Dynamic Analysis

What is malware analysis?

and

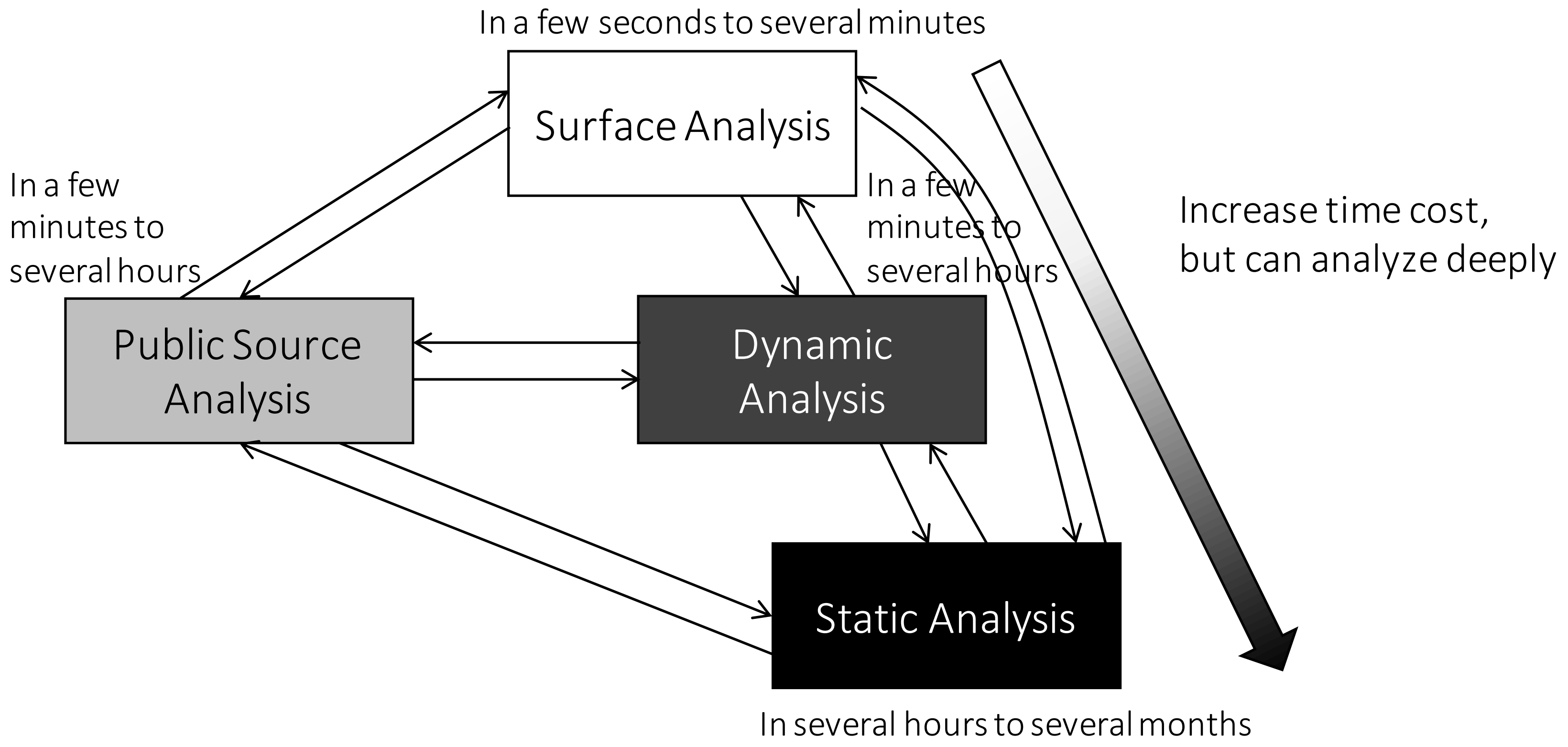
What is dynamic analysis?

What is Malware Analysis?

- It is to reveal malware's behavior combining with the below methods.
  - Surface Analysis
  - Dynamic Analysis (Runtime analysis, Black box analysis)
  - Static Analysis (White box analysis, Reverse (Code) Engineering, Reversing...)
    - Terms and definitions are not fixed.
      - Sometimes, surface analysis is included in static analysis.
    - There is "public source analysis" as well (in other words, googling ;-)).

What is Malware Analysis?

- Each analysis method is related to the others.



What is Dynamic Analysis?

- To execute malware and record malware activities with analysis tools, typically on a closed environment (e.g. virtual machine)

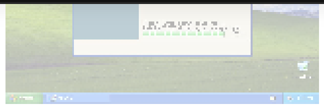
- We need to record:
  - Process Activities
  - Registry Activities
  - File Activities
  - Network Activities (with Internet emulation)
    - Internet emulation redirects communications from malware to Internet emulation software and records host names and/or IP addresses of C2 servers and its contents.

What is Dynamic Analysis?

malwr

wizSafe

Analyses Search Submit About Sign up Login



Hosts

IP
54.186.255.26
198.7.61.118
54.187.82.120
162.210.192.21

Domains

DOMAIN	IP
r1.getapplicationmy.info	54.186.255.26
c1.downlloaddatamy.info	54.186.255.26
i1.proffiiget.in	198.7.61.118
suretertminal.net	54.187.82.120
datadownloadscan.info	162.210.192.21
proffidrivergold.info	91.109.18.46

Summary

Files Registry Keys Mutexes

```
C:¥DOCUME~1¥User¥LOCALS~1¥Temp¥cfc2f0266985da92fdd3bbda494f1604
C:¥DOCUME~1¥User¥LOCALS~1¥Temp¥Tsu5DCE2BEE.dll
C:¥DOCUME~1¥User¥LOCALS~1¥Temp¥cfc2f0266985da92fdd3bbda494f1604.log
C:¥WINDOWS¥system32¥wininet.dll
```

What is Dynamic Analysis?

- If you do dynamic analysis manually, you can do it with these tools.
  - Virtual Machine environments
    - VMware
    - VirtualBox
    - Hyper-V
    - …
  - Process activities
    - Process Explorer
    - Process Hacker
    - Process Monitor
      - noriben
    - Sysmon
  - Registry activities
    - Process Monitor
    - regshot
- File activities
  - Process Monitor
  - regshot
- Internet Emulation
  - Fakenet, fakenet-ng
  - InetSim
- Network activities, packet capture
  - fakenet , fakenet-ng
  - wireshark

Exercise 1

Dynamic Analysis using Noriben, Procmon, Fakenet-ng

Exercise 1 (1)

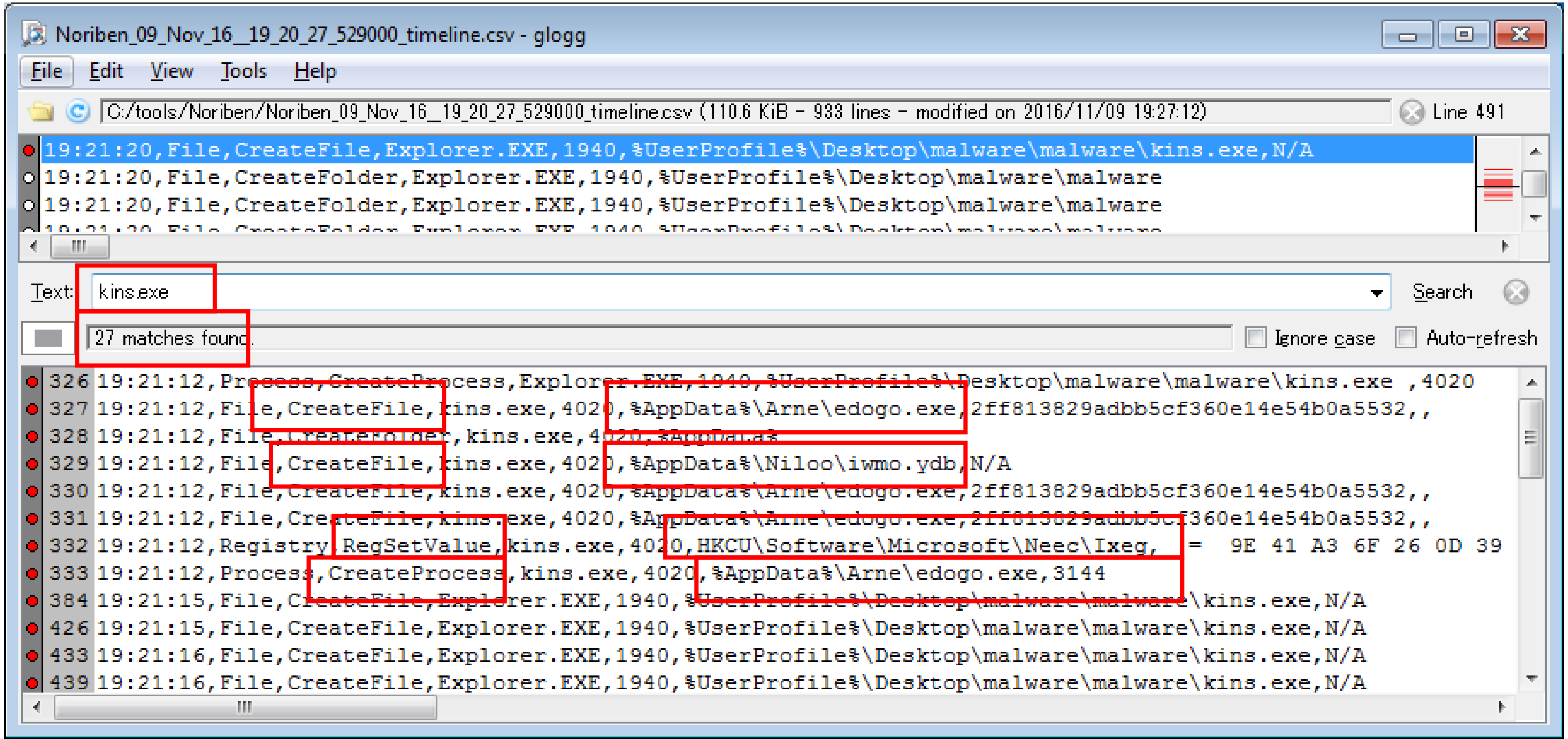
- Double-click Fakenet32.exe
  - Click “yes” when the UAC dialog shows up
- Double-click Noriben.py
  - Click “yes” when the UAC dialog shows up
- Double-click kins.exe (Banking Trojan)
  - Wait for approximately four minutes

Exercise 1 (2)

- About four minutes later, if you see suspicious communications on Fakenet-ng window, then press Ctrl + C and quit Fakenet-ng.
- Press Ctrl + C on Noriben window as well and wait for report creation for a few minutes.
- On the report of Noriben,
  - Grep activities for “kins.exe”
  - Grep file names, process names and registry key names related to “kins.exe”
- If you need further investigation, you can use these files in Noriben folder.
  - PML (raw log data of Procmon)
  - Timeline report (csv file)
- There is a pcap file in Fakenet* folder as well.

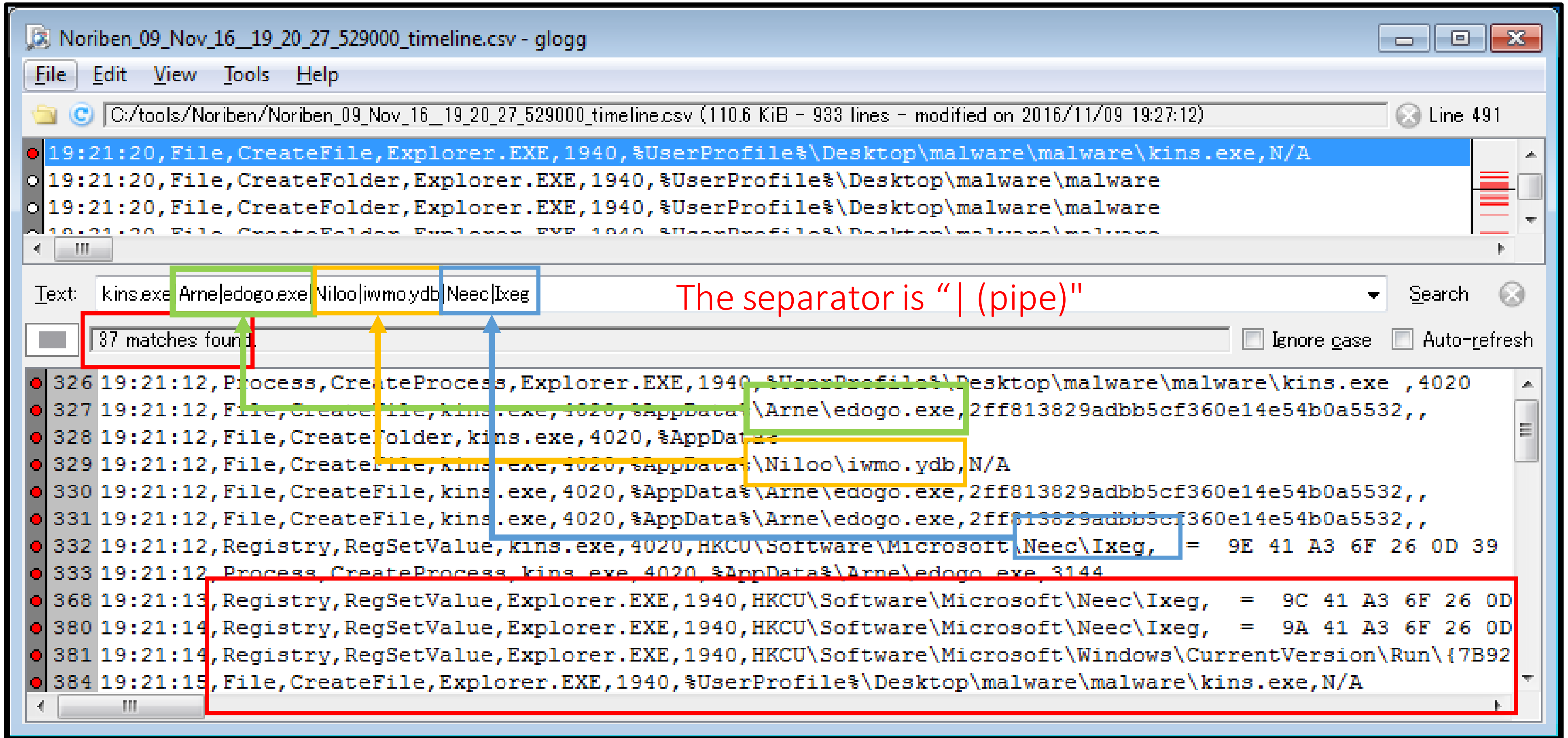
Exercise 1 (3)

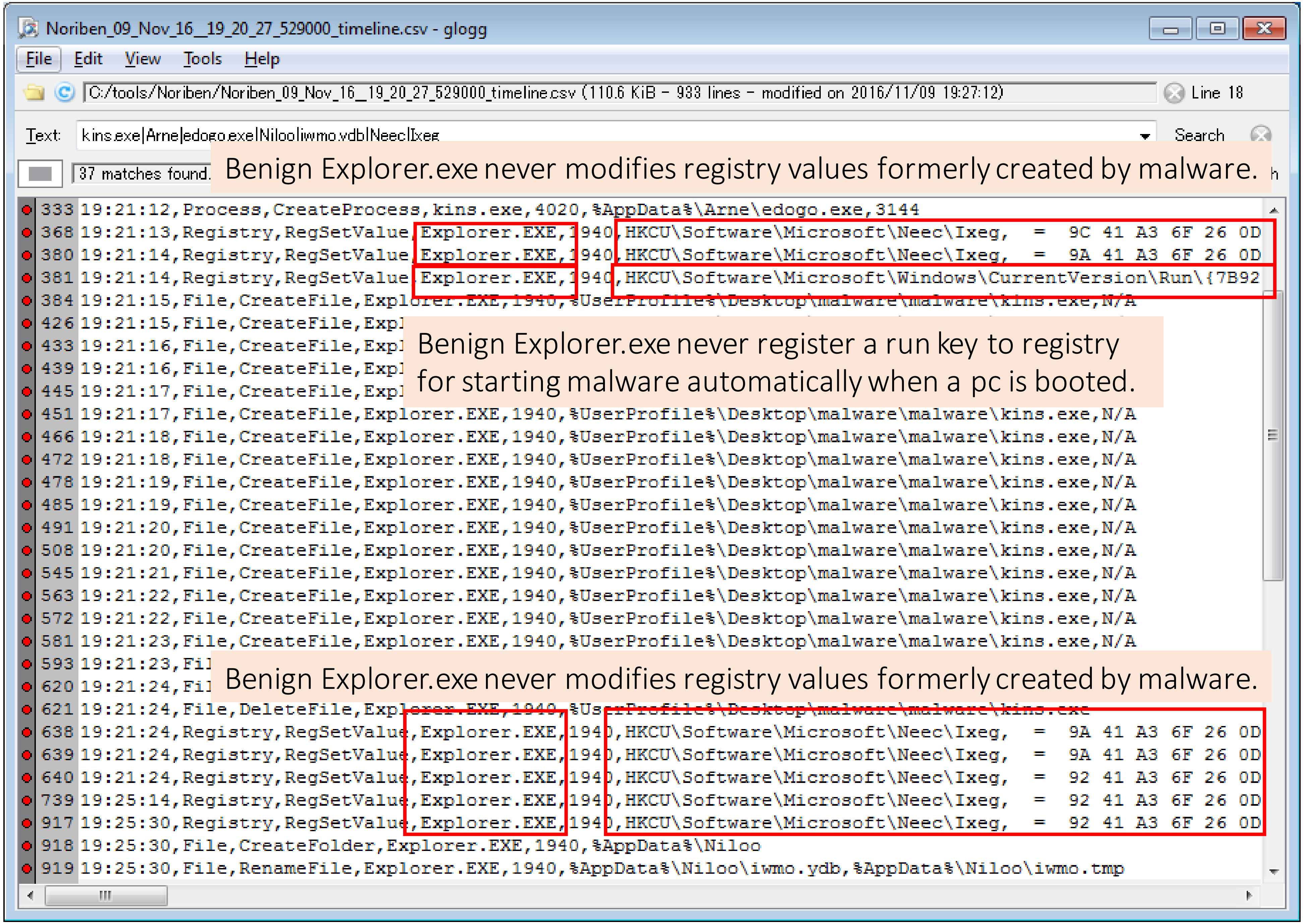
- Load a timeline report from “Noriben” into “glogg”.
  - Then type “kins.exe” to collect all “kins.exe” activities.



Exercise 1 (4)

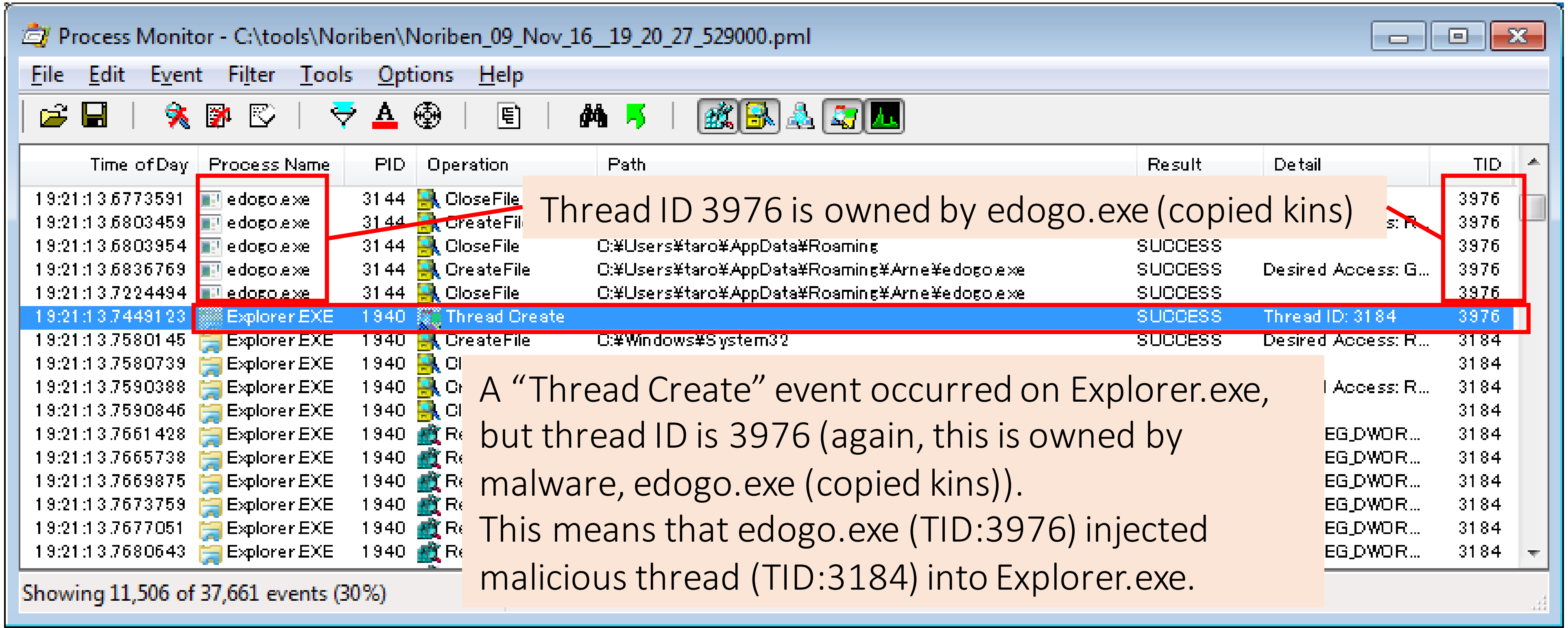
- Add the files and reg keys related to “kins.exe”
  - Then, you can find another activities related to this malware.




