẽ xuất hiện. Ở thư mục hiện tại đã có một mẫu bộ nhớ là cridex.vmem.

+ Sinh viên có thể sử dụng plugin “imageinfo” để có thông tin về mẫu bộ nhớ.

```
volatility -f cridex.vmem imageinfo
```

→ Profile của mẫu bộ nhớ này là gì?



```
ubuntu@ptit-volatility: ~  
File Edit View Search Terminal Help  
ubuntu@ptit-volatility:~$ ls  
cridex.vmem  
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem imageinfo  
Volatility Foundation Volatility Framework 2.6  
INFO : volatility.debug : Determining profile based on KDBG search...  
      Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)  
                             AS Layer1 : IA32PagedMemoryPae (Kernel AS)  
                             AS Layer2 : FileAddressSpace (/home/ubuntu/cridex.vmem)  
                             PAE type : PAE  
                             DTB : 0x2fe000L  
                             KDBG : 0x80545ae0L  
      Number of Processors : 1  
      Image Type (Service Pack) : 3  
      KPCR for CPU 0 : 0xffdff000L  
      KUSER_SHARED_DATA : 0xffdf0000L  
      Image date and time : 2012-07-22 02:45:08 UTC+0000  
      Image local date and time : 2012-07-21 22:45:08 -0400  
ubuntu@ptit-volatility:~$
```

Suggest profiles: WinXPSP2x86 hoặc WinXPSP3x86

+ Thông tin này cho khởi đầu cần thiết để thực hiện lệnh phân tích tiếp theo bằng các plugin khác. Sinh viên cần phải ghi nhớ Suggested Profiles để thay thế vào các lệnh sau này.

--profile=PROFILE (cần phải thay thế PROFILE)

--profile=WinXPSP2x86 hoặc --profile=WinXPSP3x86

- Task 2: Xác định tiến trình độc hại.

+ Có một số plugin trong Volatility cho phép xem các tiến trình đang chạy. Một số trong số này bao gồm: psscan, pstree, pslist, psxview.

+ Sinh viên có thể xem những tiến trình đã được chấm dứt trước đó và những tiến trình nào đã bị rootkit hủy liên kết bằng cách sử dụng “psscan”.

volatility -f cridex.vmem --profile=PROFILE psscan

```
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 psscan
```

Volatility Foundation Volatility Framework 2.6

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x000000002029ab8	svchost.exe	908	652	0x079400e0	2012-07-22 02:42:33 UTC+0000	
0x00000000202a3b8	lsass.exe	664	608	0x079400a0	2012-07-22 02:42:32 UTC+0000	
0x00000000202ab28	services.exe	652	608	0x07940080	2012-07-22 02:42:32 UTC+0000	
0x00000000207bda0	reader_sl.exe	1640	1484	0x079401e0	2012-07-22 02:42:36 UTC+0000	
0x0000000020b17b8	spoolsv.exe	1512	652	0x079401c0	2012-07-22 02:42:36 UTC+0000	
0x00000000225bda0	wuauclt.exe	1588	1004	0x07940200	2012-07-22 02:44:01 UTC+0000	
0x0000000022e8da0	alg.exe	788	652	0x07940140	2012-07-22 02:43:01 UTC+0000	
0x0000000023dea70	explorer.exe	1484	1464	0x079401a0	2012-07-22 02:42:36 UTC+0000	
0x0000000023dfda0	svchost.exe	1056	652	0x07940120	2012-07-22 02:42:33 UTC+0000	

+ Plugin “pslist” hiển thị địa chỉ bắt đầu, tên tiến trình, PID, PPID, số lượng luồng, số lượng handles, và ngày/giờ.

volatility -f cridex.vmem --profile=PROFILE pslist

```
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 pslist
```

Volatility Foundation Volatility Framework 2.6

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0x823c89c8	System	4	0	53	240	-----	0		
0x822f1020	smss.exe	368	4	3	19	-----	0	2012-07-22 02:42:31 UTC+0000	
0x822a0598	csrss.exe	584	368	9	326	0	0	2012-07-22 02:42:32 UTC+0000	
0x82298700	winlogon.exe	608	368	23	519	0	0	2012-07-22 02:42:32 UTC+0000	
0x81e2ab28	services.exe	652	608	16	243	0	0	2012-07-22 02:42:32 UTC+0000	
0x81e2a3b8	lsass.exe	664	608	24	330	0	0	2012-07-22 02:42:32 UTC+0000	
0x82311360	svchost.exe	824	652	20	194	0	0	2012-07-22 02:42:33 UTC+0000	
0x81e29ab8	svchost.exe	908	652	9	226	0	0	2012-07-22 02:42:33 UTC+0000	
0x823001d0	svchost.exe	1004	652	64	1118	0	0	2012-07-22 02:42:33 UTC+0000	
0x821dfda0	svchost.exe	1056	652	5	60	0	0	2012-07-22 02:42:33 UTC+0000	

- **Offset(V):** Địa chỉ offset của tiến trình trong không gian bộ nhớ ảo (virtual memory). Thông tin này có thể liên quan đến cách tiến trình đang được ánh xạ trong bộ nhớ.
- **Name:** Tên của tiến trình hoặc ứng dụng đang chạy.
- **PID (Process ID):** Mã định danh duy nhất của tiến trình.
- **PPID (Parent Process ID):** Mã định danh của tiến trình cha. Điều này cho biết tiến trình nào đã tạo ra tiến trình hiện tại.
- **Thds (Threads):** Số lượng luồng (threads) mà tiến trình đang sử dụng.
- **Hnds (Handles):** Số lượng handles mà tiến trình đã mở. Handles là các con trỏ tham chiếu tới tài nguyên hệ thống, như tệp, thiết bị hoặc các đối tượng khác.

- **Sess (Session ID):** ID của phiên đăng nhập mà tiến trình đang chạy. Điều này thường hữu ích trong môi trường đa người dùng.
- **Wow64:** Chỉ báo cho biết liệu tiến trình có đang chạy trong Windows-on-Windows 64-bit (WOW64) hay không, giúp các ứng dụng 32-bit chạy trên hệ điều hành 64-bit.
- **Start:** Thời gian tiến trình bắt đầu chạy.
- **Exit:** Thời gian tiến trình kết thúc hoặc thoát khỏi hệ thống, nếu có.

+ Ngoài ra, sinh viên cũng có thể sử dụng plugin “pstree” như một phương thức thay thế cho pslist nhưng với cấu trúc cây để thể hiện mối quan hệ giữa các tiến trình:

volatility -f cridex.vmem --profile=PROFILE pstree

```
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 pstree
Volatility Foundation Volatility Framework 2.6
Name                               Pid  PPid  Thds  Hnds  Time
-----
0x823c89c8:System                   4      0    53   240  1970-01-01 00:00:00 UTC+0000
. 0x822f1020:smss.exe               368     4     3    19  2012-07-22 02:42:31 UTC+0000
.. 0x82298700:winlogon.exe          608   368    23   519  2012-07-22 02:42:32 UTC+0000
... 0x81e2ab28:services.exe         652   608    16   243  2012-07-22 02:42:32 UTC+0000
.... 0x821dfda0:svchost.exe          1056  652     5    60  2012-07-22 02:42:33 UTC+0000
.... 0x81eb17b8:spoolsv.exe           1512  652    14   113  2012-07-22 02:42:36 UTC+0000
.... 0x81e29ab8:svchost.exe           908   652     9   226  2012-07-22 02:42:33 UTC+0000
.... 0x823001d0:svchost.exe           1004  652    64  1118  2012-07-22 02:42:33 UTC+0000
..... 0x8205bda0:wuauc.lt.exe          1588  1004     5   132  2012-07-22 02:44:01 UTC+0000
..... 0x821fcd0:wuauc.lt.exe           1136  1004     8   173  2012-07-22 02:43:46 UTC+0000
.... 0x82311360:svchost.exe           824   652    20   194  2012-07-22 02:42:33 UTC+0000
.... 0x820e8da0:alg.exe                788   652     7   104  2012-07-22 02:43:01 UTC+0000
.... 0x82295650:svchost.exe           1220  652    15   197  2012-07-22 02:42:35 UTC+0000
... 0x81e2a3b8:lsass.exe               664   608    24   330  2012-07-22 02:42:32 UTC+0000
.. 0x822a0598:csrss.exe               584   368     9   326  2012-07-22 02:42:32 UTC+0000
0x821dea70:explorer.exe            1484  1464    17   415  2012-07-22 02:42:36 UTC+0000
. 0x81e7bda0:reader_sl.exe            1640  1484     5    39  2012-07-22 02:42:36 UTC+0000
ubuntu@ptit-volatility:~$
```

- ➔ Trong danh sách này có 2 tiến trình lạ, sinh viên hãy cho biết 2 tiến trình đó là gì?
- ➔ PID của chúng là gì? Tiến trình nào là tiến trình cha và tiến trình nào là tiến trình con?

Trong danh sách các tiến trình từ lệnh ‘pstree’, hai tiến trình lạ là:

wuauc.lt.exe với PID 1588 có PPID 1004

wuauc.lt.exe với PID 1136 có PPID 1004

PID 1588 và PID 1136 đều có tên tiến trình là wuauc.lt.exe, đây là tên của dịch vụ Windows Update AutoUpdate Client. Việc có hai tiến trình wuauc.lt.exe chạy cùng lúc có thể là dấu hiệu của hoạt động đáng ngờ hoặc bị lợi dụng bởi phần mềm độc hại (malware).

+ Trước khi tiếp tục, hãy chạy plugin “psxview” để tham chiếu chéo các tiến trình với nhiều danh sách khác nhau. “psxview” được sử dụng để xem và phân tích các tiến trình ẩn và các kỹ thuật che giấu khác trên hệ thống.

volatility -f cridex.vmem --profile=PROFILE psxview

- ➔ Trong kết quả có phát hiện ra tiến trình ẩn nào không? Tại sao?

```

0 01c10000
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 psxview
Volatility Foundation Volatility Framework 2.6
Offset(P) Name PID pslist psscan thrdproc pspcid csrss session deskthrd ExitT
ime
-----
---
0x02498700 winlogon.exe 608 True True True True True True True
0x02511360 svchost.exe 824 True True True True True True True
0x022e8da0 alg.exe 788 True True True True True True True
0x020b17b8 spoolsv.exe 1512 True True True True True True True
0x0202ab28 services.exe 652 True True True True True True True
0x02495650 svchost.exe 1220 True True True True True True True
0x0207bda0 reader_sl.exe 1640 True True True True True True True
0x025001d0 svchost.exe 1004 True True True True True True True
0x02029ab8 svchost.exe 908 True True True True True True True
0x023fcda0 wuauclt.exe 1136 True True True True True True True
0x0225bda0 wuauclt.exe 1588 True True True True True True True
0x0202a3b8 lsass.exe 664 True True True True True True True
0x023dea70 explorer.exe 1484 True True True True True True True
0x023dfda0 svchost.exe 1056 True True True True True True True
0x024f1020 smss.exe 368 True True True True False False False
0x025c89c8 System 4 True True True True False False False
0x024a0598 csrss.exe 584 True True True True False True True
ubuntu@ptit-volatility:~$

```

Dựa trên kết quả cung cấp, không phát hiện ra tiến trình ẩn nào. Để giải thích:

- **Các cột:**

- **pslist:** Hiển thị tiến trình nếu nó xuất hiện trong danh sách tiến trình thông qua plugin pslist.
- **psscan:** Hiển thị tiến trình nếu nó xuất hiện trong danh sách tiến trình thông qua plugin psscan.
- **thrdproc, pspcid, csrss, session, deskthrd:** Các cột này thể hiện các thông tin về tiến trình được tìm thấy qua nhiều phương pháp khác nhau (quản lý tiến trình, kết nối với các thành phần hệ thống khác).

