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 tring 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 “nhnewfhetkmdcdyamgeczihldazjcnhhqtjylumfvlrgmhasbwjqvanafylzyemlopqosj” which has the flag “timctf{7dfadd1ee67a9c516c9efbf8f0cf43f4}”.

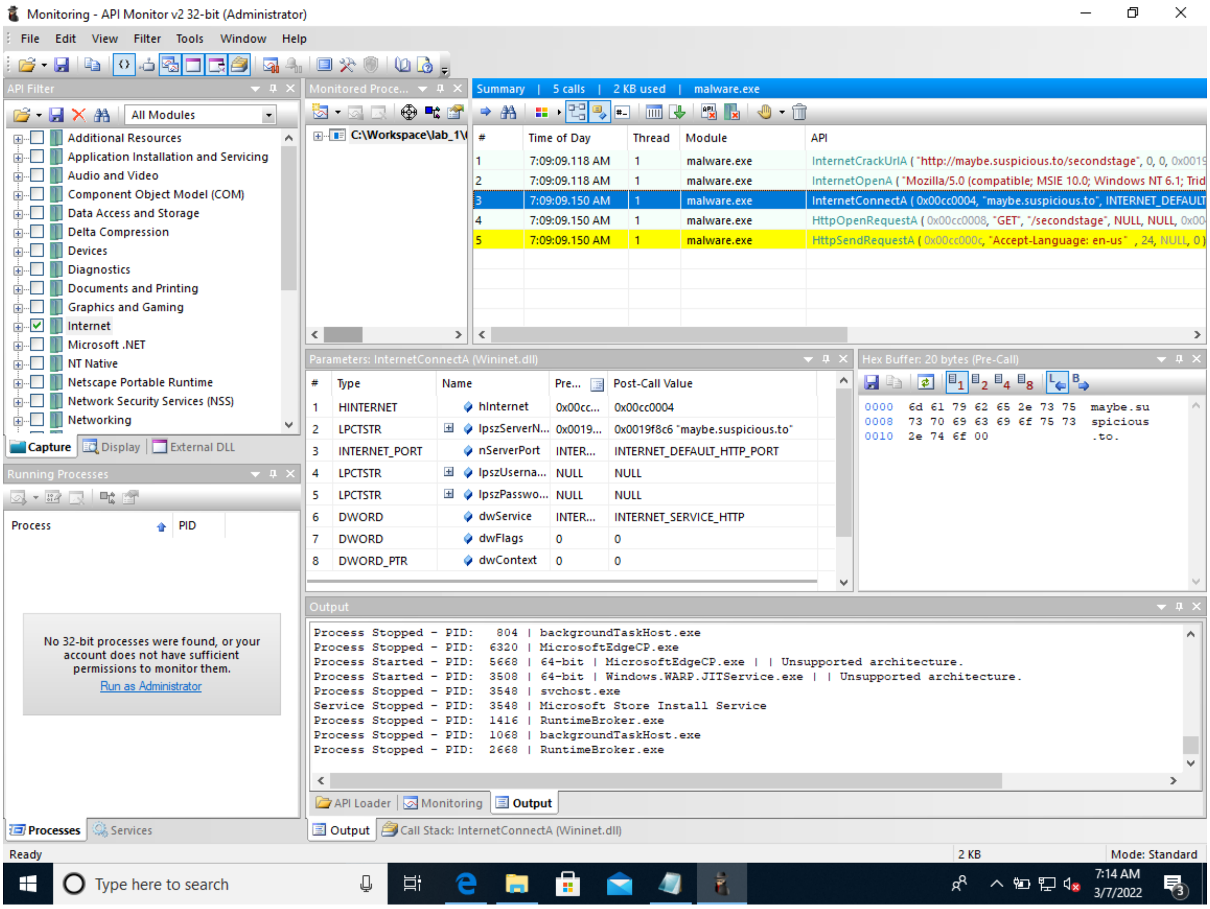
Task 2: Investigating a Windows malware

- **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”.



