

Task 1: Solving a Linux crackme (puzzle)

- **use pwntools (install it) and Python to programatically call the binary directly and get its output (1p)**

The code can be found in task11.py.

Calling the script with no arguments will run the “crackme” executable with input “1234”.

The output of the script will be printed in the console.

- **stop calling the binary directly; wrap it inside ltrace and get all the library functions called (1p)**

The code can be found in task12.py.

Calling the script with no arguments will run the “crackme” executable wrapped in "ltrace" with input “1234”.

An array containing the library function calls will be printed in the console.

- **bypass the length check by trying various inputs (2p)**

The code can be found in task13.py.

Calling the script with no arguments will run the “crackme” executable wrapped in "ltrace" with variable length input.

For each length from 0 to 99, an array containing the function calls will be written in a file called "output3.txt".

It can be seen that for an input of length 70 (71 including terminator), a new function call to strstr is made, so this must be the right length.

- **pass all the other checks (2p)**

The code can be found in task14.py.

I found those "special" strings by iteratively passing the checks (represented by “strstr” calls) from the script "crackme" wrapped in "ltrace"; each “strstr” call reveals a subsequence of characters that must be found somewhere within the correct input.

With the help of this script I found 7 strings of length 10 that are part of the correct input (those 7 strings can be found in the task14.py script)

- **find the correct password (2p)**

The code can be found in task15.py.

I generated all possible permutations of the 7 strings, used them as input for “crackme” and printed their corresponding flag.

To narrow the search I only kept the values which have more than 20 ASCII printable characters in the flag.

Looking through the output, I found the correct input as “nhnewfhethkmdcdyamgeczihldazjcnhhqtjylumfvlgmhasbwjqvanafylzyemlopqosj” which has the flag “timctf{7dfadd1ee67a9c516c9efbf8f0cf43f4}”.

Task 2: Investigating a Windows malware

- Where does it connect to? (2p)

I used “API Monitor” to analyse “malware.exe”.

To find out what connections are made, I checked the tab “Internet” to only see API calls related to Internet connections.

As it can be seen in the screenshot below, a connection is made to the “maybe.suspicious.to” URL and also a GET request is sent for “/secondstage” from “maybe.suspicious.to”.

Monitoring - API Monitor v2 32-bit (Administrator)

API Filter: All Modules

Monitored Process: C:\Workspace\lab_1v\malware.exe

Summary: 5 calls | 2 KB used | malware.exe

#	Time of Day	Thread	Module	API
1	7:09:09.118 AM	1	malware.exe	InternetCrackUrlA ("http://maybe.suspicious.to/secondstage", 0, 0, 0x0015)
2	7:09:09.118 AM	1	malware.exe	InternetOpenA ("Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident...)
3	7:09:09.150 AM	1	malware.exe	InternetConnectA (0x00cc0004, "maybe.suspicious.to", INTERNET_DEFAULT_...
4	7:09:09.150 AM	1	malware.exe	HttpOpenRequestA (0x00cc0008, "GET", "/secondstage", NULL, NULL, 0x00...
5	7:09:09.150 AM	1	malware.exe	HttpSendRequestA (0x00cc0008, "Accept-Language: en-us", 24, NULL, 0)

Parameters: InternetConnectA (Wininet.dll)

#	Type	Name	Pre-Call Value	Post-Call Value
1	HINTERNET	hinternet	0x00cc0004	0x00cc0004
2	LPTSTR	lpszServerName	0x0019f8c6 "maybe.suspicious.to"	0x0019f8c6 "maybe.suspicious.to"
3	INTERNET_PORT	nServerPort	INTERNET_DEFAULT_HTTP_PORT	INTERNET_DEFAULT_HTTP_PORT
4	LPTSTR	lpszUserName	NULL	NULL
5	LPTSTR	lpszPassword	NULL	NULL
6	DWORD	dwService	INTERNET_SERVICE_HTTP	INTERNET_SERVICE_HTTP
7	DWORD	dwFlags	0	0
8	DWORD_PTR	dwContext	0	0