Phân tích:

- Tất cả các tiến trình trong bảng đều có giá trị **True** ở cột **pslist** và **psscan**, có nghĩa là chúng đều xuất hiện khi quét qua cả danh sách tiến trình đang hoạt động và quét qua bộ nhớ để tìm dấu vết tiến trình. Điều này cho thấy rằng không có tiến trình nào bị ẩn một cách rõ ràng (ví dụ như bằng cách không xuất hiện trong một trong hai phương pháp quét).
- Tuy nhiên, có một vài tiến trình có giá trị **False** trong các cột **pspcid** (Process CID) và **session**, ví dụ như:
 - **smss.exe** (PID 368)
 - **System** (PID 4)
 - **csrss.exe** (PID 584)

Điều này có thể là do các tiến trình hệ thống đặc biệt như **smss.exe** (Session Manager Subsystem) và **System** không luôn luôn hiển thị đầy đủ thông tin khi kiểm tra qua các plugin như pspcid. Tuy nhiên, đây là các tiến trình hệ thống hợp lệ và không được coi là tiến trình ẩn.

+ “cmdline” là plugin được sử dụng để trích xuất và hiển thị các đối số của dòng lệnh được sử dụng khi khởi động các tiến trình trong hệ thống.

volatility -f cridex.vmem --profile=PROFILE cmdline

```
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=winXPSP2x86 cmdline
Volatility Foundation Volatility Framework 2.6
*****
System pid:      4
*****
smss.exe pid:    368
Command line :   \SystemRoot\System32\smss.exe
*****
csrss.exe pid:   584
Command line :   C:\WINDOWS\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,3072,512
                Windows=0n SubSystemType=Windows ServerDll=basesrv,1 ServerDll=win32srv:UserServerDllInitializatio
                n,3 ServerDll=win32srv:ConServerDllInitialization,2 ProfileControl=Off MaxRequestThreads=16
*****
winlogon.exe pid: 608
Command line :   winlogon.exe
*****
services.exe pid: 652
Command line :   C:\WINDOWS\system32\services.exe
*****
lsass.exe pid:   664
Command line :   C:\WINDOWS\system32\lsass.exe
*****
svchost.exe pid: 824
Command line :   C:\WINDOWS\system32\svchost -k DcomLaunch
*****
```

➔ Sinh viên hãy cho biết đường dẫn của tiến trình độc hại là gì?

C:\WINDOWS\system32\wuaucit.exe" /RunStoreAsComServer
Local\{3ec}SUSDSb81eb56fa3105543beb3109274ef8ec1

- **Task 3: Xác định kết nối mạng độc hại.**

+ Để xem các kết nối mở trên máy tính. Sinh viên có thể chạy các plugin như “sockets”.

volatility -f cridex.vmem --profile=PROFILE sockets


```

ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 sockets
Volatility Foundation Volatility Framework 2.6
Offset(V)      PID      Port      Proto Protocol      Address      Create Time
-----
0x81ddb780     664      500       17 UDP           0.0.0.0      2012-07-22 02:42:53 UTC+0000
0x82240d08    1484     1038       6 TCP           0.0.0.0      2012-07-22 02:44:45 UTC+0000
0x81dd7618    1220     1900       17 UDP          172.16.112.128 2012-07-22 02:43:01 UTC+0000
0x82125610     788     1028       6 TCP           127.0.0.1     2012-07-22 02:43:01 UTC+0000
0x8219cc08       4      445       6 TCP           0.0.0.0      2012-07-22 02:42:31 UTC+0000
0x81ec23b0     908      135       6 TCP           0.0.0.0      2012-07-22 02:42:33 UTC+0000
0x82276878       4      139       6 TCP          172.16.112.128 2012-07-22 02:42:38 UTC+0000
0x82277460       4      137       17 UDP          172.16.112.128 2012-07-22 02:42:38 UTC+0000
0x81e76620    1004      123       17 UDP          127.0.0.1     2012-07-22 02:43:01 UTC+0000
0x82172808     664       0      255 Reserved    0.0.0.0      2012-07-22 02:42:53 UTC+0000
0x81e3f460       4      138       17 UDP          172.16.112.128 2012-07-22 02:42:38 UTC+0000
0x821f0630    1004      123       17 UDP          172.16.112.128 2012-07-22 02:43:01 UTC+0000
0x822cd2b0    1220     1900       17 UDP          127.0.0.1     2012-07-22 02:43:01 UTC+0000
0x82172c50     664     4500       17 UDP           0.0.0.0      2012-07-22 02:42:53 UTC+0000
0x821f0d00       4      445       17 UDP           0.0.0.0      2012-07-22 02:42:31 UTC+0000
ubuntu@ptit-volatility:~$

```

+ Ngoài ra, “connscan” là một công cụ quét các kết nối TCP và sockets sẽ phát hiện các socket đang lắng nghe cho bất kỳ giao thức nào (TCP, RAW, UDP, vv).

volatility -f cridex.vmem --profile=PROFILE connscan

```

ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 connscan
Volatility Foundation Volatility Framework 2.6
Offset(P)      Local Address      Remote Address      Pid
-----
0x02087620 172.16.112.128:1038 41.168.5.140:8080    1484
0x023a8008 172.16.112.128:1037 125.19.103.198:8080 1484
ubuntu@ptit-volatility:~$

```

➔ Có kết nối nào đang được sử dụng bởi PID vừa tìm được ở nhiệm vụ kia không?

➔ Kết nối đó giao tiếp với địa chỉ IP nào? Qua cổng nào ở địa chỉ local?

Trong kết quả của lệnh connscan, có hai kết nối mạng đáng chú ý liên quan đến PID 1484:

- Kết nối thứ nhất:
 - o Địa chỉ local: 172.16.112.128
 - o Cổng local: 1038
 - o Địa chỉ remote: 41.168.5.140
 - o Cổng remote: 8080
- Kết nối thứ hai:
 - o Địa chỉ local: 172.16.112.128
 - o Cổng local: 1037
 - o Địa chỉ remote: 125.19.103.198
 - o Cổng remote: 8080

Kết luận:

- PID 1484 (tương ứng với tiến trình explorer.exe theo kết quả trước đó) có hai kết nối đang được sử dụng:
 - Kết nối đến địa chỉ IP: 41.168.5.140 qua cổng 8080 từ cổng local 1038.
 - Kết nối đến địa chỉ IP: 125.19.103.198 qua cổng 8080 từ cổng local 1037.

- Task 4: Trích xuất tiến trình và bộ nhớ độc hại.

+ Hoạt động bất thường từ hai tiến trình (explorer.exe và reader_sl.exe) gợi ý rằng có thể có một số mã độc được chèn vào. Để kiểm tra độ chính xác, sinh viên có thể sử dụng plugin “malfind” để xác nhận.

volatility -f cridex.vmem --profile=PROFILE malfind

```
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 malfind
Volatility Foundation Volatility Framework 2.6
Process: csrss.exe Pid: 584 Address: 0x7f6f0000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x7f6f0000 c8 00 00 00 91 01 00 00 ff ee ff ee 08 70 00 00 .....p..
0x7f6f0010 08 00 00 00 00 fe 00 00 00 00 10 00 00 20 00 00 .....
0x7f6f0020 00 02 00 00 00 20 00 00 8d 01 00 00 ff ef fd 7f .....
0x7f6f0030 03 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00 .....

0x7f6f0000 c8000000 ENTER 0x0, 0x0
0x7f6f0004 91 XCHG ECX, EAX
0x7f6f0005 0100 ADD [EAX], EAX
0x7f6f0007 00ff ADD BH, BH
0x7f6f0009 ee OUT DX, AL
0x7f6f000a ff DB 0xff
0x7f6f000b ee OUT DX, AL
0x7f6f000c 087000 OR [EAX+0x0], DH
0x7f6f000f 0008 ADD [EAX], CL
0x7f6f0011 0000 ADD [EAX], AL
0x7f6f0013 0000 ADD [EAX], AL
0x7f6f0015 fe00 INC BYTE [EAX]
0x7f6f0017 0000 ADD [EAX], AL
0x7f6f0019 0010 ADD [EAX], DL
0x7f6f001b 0000 ADD [EAX], AL
0x7f6f001d 2000 AND [EAX], AL
0x7f6f001f 0000 ADD [EAX], AL
```

+ Tiếp theo, sinh viên có thể phân tích sâu hơn bằng cách sử dụng các plugin “memdump” và “procdump” để trích xuất tiến trình và bộ nhớ.

volatility -f cridex.vmem --profile=PROFILE procdump -p 1640 --dump-dir .

volatility -f cridex.vmem --profile=PROFILE procdump -p 1484 --dump-dir .

volatility -f cridex.vmem --profile=PROFILE memdump -p 1640 --dump-dir .

volatility -f cridex.vmem --profile=PROFILE memdump -p 1484 --dump-dir .

```

ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 procdump -p 1640 --dump
-dir .
Volatility Foundation Volatility Framework 2.6
Process(V) ImageBase Name Result
-----
0x81e7bda0 0x00400000 reader_sl.exe OK: executable.1640.exe
ubuntu@ptit-volatility:~$ ls
cridex.vmem dumps executable.1640.exe
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 procdump -p 1484 --dump
-dir .
Volatility Foundation Volatility Framework 2.6
Process(V) ImageBase Name Result
-----
0x821dea70 0x01000000 explorer.exe OK: executable.1484.exe
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 memdump -p 1640 --dump-
-dir .
Volatility Foundation Volatility Framework 2.6
*****
Writing reader_sl.exe [ 1640] to 1640.dmp
ubuntu@ptit-volatility:~$ volatility -f cridex.vmem --profile=WinXPSP2x86 memdump -p 1484 --dump-
-dir .
Volatility Foundation Volatility Framework 2.6
*****
Writing explorer.exe [ 1484] to 1484.dmp
ubuntu@ptit-volatility:~$

```

+ Để phân tích các lệnh file **.dmp**, sinh viên có thể sử dụng “strings” và “grep” để tìm mối tương quan với địa chỉ IP 41.168.5.140.

```
strings 1640.dmp | grep -Fi "41.168.5.140" -C 5
```

```
strings 1484.dmp | grep -Fi "41.168.5.140" -C 5
```

```

ubuntu@ptit-volatility:~$ strings 1640.dmp | grep -Fi "41.168.5.140" -C 5
ABACFPFPENFDECFCFPFHDEFFFPACAB
DpI8
POST /zb/v_01_a/in/ HTTP/1.1
Accept: */*
User-Agent: Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)
Host: 41.168.5.140:8080
Content-Length: 229
Connection: Keep-Alive
Cache-Control: no-cache
>mtvR
`06!
ubuntu@ptit-volatility:~$ strings 1484.dmp | grep -Fi "41.168.5.140" -C 5
Documents and Settings
All Users
/zb/v_01_a/in/ HTTP/1.1
Accept: */*

```

+ Nếu chạy “strings” và “less”, chúng ta có thể tìm thấy một lượng đáng kể các tên miền ngân hàng và tài chính.

```
strings 1640.dmp | less
```

```
strings 1484.dmp | less
```

```
*treasurypathways.com*  
*CorporateAccounts*  
*weblink.websterbank.com*  
*secure7.onlineaccess1.com*  
*trz.tranzact.org*  
*onlineaccess1.com*  
*secureport.texascapitalbank.com*  
*/Authentication/zbf/k/*  
*ebc_ebc1961*  
*tdbank.com*  
*online.ovcb.com*  
*ebanking-services.com*  
*schwab.com*  
*billmelater.com*  
*chase.com*  
*bankofamerica.com*  
*pnc.com*  
*suntrust.com*  
*wellsfargo.com*  
*ibanking-services.com*  
*bankonline.umpquabank.com*  
*servlet/teller*  
*nsbank.com*  
*secureentry.calbanktrust.com*  
*secureentry*  
*/Common/SignOn/Start.asp*  
*telepc.net*  
*enterprise2.openbank.com*
```

- ➔ Sinh viên hãy cho liệt kê 1 số các tên miền và các địa chỉ IP đáng nghi có trong bộ nhớ.

```
*banking.calbanktrust.com*
*towernet.capitalonebank.com*
*.com/K1/*
*businessaccess.citibank.citigroup.com*
*achieveaccess.citizensbank.com*
*www8.comerica.com*
*businessclassonline.compassbank.com*
*cashanalyzer.com*
*ebanking-services.com*
*express.53.com*
*cbs.firstcitizens.com*
*efirstbank.com*
*treas-mgt.frostbank.com*
*businessonline.huntington.com*
*ibbpowerlink.com*
*access.jpmorgan.com*
*blilk.com*
*businessportal.mibank.com*
*webbankingforbusiness.mandtbank.com*
*mbachexpress.com*
*cashmanager.mizuhoe-treasurer.com*
*enternetbank.com*
*ntrs.com*
*northerntrust.com*
*treasury.pncbank.com*
*rbs.com/wps/portal/*
*sandyspringbank.com*
*ssl.selectpayment.com/mp*
*svbconnect.com*
```

+ Còn 2 file **.exe** vừa trích xuất được, để phân tích chi tiết hơn thì chúng ta cần làm phân tích tĩnh và động chuyên sâu. Tuy nhiên, sinh viên có thể tạo mã hash và kiểm tra trên các trang công cụ phân tích online như VirusTotal hay Intezer để biết thêm thông tin.

md5sum <file>

```
ubuntu@ptit-volatility:~$ md5sum executable.1484.exe
f5d61a0ccf96e07228a4818918aa33e8 executable.1484.exe
ubuntu@ptit-volatility:~$ md5sum executable.1640.exe
12cf6583f5a9171a1d621ae02b4eb626 executable.1640.exe
ubuntu@ptit-volatility:~$
```

➔ Sinh viên hãy cho biết 2 chương trình bất thường kia là loại mã độc gì?