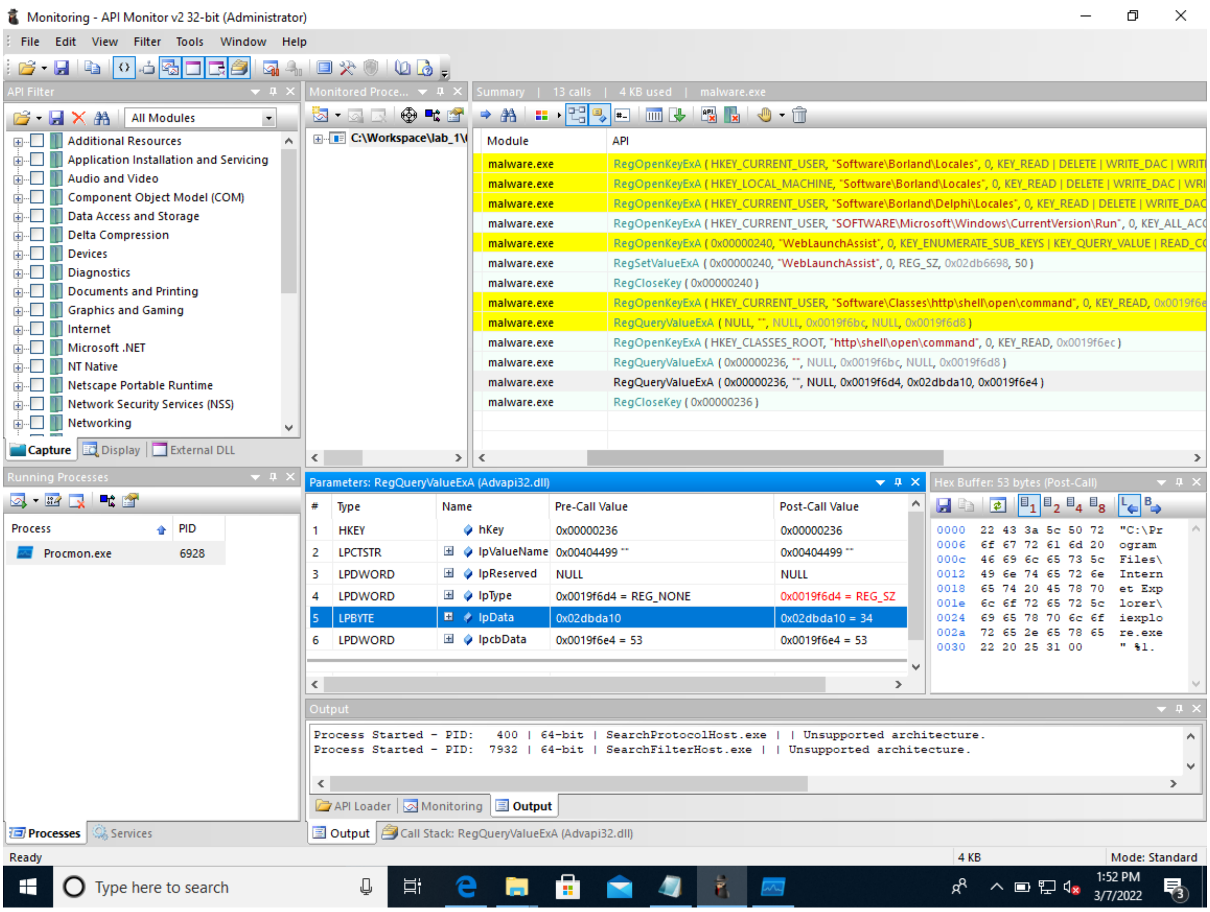
- **What registry keys does it access and why? (2p)**

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”