Hex Buffer: 20 bytes (Pre-Call)

```
0000 6d 61 79 62 65 2e 73 75 maybe.su
0008 73 70 69 63 69 65 73 73 spicious
0010 2e 74 65 00 .to.
```

Output

```
Process Stopped - PID: 804 | backgroundTaskHost.exe
Process Stopped - PID: 6320 | MicrosoftEdgeCP.exe
Process Started - PID: 5668 | 64-bit | MicrosoftEdgeCP.exe | Unsupported architecture.
Process Started - PID: 3508 | 64-bit | Windows.WARP.JITService.exe | Unsupported architecture.
Process Stopped - PID: 3548 | svchost.exe
Service Stopped - PID: 3548 | Microsoft Store Install Service
Process Stopped - PID: 1416 | RuntimeBroker.exe
Process Stopped - PID: 1068 | backgroundTaskHost.exe
Process Stopped - PID: 2668 | RuntimeBroker.exe
```

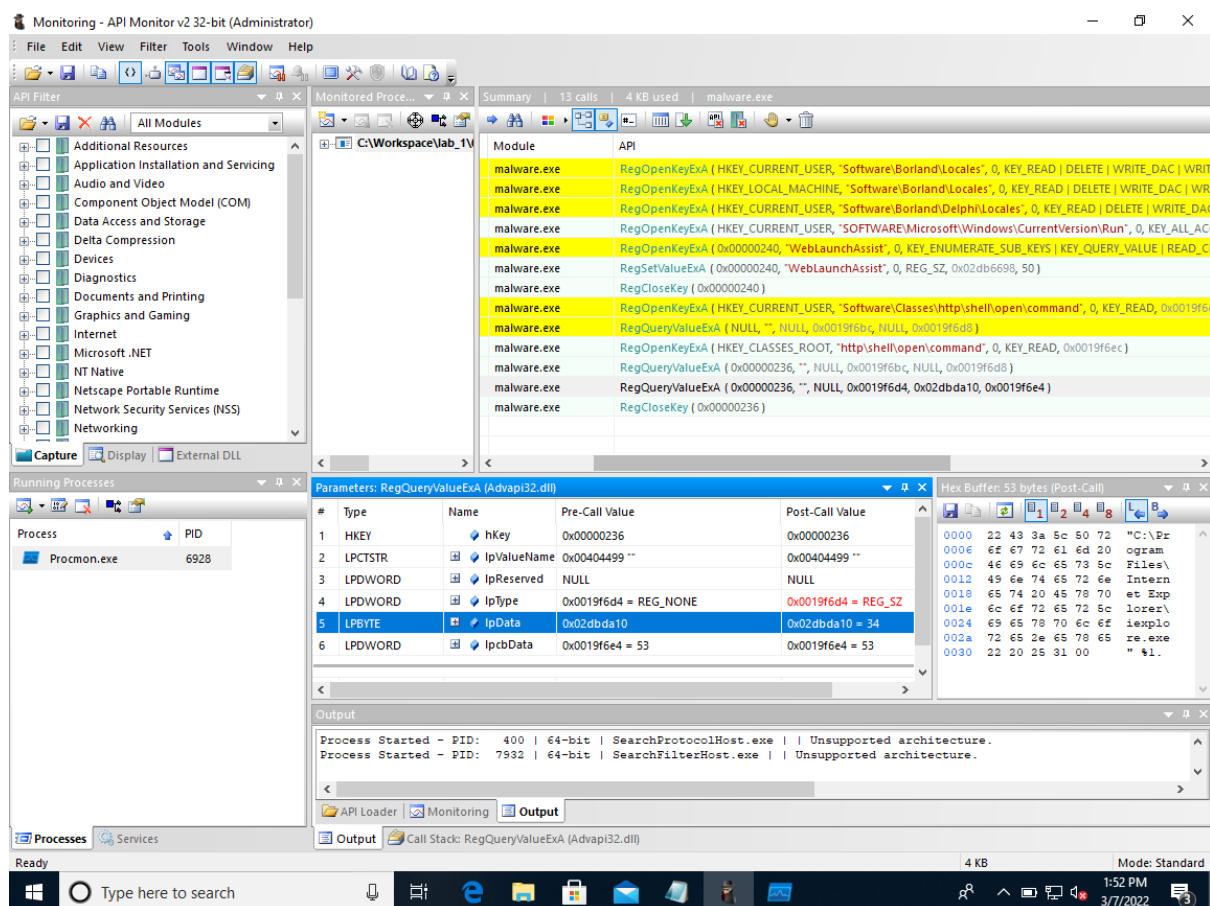
- What registry keys does it access and why? (2p)