39

/ 72

Community Score

-3

39/72 security vendors flagged this file as malicious

Reanalyze Similar More

48db195007e5ae9fc1246506564af154927e9f3fbca0b4054552804027abfb2

Size

1009.50 KB

Last Analysis Date

10 days ago

EXE

executable.1484.exe

peexe checks-user-input idle detect-debug-environment

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 8

[Join our Community](#) and enjoy additional community insights and crowdsourced detections, plus an API key to [automate checks](#).

Popular threat label

trojan.multip/amvb

Threat categories

trojan pua dropper

Family labels

multip amvb redcap

26

/ 72

Community Score

-13

26/72 security vendors flagged this file as malicious

Reanalyze Similar More

5b136147911b041f0126ce82dfd24c4e2c79553b65d3240ecea2dcab4452dcb5

Size

28.50 KB

Last Analysis Date

21 days ago

EXE

AcroSpeedLaunch.exe

peexe idle checks-user-input direct-cpu-clock-access

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 6

[Join our Community](#) and enjoy additional community insights and crowdsourced detections, plus an API key to [automate checks](#).

Popular threat label

trojan.multip

Threat categories

trojan pua

Family labels

multip

+ Sau đó, sinh viên kết nối đến 1 container khác thông qua ssh bằng lệnh sau:

ssh ubuntu@192.168.0.20

```
ubuntu@ptit-volatility:~/dumps$ ls
1484.dmp 1640.dmp executable.1484.exe executable.1640.exe
ubuntu@ptit-volatility:~/dumps$ md5sum executable.1484.exe
f5d61a0ccf96e07228a4818918aa33e8 executable.1484.exe
ubuntu@ptit-volatility:~/dumps$ md5sum executable.1640.exe
12cf6583f5a9171a1d621ae02b4eb626 executable.1640.exe
ubuntu@ptit-volatility:~/dumps$ ssh ubuntu@192.168.0.20
The authenticity of host '192.168.0.20 (192.168.0.20)' can't be established.
ECDSA key fingerprint is SHA256:ZtE8xi5Y50aUktZ/XtgjIs1c5jxYQB84Vq5ofmlgGng.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.20' (ECDSA) to the list of known hosts.
ubuntu@192.168.0.20's password:
Permission denied, please try again.
ubuntu@192.168.0.20's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-150-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

ubuntu@server:~$
```


+ Mật khẩu chính là 1 trong 2 mã hash md5 mà sinh viên vừa tìm được ở bước trên. Tiếp theo, sinh viên hãy in ra màn hình tệp tin filetoview:

cat filetoview

```
ubuntu@ptit-volatility:~/dumps$ ls
1484.dmp 1640.dmp executable.1484.exe executable.1640.exe
ubuntu@ptit-volatility:~/dumps$ md5sum executable.1484.exe
f5d61a0ccf96e07228a4818918aa33e8 executable.1484.exe
ubuntu@ptit-volatility:~/dumps$ md5sum executable.1640.exe
12cf6583f5a9171a1d621ae02b4eb626 executable.1640.exe
ubuntu@ptit-volatility:~/dumps$ ssh ubuntu@192.168.0.20
The authenticity of host '192.168.0.20 (192.168.0.20)' can't be established.
ECDSA key fingerprint is SHA256:ZtE8xi5Y50aUktZ/XtgjIs1c5jxYQB84Vq5ofmlgGng.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.20' (ECDSA) to the list of known hosts.
ubuntu@192.168.0.20's password:
Permission denied, please try again.
ubuntu@192.168.0.20's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-150-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

ubuntu@server:~$
```

Kết thúc bài lab:

- Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

stoplab ptit- volatility

- Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

Khởi động lại bài lab:

- Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

labtainer -r ptit- volatility

```
student@ubuntu:~/labtainer/labtainer-student$ checkwork
Results stored in directory: /home/student/labtainer_xfer/ptit-volatility
Labname ptit-volatility

Student          | imageinfo_check | process_check | network_check |
=====          | =====          | =====          | =====          |
B21DCAT151      | Y                | Y              | Y              |
What is automatically assessed for this lab:
```

2. PTIT – MALDET

2.1. Mục đích

Bài thực hành này giới thiệu công cụ Linux Malware Detect. Bài thực hành giúp sinh viên hiểu về khái niệm và biết cách sử dụng công cụ rà quét mã độc Linux Malware Detect.

2.2. Yêu cầu đối với sinh viên

Có kiến thức cơ bản về hệ điều hành Linux, khả năng phát hiện và phân tích mã độc.

2.3. Nội dung thực hành

- Khởi động bài lab:
 - Vào terminal, gõ:

labtainer ptit-maldet

(Chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Sau khi khởi động xong một terminal ảo sẽ xuất hiện, đại diện cho máy **ptit-maldet**

- Trên terminal **ptit-maldet** sử dụng lệnh "**maldet -h**" xem các chức năng được cung cấp bởi công cụ và nhận được thông báo công cụ cần được cấu hình để có thể hoạt động.
- Sinh viên làm các nhiệm vụ dưới đây để hoàn thành bài lab:
 - **Nhiệm vụ 1: Cấu hình và rà quét mã độc với maldet**
 - Linux Malware Detect (LMD) là một công cụ mã nguồn mở được thiết kế để phát hiện và loại bỏ malware trên hệ thống Linux. Nó sử dụng cơ sở dữ liệu chữ ký để so sánh các tập tin trên hệ thống với các mẫu malware đã biết và mẫu nghi ngờ. Công cụ này thường được sử dụng như một phần của chiến lược bảo mật đa lớp, bổ sung cho các giải pháp antivirus khác và các biện pháp bảo mật khác trên hệ thống Linux.

Bảng 1.1: Một số lệnh trong maldet

Câu lệnh	Mô tả
-a	Để quét tất cả các tập tin trong đường dẫn

-b	Để thực hiện các hoạt động ở chế độ nền.
-c	Để tải tệp phần mềm độc hại bị nghi ngờ lên rfxn.com để xem xét và bấm thành chữ ký
-d	Để cập nhật phiên bản đã cài đặt
-e	Để xem lần quét gần đây nhất hoặc ID quét cụ thể và báo cáo quét email tới địa chỉ email được cung cấp
-h	Để liệt kê tất cả các tùy chọn trợ giúp có sẵn của maldet.
-l	Để xem các sự kiện trong tệp nhật ký maldet
-n	Để dọn dẹp và khôi phục các lượt truy cập phần mềm độc hại từ báo cáo
-p	Để xóa nhật ký, phiên và dữ liệu tạm thời
-q	Để cách ly tất cả phần mềm độc hại khỏi báo cáo
-r	Để quét tệp được tạo hoặc sửa đổi trong X ngày qua (7 ngày theo mặc định)
-s	Để khôi phục tệp đã cách ly từ hàng đợi cách ly về đường dẫn ban đầu hoặc khôi phục tất cả các tệp đã cách ly từ một ID quét cụ thể
-u	Để cập nhật chữ ký phát hiện phần mềm độc hại.

- Đầu tiên sử dụng lệnh "*sudo nano /usr/local/maldetect/conf.maldet*" để truy cập và chỉnh sửa file cấu hình
 - ✓ Thay đổi trường *scan_user_access="0"* thành 1 để bật chế độ rà quét công cụ với quyền root. Thay đổi trường *scan_ignore_root="1"* thành 0 để cho phép công cụ có quyền rà quét các file do root sở hữu.

```
ubuntu@ptit-maldet: ~  
File Edit View Search Terminal Help  
GNU nano 4.8 /usr/local/maldetect/conf.maldet  
  
# Include the scanning of known temporary world-writable paths for  
# -a|--all and -r|--recent scan types.  
scan_tmpdir_paths="/tmp /var/tmp /dev/shm /var/fcgi_ipc"  
  
# Allows non-root users to perform scans. This must be enabled when  
# using mod_security2 upload scanning or if you want to allow users  
# to perform scans. When enabled, this will populate 'pub/' with user  
# owned quarantine, session and temporary paths to facilitate scans.  
# [ 0 = disabled, 1 = enabled, disabled by default ]  
scan_user_access="1"  
  
# Process CPU scheduling (nice) priority level for scan operations.  
# [ -19 = high prio , 19 = low prio, default = 19 ]  
scan_cpunice="19"  
  
# Process IO scheduling (ionice) priority levels for scan operations.  
# (uses cbq best-effort scheduling class [-c2])  
# [ 0 = most favorable IO, 7 = least favorable IO ]  
scan_ionice="6"  
  
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C  
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^
```

- ✓ Chạy lệnh `cat /usr/local/maldetect/conf.maldet`, maldet sẽ hiển thị một loạt cấu hình cho công cụ và lệnh `maldet -h` để xem các chức năng giúp công cụ hoạt động.
- Sau khi xác định được công cụ hoạt động thành công. Sử dụng lệnh `sudo maldet -a /home/ubuntu/malware2` để rà quét thư mục `/malware2` được cài sẵn trên máy ảo.

```

ubuntu@ptit-maldet:~$ ls
malware2
ubuntu@ptit-maldet:~$ sudo maldet -a /home/ubuntu/malware2
Linux Malware Detect v1.6.5
      (C) 2002-2023, R-fx Networks <proj@rfxn.com>
      (C) 2023, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms of the GNU GPL v2

maldet(435): {scan} signatures loaded: 17638 (14801 MD5 | 2054 HEX | 783 YARA | 0 USER)
maldet(435): {scan} building file list for /home/ubuntu/malware2, this might take awhile...
maldet(435): {scan} setting nice scheduler priorities for all operations: cpunice 19 , ionice 6
maldet(435): {scan} file list completed in 0s, found 7 files...
maldet(435): {scan} scan of /home/ubuntu/malware2 (7 files) in progress...
maldet(435): {scan} 7/7 files scanned: 3 hits 0 cleaned

maldet(435): {scan} scan completed on /home/ubuntu/malware2: files 7, malware hits 3, cleaned hits 0
, time 1s
maldet(435): {scan} scan report saved, to view run: maldet --report 241022-0531.435
maldet(435): {scan} quarantine is disabled! set quarantine_hits=1 in conf.maldet or to quarantine re
sults run: maldet -q 241022-0531.435
ubuntu@ptit-maldet:~$

```

```

File Edit View Search Terminal Help
ubuntu@ptit-maldet:~$ sudo nano /usr/local/maldetect/conf.maldet
ubuntu@ptit-maldet:~$ sudo maldet -a /home/ubuntu/malware2/
Linux Malware Detect v1.6.5
      (C) 2002-2023, R-fx Networks <proj@rfxn.com>
      (C) 2023, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms of the GNU GPL v2

maldet(388): {scan} signatures loaded: 17638 (14801 MD5 | 2054 HEX | 783 YARA | 0 USER)
maldet(388): {scan} building file list for /home/ubuntu/malware2/, this might take awhile...
maldet(388): {scan} setting nice scheduler priorities for all operations: cpunice 19 , ionice 6
maldet(388): {scan} file list completed in 0s, found 7 files...
maldet(388): {scan} scan of /home/ubuntu/malware2/ (7 files) in progress...
maldet(388): {scan} 7/7 files scanned: 3 hits 0 cleaned

maldet(388): {scan} scan completed on /home/ubuntu/malware2/: files 7, malware hits 3, cleaned hits 0, time 6s
maldet(388): {scan} scan report saved, to view run: maldet --report 241022-0557.388