Noriben_09_Nov_16__19_20_27_529000_timeline.csv - glogg
File Edit View Tools Help
C:/tools/Noriben/Noriben_09_Nov_16__19_20_27_529000_timeline.csv (110.6 KiB - 933 lines - modified on 2016/11/09 19:27:12)
Line 18
Text: kins.exe|Arne|edogo.exe|Niloo|iwmo.ydb|Neec|Ixeg
Search
37 matches found.
Benign Explorer.exe never modifies registry values formerly created by malware.
333 19:21:12,Process,CreateProcess,kins.exe,4020,%AppData%\Arne\edogo.exe,3144
368 19:21:13,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Neec\Ixeg, = 9C 41 A3 6F 26 0D
380 19:21:14,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Neec\Ixeg, = 9A 41 A3 6F 26 0D
381 19:21:14,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{7B92
384 19:21:15,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
426 19:21:15,File,CreateFile,Expl
433 19:21:16,File,CreateFile,Expl
439 19:21:16,File,CreateFile,Expl
445 19:21:17,File,CreateFile,Expl
Benign Explorer.exe never register a run key to registry for starting malware automatically when a pc is booted.
451 19:21:17,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
466 19:21:18,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
472 19:21:18,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
478 19:21:19,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
485 19:21:19,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
491 19:21:20,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
508 19:21:20,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
545 19:21:21,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
563 19:21:22,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
572 19:21:22,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
581 19:21:23,File,CreateFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe,N/A
593 19:21:23,Fil
620 19:21:24,Fil
Benign Explorer.exe never modifies registry values formerly created by malware.
621 19:21:24,File,DeleteFile,Explorer.EXE,1940,%UserProfile%\Desktop\malware\malware\kins.exe
638 19:21:24,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Neec\Ixeg, = 9A 41 A3 6F 26 0D
639 19:21:24,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Neec\Ixeg, = 9A 41 A3 6F 26 0D
640 19:21:24,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Neec\Ixeg, = 92 41 A3 6F 26 0D
739 19:25:14,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Neec\Ixeg, = 92 41 A3 6F 26 0D
917 19:25:30,Registry,RegSetValue,Explorer.EXE,1940,HKCU\Software\Microsoft\Neec\Ixeg, = 92 41 A3 6F 26 0D
918 19:25:30,File,CreateFolder,Explorer.EXE,1940,%AppData%\Niloo
919 19:25:30,File,RenameFile,Explorer.EXE,1940,%AppData%\Niloo\iwmo.ydb,%AppData%\Niloo\iwmo.tmp

Exercise 1 (5)

- This is a suspicious sign for remote code injection into legitimate and existing explorer.exe!
  - Further investigation, you can find the evidence of remote thread injection from raw procmon log (.pml file).



Exercise 1 (6)

Summary of malicious activities

Activities		Value	Source
Network activities	URL	https://dimitfruit.com/s186/lkp13.jpg *1	Fakenet
	Method	GET	Fakenet
File activities	Create	%AppData%_4-6random__4-6random_.exe (copied itself)	Noriben/procmon
	Delete	Itself (original one)	Noriben/procmon
	Modify	Itself (copied one)	Hash value
Process activities	Create	Itself (copied one)	Noriben/procmon
	Thread injection	Target: Explorer.exe (legitimate and existing process)	procmon
Registry activities	Create	HKCU\Software\Microsoft_4-6random__random_ Value: _unknown binary_ *2	Noriben/procmon
	Create	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\\{_GUID_} Value: _path_to_itself_ (copied one)	Noriben/procmon

*1: Path does not appear in the actual proxy log because this malware uses https.
*2: Actually, this value is encrypted “BaseConfig”.

Exercise 1 (7)

- We can get various results like the previous slide even if we don't have commercial sandboxes. Those free tools we mentioned earlier help us.
  - Communications
    - C2 servers
    - URL / method ...
  - File activities
  - Registry activities
- We can do first response using this information.
  - E.g. finding other infected machines

Exercise 1 (8)

- But sometimes, we may encounter that malware doesn’t work or its behavior is different between in real PCs and in VMs.

- Possible reasons why a malware may not work properly include:
  - VM or analysis environment detection
  - Difference in OSes, hardware, language environments
  - Time bomb…

The Reason Why Malware Does not Work

- One of the most likely causes of the problem is analysis environment detection.
  - There are many techniques to detect analysis environment.
    - VM detection
      - Detecting backdoor ports (Host-Guest communication)
      - Detecting differences between real PCs and VMs by executing some specific machine instructions
      - Detecting virtual devices (e.g. Motherboard, NIC, HDD) from registry or via COM
    - Product IDs of OSes
    - Detecting process and file names that works only in VM or sandbox environments
    - Detecting analysis tools
    - Checking the number of CPUs
    - …

Avoiding Analysis Environment Detection (1)

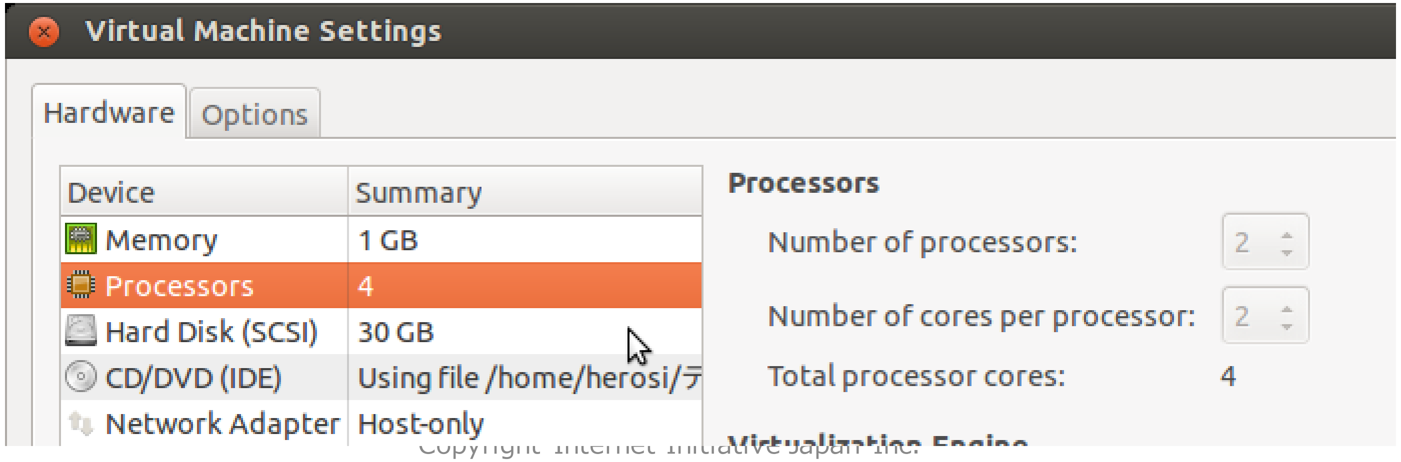
- The easiest way against such detections is to execute malware on real machines.
  - It’s tough to recover though.
    - Recovering real machines can be done by using “FOG” or similar tools.

- The second best is to try multiple analysis environments.
  - Because some malware detects only some specific VM environments.
    - VMware
    - KVM
    - Hyper-V
    - VirtualBox
    - …

- However, these are not the perfect solutions for avoiding analysis environment detection.
  - Some malware samples might run okay, but many samples will detect that it is being executed on an analysis environment and quit running.

Avoiding Analysis Environment Detection (2)

- We can deal with some VM detection techniques in advance.
  - Disabling the backdoor port of VMware
    - monitor_control.restrict_backdoor = "TRUE"
    - vmci0.present = "FALSE"
  - Increasing the number of virtual CPUs to two or more



Avoiding Analysis Environment Detection (3)

- We still can deal with such malware even when the solutions mentioned earlier don’t work .

Avoiding Analysis Environment Detection (4)

- The solution is to read Windows APIs which malware use.

Avoiding Analysis Environment Detection (5)

- Malware needs to request many important operations to the Windows OS through APIs such as below.
  - Communications with other hosts
  - File handling
  - Registry handling
  - Process creation
  - Code injection
  - Memory management
    - including reading and writing data from/to memory regions of other processes
  - Enumerating processes
  - …

- So, if we understand strategies of malware authors, and if we observe APIs that the malware uses, we can figure out the answer why the malware doesn't work properly and how to deal with the problems.

Exercise 2
Rewriting API Responses with Debuggers

Avoiding HDD device name detection

Exercise 2 (1)

- Next let’s see one of the VM detection techniques.
- First, revert your VM.
- Next, execute “Noriben.py” and “Fakenet-ng”.
- Double-click “gozi_ursnif.exe”.

wizSafe

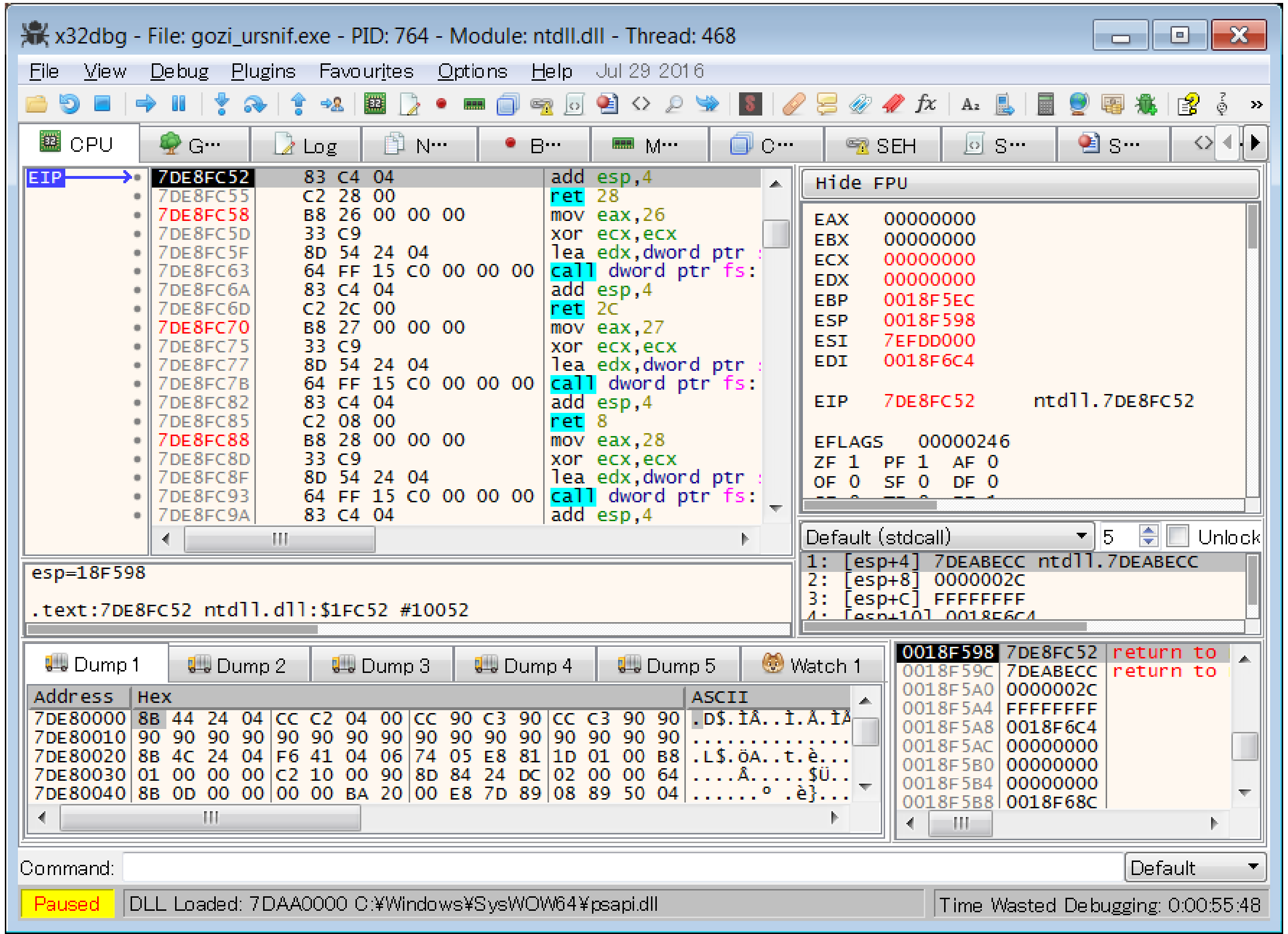
Exercise 2 (2)

- Nothing happens ☹
  - Quit “Noriben.py” and “Fakenet-ng”.
- Actually, this malware checks HDD names with this API. Let’s check this with a debugger!
  - SetupDiGetDeviceRegistryPropertyA

```
BOOL SetupDiGetDeviceRegistryProperty(
  _In_      HDEVINFO         DeviceInfoSet,
  _In_      PSP_DEVINFO_DATA DeviceInfoData,
  _In_      DWORD            Property,
  _Out_opt_ PDWORD           PropertyRegDataType,
  _Out_opt_ PBYTE            PropertyBuffer,
  _In_      DWORD            PropertyBufferSize,
  _Out_opt_ PDWORD           RequiredSize
);
```

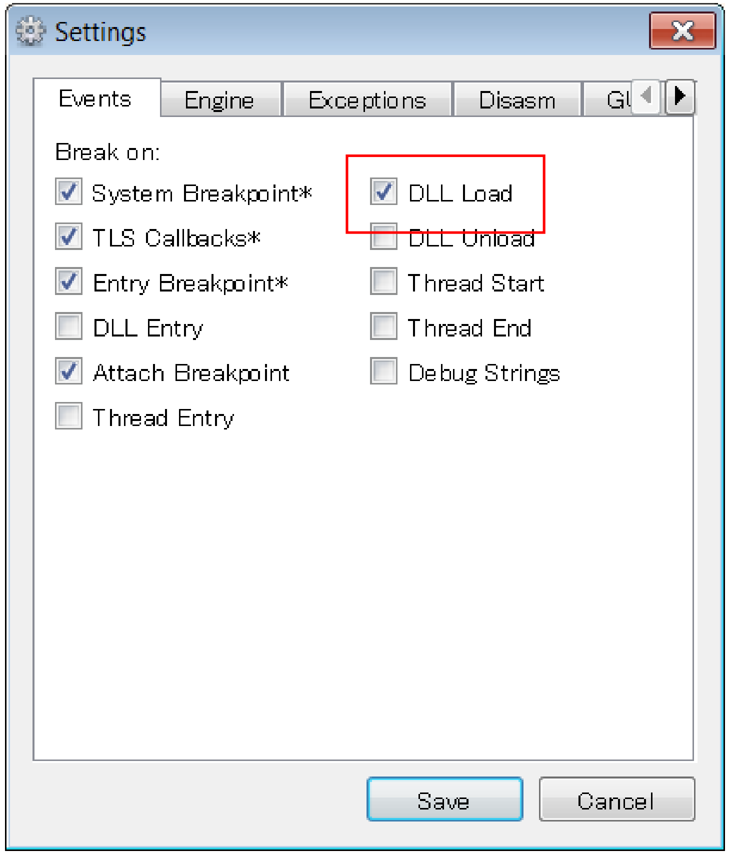
Exercise 2 (3)

- Load “gozi_ursnif.exe” into x32dbg



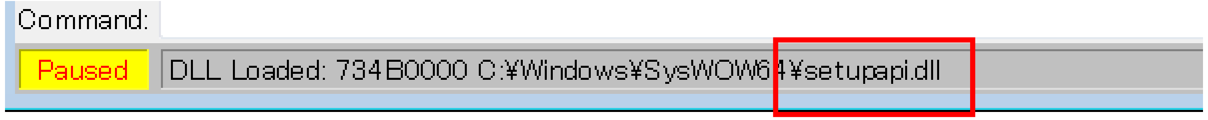
Exercise 2 (4)

- Options -> Preferences
  - Go to “Events” tab, and check “DLL Load”.

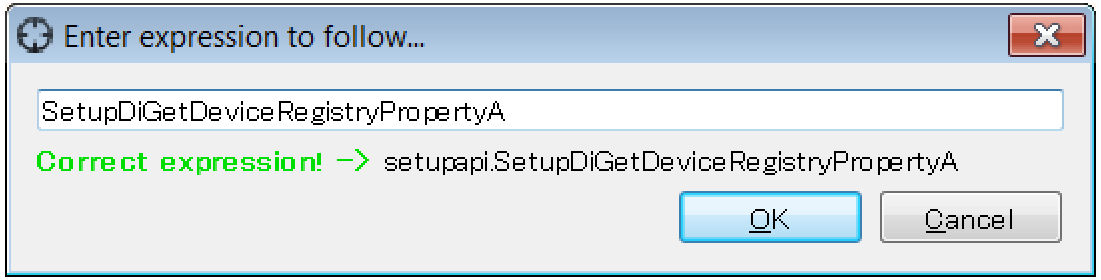


Exercise 2 (5)

- Press “F9” several times until you see “setupapi.dll” at the left bottom of the x32dbg window.

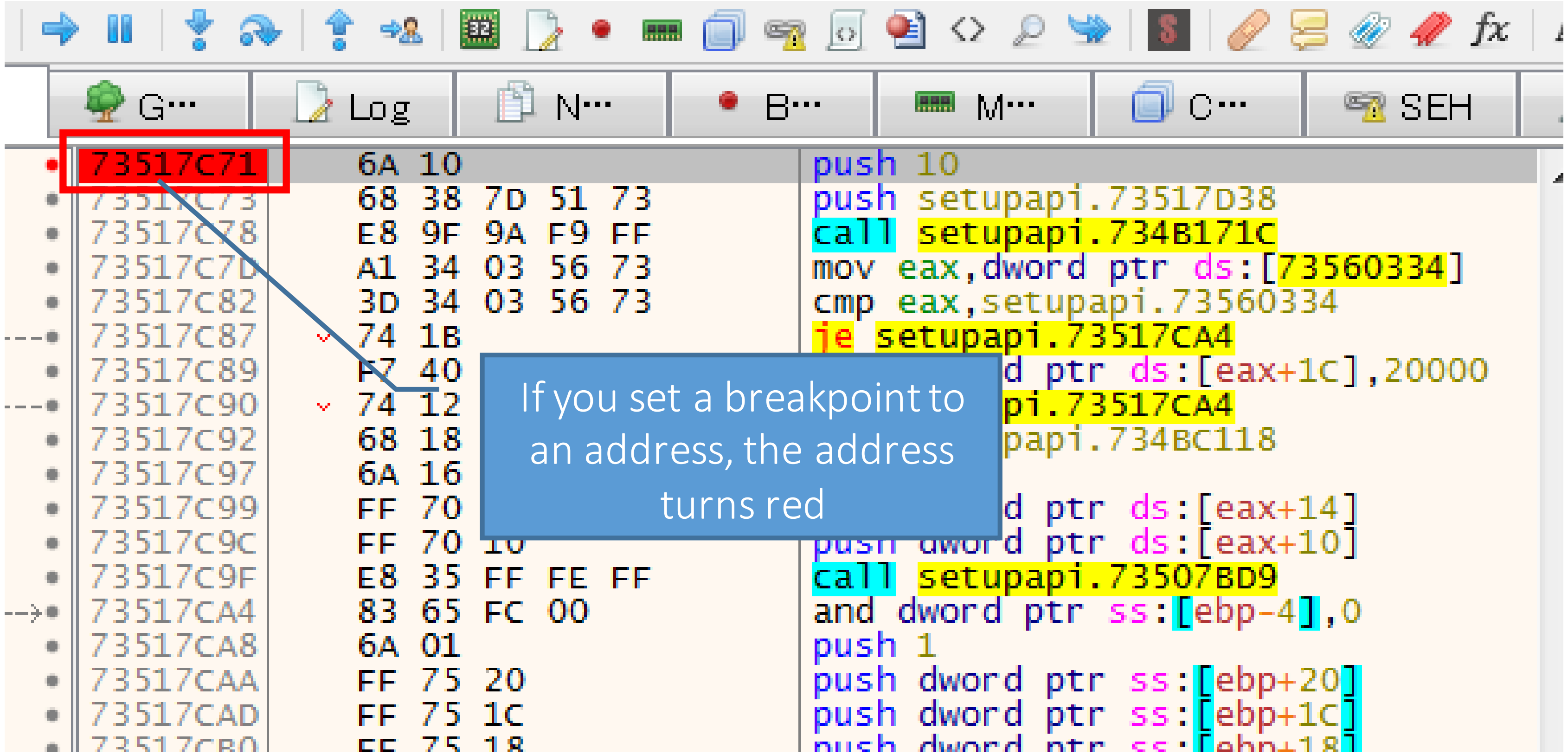


- Press “Ctrl + G” and type “SetupDiGetDeviceRegistryPropertyA” in the text box below. And then click “OK”.