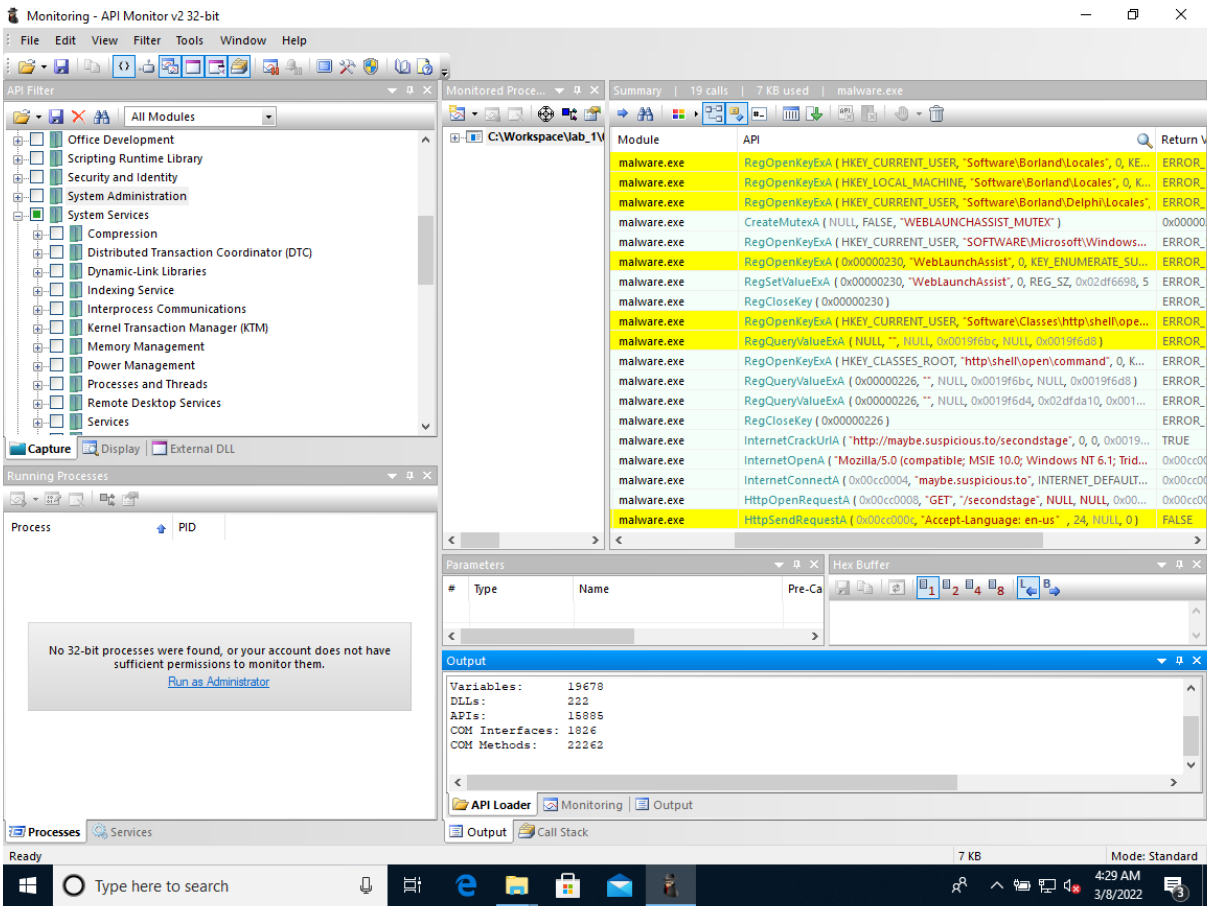


The purpose for accessing these registries is to set the value of a register “WebLaunchAssist” to the path of executable “weblaunchassist.exe” (this is done through the RegSetValueExA API call) and to find what the default browser is (this is done by querying the “http\shell\open\command” key - using 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