```

○ *Nhiệm vụ 2: Xem chi tiết báo cáo rà quét của công cụ*

- *Lệnh --report của công cụ maldet được sử dụng để tạo báo cáo về kết quả quét của chương trình. Khi chạy maldet với tùy chọn --report, nó sẽ tạo ra một báo cáo chi tiết về các tập tin mà chương trình đã phát hiện là có khả năng chứa mã độc hại. Báo cáo này thường chứa thông tin về các tập tin, loại malware, đường dẫn, và các hành động khuyến nghị.*

```

ubuntu@ptit-maldet: ~
File Edit View Search Terminal Help
ubuntu@ptit-maldet:~$ sudo maldet --report 241022-0531.435
Linux Malware Detect v1.6.5
      (C) 2002-2023, R-fx Networks <proj@rfxn.com>
      (C) 2023, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms of the GNU GPL v2

ubuntu@ptit-maldet:~$

```

- Sau khi chạy lệnh `sudo --report [SCAN_ID]`, maldet sẽ tạo ra một báo cáo về kết quả của quét và hiển thị thông tin tương ứng trên màn hình. Trong đó, `SCAN_ID` là một định danh duy nhất được tạo ra mỗi khi sinh viên chạy rà quét. `SCAN_ID` giúp xác định phiên quét cụ thể và cho phép sinh viên theo dõi kết quả của quét đó. Sinh viên có thể kiểm tra báo cáo này để xem thông tin chi tiết về các tập tin bị nghi ngờ và các hành động khuyến nghị để xử lý chúng.

```

ubuntu@ptit-maldet: ~
File Edit View Search Terminal Help
GNU nano 4.8 /usr/local/maldetect/sess/session.241022-0531.435
HOST:      ptit-maldet
SCAN ID:   241022-0531.435
STARTED:   Oct 22 2024 05:31:44 +0000
COMPLETED: Oct 22 2024 05:31:45 +0000
ELAPSED:   1s [find: 0s]

PATH:      /home/ubuntu/malware2
TOTAL FILES: 7
TOTAL HITS: 3
TOTAL CLEANED: 0

WARNING: Automatic quarantine is currently disabled, detected threats are still
To enable, set quarantine_hits=1 and/or to quarantine hits from this scan run:
/usr/local/sbin/maldet -q 241022-0531.435

FILE HIT LIST:
{HEX}php.cmdshell.anticat.202 : /home/ubuntu/malware2/f1375cf097b3f28247762147
{HEX}php.cmdshell.egyspider.244 : /home/ubuntu/malware2/3c3df3e64e4a8ae63bb963a
{HEX}php.cmdshell.cih.237 : /home/ubuntu/malware2/4bc59b24b0def2afc418dda48b09e
=====
[ Read 21 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File ^\ Replace  ^U Paste Text ^T To Spell  ^_ Go To Line

```

○ **Nhiệm vụ 3: Trả lời câu hỏi và thực hiện yêu cầu đưa ra**

- Chức năng "*quarantine*" của công cụ maldet là một tính năng quan trọng giúp cách ly và ngăn chặn các tập tin bị nghi ngờ chứa mã độc hại. Khi maldet phát hiện một tập tin có khả năng là malware trong quá trình quét, sinh viên có thể chọn để đưa tập tin đó vào chế độ *quarantine*.
- Khi tập tin được đưa vào quarantine, nó sẽ được di chuyển đến một vị trí được chỉ định trước đó, thường là một thư mục riêng biệt trên hệ thống. Điều này giúp ngăn chặn các tập tin bị nghi ngờ gây ra hậu quả tiêu cực cho hệ thống bằng cách ngăn chặn truy cập và thực thi từ các tập tin đó.

- Sinh viên cần sử dụng lệnh *maldet -h* để đọc các chức năng của công cụ và tìm hiểu về cách sử dụng chức năng *quarantine* để đưa các file mã độc vừa quét được vào thư mục *quarantine* của công cụ.

```

ubuntu@ptit-maldet: ~
File Edit View Search Terminal Help
ubuntu@ptit-maldet:~$ sudo maldet -q 241011-1331.726
Linux Malware Detect v1.6.5
      (C) 2002-2023, R-fx Networks <proj@rfxn.com>
      (C) 2023, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms of the GNU GPL v2

chattr: Operation not permitted while setting flags on /home/ubuntu/malware2/f1375cf097b3f28247762147f8ee3755e0ce26e24fbf8a785fe4e5b42c1fed05.unknown
maldet(1447): {quar} malware quarantined from '/home/ubuntu/malware2/f1375cf097b3f28247762147f8ee3755e0ce26e24fbf8a785fe4e5b42c1fed05.unknown' to '/usr/local/maldetect/quarantine/f1375cf097b3f28247762147f8ee3755e0ce26e24fbf8a785fe4e5b42c1fed05.unknown.175626280'
chattr: Operation not permitted while setting flags on /home/ubuntu/malware2/3c3df3e64e4a8ae63bb963a5b9f8b135acf2e5f76ab755c89f2a8a311edd2230.php
maldet(1447): {quar} malware quarantined from '/home/ubuntu/malware2/3c3df3e64e4a8ae63bb963a5b9f8b135acf2e5f76ab755c89f2a8a311edd2230.php' to '/usr/local/maldetect/quarantine/3c3df3e64e4a8ae63bb963a5b9f8b135acf2e5f76ab755c89f2a8a311edd2230.php.2017331724'
chattr: Operation not permitted while setting flags on /home/ubuntu/malware2/4bc59b24b0def2afc418dda48b09e2d21cc1b581db6b584d5f44902195565559.php
maldet(1447): {quar} malware quarantined from '/home/ubuntu/malware2/4bc59b24b0def2afc418dda48b09e2d21cc1b581db6b584d5f44902195565559.php' to '/usr/local/maldetect/quarantine/4bc59b24b0def2afc418dda48b09e2d21cc1b581db6b584d5f44902195565559.php.2555925299'
ubuntu@ptit-maldet:~$

```

○ *Nhiệm vụ 4: Tự động hóa việc xử lý mã độc cho công cụ*

- Sinh viên cần tiếp tục cấu hình file *conf.maldet* để sau khi công cụ rà quét xong, các file mã độc sẽ được tự động chuyển đến thư mục cách ly của công cụ. Mặc định trường *quarantine_hits="0"* sẽ được đổi thành 1 để bật chức năng này.

```

##
# [ QUARANTINE OPTIONS ]
##
# The default quarantine action f
# [0 = alert only, 1 = move to qu
quarantine_hits="1"

# Try to clean string based malwa
# [NOTE: quarantine_hits=1 requir

```

- Sử dụng công cụ quét lại file */malware1*. Sau khi quét xong, mở file report sẽ thấy thông báo các file mã độc ban đầu đã được tự động chuyển sang thư mục */usr/local/maldetect/quarantine*.

```
File Edit View Search Terminal Help
ubuntu@ptit-maldet:~$ sudo maldet --report 241022-0531.435
Linux Malware Detect v1.6.5
      (C) 2002-2023, R-fx Networks <proj@rfxn.com>
      (C) 2023, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms of the GNU GPL v2

ubuntu@ptit-maldet:~$ sudo nano /usr/local/maldetect/conf.maldet
ubuntu@ptit-maldet:~$ ls
malware2
ubuntu@ptit-maldet:~$ cd ..
ubuntu@ptit-maldet:/home$ ls
ubuntu
ubuntu@ptit-maldet:/home$ cd ..
ubuntu@ptit-maldet:/$ ls
bin  dev  home  lib32  libx32  maldetect-current.tar.gz  media  opt  root  sbin  sys  tmp  var
boot  etc  lib  lib64  maldetect-1.6.5  malware1  mnt  proc  run  srv  sys.tar  usr
ubuntu@ptit-maldet:/$ pwd
/
ubuntu@ptit-maldet:/$ sudo maldet -a /malware1/
```

```
/
ubuntu@ptit-maldet:/$ sudo maldet -a /malware1/
Linux Malware Detect v1.6.5
      (C) 2002-2023, R-fx Networks <proj@rfxn.com>
      (C) 2023, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms of the GNU GPL v2

maldet(1473): {scan} signatures loaded: 17638 (14801 MD5 | 2054 HEX | 783 YARA | 0 USER)
maldet(1473): {scan} building file list for /malware1/, this might take awhile...
maldet(1473): {scan} setting nice scheduler priorities for all operations: cpunice 19 , ionice 6
maldet(1473): {scan} file list completed in 0s, found 9 files...
maldet(1473): {scan} scan of /malware1/ (9 files) in progress...
maldet(1473): {scan} 9/9 files scanned: 4 hits 0 cleaned

maldet(1473): {scan} scan completed on /malware1/: files 9, malware hits 4, cleaned hits 0, time 5s
maldet(1473): {scan} scan report saved, to view run: maldet --report 241022-0546.1473
ubuntu@ptit-maldet:/$
```

- Sử dụng lệnh để kiểm tra thư mục: `ls /usr/local/maldetect/quarantine`.

```
ubuntu@ptit-maldet:/$ sudo ls /usr/local/maldetect/quarantine
eicar.com.259589000      eicar.com.txt.259319375      eicar_com.zip.1650215394      eicarcom2.zip.2385515541
eicar.com.259589000.info  eicar.com.txt.259319375.info  eicar_com.zip.1650215394.info  eicarcom2.zip.2385515541.info
ubuntu@ptit-maldet:/$
```

- Kết thúc bài lab:
 - Trên *terminal* đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:
stoplab ptit-maldet
 - Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.
- Khởi động lại bài lab:
 - Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

labtainer -r ptit-maldet

3. PTIT-RADARE2

3.1. Mục đích

Bài thực hành này giới thiệu công cụ Radare2. Sinh viên sẽ sử dụng Radare2 để phân tích một tập tin thực thi nhằm xác định một số thuộc tính cơ bản.

3.2. Yêu cầu đối với sinh viên

Có kiến thức cơ bản về hệ điều hành Linux, mã độc...

3.3. Nội dung thực hành

- Khởi động bài lab:

Vào terminal, gõ: *labtainer ptit-radare2*

(Chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Sau khi khởi động xong một terminal ảo sẽ xuất hiện, đại diện cho máy radare2

- Trong bài lab này sẽ có sẵn file mã độc *sample.c*, được sử dụng để lây nhiễm mã độc vào một số các file khác có đuôi file là “.c, .cpp, .txt”.
- File mã độc này bắt đầu được thực thi bằng một payload đơn giản, đầu tiên nó sẽ in ra "YOU HAVE BEEN INFECTED ...!!!".
- Sinh viên bắt đầu làm bài thực hành bằng cách chạy lệnh “./run.sh” để biên dịch và thực thi đoạn code mã độc trên.

```
File Edit View Search Terminal Help
ubuntu@radare2:~$ ls
doixung.cpp  file.txt  hello.c  malware.txt  run.sh  sample.c  sodx.cpp
ubuntu@radare2:~$ ./run.sh
ubuntu@radare2:~$ ls
doixung.cpp  file.txt  hello.c  malware.txt  run.sh  sample  sample.c  sodx.cpp
ubuntu@radare2:~$
```

- Sau khi biên dịch xong thì sẽ có file sample sinh ra.