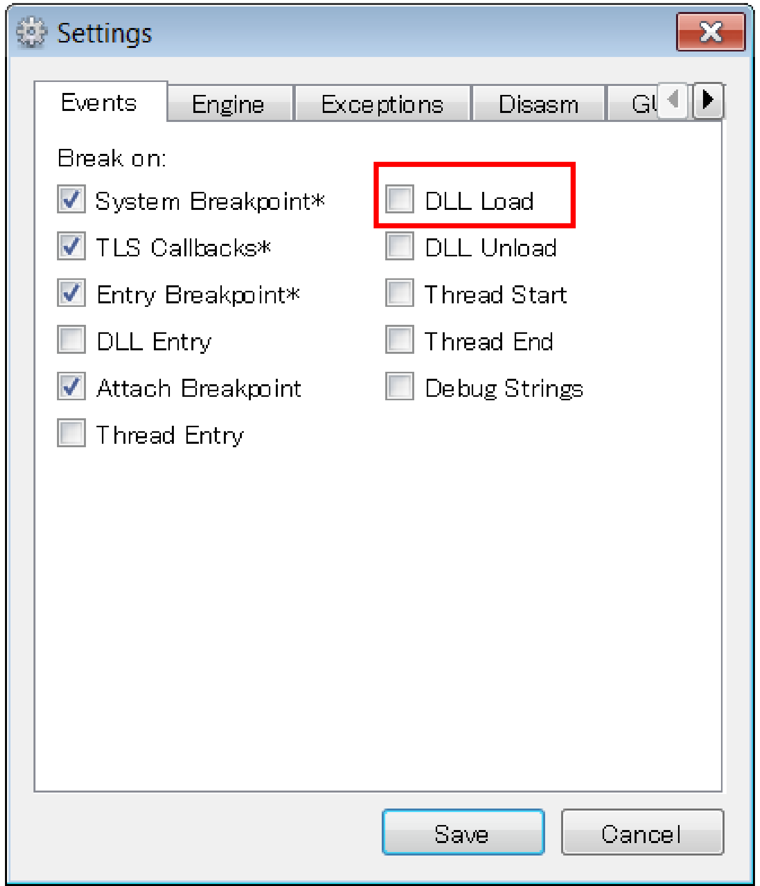
Exercise 2 (6)

- Press F2 to set a breakpoint to the head of API.



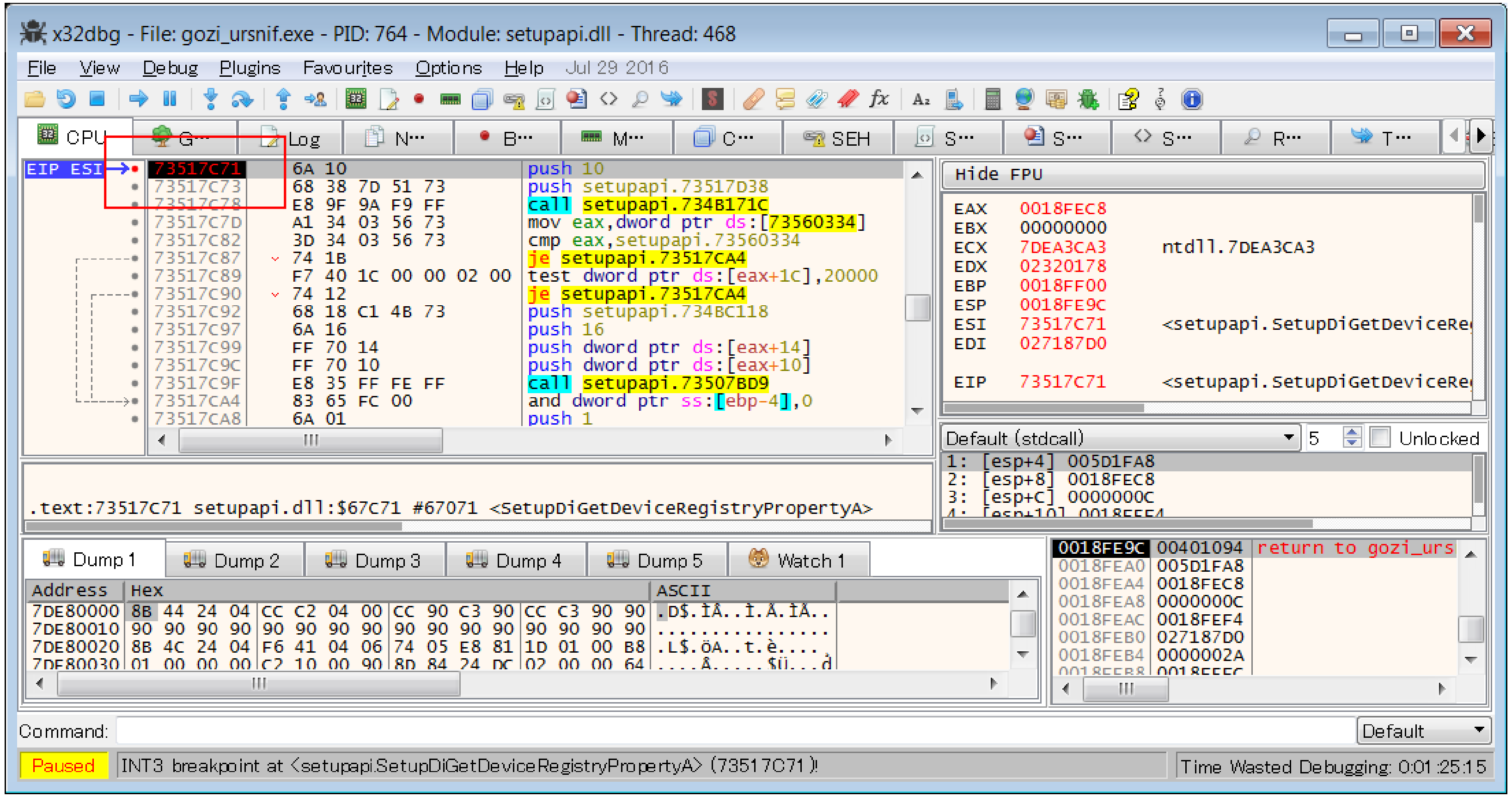
Exercise 2 (7)

- Options -> Preferences
  - Go to “Events” tab again and uncheck “DLL Load”.



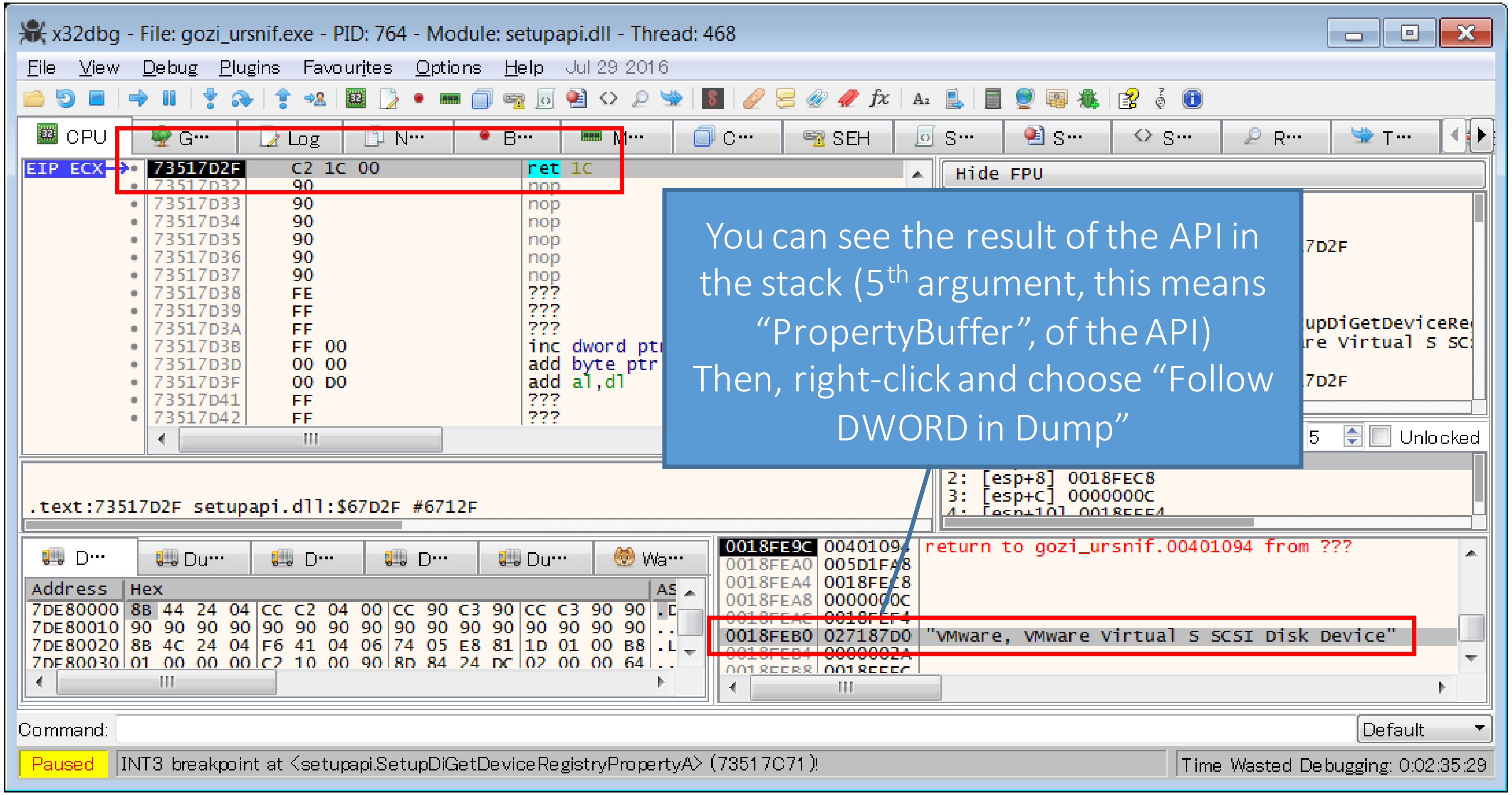
Exercise 2 (8)

- Then press F9 twice.
  - Since the first API call always fails, we need to take a look at the second call.



Exercise 2 (9)

- Execute up to ret instruction by pressing “Ctrl+F9”.



Exercise 2 (10)

- Let’s see the second call.
  - DeviceInfoData->ClassGuid (The second argument)
    - {4d36e967-e325-11ce-bfc1-08002be10318}
      - Hard Disk
  - Property (The third argument)
    - SPDRP_FRIENDLYNAME (0xC)

Disk Drives

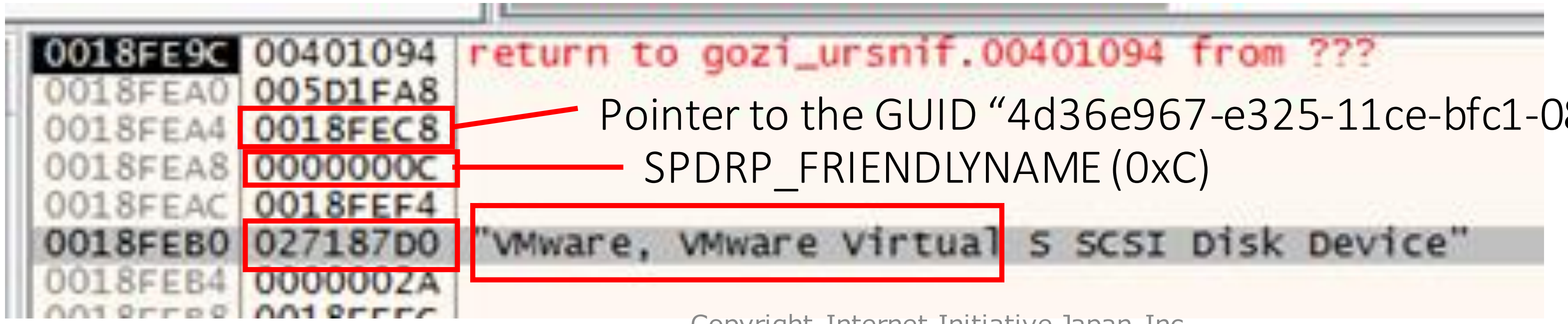
Class = DiskDrive

ClassGuid = {4d36e967-e325-11ce-bfc1-08002be10318}

This class includes hard disk drives. See also the HDC and SCSIAdapter classes.

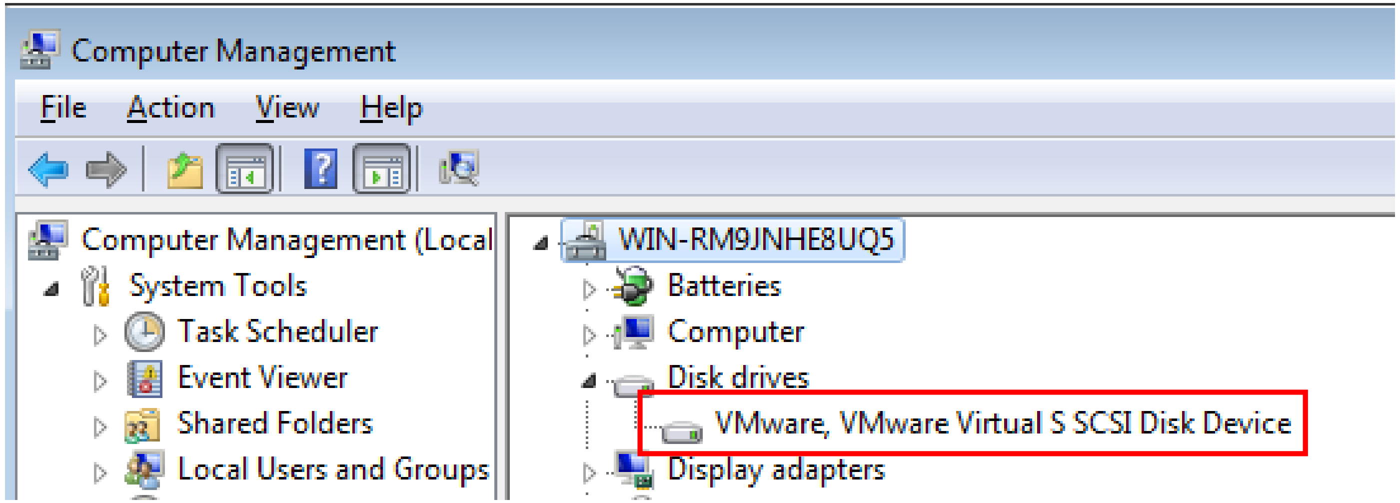
https://msdn.microsoft.com/en-us/library/windows/hardware/ff553426(v=vs.85).aspx

- PropertyBuffer (The fifth argument) (Post-Call)



Exercise 2 (11)

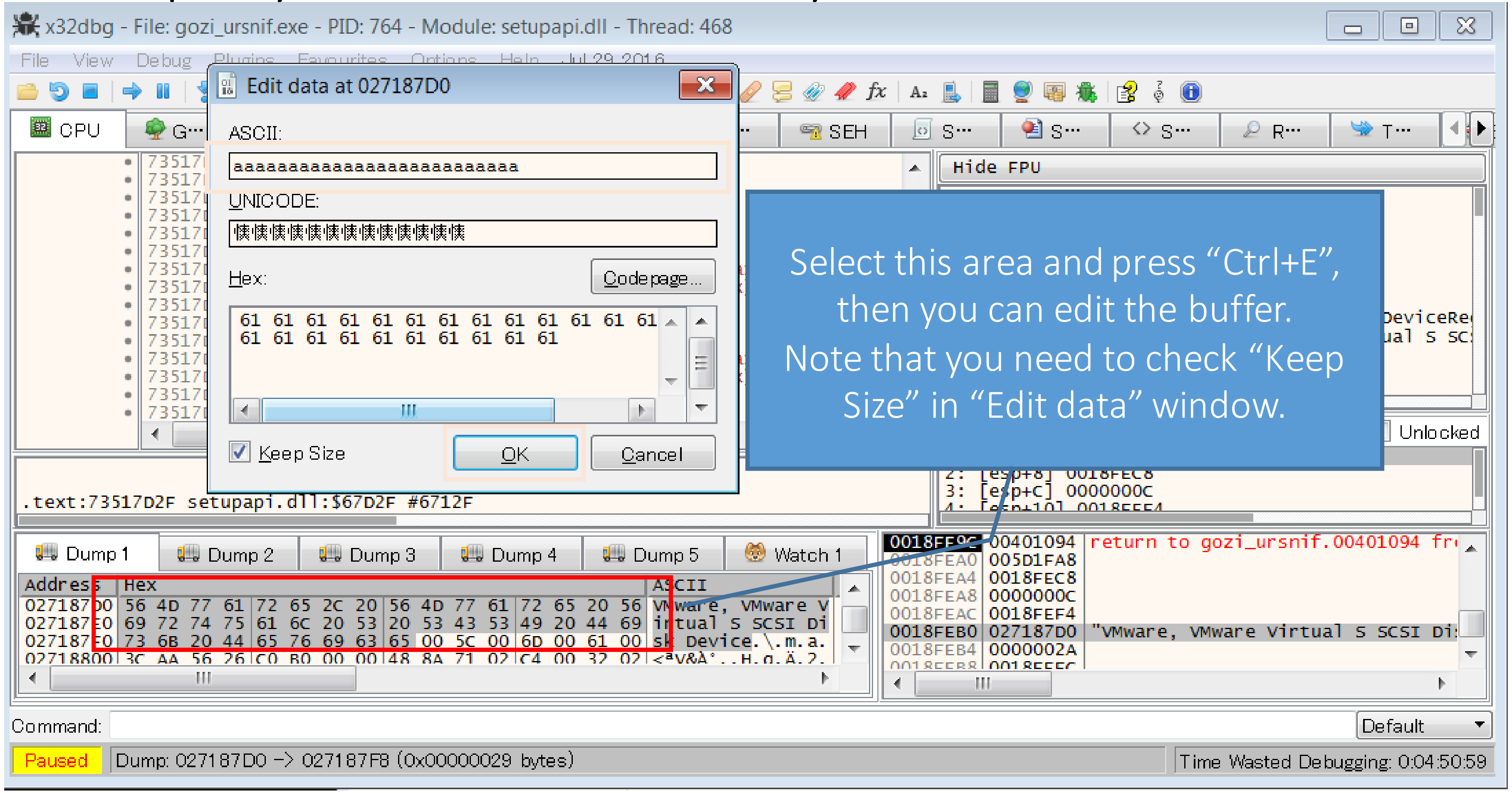
- This malware is likely to detect virtual HDD device in your VM environment.



- How can we deal with this problem?
  - We need to rewrite API responses.

Exercise 2 (12)

- Replace "PropertyBuffer" with arbitrary characters.



Exercise 2 (13)

- Let’s create a snapshot of your VM.
- And start Noriben and FakeNet.
- Press F9 to execute malware and then process is terminated.