I used "API Monitor" to analyse "malware.exe".

To find out what registry keys are accessed, I checked the tab "System Services / Windows System Information / Registry" to only see the API calls related to registries.

As it can be seen in the screenshot below, there are multiple keys being accessed:

- Under the root key HKEY_CURRENT_USER:
 - "Software\Borland\Locales"
 - "Software\Borland\Delphi\Locales"
 - "Software\Microsoft\Windows\Current\Version\Run"
 - "Software\Classes\http\shell\open\command"
- Under the root key HKEY_LOCAL_MACHINE:
 - "Software\Borland\Locales"
- Under the root key HKEY_CLASSES_ROOT:
 - "http\shell\open\command"



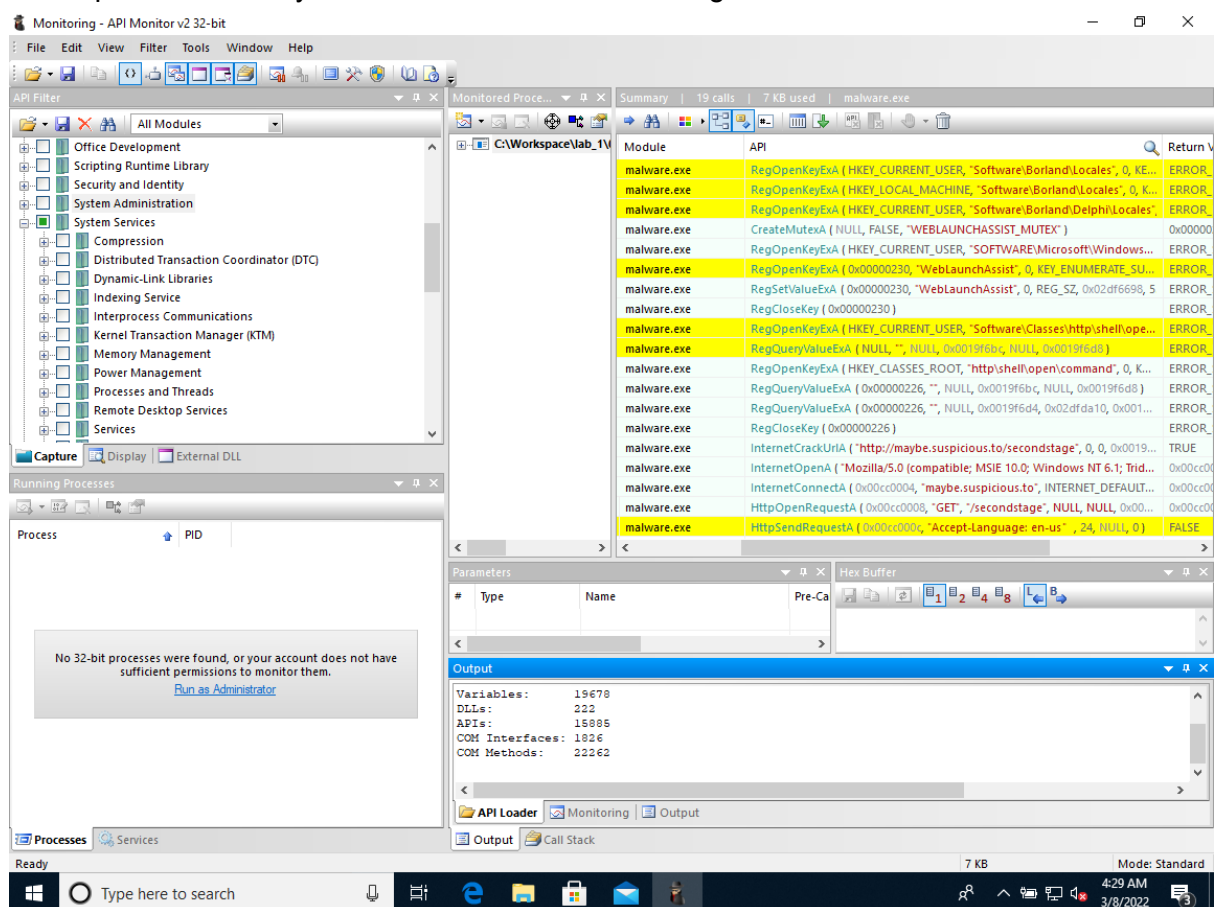
The purpose for accessing these registries is to set the value of a register "WebLaunchAssist" to the path of executable "weblaunchassist.exe" (this is done through the RegSetValueExA API call) and to find what the default browser is (this is done by querying the "http\shell\open\command" key - using RegQueryValueExA API call - from multiple locations based on priority); the value from the register containing the path to the