- Chức năng chính của virus là chịu trách nhiệm thực thi payload và tự sao chép. Nó bắt đầu bằng cách gọi hàm mã độc và khởi tạo một số biến. Nó mở tệp hiện tại, chính là chương trình vi-rút, bằng cách sử dụng chức năng fopen.
- Sau đó, nó đọc từng dòng tệp hiện tại, tìm kiếm một chuỗi cụ thể "// VIRUS SAYS HI!" đánh dấu sự bắt đầu của mã virus và "// VIRUS SAYS BYE!" đánh dấu sự kết thúc của mã virus. Nó sao chép mã virus vào một mảng, vì vậy nó có thể được sử dụng để lây nhiễm các tệp khác.
- Sinh viên làm các nhiệm vụ dưới đây để hoàn thành bài lab:
 - **Nhiệm vụ 1: Xem tổng quan chương trình với radare2**
 - Radare2 là một công cụ phân tích mã nguồn mở và phục hồi hệ thống mạnh mẽ. Nó có thể được sử dụng để phân tích mã nguồn của nhiều loại tệp khác nhau, bao gồm cả mã độc. Radare2 cung cấp các chức năng để xem mã hợp ngữ, tìm kiếm chuỗi hoặc giá trị nhất định, đặt điểm dừng, và nhiều hơn nữa.
 - Đầu tiên sinh viên sử dụng câu lệnh "`r2 ./sample`" để khởi chạy chương trình, radare2 sẽ mở tệp "sample" và chờ nhập các lệnh khác để phân tích.
 - ✓ Sau khi khởi chạy chương trình thì sử dụng "`aaa`" để xem mô-đun và các hàm.

```
ubuntu@radare2:~$ r2 ./sample
WARN: Relocs has not been applied. Please use '-e bin.relocs.apply=true' or '-e bin.cache=true' n
ext time
-- r2 talks to you. tries to make you feel well.
[0x00001310]> aaa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Recovering variables
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods
INFO: Recovering local variables (afva)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
```


- ✓ Chạy lệnh “*afl*”, radare2 sẽ hiển thị một danh sách các hàm, cùng với một số thông tin về mỗi hàm.

```
[0x00001310]> afl
0x000011b0 1 10 sym.imp.strcmp
0x000011c0 1 10 sym.imp.printf
0x000011d0 1 10 sym.imp.fgets
0x000011e0 1 10 sym.imp.fclos
0x000011f0 1 10 sym.imp.time
0x00001200 1 10 sym.imp.popen
0x00001210 1 10 sym.imp.__stack_chk_fail
0x00001220 1 10 sym.imp.strcpy
0x00001230 1 10 sym.imp.malloc
0x00001240 1 10 sym.imp.puts
0x00001250 1 10 sym.imp.exit
0x00001260 1 10 sym.imp.pclose
0x00001270 1 10 sym.imp.srand
0x00001280 1 10 sym.imp.strlen
0x00001290 1 10 sym.imp.__libc_start_main
0x000012a0 1 10 sym.imp.fprintf
0x000012b0 1 10 sym.imp.fopen
0x000012c0 1 10 sym.imp.rand
0x000012d0 1 10 sym.imp.fputc
0x000012e0 1 10 sym.imp.sprintf
```

○ Nhiệm vụ 2: Xem các hàm code trong chương trình

- “s (địa chỉ hàm) hoặc s (tên hàm)” được sử dụng để di chuyển con trỏ đến hàm trong chương trình.
Ví dụ, s main: di chuyển con trỏ đến hàm main cho phép phân tích và kiểm tra cấu trúc của hàm main.
- Lệnh “*pdf*” trong Radare2 được sử dụng để phân tích cú pháp một hàm. Khi gọi lệnh pdf mà không có tham số, nó sẽ phân tích cú pháp hàm hiện tại (nơi con trỏ đang trỏ đến). Nếu muốn phân tích cú pháp một hàm cụ thể, có thể chỉ định nó như một tham số, ví dụ: pdf @ main để phân tích cú pháp hàm main.
- Ở đây sinh viên sẽ cần phải xem thông tin của 2 hàm “*private_Function*” mô tả chức năng chính của virus lây nhiễm trong chương trình và hàm “*gene*” được sử dụng để sinh các chữ số và ký tự ngẫu nhiên.

```

|      :| 0x00001685      50      push eax      ; const char *s2
|      :| 0x00001686      8d850cfcffff  lea eax, [src]
|      :| 0x0000168c      50      push eax      ; const char *s1
|      :| 0x0000168d      e81efbffff  call sym.imp.strcmp ; int strcmp(const char *s
1, const char *s2)
|      :| 0x00001692      83c410      add esp, 0x10
|      :| 0x00001695      85c0      test eax, eax
|      :|= 0x00001697      7427      je 0x16c0
|      :| ; CODE XREF from sym.private_Function @ 0x15ff(x)
|      :| -> 0x00001699      83ec04      sub esp, 4
|      :| 0x0000169c      ffb5fcb8f0ff push dword [ebp - 0xf4704] ; FILE *stream
|      :| 0x000016a2      68e8030000 push 0x3e8 ; int size
|      :| 0x000016a7      8d850cfcffff  lea eax, [src]
|      :| 0x000016ad      50      push eax      ; char *s
|      :| 0x000016ae      e81dfbffff  call sym.imp.fgets ; char *fgets(char *s, int
size, FILE *stream)
|      :| 0x000016b3      83c410      add esp, 0x10
|      :| 0x000016b6      85c0      test eax, eax
|      :|= 0x000016b8      0f8563ffffff jne 0x1621
|      :| ,= 0x000016be      eb01      jmp 0x16c1
|      :| ; CODE XREF from sym.private_Function @ 0x1697(x)
|      :| -> 0x000016c0      90      nop
|      :| ; CODE XREF from sym.private_Function @ 0x16be(x)
|      :| -> 0x000016c1      83ec0c      sub esp, 0xc
|      :| 0x000016c4      ffb5fcb8f0ff push dword [ebp - 0xf4704] ; FILE *stream

```

Hàm sym.private_Function

```

; CALL XREF from sym.private_Function @ 0x1881(x)
/ 264: sym.gene ();
; var int32_t var_ch @ ebp-0xc
; var size_t var_1ch @ ebp-0x1c
; var int32_t var_5bh @ ebp-0x5b
; var int32_t var_60h @ ebp-0x60
; var size_t var_64h @ ebp-0x64
; var size_t size @ ebp-0x68
; var int32_t var_6ch @ ebp-0x6c
; var int32_t var_7ch @ ebp-0x7c
0x0000147c      f30f1efb      endbr32
0x00001480      55           push ebp
0x00001481      89e5         mov ebp, esp
0x00001483      57           push edi
0x00001484      56           push esi
0x00001485      53           push ebx
0x00001486      83ec7c       sub esp, 0x7c
0x00001489      e8c9050000   call sym.__x86.get_pc_thunk.si
0x0000148e      81c6fa2a0000 add esi, 0x2afa
0x00001494      89f3         mov ebx, esi
0x00001496      895d84       mov dword [var_7ch], ebx
0x00001499      65a114000000 mov eax, dword gs:[0x14]
0x0000149f      8945e4       mov dword [var_1ch], eax
0x000014a2      31c0         xor eax, eax
0x000014a4      c745980a00.. mov dword [size], 0xa
0x000014ab      8b4598       mov eax, dword [size]
0x000014ae      83c001       add eax, 1
0x000014b1      83ec0c       sub esp, 0xc
0x000014b4      50           push eax
0x000014b5      e876fdffff   call sym.imp.malloc      ; size_t size
                                ; void *malloc(size_t siz

e)
0x000014ba      83c410       add esp, 0x10
0x000014bd      89459c       mov dword [var_64h], eax
0x000014c0      83ec0c       sub esp, 0xc
0x000014c3      6a00         push 0
0x000014c5      e826fdffff   call sym.imp.time        ; time_t time(time_t *time

r)
0x000014ca      83c410       add esp, 0x10
0x000014cd      83ec0c       sub esp, 0xc
0x000014d0      50           push eax
                                ; int seed

```

Hàm sym.gene

o Nhiệm vụ 3: Khởi chạy debug

- Trong Radare2, “db, dc” là các lệnh liên quan đến việc gỡ lỗi:
 - ✓ db: Lệnh này được sử dụng để thiết lập các điểm dừng (breakpoints). Có thể sử dụng db kèm theo điểm vào của chương trình để bắt đầu trình gỡ lỗi và tạm dừng thực thi tại hàm đó. Ví dụ, db main sẽ bắt đầu trình gỡ lỗi và tạm dừng thực thi tại hàm main.
 - ✓ dc: Lệnh này được sử dụng để tiếp tục thực thi chương trình sau khi đã tạm dừng tại một điểm dừng. Nó cho phép kiểm soát quá trình thực thi của chương trình và dừng lại tại các điểm quan trọng để kiểm tra trạng thái của chương trình.

- ✓ Sau quá trình nhập các lệnh *db*, *dc* một vài lần thì chương trình sẽ yêu cầu sinh viên phải nhập thông tin, ở đây sinh viên sẽ nhập input là mã sinh viên của mình và tiếp tục sử dụng lệnh *dc* để thực thi chương trình đến khi xuất hiện dòng chữ “YOU HAVE BEEN INFECTED ...” → hoàn thành nhiệm vụ.

```
[0x00001310]> db
[0x00001310]> dc
ERROR: Cannot continue, run ood?
[0x00000000]> ood
INFO: File dbg:///home/ubuntu/sample reopened in read-write mode
WARN: Relocs has not been applied. Please use '-e bin.relocs.apply=true' or '-e bin.cache=true' next time
349
[0xf7f86120]> dc
địa chỉ của buf: 0x56619040
Nhập thông tin: B21DCAT151
B21DCAT151
(349) Created process 350
[0xf7e92d78]>
```

Nhập thông tin

```
(349) Created process 350
[0xf7e92d78]> db
[0xf7e92d78]> dc
[+] SIGNAL 17 errno=0 addr=0x3e80000015e code=1 si_pid=350 ret=0
[+] signal 17 aka SIGCHLD received 0 (Child)
[0xf7f84059]> dc
YOU HAVE BEEN INFECTED 973da1a8689e91f697a995df112a0f30HA !!!(349) Process exited with status=0x0
[0xf7f84059]>
```

“YOU HAVE BEEN INFECTED ...”

○ **Nhiệm vụ 4: Tìm kiếm các chuỗi**

- Lệnh “*iz*” trong Radare2 được sử dụng để liệt kê tất cả các chuỗi trong chương trình.

```
[0xf7f84059]> iz
[Strings]
nth paddr      vaddr      len size section type  string
-----
0  0x00002008 0x56617008 61  62  .rodata ascii YOU HAVE BEEN INFECTED 973da1a8689e91f697a995df1
12a0f30HA !!!
1  0x00002048 0x56617048 62  63  .rodata ascii abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
WXYZ0123456789
2  0x00002089 0x56617089 26  27  .rodata ascii Error reading current file
3  0x000020a4 0x566170a4 18  19  .rodata ascii // VIRUS SAYS HI!\n
4  0x000020b7 0x566170b7 19  20  .rodata ascii // VIRUS SAYS BYE!\n
5  0x000020cc 0x566170cc 52  53  .rodata ascii find . -name '*.c' -o -name '*.cpp' -o -name '*.
txt'
6  0x00002101 0x56617101 19  20  .rodata ascii Error listing files
7  0x00002115 0x56617115 23  24  .rodata ascii Error reading file: %s\n
8  0x0000212f 0x5661712f 26  27  .rodata ascii Error writing to file: %s\n
9  0x0000214c 0x5661714c 32  33  .rodata ascii 2db71b45bc52113fc4c774763375d7b0
10 0x0000216d 0x5661716d 16  17  .rodata ascii %s_?%s!!!%s____%s
11 0x0000217e 0x5661717e 20  21  .rodata ascii địa chỉ của buf: %p\n
12 0x00002193 0x56617193 16  17  .rodata ascii Nhập thông tin:
[0xf7f84059]>
```

- Sau khi chạy xong mã độc, sinh viên có thể thực hiện đọc các file trong hệ thống để xem được hành động lây nhiễm của virus vào các file.
- Kết thúc bài lab:
Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab: *stoplab radare2*
Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.
- Khởi động lại bài lab:
Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

labtainer -r radare2

```
student@ubuntu:~/labtainer/labtainer-student$ checkwork
Results stored in directory: /home/student/labtainer_xfer/ptit-radare2
Labname ptit-radare2

Student      | view_function | check_analysis | debug | strings |
===== | ===== | ===== | ===== | ===== |
B21DCAT151  | Y | Y | Y | Y |
What is automatically assessed for this lab:
```

4. PTIT – STATIC- ANALYSIS

4.1. Mục đích

- Giúp sinh viên hiểu về quy trình, các công cụ trong quá trình phân tích mã độc và thực hiện phân tích tĩnh mã độc.