- What happened?
- Let’s take a look at Noriben report and FakeNet log.
  - There is no suspicious communication in FakeNet log. But…

Exercise 2 (14)

- There are suspicious activities in Noriben report.
  - We found a batch file which gozi executed in process activities.

```
Processes Created:
==================
[CreateProcess] gozi_ursnif.exe:1892 > "cmd /c %LocalAppData%¥Temp¥4B61¥2.bat %AppData%¥COLOsnap¥d3diound.exe %User
[CreateProcess] csrss.exe:400 > "¥??¥%WinDir%¥system32¥conhost.exe"     [Child PID: 2972]
[CreateProcess] cmd.exe:2880 > "cmd  /C %AppData%¥COLOsnap¥d3diound.exe %UserProfile%¥Desktop¥malware¥GOZI_U~1.EXE"
[CreateProcess] cmd.exe:2000 > "%AppData%¥COLOsnap¥d3diound.exe  %UserProfile%¥Desktop¥malware¥GOZI_U~1.EXE"    [Ch
```

```
"cmd /c %LocalAppData%¥Temp¥4B61¥2.bat %AppData%¥COLOsnap¥d3diound.exe %UserProfile%¥Desktop¥malware¥GOZI_U~1.EXE"
```

  - We also found the file creation of the batch file and an executable file which gozi created.
    - Actually, this new executable has the same md5 hash as the original file, so this activity implies copy itself to another folder.

```
File Activity:
==================
[CreateFile] gozi_ursnif.exe:1892 > %AppData%¥COLOsnap¥d3diound.exe    [MD5: a780221be9d11249ea3845794714ba67]
[CreateFile] gozi_ursnif.exe:1892 > %AppData%¥COLOsnap¥d3diound.exe    [MD5: a780221be9d11249ea3845794714ba67]
[CreateFile] gozi_ursnif.exe:1892 > %LocalAppData%¥Temp¥4B61¥2.bat     [MD5: 31a6d044726d3d45eef3e23c0f9a703a]
```

```
$ md5sum gozi_ursnif.exe
a780221be9d11249ea3845794714ba67  gozi_ursnif.exe
$
```

Exercise 2 (15)

- You can find suspicious activities in Noriben report.
  - We can also find the file registration which registered by gozi in run key in registry activities.

```
Registry Activity:
==================
[RegCreateKey] svchost.exe:816 > HKLM¥System¥CurrentControlSet¥Control¥Network¥{4D36E972-E325-11CE-BFC1-08002BE10318}¥{3D8B7A06-B093-4612-B540-9FD777574A2
[RegSetValue] svchost.exe:816 > HKLM¥System¥CurrentControlSet¥Control¥Network¥{4D36E972-E325-11CE-BFC1-08002BE10318}¥{3D8B7A06-B093-4612-B540-9FD777574A20
[RegCreateKey] svchost.exe:816 > HKCU¥Software¥Microsoft¥RAS AutoDial
[RegCreateKey] svchost.exe:816 > HKCU¥Software¥Microsoft¥RAS AutoDial¥Default
[RegCreateKey] svchost.exe:816 > HKLM¥Software¥Microsoft¥RAS AutoDial
[RegCreateKey] svchost.exe:816 > HKLM¥SOFTWARE¥Microsoft¥RAS AutoDial¥Default
[RegSetValue] svchost.exe:816 > HKLM¥System¥CurrentControlSet¥Control¥Network¥{4D36E972-E325-11CE-BFC1-08002BE10318}¥{3D8B7A06-B093-4612-B540-9FD777574A20
[RegSetValue] svchost.exe:816 > HKLM¥System¥CurrentControlSet¥Control¥Network¥{4D36E972-E325-11CE-BFC1-08002BE10318}¥{3D8B7A06-B093-4612-B540-9FD777574A20
[RegSetValue] gozi_ursnif.exe:1892 > HKCU¥Software¥Microsoft¥Windows¥CurrentVersion¥Run¥apilkmon  =  C:¥Users¥taro¥AppData¥Roaming¥COLOsnap¥d3diound.exe
[RegDeleteValue] taskhost.exe:1772 > HKCU¥Software¥Microsoft¥Windows¥CurrentVersion¥Run¥...
```

```
HKCU¥Software¥Microsoft¥Windows¥CurrentVersion¥Run¥apilkmon
```

```
=  C:¥Users¥taro¥AppData¥Roaming¥COLOsnap¥d3diound.exe
```

- These activities are the installation task of the malware.
  - We can assume that this malware changes its behavior when the executable is located in a specific folder.

Exercise 2 (16)

- We still have some unclear points:
  - What is the content of the batch file?
  - What API does the malware use to execute the batch file?
  - Why this malware doesn’t communicate with C2 servers?

- Revert the VM, and let’s investigate those points.

Exercise 2 (17)

- What API does malware use to execute the batch file?
  - Typically, we use the following APIs to execute files.
    - CreateProcess
    - ShellExecute, ShellExecuteEx
    - WinExec
  - Set breakpoints at APIs below to find this activity.
    - CreateProcessA
    - ShellExecuteA
    - WinExec
    - To set break points: use “Ctrl+G” and type API name, and then press F2

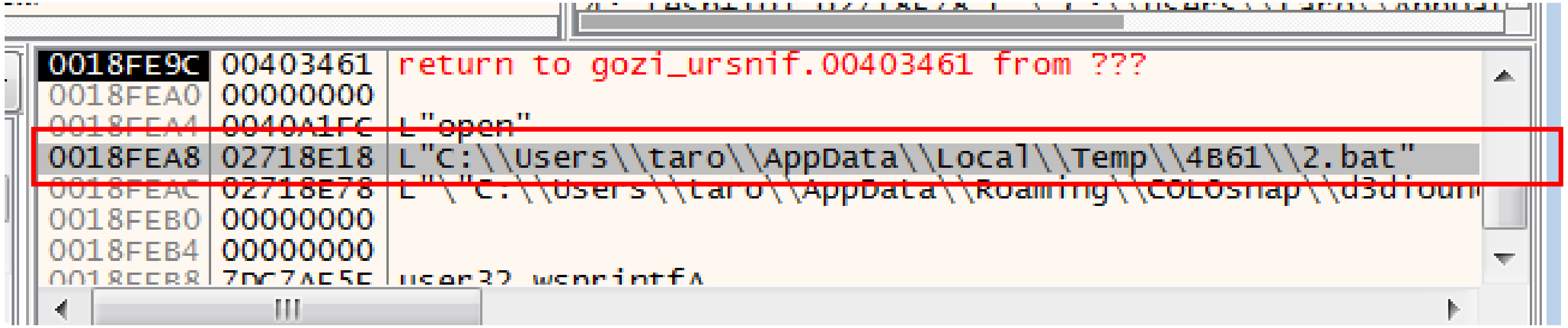
Exercise 2 (18)

- What API does malware use to execute the batch file? (Cont.)
  - If you have finished setting the breakpoints, hit F9.
  - If you hit a breakpoint, you can get the detail of this activities.
    - If you see the process termination at the left bottom of the x64dbg window, it's sign that it was failed.
    - Then, revert your VM, and try the following APIs.
  - Note that some malware use UNICODE version of API.
    - In this case, the last character of API name becomes "W" instead of "A".
    - E.g. CreateProcessW or ShellExecuteW or ShellExecuteExW
  - And some malware also might use low layer versions of the APIs.
    - E.g. ZwCreateUserProcess or ZwCreateProcess is used instead of CreateProcess*.

Exercise 2 (19)

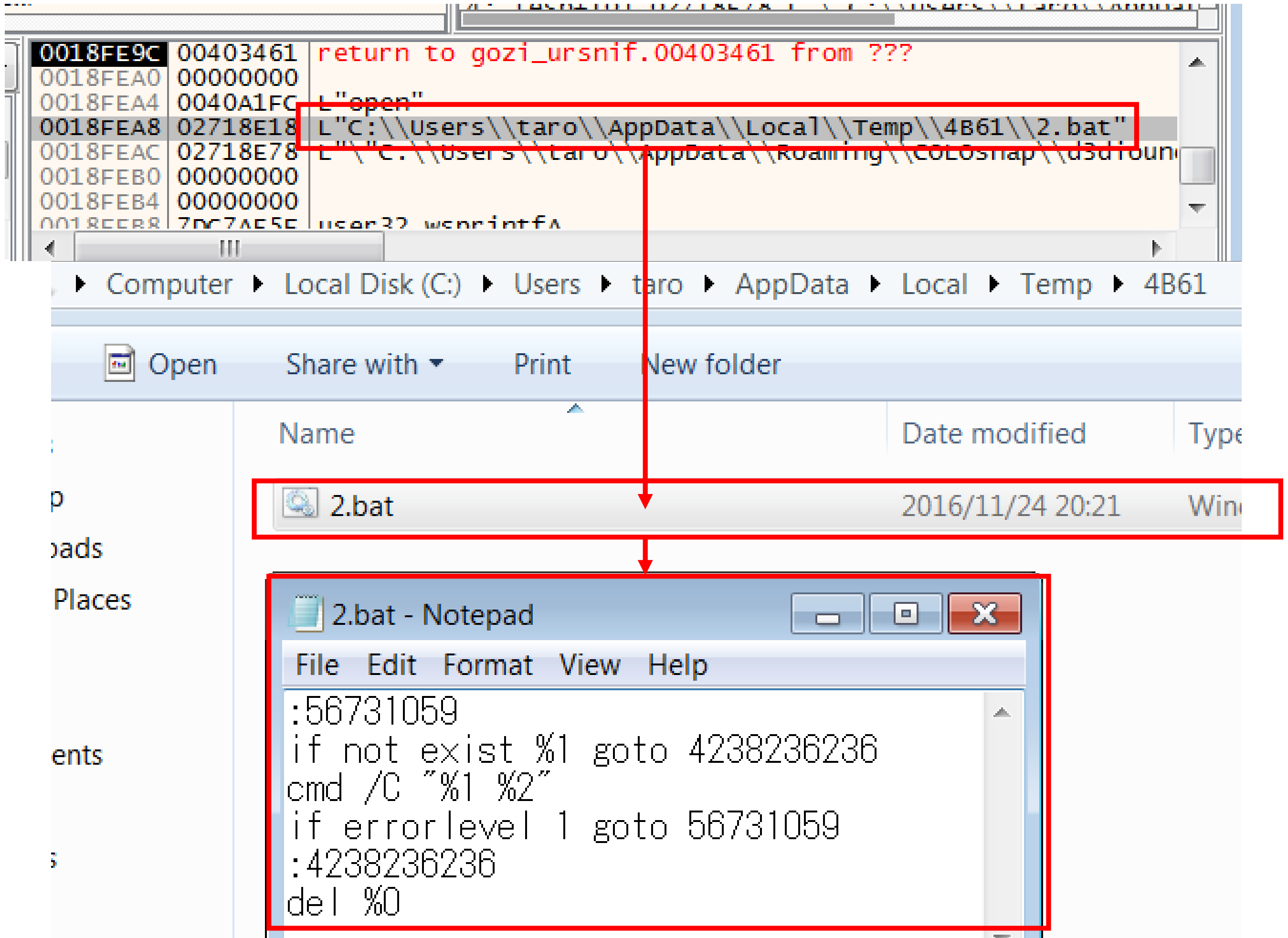
- Actually, we can break at ShellExecuteW!

- Now we can find the batch location.



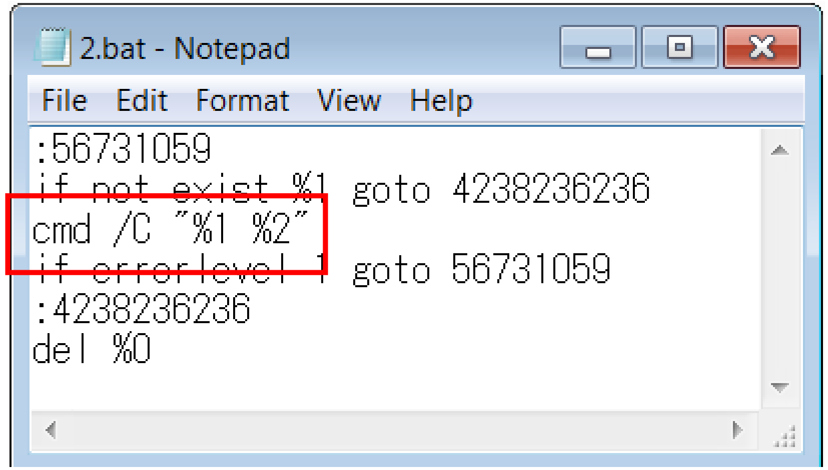
Exercise 2 (20)

- Then we can get the contents of the batch file.



Exercise 2 (21)

- The batch file simply executes the first argument, with the second argument as an argument to the executables specified as the first argument, on command prompt.



- And you already know the first and the second arguments (from Noriben log).

```
"cmd /c %LocalAppData%¥Temp¥4B61¥2.bat %AppData%¥COLOsnap¥d3diound.exe %UserProfile%¥Desktop¥malware¥GOZI_U~1.EXE"
```

Exercise 2 (22)

- We now have the contents of the batch file.
- And we also have “Run” key of the registry from Noriben report.

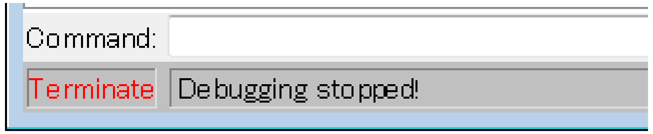
```
HKCU¥Software¥Microsoft¥Windows¥CurrentVersion¥Run¥apilkmon
```

```
 =  C:¥Users¥taro¥AppData¥Roaming¥COLOsnap¥d3diound.exe
```

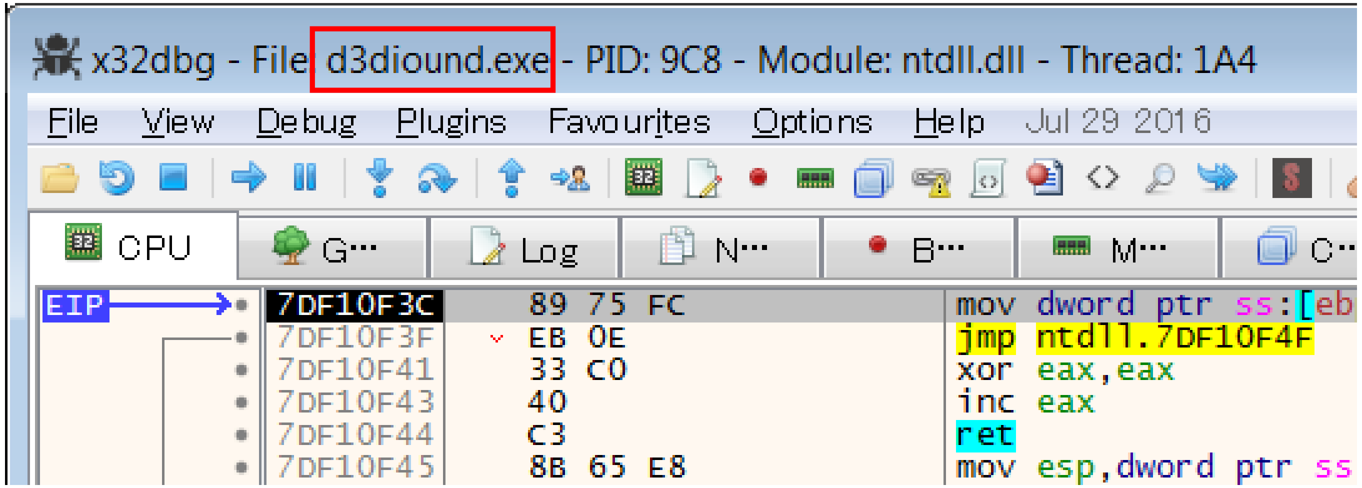
- Then we have two strategies here.
  - Execute copied gozi with original one as the argument in a debugger.
    - The batch file uses this method.
  - Execute copied gozi simply in a debugger.
    - If the installation task is finished, this method is used because of “Run” key.
- Let’s take 2nd method!

Exercise 2 (23)

- Hit F9 until the debugging process is terminated.

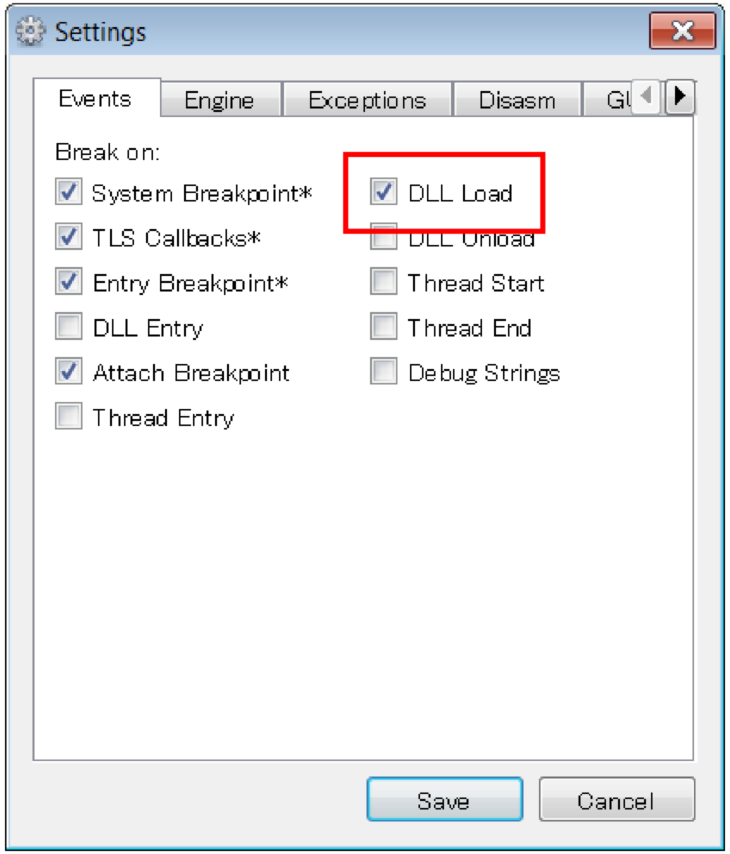


- Then load copied gozi into x32dbg.



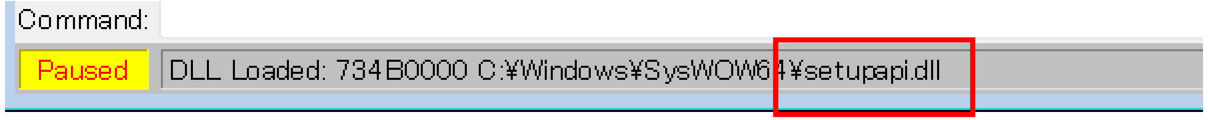
Exercise 2 (24)

- Options -> Preferences
  - Go to “Events” tab, and check “DLL Load”.

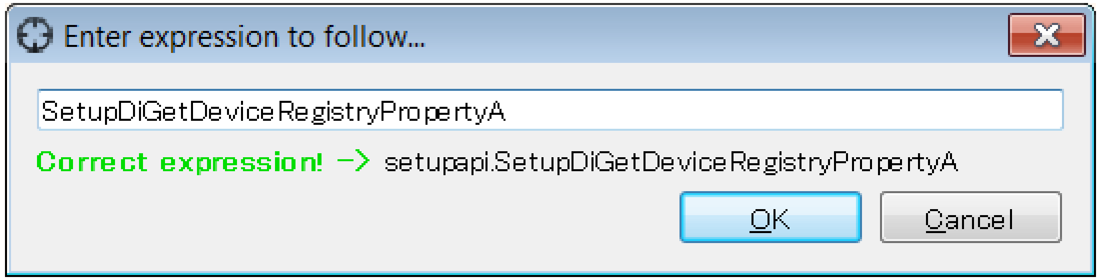


Exercise 2 (25)

- Press “F9” several times until you see “setup.dll” at the left bottom of the x32dbg window.