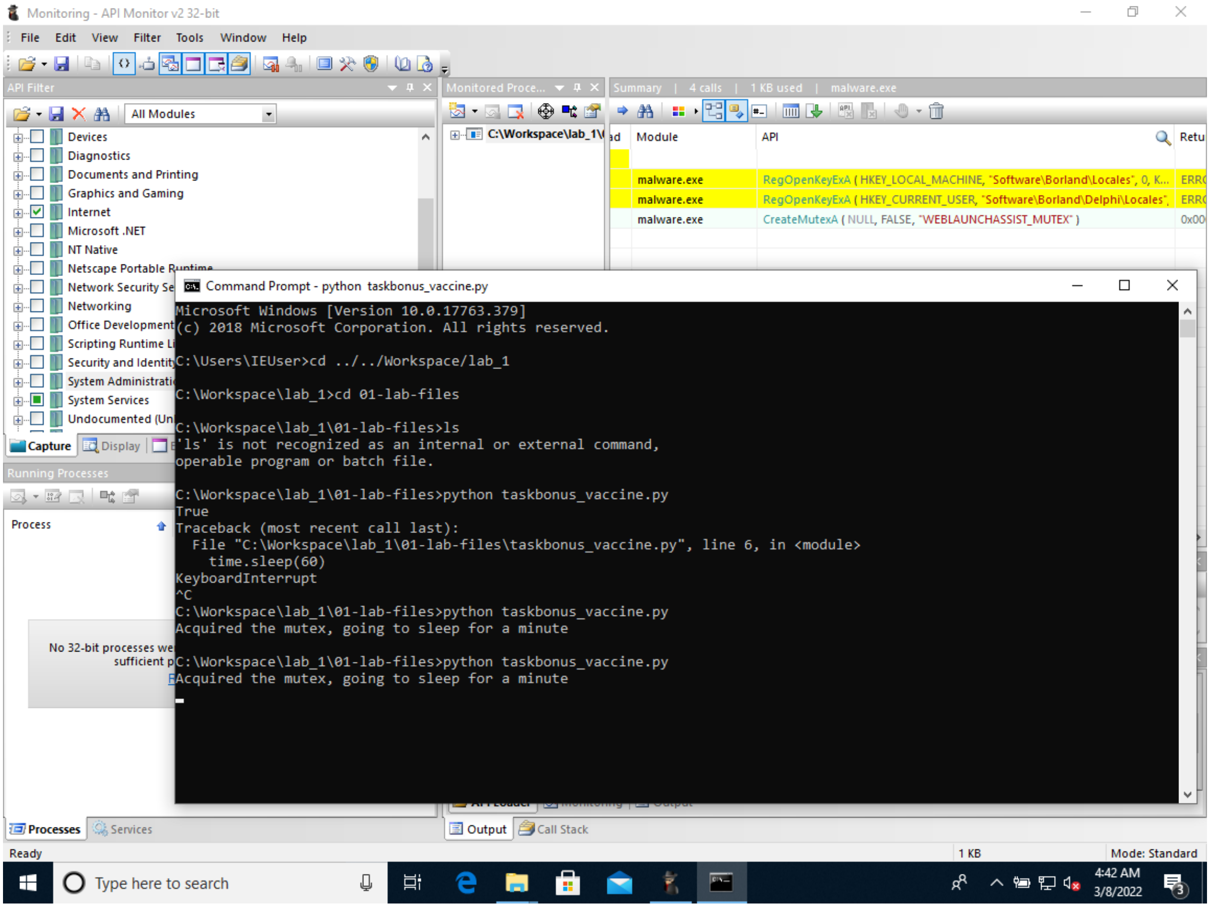
- **To get the bonus points, figure out how this malware uses synchronization to avoid reinfection and then devise a way to “vaccinate” machines against this malware. (4p)**

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
All Modules
Devices
Diagnostics
Documents and Printing
Graphics and Gaming
Internet
Microsoft .NET
NT Native
Monitored Proce...
C:\Workspace\lab_1\
Summary | 4 calls | 1 KB used | malware.exe
Module
API
malware.exe
RegOpenKeyExA (HKEY_LOCAL_MACHINE, "Software\Borland\Locales", 0, K...
malware.exe
RegOpenKeyExA (HKEY_CURRENT_USER, "Software\Borland\Delphi\Locales",
malware.exe
CreateMutexA (NULL, FALSE, "WEBLAUNCHASSIST_MUTEX")
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