4.2. Yêu cầu đối với sinh viên

- Có kiến thức cơ bản về mã độc và cách sử dụng các công cụ để phân tích mã độc.

4.3. Nội dung thực hành

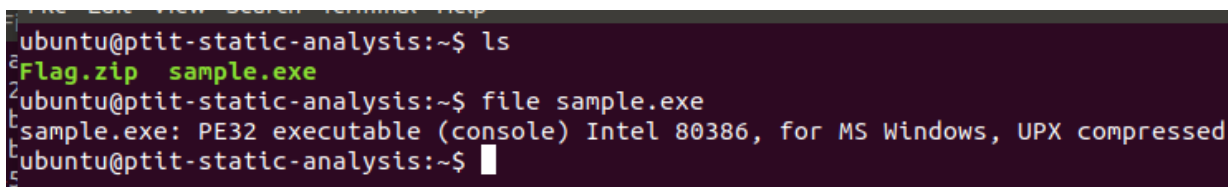
- Khởi động bài lab:
 - Vào terminal, gõ:

Labtainer -r ptit-static-analysis

(chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Sau khi khởi động xong một terminal ảo sẽ xuất hiện.

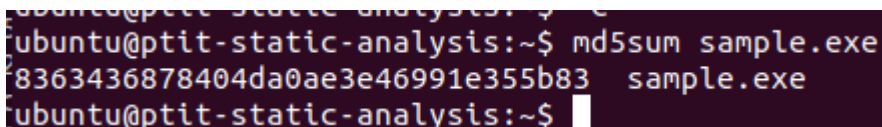
- Các nhiệm vụ thực hành:
 - **Nhiệm vụ 1: Thu thập thông tin file của mẫu mã độc.**
 - Trên terminal *ptit-static-analysis* sử dụng lệnh “file”, xác định thông tin của mẫu. Từ đó có thể biết được những thông tin gì về sample.exe? Đây là file thực thi của hệ điều hành Windows 32-bit, tệp đã được nén bằng **UPX** (Ultimate Packer for Executables)



```
ubuntu@ptit-static-analysis:~$ ls
Flag.zip  sample.exe
ubuntu@ptit-static-analysis:~$ file sample.exe
sample.exe: PE32 executable (console) Intel 80386, for MS Windows, UPX compressed
ubuntu@ptit-static-analysis:~$
```

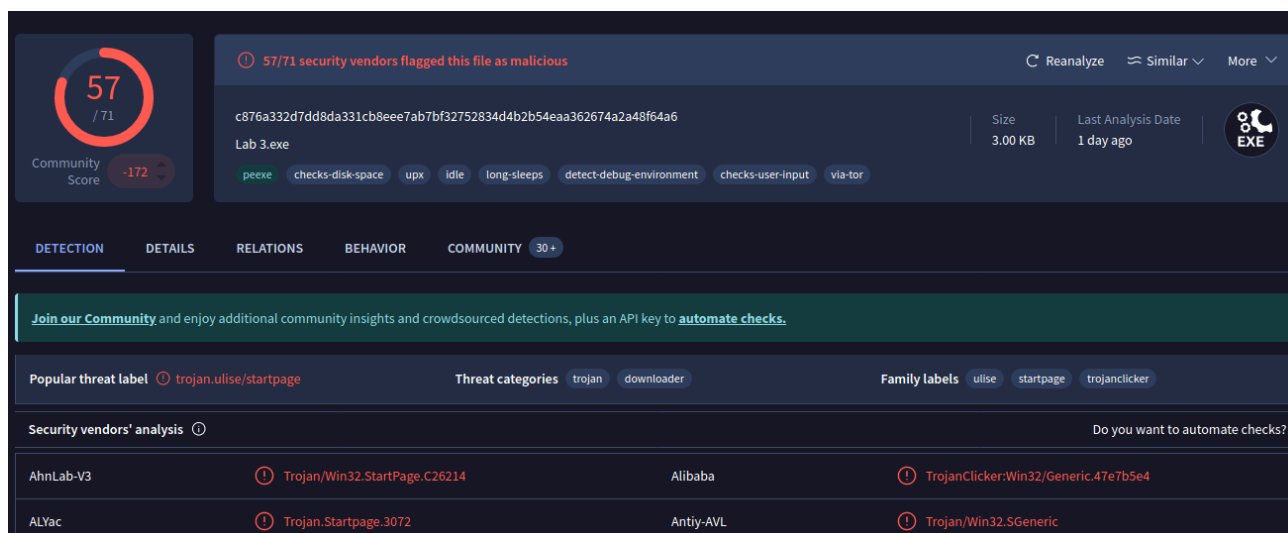
- **Nhiệm vụ 2: Thu thập thông tin file của mẫu mã độc.**

- Sử dụng lệnh “md5sum” để lấy chữ ký của mẫu.



```
ubuntu@ptit-static-analysis:~$ md5sum sample.exe
8363436878404da0ae3e46991e355b83 sample.exe
ubuntu@ptit-static-analysis:~$
```

- Mở trình duyệt (gõ firefox ở máy *ptit-static-analysis*), truy cập vào <https://www.virustotal.com/gui/home/upload> để tìm kiếm thông tin từ chữ ký của mẫu. Có bao nhiêu nhà cung cấp và môi trường cho rằng đây là tệp tin độc hại?



The screenshot shows the VirusTotal interface for a file named 'Lab 3.exe'. The file's SHA-256 hash is c876a332d7dd8da331cb8eee7ab7bf32752834d4b2b54eaa362674a2a48f64a6. It has a size of 3.00 KB and was last analyzed 1 day ago. The file is flagged as malicious by 57 out of 71 security vendors. The community score is -172. The file is categorized as a trojan and downloader. The security vendors' analysis table shows detections from AhnLab-V3, ALYac, Alibaba, and Antiy-AVL.

Security vendors' analysis	Detection
AhnLab-V3	Trojan.Win32.StartPage.C26214
ALYac	Trojan.Startpage.3072
Alibaba	TrojanClicker.Win32/Generic.47e7b5e4
Antiy-AVL	Trojan.Win32.SGeneric

○ Nhiệm vụ 3: Phân tích chuỗi

- Thực hiện phân tích các chuỗi trong mẫu, sử dụng lệnh “strings” để xem xét các chuỗi đáng ngờ như các URL, tên tệp, tên các thư viện được sử dụng, chuỗi mã hóa, ...

```
ubuntu@ptit-static-analysis:~$ strings sample.exe
!This program cannot be run in DOS mode.
Rich
UPX0
UPX1
UPX2
3.04
UPX!
a\`Y
(23h
MalService
sHGL345
http://w
warean
ysisbook.co
om#Int6net Explo!r 8FEI
SystemTimeToFile
GetMo
*Waitab'r
Process
OpenMu$x
ZSB+
ForS
ObjectU4
[Vrtb
```



```

ubuntu@ptit-static-analysis:~$ strings sample.exe | grep -i "http"
http://w
ubuntu@ptit-static-analysis:~$ strings sample.exe | grep -i "\.exe\\|\.dll\\|\.sys"
KERNEL32.DLL
ADVAPI32.dll
MSVCRT.dll
WININET.dll
ubuntu@ptit-static-analysis:~$ strings sample.exe | awk 'length($0) > 15'
!This program cannot be run in DOS mode.
om#Int6net Explo!r 8FEI
SystemTimeToFile
ubuntu@ptit-static-analysis:~$

```

- Sử dụng lệnh “echo” để in ra màn hình các chuỗi đáng ngờ.

```

ubuntu@ptit-static-analysis:~$ echo "$(cat stringkhanghi.txt)"
!This program cannot be run in DOS mode.
Rich
UPX0
UPX1
UPX2
3.04
UPX!
a\`Y
(23h
MalService
SHGL345
http://w
warean
ysisbook.co
om#Int6net Explo!r 8FEI
SystemTimeToFile
GetMo
*Waitab'r

```

○ **Nhiệm vụ 4: Phân tích Upacking:**

- Từ các chuỗi thu được có thể biết được mẫu này đã sử dụng kỹ thuật gì để che dấu và tránh bị phát hiện?

Sử dụng UPX để nén tệp thực thi:

Các chuỗi như UPX0, UPX1, UPX2, và UPX! cho thấy tệp này đã được nén bằng UPX (Ultimate Packer for Executables), một công cụ nén tệp thực thi. UPX giúp giảm kích thước tệp và cũng có thể làm cho mã khó đọc hơn, vì phần mã bị nén sẽ không hiển thị rõ ràng cho đến khi được giải nén khi chạy.

Sử dụng các thư viện Windows phổ biến:

Các chuỗi **KERNEL32.DLL**, **ADVAPI32.dll**, **MSVCRT.dll**, **WININET.dll** là các thư viện Windows tiêu chuẩn, thường được mã độc sử dụng để thực hiện các

chức năng hệ thống, thao tác mạng và truy cập tài nguyên. Các API như **VirtualProtect**, **VirtualAlloc**, **VirtualFree** có thể được dùng để cấp phát bộ nhớ và thực thi mã trong bộ nhớ – kỹ thuật phổ biến để tránh bị phát hiện qua đĩa.

Các API liên quan đến mạng:

InternetOpenA và **Internet Explorer** cho thấy mã độc có thể có khả năng kết nối đến các địa chỉ mạng từ xa, như URL và tên miền (vd. warean, ysisbook.co có thể là một phần của URL đầy đủ). Các mã độc thường kết nối đến các máy chủ điều khiển để tải xuống mã độc khác hoặc gửi dữ liệu đánh cắp.

Tên dịch vụ đáng ngờ:

Chuỗi **MalService** có thể ám chỉ mã độc đăng ký một dịch vụ (service) trên máy nạn nhân để duy trì sự hiện diện lâu dài. Dịch vụ này có thể được thiết lập để khởi động cùng hệ thống, đảm bảo mã độc hoạt động ngay khi hệ thống được bật.

Các hàm tạo và quản lý dịch vụ:

CreateServiceA là hàm tạo dịch vụ trong Windows, giúp mã độc cài đặt chính nó như một dịch vụ hợp lệ của hệ điều hành, tránh bị phát hiện do chạy liên tục.

Obfuscation:

Các chuỗi như **mArg**, **CtrlDisp**, **Xcpt**, **sHGL345**, và **t_fd** là các tên biến hoặc hàm khó hiểu, có thể là một phần của kỹ thuật obfuscation (làm xáo trộn mã) nhằm làm cho mã khó hiểu và khó phân tích.

- Dùng công cụ UPX để unpack mẫu, sử dụng lệnh: “upx -d sample.exe -o unpack_sample.exe”.

```

ubuntu@ptit-static-analysis:~$ upx -d sample.exe -o unpack_sample.exe
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2013
UPX 3.91      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

      File size      Ratio      Format      Name
-----
16384 <- 3072 18.75% win32/pe  unpack_sample.exe

Unpacked 1 file.

```

- Phân tích lại các chuỗi trong mẫu sau khi đã unpack. Từ đó thu thập được các thông tin đáng ngờ nào?

Các chuỗi thu được từ mẫu này cung cấp một số dấu hiệu cho thấy mã độc đang sử dụng các kỹ thuật sau để che giấu hoặc tránh bị phát hiện:

– Các thư viện và API Windows tiêu chuẩn:

- Mẫu mã độc sử dụng các thư viện chuẩn của Windows như KERNEL32.DLL, ADVAPI32.dll, MSVCRT.dll, và WININET.dll. Đây là các thư viện thường được sử dụng để thực hiện các chức năng hệ thống, mạng, và quản lý tài nguyên. Việc sử dụng các thư viện tiêu chuẩn giúp mã độc dễ dàng ẩn mình hơn, do các thư viện này là thành phần hợp lệ của hệ điều hành.

– Tạo và điều khiển dịch vụ Windows:

- Các API như CreateServiceA, StartServiceCtrlDispatcherA, và OpenSCManagerA cho thấy rằng mã độc có khả năng tự cài đặt như một dịch vụ trên hệ thống. Việc chạy mã độc như một dịch vụ giúp nó có quyền truy cập cao hơn và có thể tự động khởi động lại khi hệ thống khởi động, giúp mã độc duy trì sự tồn tại lâu dài.