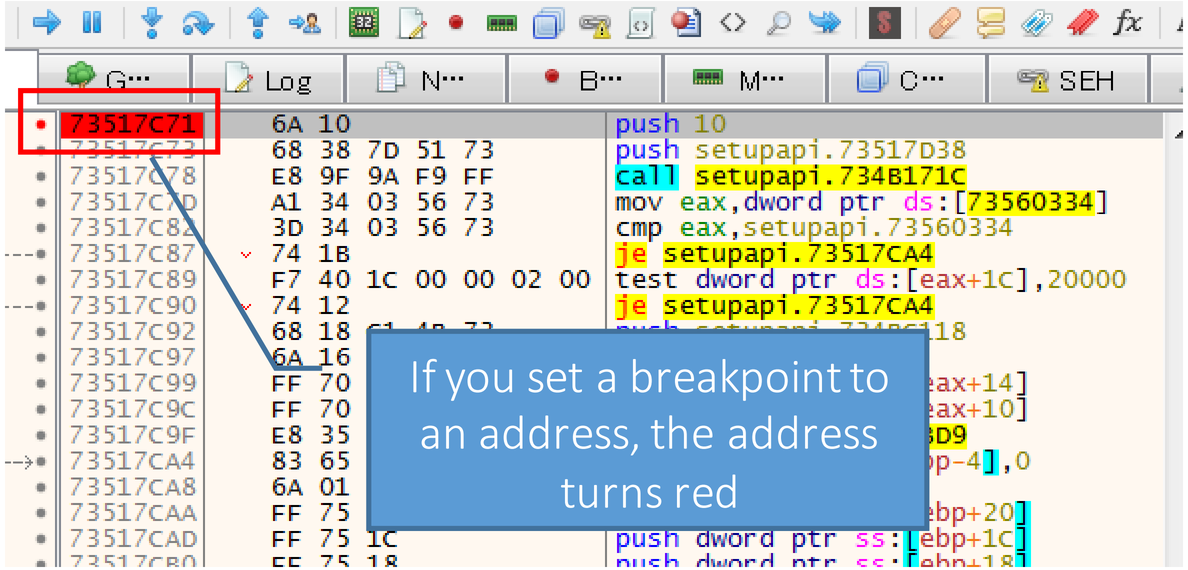


- Press “Ctrl + G” and type “SetupDiGetDeviceRegistryPropertyA” in the text box below. And then click “OK”.



Exercise 2 (26)

- Press F2 to set a breakpoint at the head of API.



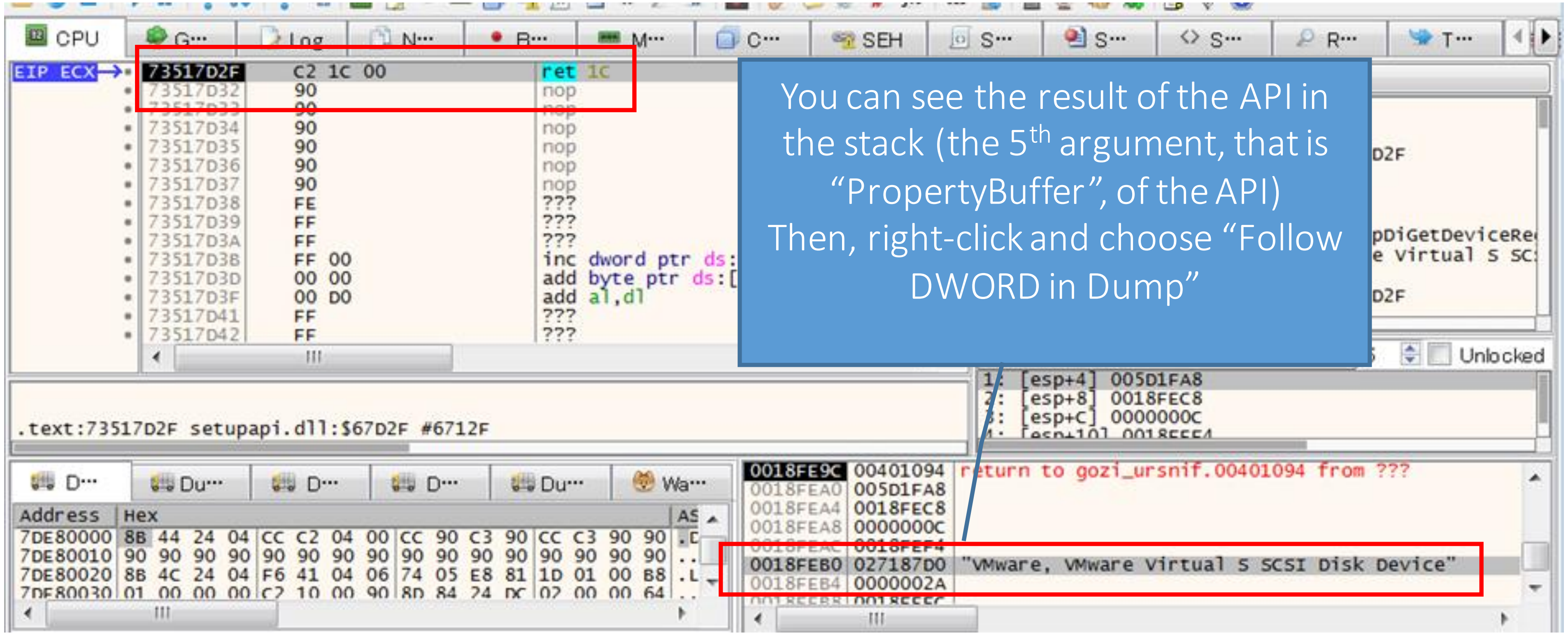
Exercise 2 (27)

- Options -> Preferences
  - Go to “Events” tab again and uncheck “DLL Load”.

- Then press F9 twice.
  - The first API call always fails.

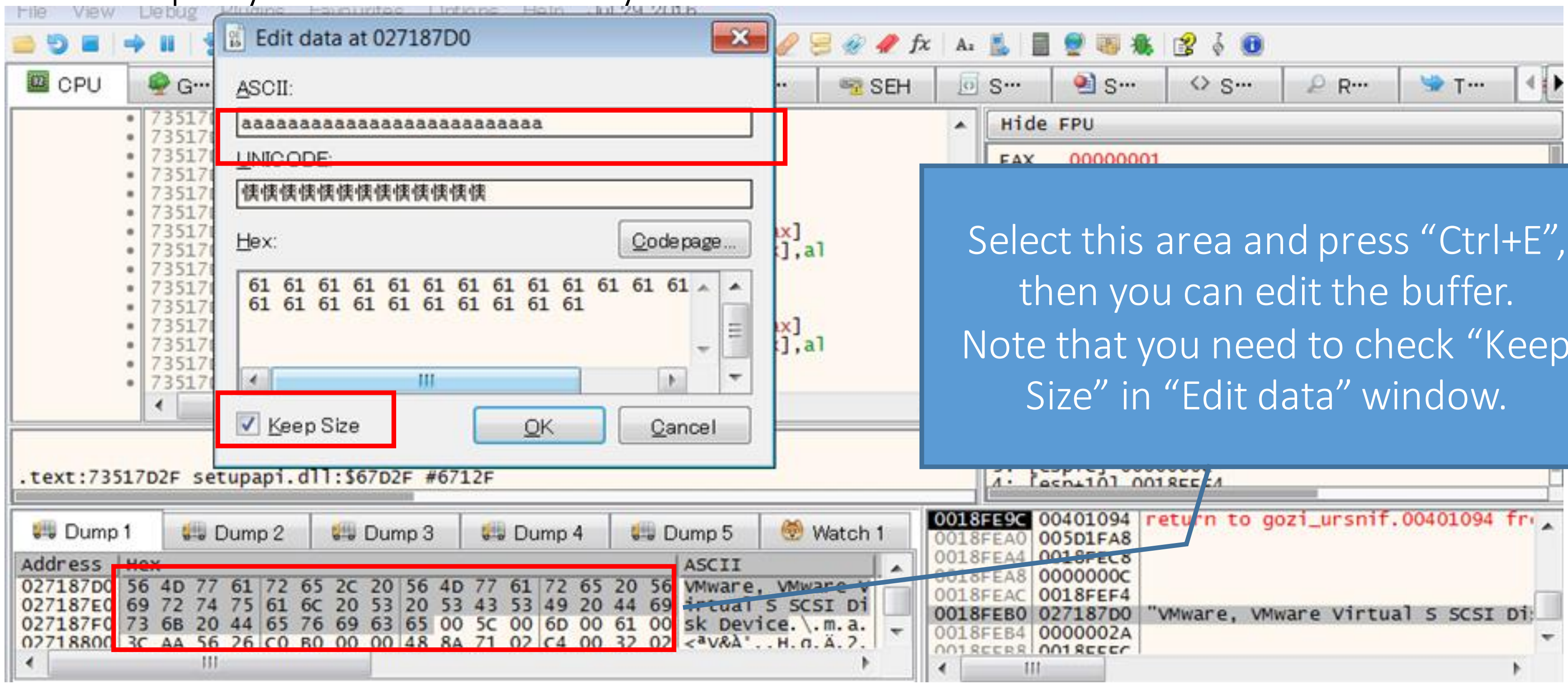
Exercise 2 (28)

- Execute until the “ret” instruction by pressing “Ctrl+F9”.



Exercise 2 (29)

- Replace “PropertyBuffer” with arbitrary characters.



Select this area and press “Ctrl+E”, then you can edit the buffer. Note that you need to check “Keep Size” in “Edit data” window.

- And then, Hit F9 until the process is terminated, and after taking for a while, you will see suspicious communications.

Exercise 3

Dealing with the Process Hollowing / PE Reflective Injection Technique with Debuggers

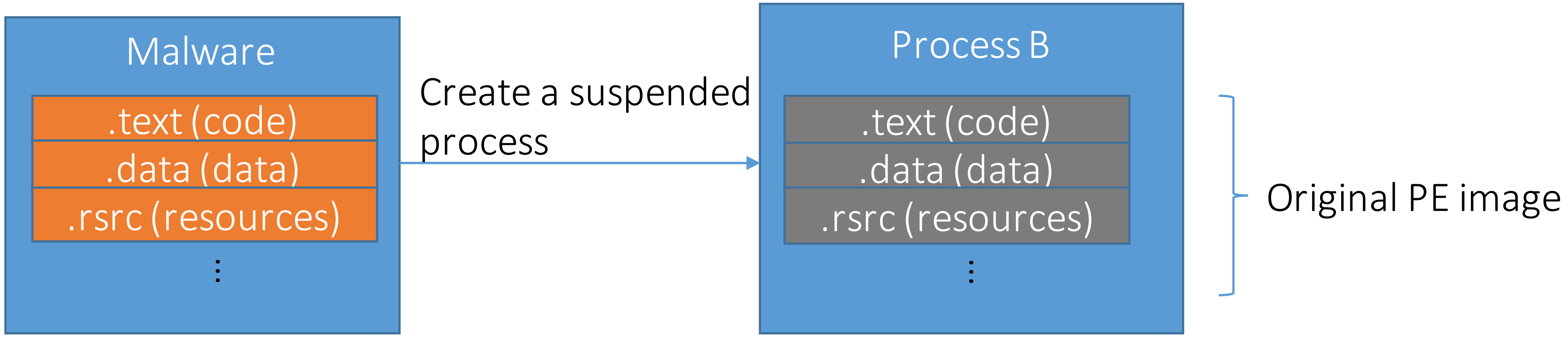
What is Process Hollowing / PE Reflective Injection?

- Process Hollowing / PE Reflective Injection are kinds of remote code injection technique.
  - A.k.a process replacement or Nebbett's Shuttle.

  - If these techniques are used, almost all API monitoring tools including APIMonitor can't monitor the APIs that are used in these techniques. Those tools cannot set hooks when a target process is created because the process is created with the suspended option.
    - Even debuggers cannot attach the suspended process at the moment.
    - You need to use debuggers with a certain technique!

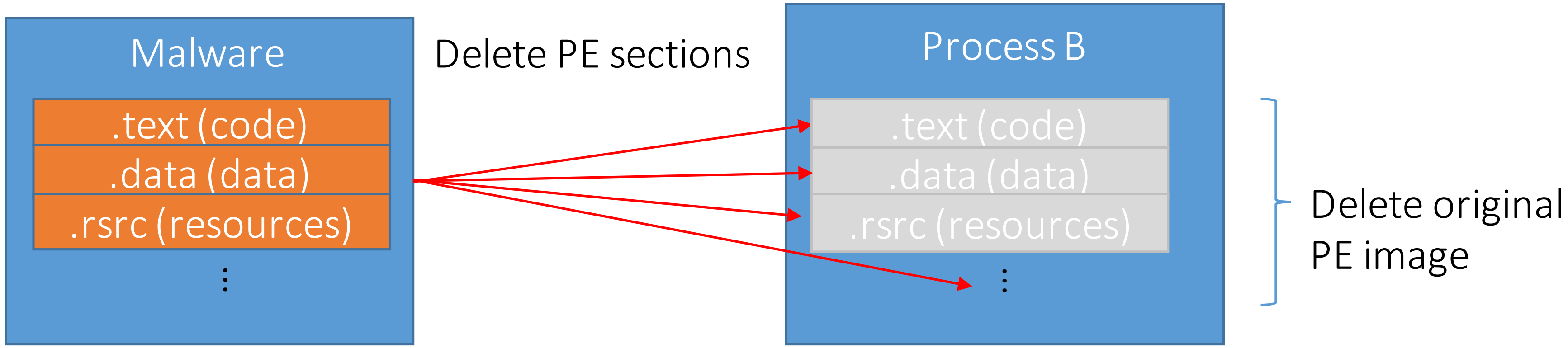
What is Process Hollowing / PE Reflective Injection?

- How does the Process Hollowing / PE Reflective Injection work?
- First, malware creates Process B (e.g. svchost.exe) using CreateProcess API with CREATE_SUSPENDED flag



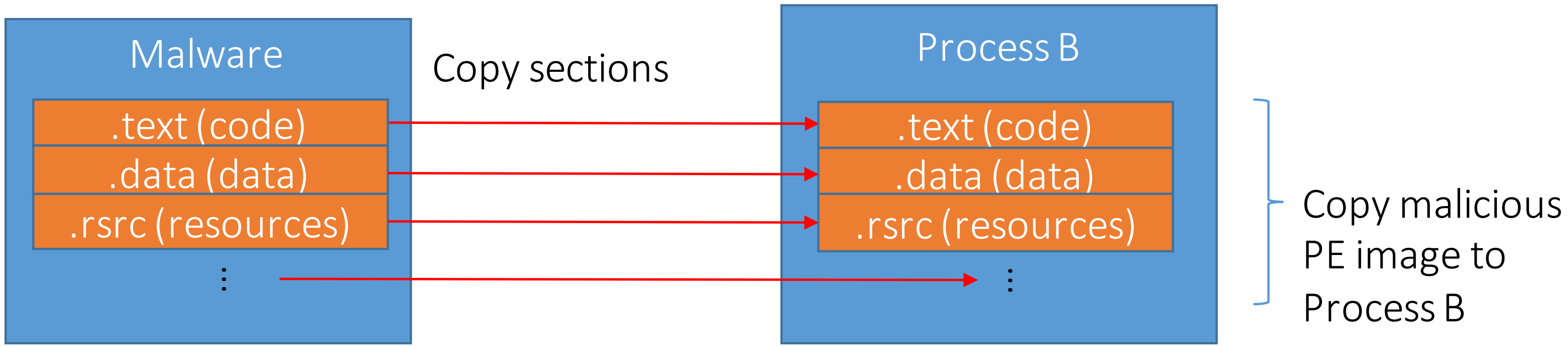
What is Process Hollowing / PE Reflective Injection?

- Second, malware removes original PE image from the memory of Process B using ZwUnmapViewOfSection API.
  - If PE reflective injection technique is used, then this step is skipped.



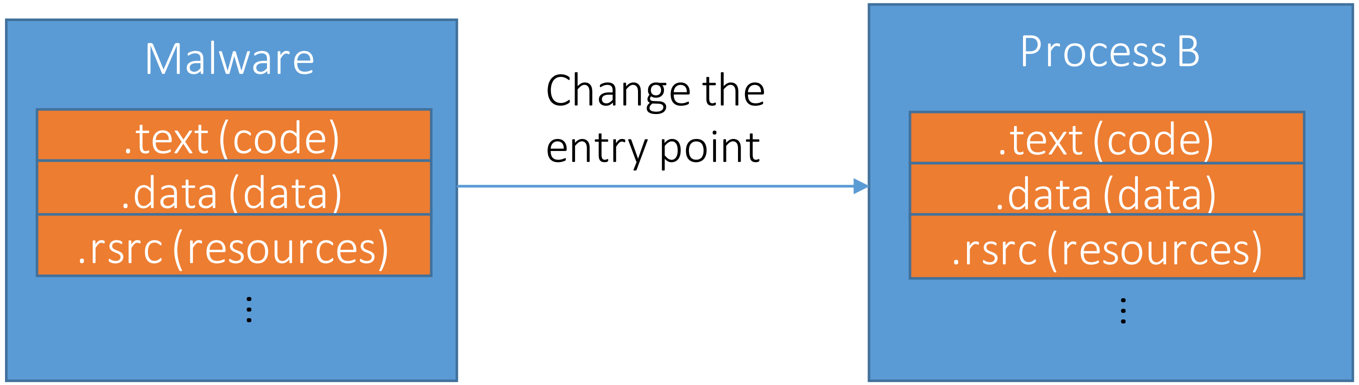
What is Process Hollowing / PE Reflective Injection?

- Next, it copies malicious code and data in malware to Process B using ZwMapViewOfSection API or VirtualAllocEx and WriteProcessMemory API.



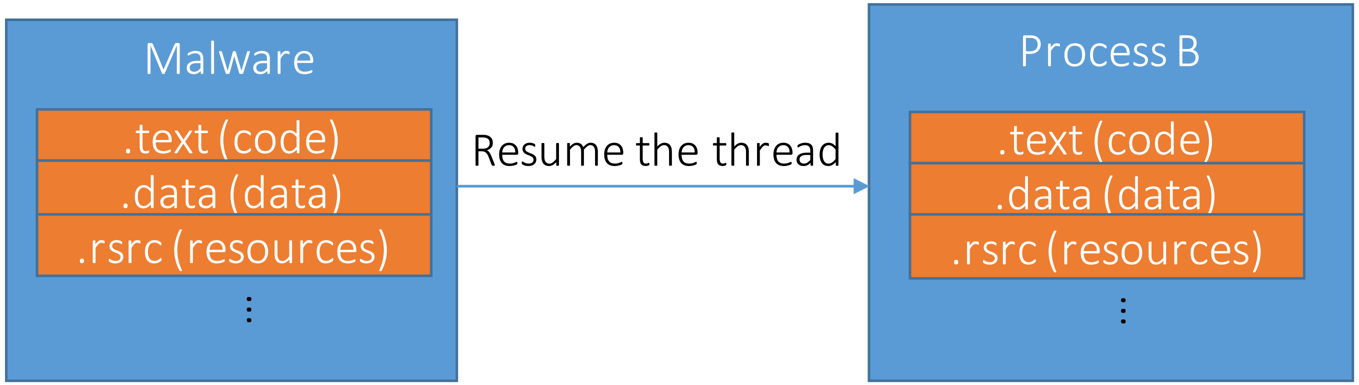
What is Process Hollowing / PE Reflective Injection?

- Then it replaces the current entry point in Process B with malware's one using GetThreadContext and SetThreadContext API.
  - If ZwMapViewOfSection API is used, the malware might replace the legitimate code at the entry point with the malicious code directly without SetThreadContext API.



What is Process Hollowing / PE Reflective Injection?

- Finary, it execute malicious code in process B using ResumeThread API.
  - Note that the malicious code is executed with the access rights of the “Process B”. If the “Process B” is an Internet Explorer, the process can access the Internet because typical personal firewall allows IE to access the Internet communication.



Exercise 3 (1)

- At this time, we will see another gozi sample that uses the reflective PE injection.
- Load “gozi_ursnif_201610.exe”.
  - This another gozi sample has multiple anti-analysis techniques.
- First, we need to deal with “file handle” issue.
  - This malware opens itself using CreateFile API, but this activity is failed on some debuggers because those debuggers don’t close a file handle of debuggee when they finish to load.