default browser (“C:\Program Files\Internet Explorer\iexplore.exe”) can be seen in the screenshot above.

Bonus task: Malware vaccine

- To get the bonus points, figure out how this malware uses synchronization to avoid reinfection and then devise a way to “vaccinate” machines against this malware. (4p)

Similar to exercise 2, I use API Monitor to analyse “malware.exe” with all the filters previously mentioned to which I added the filter “System Services / Synchronization / Mutex”; the output for this analysis can be found in the following screenshot.



It can be seen that a mutex called “WEBLAUNCHASSIST_MUTEX” is created before most of the actions, so I write a script “taskbonus_vaccine.py” that creates and acquires that mutex before going to sleep for a period of time; in a real time scenario, an indefinite amount of time can be used and the script can be added to an autorun location, to have protection all the time.

Now, with the “vaccine” script running, I also start the analysis from API Monitor using the same filters; as it can be seen in the screenshot below no actions are made after the API call CreateMutexA.

Monitoring - API Monitor v2 32-bit

File Edit View Filter Tools Window Help

API Filter

Monitored Process: C:\Workspace\lab_1\malware.exe

Summary 4 calls 1 KB used malware.exe

Module	API	Return
malware.exe	RegOpenKeyEx (HKEY_LOCAL_MACHINE, "Software\Borland\Locales", 0, K...	ERROR
malware.exe	RegOpenKeyEx (HKEY_CURRENT_USER, "Software\Borland\Delphi\Locales",	ERROR
malware.exe	CreateMutexA (NULL, FALSE, "WEBLAUNCHASSIST_MUTEX")	0x00

Running Processes

Process

No 32-bit processes were sufficient p...

Command Prompt - python taskbonus_vaccine.py

```
Microsoft Windows [Version 10.0.17763.379]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\IEUser>cd ../../Workspace/lab_1
C:\Workspace\lab_1>cd 01-lab-files
C:\Workspace\lab_1\01-lab-files>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Workspace\lab_1\01-lab-files>python taskbonus_vaccine.py
True
Traceback (most recent call last):
  File "C:\Workspace\lab_1\01-lab-files\taskbonus_vaccine.py", line 6, in <module>
    time.sleep(60)
KeyboardInterrupt
^C
C:\Workspace\lab_1\01-lab-files>python taskbonus_vaccine.py
Acquired the mutex, going to sleep for a minute

C:\Workspace\lab_1\01-lab-files>python taskbonus_vaccine.py
Acquired the mutex, going to sleep for a minute
```

Processes Services

Output Call Stack

Ready 1 KB Mode: Standard

Type here to search

4:42 AM 3/8/2022