– Khả năng kết nối mạng:

- Các hàm như InternetOpenUrlA và InternetOpenA cho thấy mã độc có khả năng kết nối với internet, có thể nhằm mục đích liên lạc với máy chủ điều khiển từ xa (Command and Control Server) để tải thêm mã độc, gửi thông tin đánh cắp hoặc nhận chỉ thị. URL <http://www.malwareanalysisbook.com> được liệt kê có thể là

một phần trong chuỗi mã, nhưng có thể đã được thay đổi trong môi trường thực tế.

– **Sử dụng Mutex để tránh chạy nhiều phiên bản:**

- Các hàm OpenMutexA và CreateMutexA cho thấy mã độc có thể sử dụng mutex để đảm bảo rằng chỉ một phiên bản của nó chạy tại một thời điểm. Đây là kỹ thuật phổ biến giúp tránh việc mã độc tự sao chép chính nó khi đã có một phiên bản đang hoạt động.

– **Các hàm xử lý ngoại lệ và điều chỉnh:**

- Các hàm như _XcptFilter và _except_handler3 được sử dụng để xử lý lỗi và ngoại lệ. Điều này cho thấy mã độc có thể đang cố gắng chạy ổn định và xử lý các tình huống ngoại lệ mà không làm sập hệ thống, giảm khả năng bị phát hiện khi có lỗi.

– **Các hàm liên quan đến việc đặt thời gian:**

- Các hàm như CreateWaitableTimerA, SetWaitableTimer, WaitForSingleObject cho thấy mã độc có thể sử dụng các kỹ thuật liên quan đến thời gian để trì hoãn hoạt động, đồng bộ hóa các sự kiện hoặc gây khó khăn cho các công cụ phân tích (các công cụ phân tích thường thực thi mã một cách nhanh chóng, trong khi mã độc có thể trì hoãn hoạt động để đánh lừa công cụ).

– **Tên dịch vụ đáng ngờ:**

- Chuỗi MalService hoặc Malservice có thể là tên dịch vụ mà mã độc tạo ra. Điều này có thể là một dấu hiệu cho thấy mã độc đang giả danh một dịch vụ hệ thống hợp lệ để tránh bị nghi ngờ.

Kết luận

Các kỹ thuật được sử dụng bao gồm:

- **Cài đặt và chạy như một dịch vụ hệ thống** để đảm bảo mã độc có quyền truy cập cao và tồn tại lâu dài.

- **Kết nối mạng và tải xuống từ máy chủ từ xa** để thực hiện các hoạt động độc hại.
- **Sử dụng Mutex** để ngăn việc chạy nhiều phiên bản.
- **Xử lý ngoại lệ** để tránh gây lỗi hệ thống, giúp mã độc chạy ổn định và ẩn mình tốt hơn.
- **Sử dụng các kỹ thuật kiểm soát ngoại lệ và thời gian** để gây khó khăn cho các công cụ phân tích.
- Sử dụng lệnh “echo” để in ra những chuỗi đáng ngờ.

```

CreateWaitableTimerA
ExitProcess
OpenMutexA
SetWaitableTimer
WaitForSingleObject
CreateMutexA
CreateThread
CreateServiceA
StartServiceCtrlDispatcherA
OpenSCManagerA
_exit
_XcptFilter
exit
__p__initenv
__getmainargs
__initterm
__setusermatherr
_adjust_fdiv
__p__commode
__p__fmode
__set_app_type
_except_handler3
_controlfp
InternetOpenUrlA
InternetOpenA
MalService
MalService
HGL345
http://www.malwareanalysisbook.com
Internet Explorer 8.0
ubuntu@ptit-static-analysis:~$

```

- **Nhiệm vụ 5: Xác định các hành vi độc hại của mẫu**
 - Sử dụng công cụ Radare2 để tìm chức năng chính của mẫu, dùng lệnh:

r2 ./unpack_sample.exe

```
ubuntu@ptit-static-analysis:~$ r2 ./unpack_sample.exe
WARN: Relocs has not been applied. Please use '-e bin.relocs.apply=true' or '-e bin.cache=true' next time
-- Most of commands accept '?' as a suffix. Use it to understand how they work :)
```

❖ Lệnh “aaaaa”: Tự động phân tích một file nhị phân hoặc một đoạn mã.

```
[0x00401190]> aaaaa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Recovering variables
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods
INFO: Recovering local variables (afva)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Scanning for strings constructed in code (/azs)
INFO: Finding function preludes (aap)
INFO: Enable anal.types.constraint for experimental type propagation
INFO: Reanalyzing graph references to adjust functions count (aarr)
```

❖ Liệt kê tất cả các hàm (Analysis Functions List), sử dụng lệnh “afl” sẽ quét qua mã máy và xác định tất cả các hàm có trong chương trình. Điều này bao gồm các hàm hợp lệ và các hàm có thể không hoàn chỉnh hoặc không được sử dụng. Lệnh “afl” sẽ hiển thị thông tin của mỗi hàm gồm: đại chỉ bắt đầu, địa chỉ kết thúc, kích thước của hàm và thông tin khác liên quan.

```
[0x00401190]> afl
0x00401190 3 260 sub.entry0_401190
0x004012c1 1 1 fcn.004012c1
0x004012ac 1 18 sub.sub.sub.MSVCRT.dll__controlfp_4012d6_4012ac
0x004012d6 1 6 sub.sub.MSVCRT.dll__controlfp_4012d6
0x004012a6 1 6 sub.sub.MSVCRT.dll__initterm_4012a6
0x00401000 1 59 sub.text_401000
0x00401040 7 265 sub.HGL345_401040
0x004012a0 1 6 sub.sub.MSVCRT.dll__XcptFilter_4012a0
0x00401150 2 50 sub.Internet_Explorer_8.0_401150
[0x00401190]> s 0x00401190
```

❖ Để hiển thị mã hợp ngữ của một hàm, sử dụng lệnh “s <địa chỉ bắt đầu của hàm>” để chuyển vùng nhớ hiện tại đến đại chỉ bắt đầu của hàm cụ thể trong mã

máy (ví dụ: s 0x00401190). Sau đó sử dụng lệnh “pdf” để hiển thị mã máy của hàm mà con trỏ vùng nhớ hiện tại đang trỏ tới. Nó tạo ra một phiên bản của mã hợp ngữ, cho phép xem và phân tích mã máy cụ thể của hàm đó.

```
[0x00401190]> s 0x00401190
[0x00401190]> pdf
;-- entry0:
;-- eip:
/ 260: sub.entry0_401190 ();
; var int32_t var_4h @ ebp-0x4
; var int32_t var_14h @ ebp-0x14
; var int32_t var_18h @ ebp-0x18
; var int32_t var_1ch @ ebp-0x1c
; var int32_t var_20h @ ebp-0x20
; var int32_t var_24h @ ebp-0x24
; var int32_t var_28h @ ebp-0x28
; var int32_t var_2ch @ ebp-0x2c
; var int32_t var_30h @ ebp-0x30
0x00401190      55          push ebp
0x00401191      8bec        mov ebp, esp
0x00401193      6aff        push 0xffffffffffffffff
0x00401195      6880204000  push 0x402080
0x0040119a      68d0124000  push 0x4012d0
0x0040119f      64a100000000 mov eax, dword fs:[0]
0x004011a5      50          push eax
```

❖ Quan sát tất cả các hàm để xác định hành vi chính của mẫu mã độc sample.exe là gì (phải xem 2 hàm chính mới có thể hoàn thành nhiệm vụ)?

Hàm entry0 (0x00401190):

Đây là hàm khởi đầu, có nhiệm vụ thiết lập môi trường thực thi và gọi các hàm quan trọng khác.

Hàm sub.text_401000 (0x0040126e):

Đây là một trong những hàm con chính mà hàm entry0 đã gọi. Hàm này có vai trò thực thi hành vi chính của mã độc:

```

[0x00401150]> s 0x00401040
[0x00401040]> pdf
; CALL XREFS from sub.text_401000 @ 0x401010(r), 0x401032(x)
/ 265: sub.HGL345_401040 ();
; var int32_t var_20h @ esp+0x50
; var SYSTEMTIME *lpSystemTime @ esp+0x54
; var int32_t var_8h @ esp+0x58
; var int32_t var_10h @ esp+0x5c
; var int32_t var_18h @ esp+0x60
; var LPFILETIME lpFileTime @ esp+0x64
; var int32_t var_2ch @ esp+0xa0
; var int32_t var_1ch @ esp+0xac
0x00401040      81ec00040000      sub esp, 0x400
0x00401046      6828304000      push str.HGL345          ; 0x403028 ; "HGL345"
0x0040104b      6a00            push 0
0x0040104d      6801001f00      push 0x1f0001          ; DWORD dwDesiredAccess
0x00401052      ff1520204000      call dword [sym.imp.KERNEL32.DLL_OpenMutexA] ; 0x40202
0 ; HANDLE OpenMutexA(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCSTR lpName)
0x00401058      85c0            test eax, eax
;,<= 0x0040105a      7408            je 0x401064
0x0040105c      6a00            push 0
0x0040105e      ff151c204000      call dword [sym.imp.KERNEL32.DLL_ExitProcess] ; 0x40202
1c ; VOID ExitProcess(UINT uExitCode)
; CODE XREF from sub.HGL345_401040 @ 0x40105a(x)
-> 0x00401064      56              push esi
0x00401065      6828304000      push str.HGL345          ; 0x403028 ; "HGL345"
0x0040106a      6a00            push 0

```

○ Nhiệm vụ 6: Giải mã flag:

- Phân tích kỹ các hàm của mã độc trong nhiệm vụ 5. Từ đó xác định được các hành vi độc hại mà mã độc thực hiện lên máy của nạn nhân. Mã độc đã tạo một bộ hẹn giờ để chờ đến thời điểm thực hiện hành vi độc hại.

sub.HGL345_401040:

- Đây là hàm với tên bắt đầu bằng cụm mã HGL345, có thể là một chuỗi đáng ngờ từ dữ liệu phân tích trước đó. Hàm này chứa **7 lệnh** và có chiều dài **265 byte**, có khả năng chứa các lệnh liên quan đến các hành vi độc hại, bao gồm cả **bộ hẹn giờ** hoặc bộ đếm thời gian để thực hiện hành vi độc hại.

Các chi tiết quan trọng:

- **Tạo Waitable Timer:**
 - Ở địa chỉ **0x004010eb**, có lời gọi tới hàm **CreateWaitableTimerA** từ thư viện KERNEL32, đây là hàm dùng để tạo một bộ hẹn giờ có thể đợi.
- **Đặt thời gian hẹn giờ:**
 - Tại địa chỉ **0x00401101**, hàm **SetWaitableTimer** được gọi để thiết lập bộ hẹn giờ. Tham số lpDueTime là một **LARGE_INTEGER**, chỉ thời gian đếm ngược cho đến khi bộ hẹn giờ kích hoạt.
- **Chuyển đổi thời gian từ SYSTEMTIME sang FILETIME:**
 - Tại địa chỉ **0x004010df**, mã gọi hàm **SystemTimeToFileTime**, chuyển đổi từ cấu trúc **SYSTEMTIME** sang **FILETIME** để sử dụng trong hẹn giờ. Trong đó, **SYSTEMTIME** đã được thiết lập với giá trị năm là **2100**

ở dòng **mov word [lpSystemTime], 0x834**. (0x834 = 2100 trong hệ thập phân).

– **Năm sinh ra bộ hẹn giờ:**

- Năm 2100 là thời điểm mà bộ hẹn giờ được thiết lập để thực hiện hành vi độc hại. Điều này được xác định bởi dòng lệnh **mov word [lpSystemTime], 0x834**, nơi năm được chuyển đổi thành dạng hệ thập phân và gán vào cấu trúc thời gian.