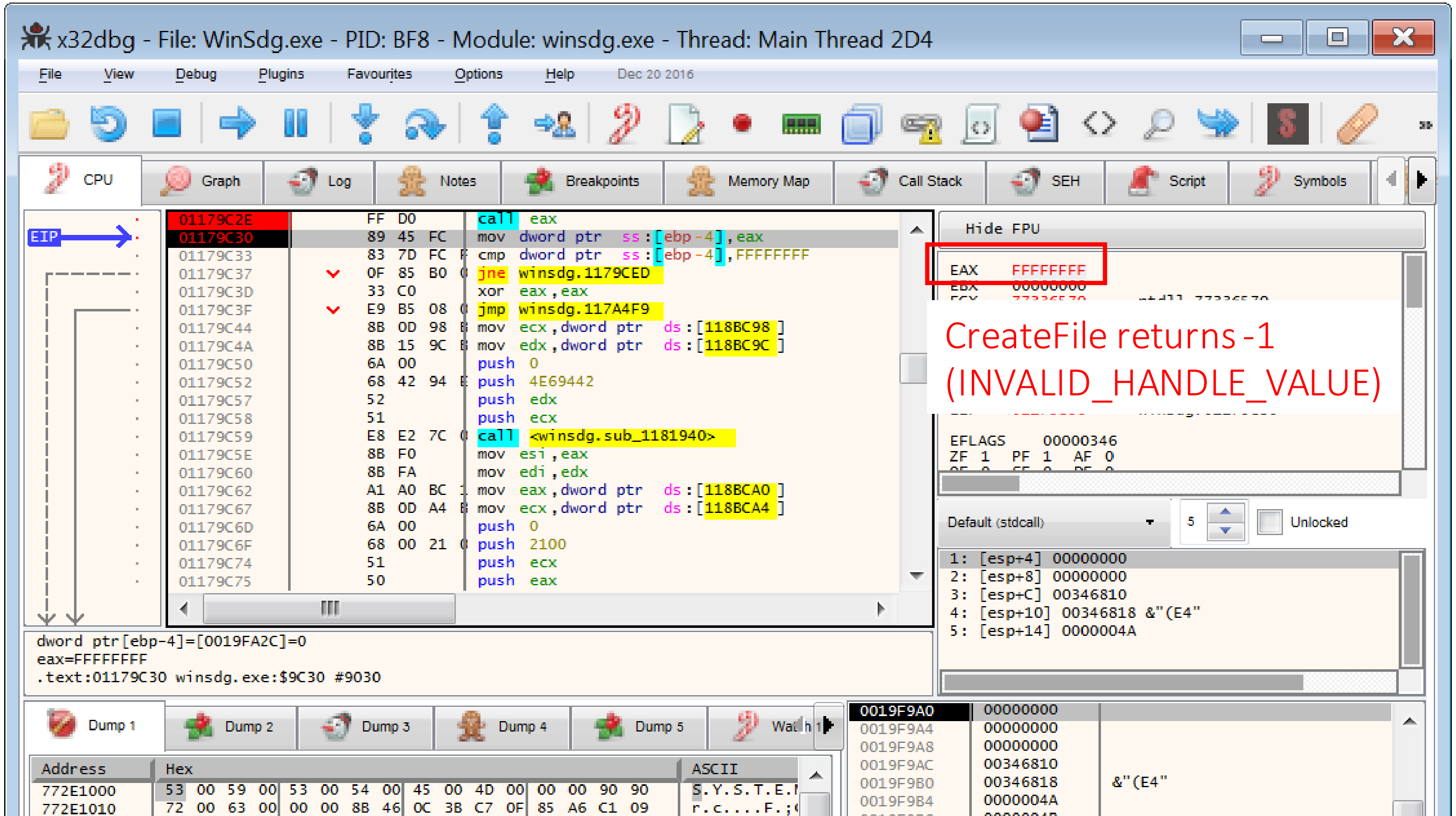
Exercise 3 (2)

- Some debuggers don’t close debuggee’s file handle.

Debugger	Close
OllyDbg 1.10 / 2.01	OK
Immunity Debugger 1.85	OK
x64dbg / x32dbg (Jan 27 2017)	NG
WinDbg 6.2 / 6.3	NG
IDA Pro 6.95 (Local Win32 Debugger)	NG

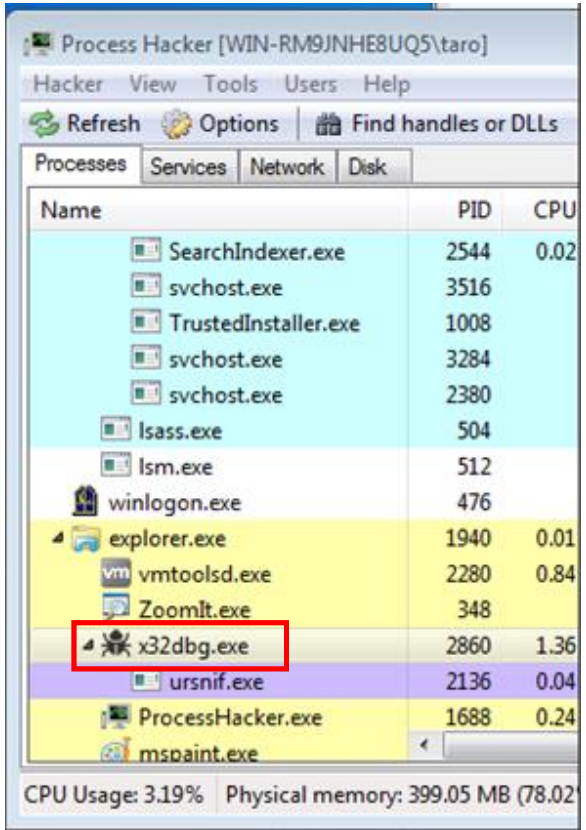
Exercise 3 (3)

- E.g. This sample fails to open itself on x64dbg.



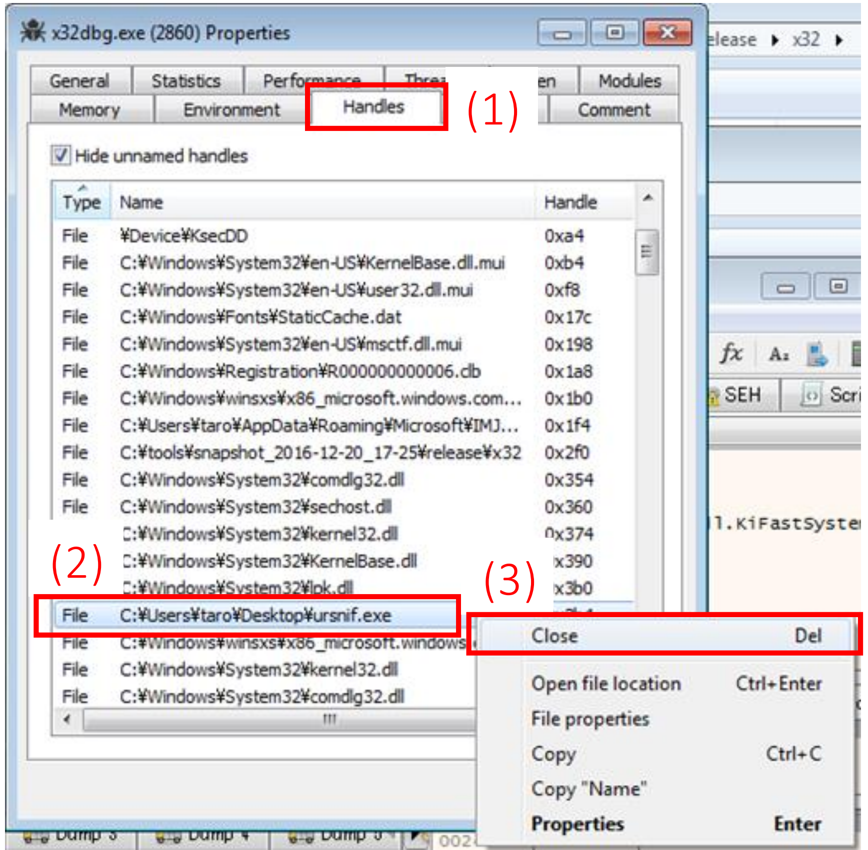
Exercise 3 (4)

- How to close debuggee’s file handle forcibly.
  - First, start “Process Hacker” and double click on your debugger process.



Exercise 3 (5)

- How to close debuggee’s file handle forcibly (cont).
  - Click “Handles” tab, and find “File” type and debuggee’s file path, then right click and choose “Close”.



Exercise 3 (6)

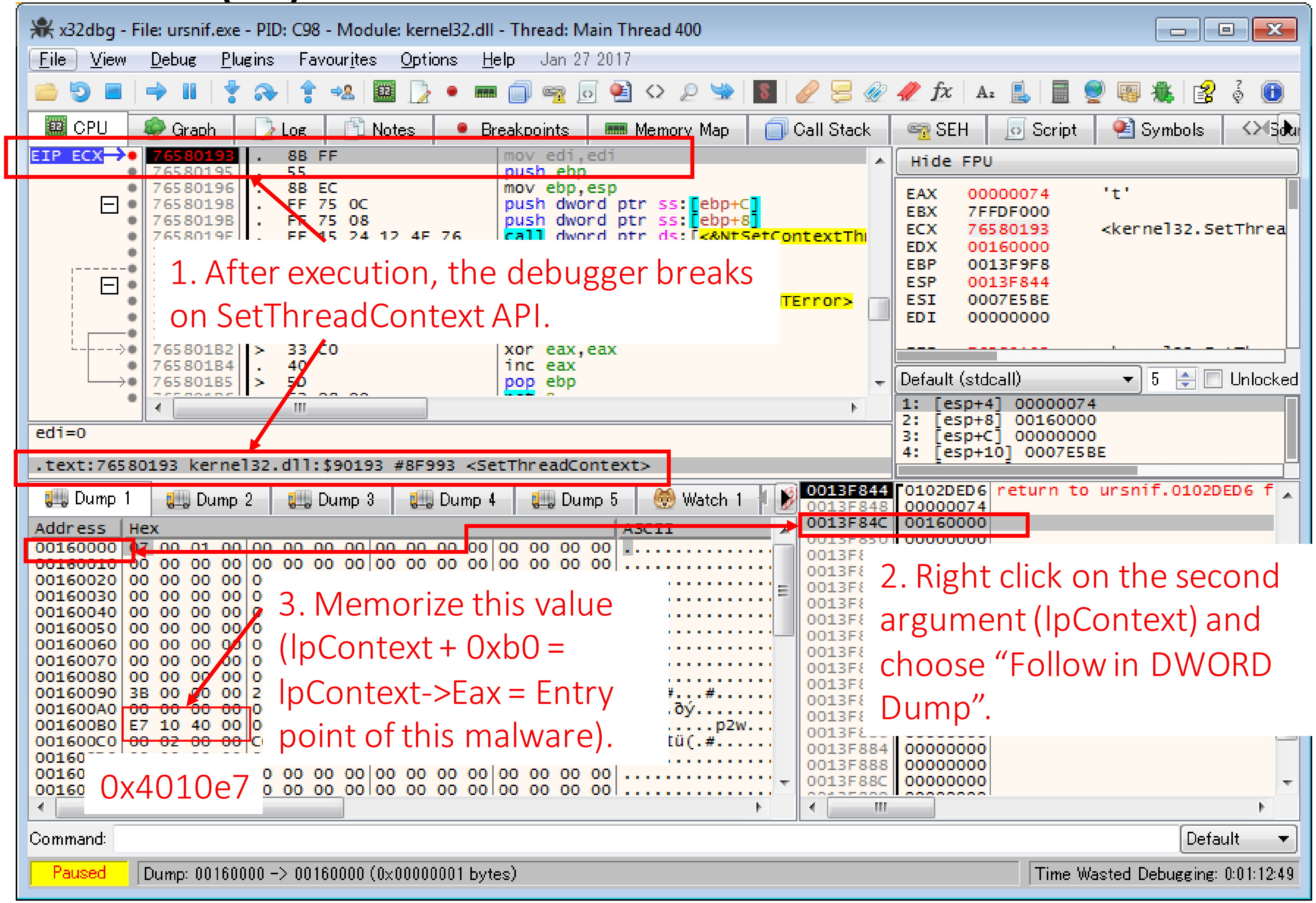
- Next, we need to deal with Reflective PE Injection.
- Set breakpoint at "SetThreadContext" API and press "F9" to execute malware.
  - For Win7 64 bit users (only for Win7 64 bit users), you need to set the breakpoint on "Wow64SetThreadContext" or "ZwSetContextThread" instead.

```
BOOL WINAPI SetThreadContext(
  _In_       HANDLE  hThread,
  _In_ const CONTEXT *lpContext
);
```

https://msdn.microsoft.com/ja-jp/library/windows/desktop/ms680632(v=vs.85).aspx

```
> dt _CONTEXT
ntdll!_CONTEXT
   +0x000 ContextFlags     : Uint4B
   +0x004 Dr0              : Uint4B
   +0x008 Dr1              : Uint4B
   +0x00c Dr2              : Uint4B
   +0x010 Dr3              : Uint4B
   +0x014 Dr6              : Uint4B
   +0x018 Dr7              : Uint4B
   +0x01c FloatSave        : _FLOATING_SAVE_AREA
   +0x08c SegGs            : Uint4B
   +0x090 SegFs            : Uint4B
   +0x094 SegEs            : Uint4B
   +0x098 SegDs            : Uint4B
   +0x09c Edi              : Uint4B
   +0x0a0 Esi              : Uint4B
   +0x0a4 Ebx              : Uint4B
   +0x0a8 Edx              : Uint4B
   +0x0ac Ecx              : Uint4B
   +0x0b0 Eax              : Uint4B
   +0x0b4 Ebp              : Uint4B
   +0x0b8 Eip              : Uint4B
   +0x0bc SegCs            : Uint4B
   +0x0c0 EFlags           : Uint4B
```

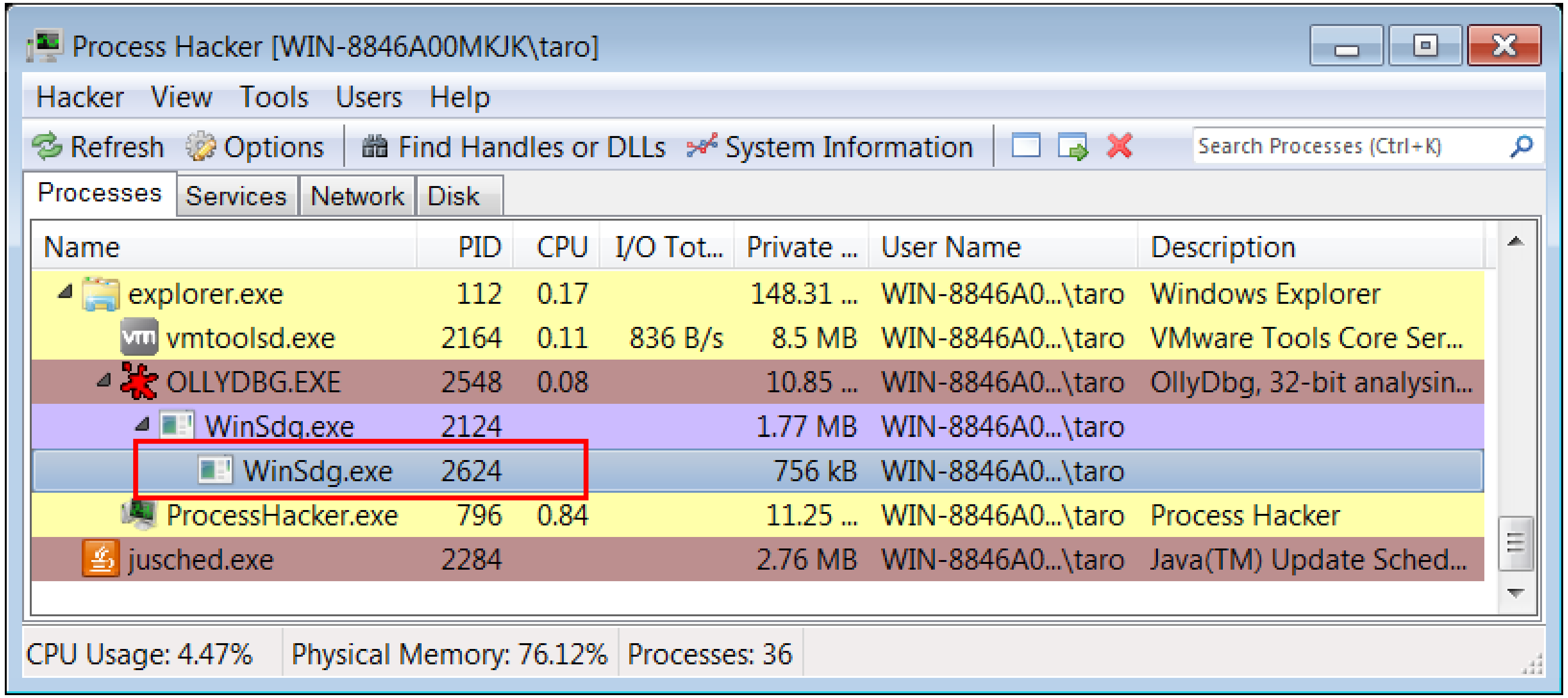
Exercise 3 (7)



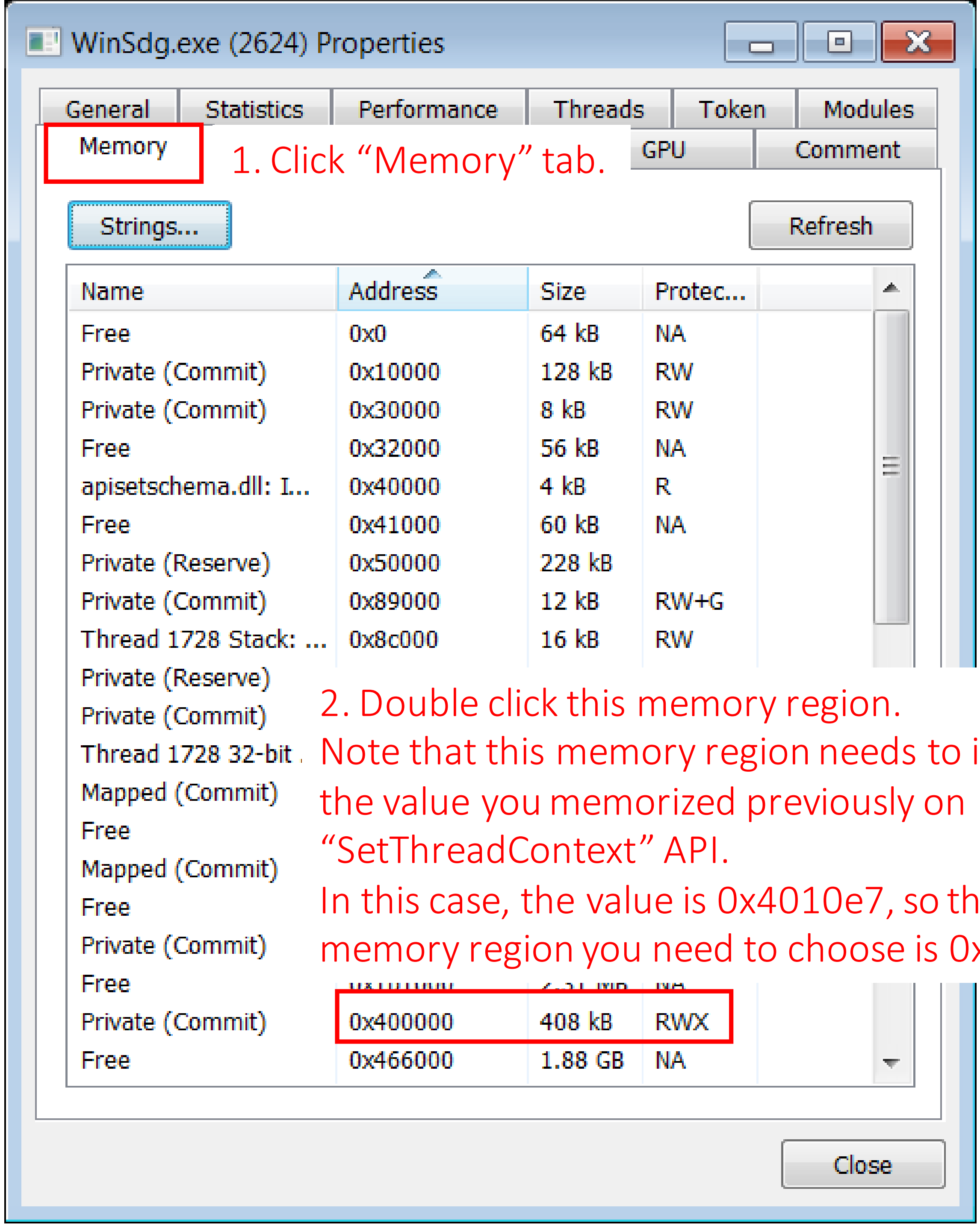
x32dbg - File: ursnif.exe - PID: C98 - Module: kernel32.dll - Thread: Main Thread 400
1. After execution, the debugger breaks on SetThreadContext API.
.text:76580193 kernel32.dll:$90193 #8F993 <SetThreadContext>
2. Right click on the second argument (lpContext) and choose "Follow in DWORD Dump".
3. Memorize this value (lpContext + 0xb0 = lpContext->Eax = Entry point of this malware).
0x4010e7

Exercise 3 (8)

- Execute “Process Hacker” and right click on the child process of malware and choose “Properties”.