Kết luận:

- **Năm 2100** là mật khẩu để giải nén file **Flag.zip** vì đó là thời điểm mà mã độc đã đặt bộ hẹn giờ thông qua hàm **SetWaitableTimer**. Bạn có thể sử dụng lệnh sau để giải nén file:
 - Sử dụng công cụ 7zip để giải nén file Flag.zip với câu lệnh là **7z x Flag.zip -p<password>**. password là năm mà mã độc đã tạo bộ hẹn giờ. Giải mã chuỗi ký tự trong file Flag.txt để thu được flag ở dạng bản rõ và in ra màn hình bản rõ của flag với lệnh “echo”.

```
ubuntu@ptit-static-analysis:~$ 7z x Flag.zip
7-Zip [64] 9.20 Copyright (c) 1999-2010 Igor Pavlov 2010-11-18
p7zip Version 9.20 (locale=C,Utf16=off,HugeFiles=on,2 CPUs)

Processing archive: Flag.zip

Extracting Flag.txt
Enter password (will not be echoed) :

Everything is Ok

Size:          52
Compressed: 270
ubuntu@ptit-static-analysis:~$
```

```
ubuntu@ptit-static-analysis:~$ more Flag.txt
BRDpKPMuZ3FBqYWr6pfjJoRtpJrHXXW6zEcBNndBSq6tEjWGVx6p
ubuntu@ptit-static-analysis:~$
```


Mã hóa bằng Base58 → Giải mã bằng công cụ Base58 decoder online → Thu kết quả

```
ubuntu@ptit-static-analysis:~$ echo "Flag={FLAG_PTIT_ATTT_MALWARE_ANALYSIS}"
Flag={FLAG_PTIT_ATTT_MALWARE_ANALYSIS}
ubuntu@ptit-static-analysis:~$
```

- Kết thúc bài lab:

- Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

stoplab ptit-static-analysis

- Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

- Khởi động lại bài lab:

- Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

labtainer -r ptit-static-analysis

```
student@ubuntu:~/labtainer/labtainer-student$ checkwork
Results stored in directory: /home/student/labtainer_xfer/ptit-static-analysis
Labname ptit-static-analysis

Student      | String | Unpack | Analy_Function | File_Infor | VirusTotal | Flag |
=====|=====|=====|=====|=====|=====|=====|
B21DCAT151  | Y      | Y      | Y              | Y          | Y          | Y    |
What is automatically assessed for this lab:
```

5. PTIT-STATIC-DANABOT

5.1. Mục đích

Bài lab này giúp sinh viên hiểu và thực hành các nhiệm vụ trong quy trình phân tích tĩnh.

Cụ thể, sinh viên sẽ phân tích mã độc Danabot trên nền tảng Linux với các nhiệm vụ bao gồm **phân tích các đặc tính của file, phân tích mã hash, phân tích string, kiểm tra đóng gói và ẩn giấu, kiểm tra các import**. Qua các nhiệm vụ, sinh viên sẽ tiếp cận và hiểu rõ về quy trình phân tích tĩnh đồng thời đưa ra xác định và dự đoán về hành vi của mã độc Danabot.

5.2. Yêu cầu đối với sinh viên

Có kiến thức cơ bản về hệ điều hành Linux, biết quy trình phân tích tĩnh, biết sử dụng công cụ Detect It Easy và Ghidra.

5.3. Nội dung thực hành

Tìm hiểu về phân tích tĩnh:

Phân tích tĩnh không yêu cầu thực thi mã độc và thường bao gồm các bước sau:

- **Phân tích thuộc tính tập tin:** Xác định tên tập tin, kích thước, dấu thời gian và loại tập tin.
- **Phân tích băm:** Tạo ra giá trị băm để so sánh và xác định sự trùng lặp với các mẫu mã độc đã biết.
- **Phân tích chuỗi:** Tìm kiếm các chuỗi văn bản có thể tiết lộ thông tin về chức năng của mã độc.
- **Kiểm tra đóng gói và mã hóa:** Phát hiện các kỹ thuật đóng gói hoặc mã hóa nhằm ẩn mã độc khỏi các công cụ phân tích.
- **Giải gói:** Nếu mã độc được đóng gói, cần thực hiện giải gói để có thể phân tích mã nguồn.
- **Phân loại malware:** Dựa vào các đặc điểm đã phân tích để xác định họ malware mà mã độc thuộc về.

Giới thiệu mã độc Danabot:

Danabot là một trojan ngân hàng được phát hiện thực tế vào năm 2018. Kể từ lần đầu tiên xuất hiện, Danabot đã nhận được sự yêu thích cao của tội phạm mạng và trở thành mối đe dọa tích cực ở nhiều khu vực trên thế giới. Mã độc này được viết bằng ngôn ngữ lập trình Delphi chứa một số tính năng chống phân tích, cũng như các mô-đun đánh cắp thông tin và điều khiển từ xa được cập nhật thường xuyên, làm tăng thêm sự đe dọa của nó đối với các mục tiêu.

Giới thiệu công cụ:

- Detect It Easy (DIE) là một chương trình đa nền tảng để xác định loại tệp và biết được tệp có bị mã hóa hay không với kiến trúc chữ ký hoàn toàn mở (MS-DOS, PE, ELF, MACH,...). Ngoài ra DIE ở chế độ nâng cao còn có thể cho biết nhiều thông tin của tệp (loại packet, mã hash, entropy, hex, string, import,...). DIE có 3 phiên bản: basic version (die), lite version (diel) và console version ("diec"). Ở bài thực hành này sẽ sử dụng die và diec.
- Ghidra là một công cụ phân tích mã nguồn mở được phát triển bởi Cơ quan Tình báo Trung ương Hoa Kỳ (NSA). Được công bố công khai vào tháng 3 năm 2019, Ghidra cung cấp các tính năng mạnh mẽ để **phân tích mã nhị phân và mã nguồn**.

Khởi động bài lab:

- Vào terminal, gõ:

labtainer ptit-static-danabot

(chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)

Sau khi khởi động xong hai terminal ảo sẽ xuất hiện, một cái là **die** (với công cụ chính là Detect It Easy), một cái là **ghidra**.

Các nhiệm vụ:

- Task 1: Xác định loại tệp.

- Khởi động die chế độ GUI tại terminal **die**: *die danabot_sample.exe &*
- Trên terminal **die** xem thông tin cơ bản của file: *diec -S Info danabot_sample.exe*
- Gõ lệnh quét loại của tệp và kiểm tra mã hóa (lệnh này tương ứng với chọn all types và chọn Scan tại giao diện): *diec -a danabot_sample.exe*

→ File mã độc trên thuộc loại mã độc nào và có kiểu mã hóa nào không?

- Task 2: Phân tích mã hash.

- Tại terminal **die** tính giá trị hash của file mã độc: *diec -S Hash danabot_sample.exe*
- Tại giao diện chọn chế độ Advanced và chọn Hash để có thể xem thêm giá trị hash của các thành phần khác trong mã độc
- Phân tích mẫu với virustotal.com với 1 trong 2 cách
 - ✦ Cách 1: Sử dụng mã hash thu được bên trên vào phần tìm kiếm trên virustotal.com. Hoặc tại giao diện die chọn mục VirusTotal và sử dụng link tại cảnh báo hiện ở terminal.
 - Tại trang virustotal trên trình duyệt. Ở phần Detection, sinh viên bôi đen chuột và copy tất cả các phát hiện và dán lưu vào 1 file mới <detecion> trên terminal.
 - ✦ Cách 2: Đăng ký tài khoản tại trang virultotal.com và lấy apikey. Tại giao diện die chọn Option -> Online tools -> Nhập apikey -> Ok. Sau đó sinh viên chọn mục VirusTotal tại giao diện die.

- Chọn mục sẽ cho kết quả quét file với các phát hiện và chọn nút Save để lưu vào file <detecion>.
- Sinh viên có thể chọn mục Website trên giao diện VirusTotal(on die) để lấy link tại cảnh báo hiện ở terminal

o Sau khi lưu các phát hiện vào file, kiểm tra có bao nhiêu phát hiện dự đoán mã độc là Danabot *grep "Dana" <detecion>*

→ Có bao nhiêu phát hiện đây là mã độc Danabot?

- Task 3: Phân tích string.

- o Tại terminal die, sinh viên tìm các chuỗi có trong chương trình: *strings -n 6 danabot_sample.exe*
- o Tại giao diện die, sinh viên chọn Strings để có thể sử dụng 1 số filter và các kiểu định dạng để dễ tìm kiếm các chuỗi đáng ngờ

→ Kết quả sau khi quét như thế nào (các chuỗi có thể đọc rõ như nào, số lượng các chuỗi khó đọc, địa chỉ ip tìm thấy và so sánh virustotal có địa chỉ ip đó không, qua đó rút ra kết luận gì) ?

- Task 4: Kiểm tra mã độc packed và obfuscated.

- o Tính giá trị entropy của chương trình:
diec -S Entropy danabot_sample.exe
- o Tại giao diện die, sinh viên chọn Entropy để có thể xem rõ giá trị entropy của từng section

⑨ Nhận xét giá trị entropy? Tại sao section .text có entropy cao? Tại sao ở nhiệm vụ 1 không phát hiện mã hóa nào nhưng ở đây die lại dự đoán tỉ lệ packed cao ?

o Phân tích obfuscated strings:

floss -s danabot_sample.exe --no-static-strings Kết quả ra 1 số stackstring nhưng khó có thể đọc.

⑨ Rút ra kết luận về việc section .text có chứa các stackstring và việc sử dụng nó để làm mờ, che giấu mã độc để thực hiện các hành vi đáng ngờ khác

- Task 5: Phân tích tập tin PE.

- Tại giao diện die, chọn Import để xem các import từ các dll có trong chương trình
 - Chọn Save các import từ KERNEL32.dll vào file <import> và thực hiện hiển thị các import đó trong terminal: `cat <import>`
 - ⑨ Từ các import được tìm thấy kết hợp với mục Behavior trên virustotal.com, chỉ ra các import đáng ngờ có thể khớp với MITRE ATT&CK Tactics and Techniques trên virustotal.
 - ⑨ Nhận xét về khả năng của chương trình? Khả năng này có giống với Trojan không?
- **Task 6: Phân tích mã nguồn** o Phân tích mã nguồn là 1 nhiệm vụ khó khăn, đặc biệt với mã độc có khả năng che giấu là 1 đặc điểm chính của Trojan. Ở đây sinh viên chỉ cần thực hiện tìm ra đoạn mã có chứa nơi bị che giấu.
- Tại terminal **ghidra**, sinh viên khởi động ./ghidra và thực hiện phân tích mã nguồn của file danabot_sample.exe
 - Để tìm nơi bị che giấu, sinh viên chọn vào section có entropy cao, và dùng chức năng Function Call Tree có trong ghidra. Để chắc chắn hãy kiểm tra xem function call tree nhận được có node gốc từ entry hoặc main (là các hàm khởi tạo thường thấy của chương trình)
 - Sinh viên chọn lấy 1 function cách xa entry nhất trong function call tree nhận được và xem mã C
- Nhận xét về function nhận được (Số lượng giá trị khởi tạo, các vòng lặp và phép tính, số lượng hàm được gọi) ? Đưa ra dự đoán về khả năng của function này ?
- o Tìm địa chỉ đầu và địa chỉ cuối của function có dạng 0x004d__ và nhập vào terminal die với câu lệnh sau để hiển thị mã nguồn gốc của function đó.
- ```
objdump -D --section=< section_name> --startaddress=<start_address> -
-stop-address=<stop_address> danabot_sample.exe
```
- **Task 7: Phân tích khả năng chính (Nâng cao)** o Thông qua các phân tích trên, chúng ta chỉ biết được 1 số thông tin và có thể rút ra đây là mã độc Trojan còn các khả năng chính của nó đã được ẩn giấu. Để phân tích rõ xem hoạt động của nó, tại terminal
- đã có file danabot\_sample.dll là 1 file dll được tạo ra từ hoạt động ẩn giấu của chương trình chính, phân tích file dll này sẽ biết được nhiều hơn về khả năng của danabot.

- Dùng lệnh strings tìm ra link virustotal.com được giấu trong o Scan nó trên giao diện die và chọn Import để xem các import được sử dụng.
- So sánh các import và Behavior trên trang virustotal tìm được **Kết thúc**

**bài lab:**

- Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab: *stoplab ptit-static-danabot*
- Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

**Khởi động lại bài lab:**

- Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh: *labtainer -r ptit-static-danabot*