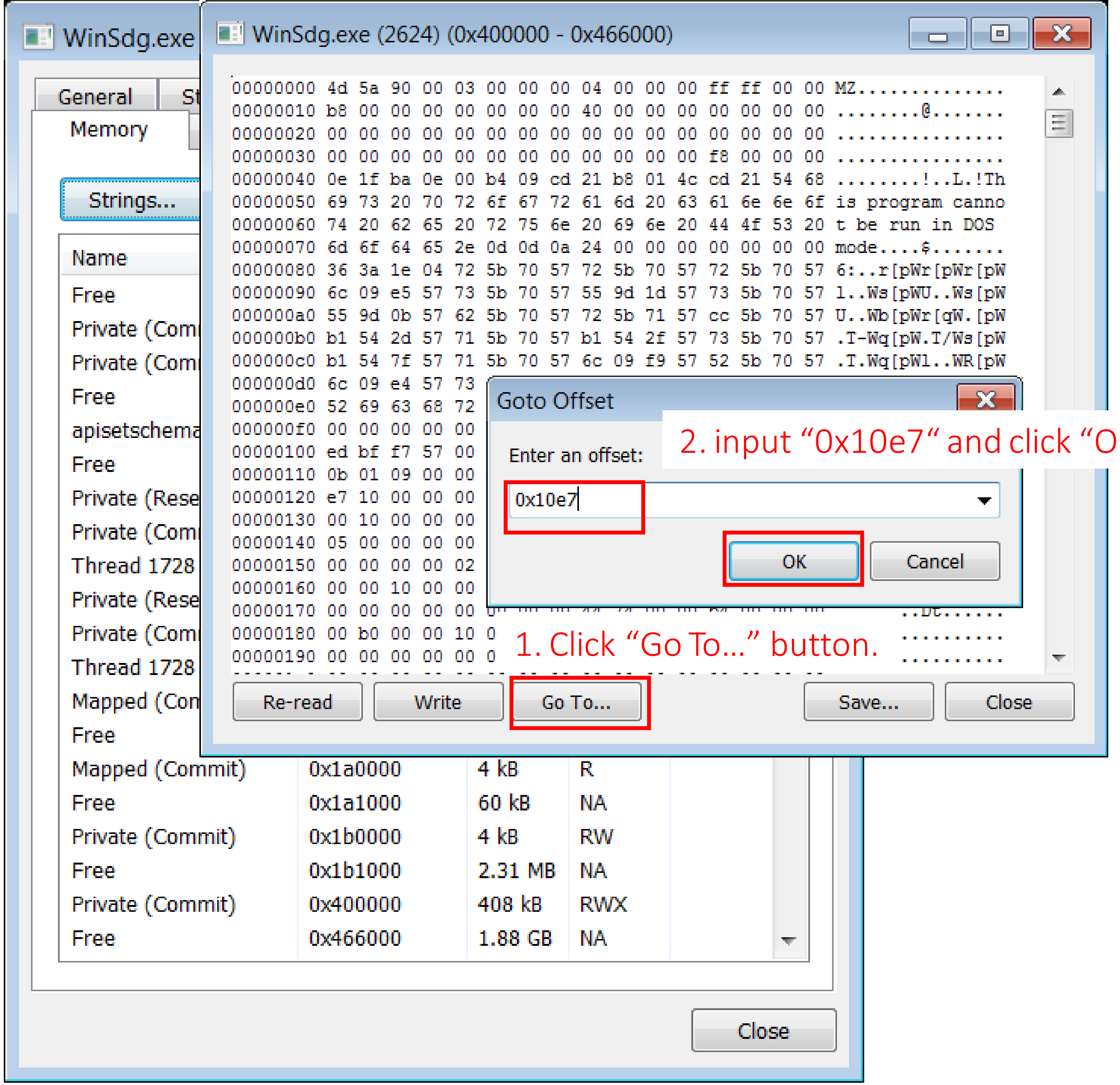
Exerci



1. Click "Memory" tab.

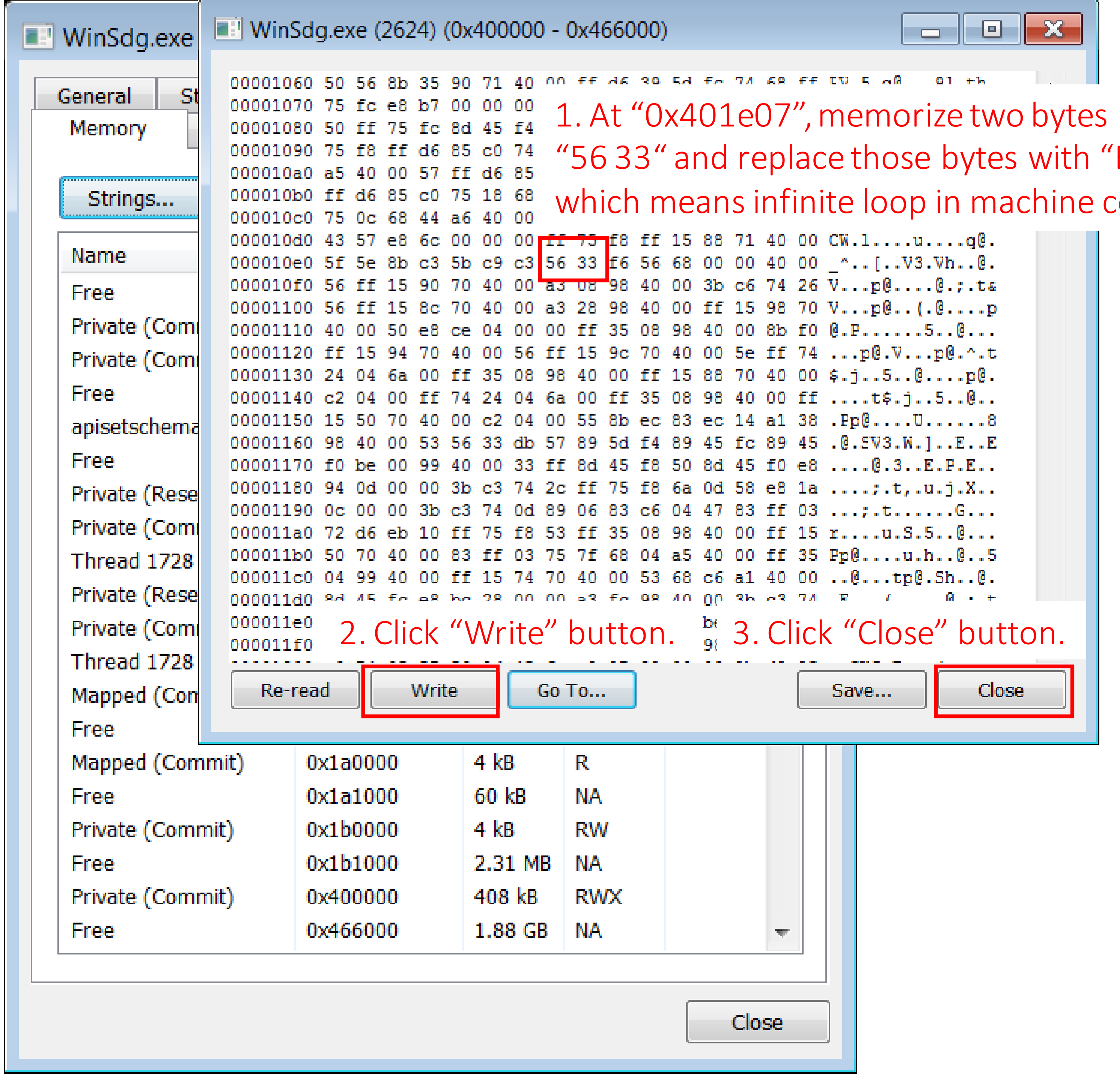
2. Double click this memory region.
Note that this memory region needs to include the value you memorized previously on "SetThreadContext" API.
In this case, the value is 0x4010e7, so the memory region you need to choose is 0x400000.

Exerci



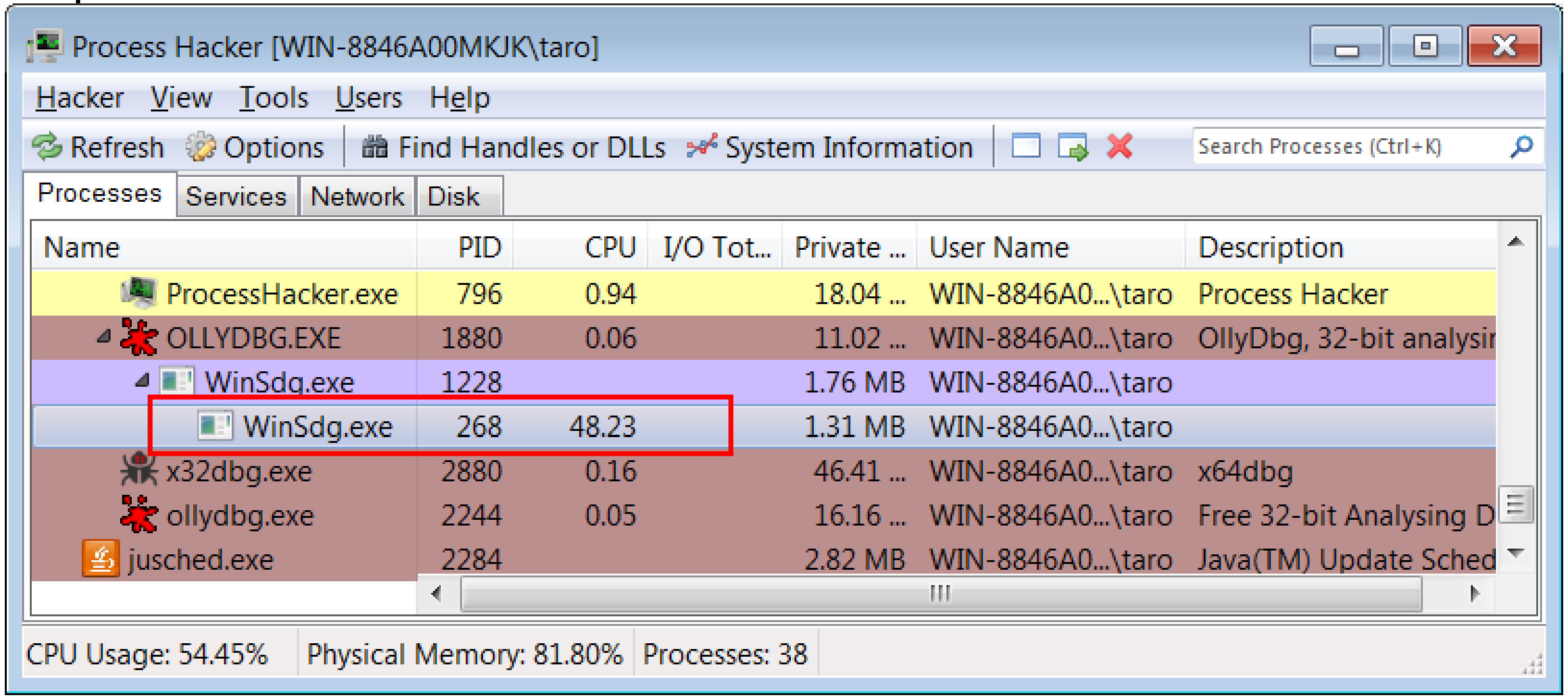
wizSafe

Exerci


WinSdg.exe (2624) (0x400000 - 0x466000)
1. At "0x401e07", memorize two bytes "56 33" and replace those bytes with "EB FE" which means infinite loop in machine code!
2. Click "Write" button.
3. Click "Close" button.
Re-read
Write
Go To...
Save...
Close

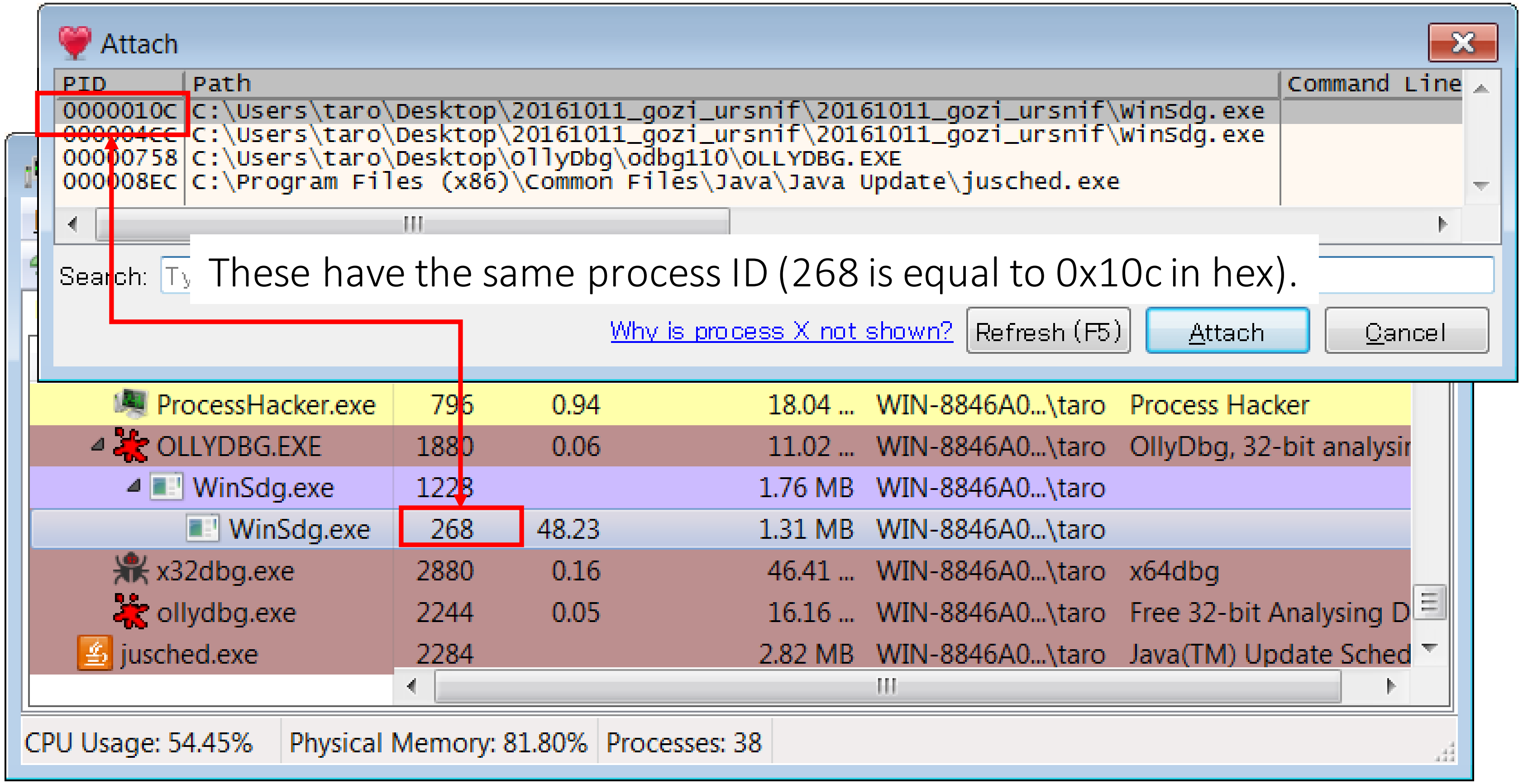
Exercise 3 (12)

- Go back to x32dbg and press F9 to execute malware. Then malware is terminated. But the child process of the malware raise the CPU rate because we replaced the first instruction of the target process with an infinite loop instruction.



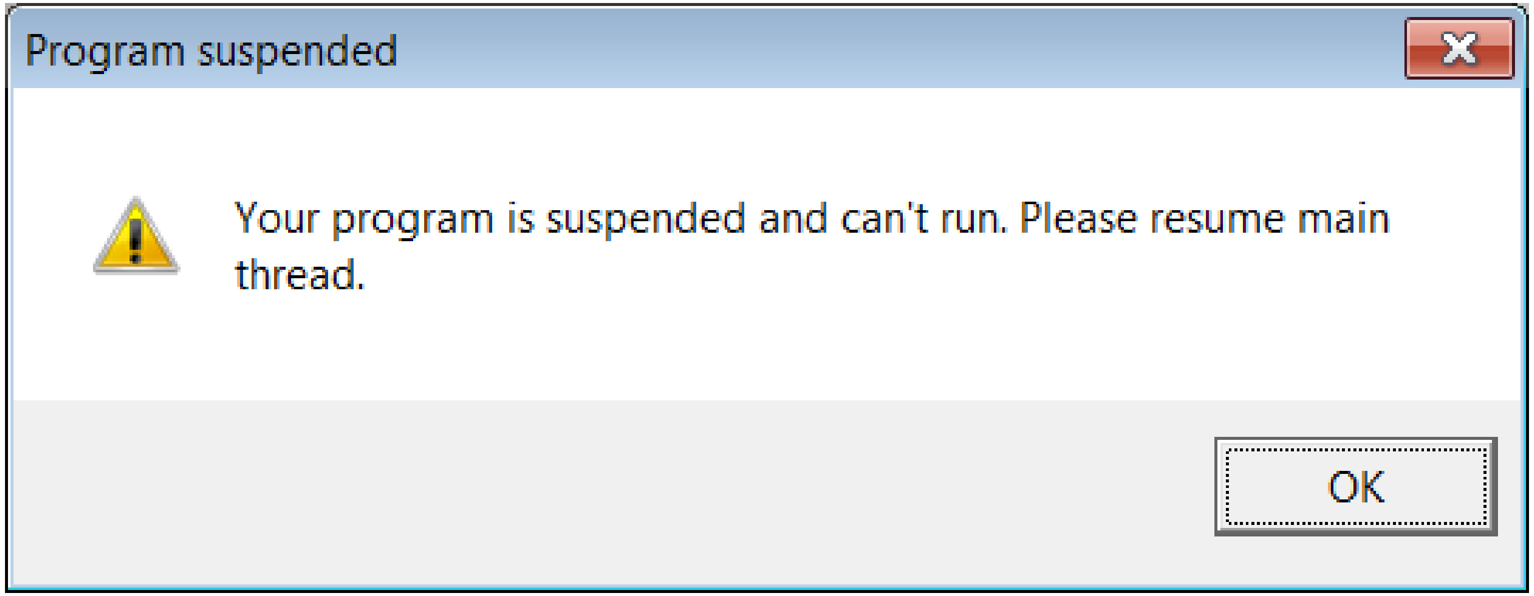
Exercise 3 (13)

- Attach the child process of the malware.
  - From menu bar of OllyDbg / x32dbg, choose “File” -> “Attach” and pick the child process.



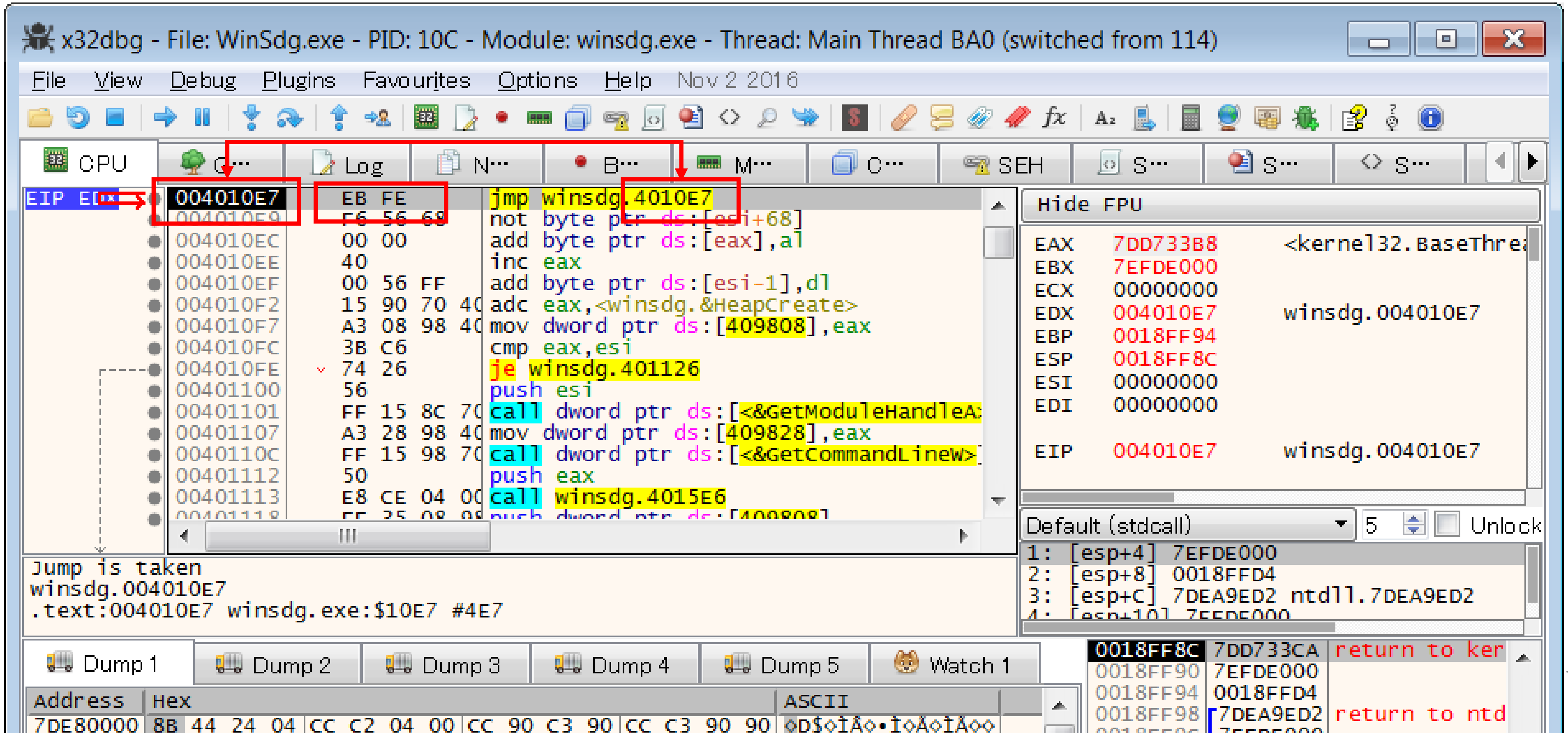
Exercise 3 (14)

- When we attach the process, OllyDbg 1.10 might show us the below popup and can't resume the thread execution. If you encounter this issue, use OllyDbg 2.0 or other debuggers (e.g. x32dbg).



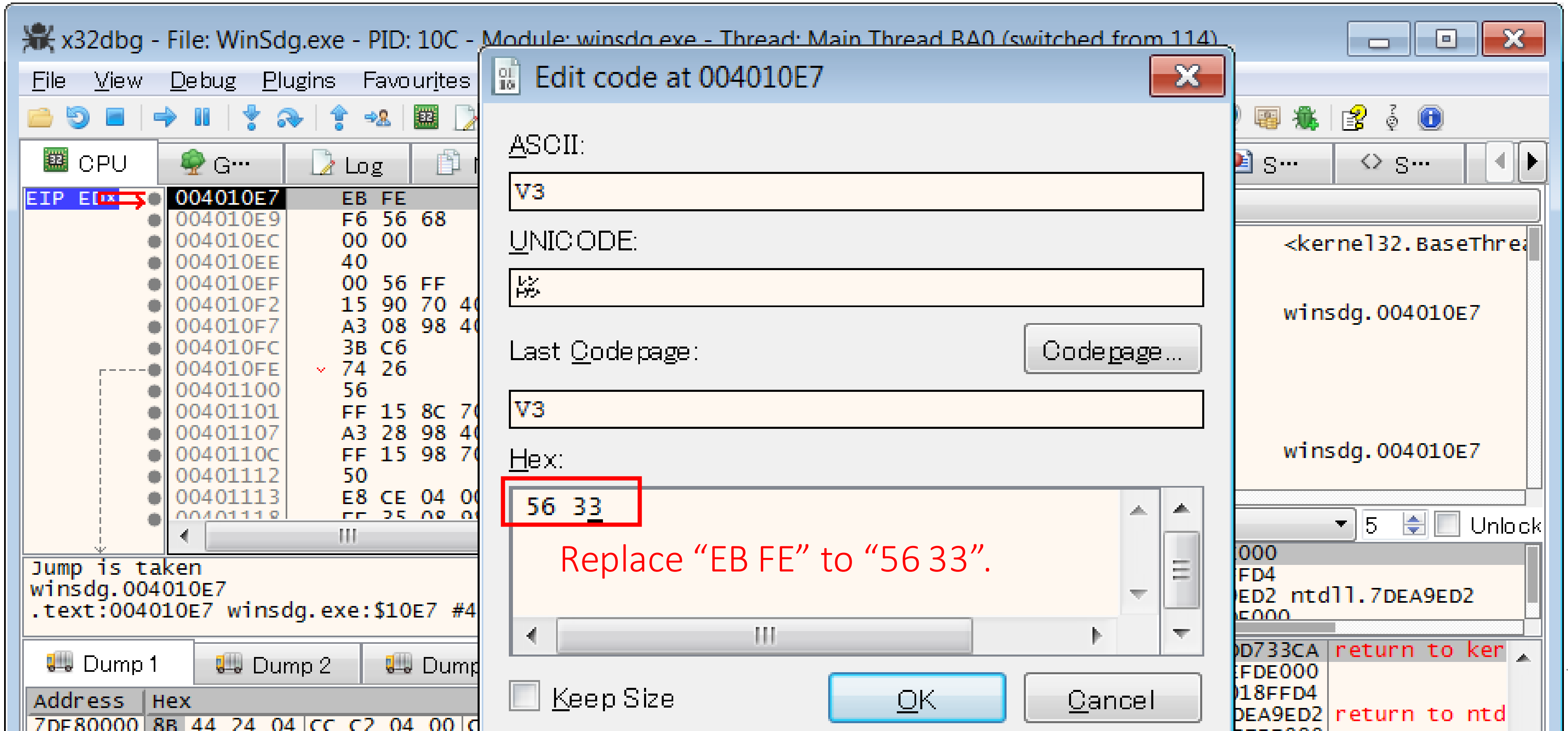
Exercise 3 (15)

- Hit F9 (execution) and F12 (pause) in a debugger, then you will see the infinite loop.



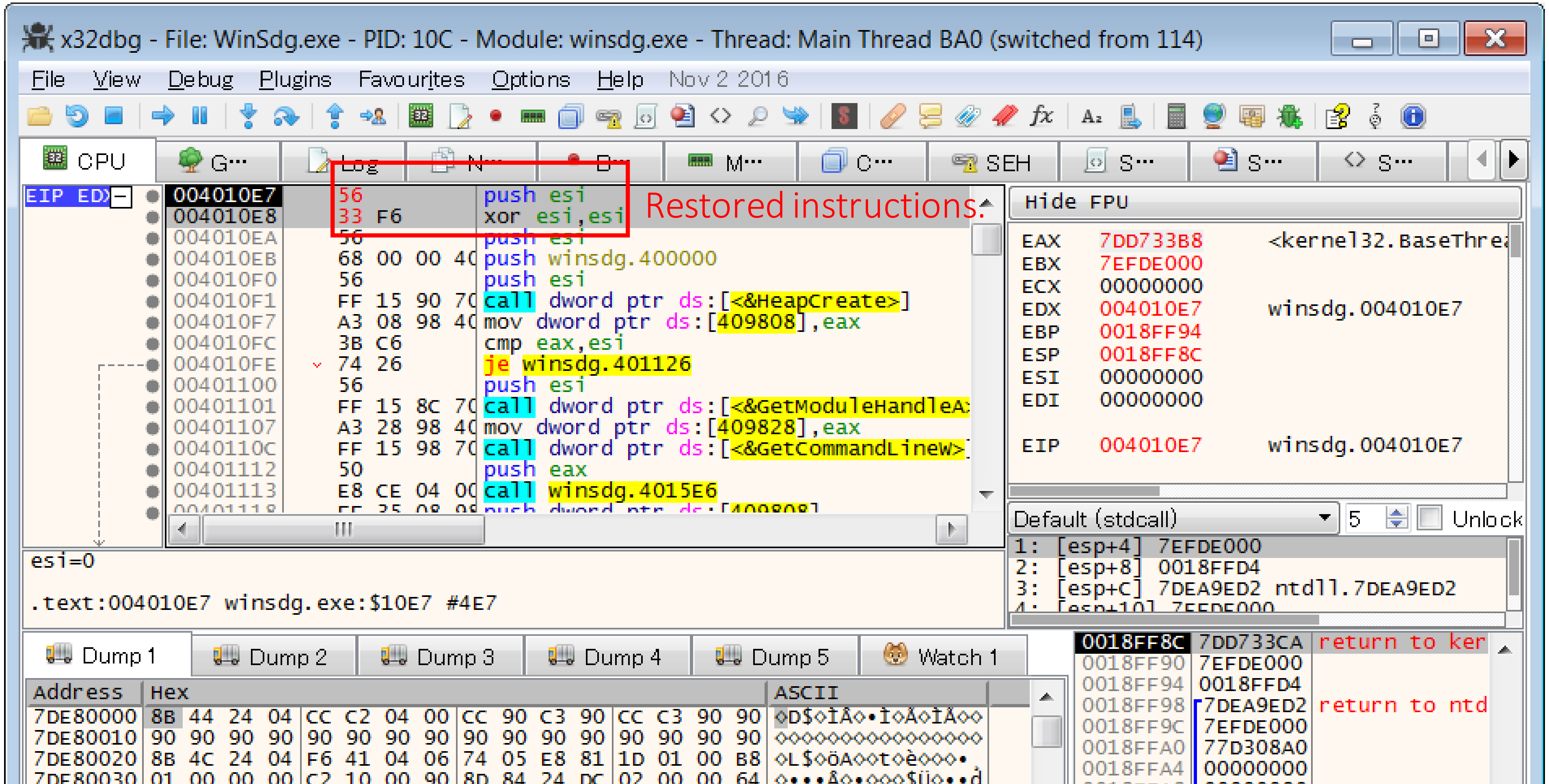
Exercise 3 (16)

- Then press “Ctrl+E” and restore the original bytes you memorized previously. (in this case, “56 33”)



Exercise 3 (17)

- Now you can debug the malicious code in the target process.



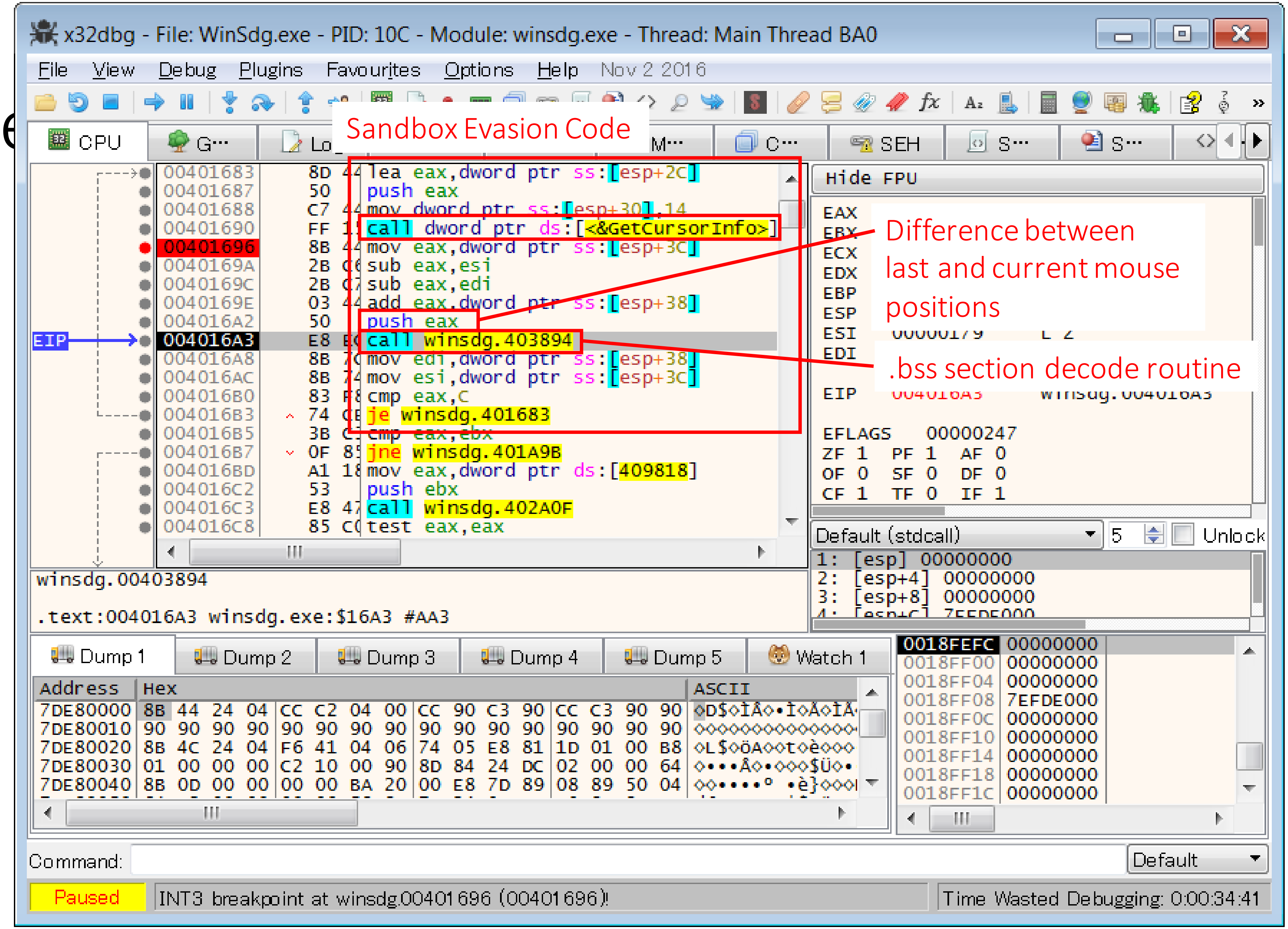
Exercise 3 (18)

- Press “Ctrl + G” and type “SetupDiGetDeviceRegistryPropertyA” in the text box below. And then click “OK”.
- Then you can apply Exercise 2 (6) and later.
  - But see next slide…
- At Exercise 2 (23), you will need to deal with Reflective PE Injection again and the file handle technique when you analyze “copied gozi” again.
  - So you need to combine Exercise 2 with this exercise.

Exercise 3 (19)

- GetCursorInfo
  - Actually, this malware sample has another sandbox evasion technique.
  - Malware checks mouse movement with GetCursorInfo API.
  - If your sandbox system has no mouse activity emulation, this malware never takes any action.

Exe


x32dbg - File: WinSdg.exe - PID: 10C - Module: winsdg.exe - Thread: Main Thread BA0
Sandbox Evasion Code
call dword ptr ds:[<&GetCursorInfo>]
Difference between last and current mouse positions
push eax
call winsdg.403894
.bss section decode routine
Paused
INT3 breakpoint at winsdg.00401696 (00401696)!
Time Wasted Debugging: 0:00:34:41

Exercise 3 (21)

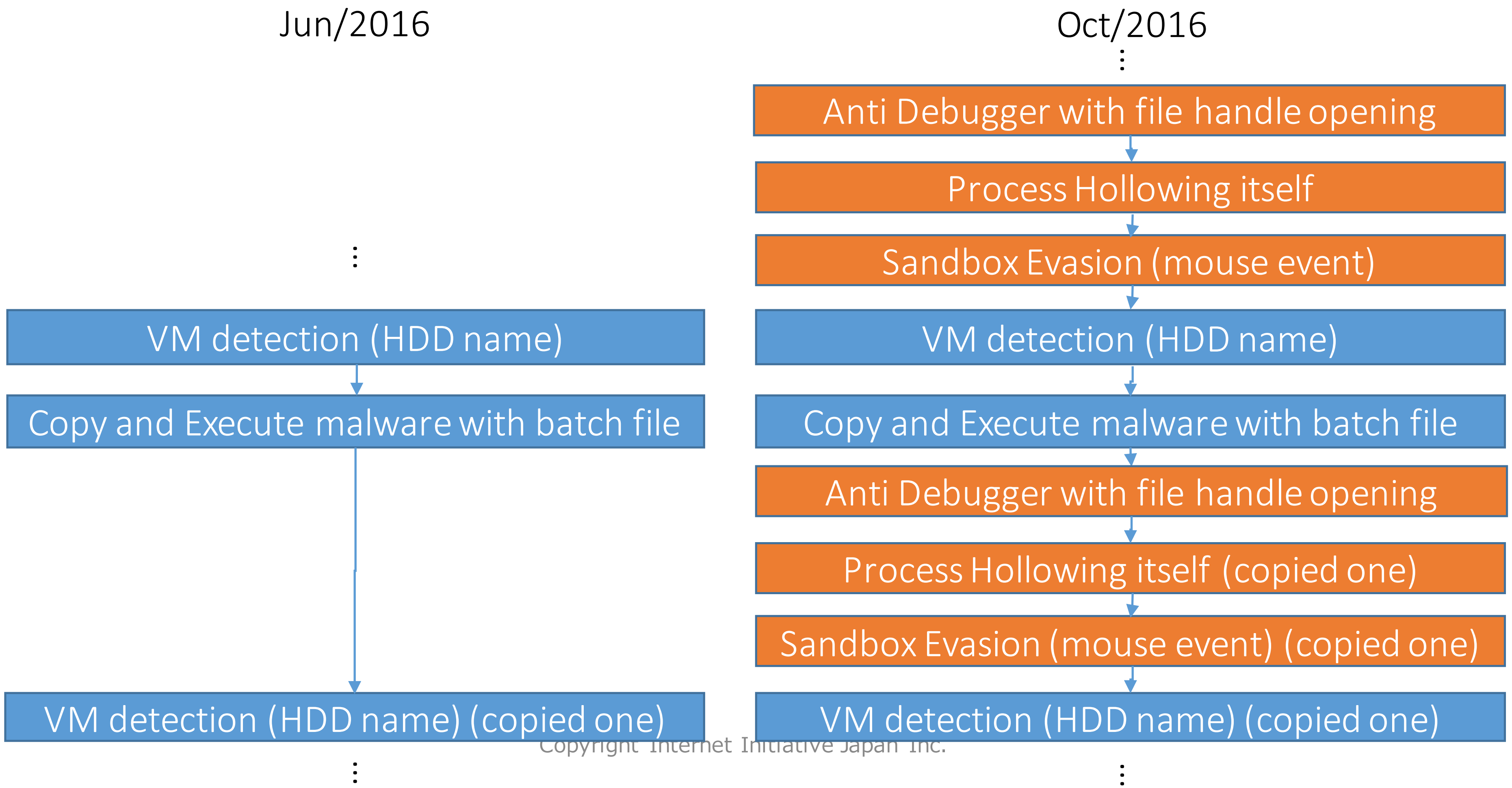
- Then you can get suspicious HTTP communications.

fakenet64

```
12/26/16 08:30:19 PM [    HTTPListener80] --------------------------------------
------------------------------------------
12/26/16 08:30:19 PM [    HTTPListener80] GET /images/ka4rMOmH4nxjXh1uaVcB2WG/ZO
_2BZa5Tn/maZts2dOjUXt_2FTf/2b3x3JquXbLO/s9YfPhHGb_2/FQypXsqMNhpCZY/v_2FTaZM2PF9g
FDVzcOap/Gzg1NGOcunfMUO9B/kc9JCnamqUg3LO4/DLtX1_2BkrJeQFmZHk/t47jy6XdN/ATpN9cmA/
tskWMuY.gif HTTP/1.1
12/26/16 08:30:19 PM [    HTTPListener80] User-Agent: Mozilla/4.0 (compatible; M
SIE 8.0; Windows NT 6.1; Win64; x64)
12/26/16 08:30:19 PM [    HTTPListener80] Host: ceeoerunw10.com
12/26/16 08:30:19 PM [    HTTPListener80] Connection: Keep-Alive
12/26/16 08:30:19 PM [    HTTPListener80] Cache-Control: no-cache
12/26/16 08:30:19 PM [    HTTPListener80]
12/26/16 08:30:19 PM [    HTTPListener80] --------------------------------------
------------------------------------------
12/26/16 08:30:19 PM [    HTTPListener80] Responding with mime type: image/gif f
ile: defaultFiles/FakeNet.html
```

Exercise 3 (22)

- Difference between gozi samples Jun/2016 and Oct/2016



Detection techniques

- Most of VM or sandbox detection techniques are the same old.
  - If you know those techniques, you will handle almost all cases.
    - Search keyword
      - Sandbox detection technique
      - VMdetect technique
      - …
  - Sometimes we might encounter new techniques though.
    - http://joe4security.blogspot.jp/2016/10/pafish-for-office-macro.html
  - References
    - https://github.com/a0rtega/pafish
    - http://artemonsecurity.com/vmde.pdf
    - http://resources.infosecinstitute.com/how-malware-detects-virtualized-environment-and-its-countermeasures-an-overview/

wizSafe

To be continued…