

Dynamic Analysis

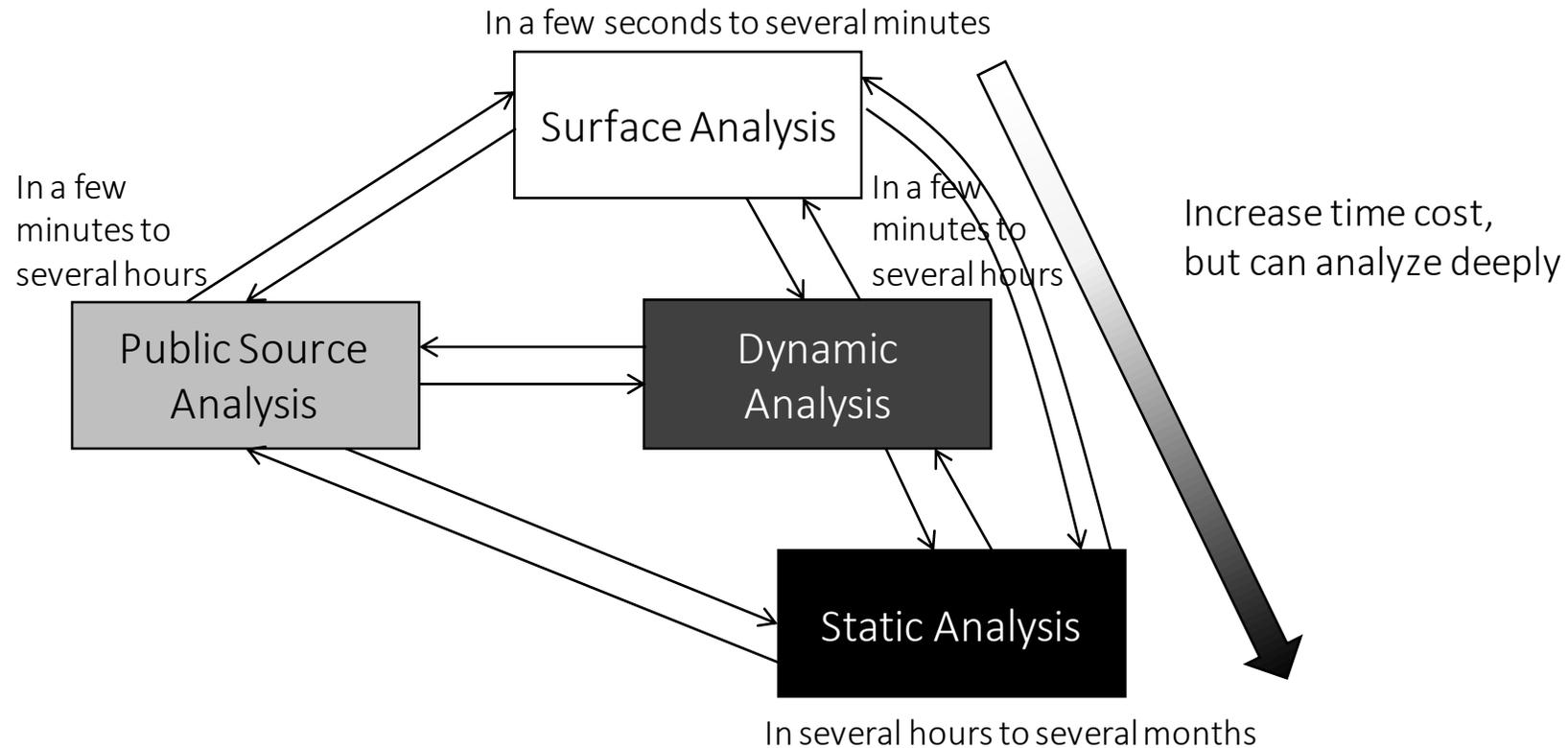
What is malware analysis?
and
What is dynamic analysis?

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?





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
suretterminal.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)

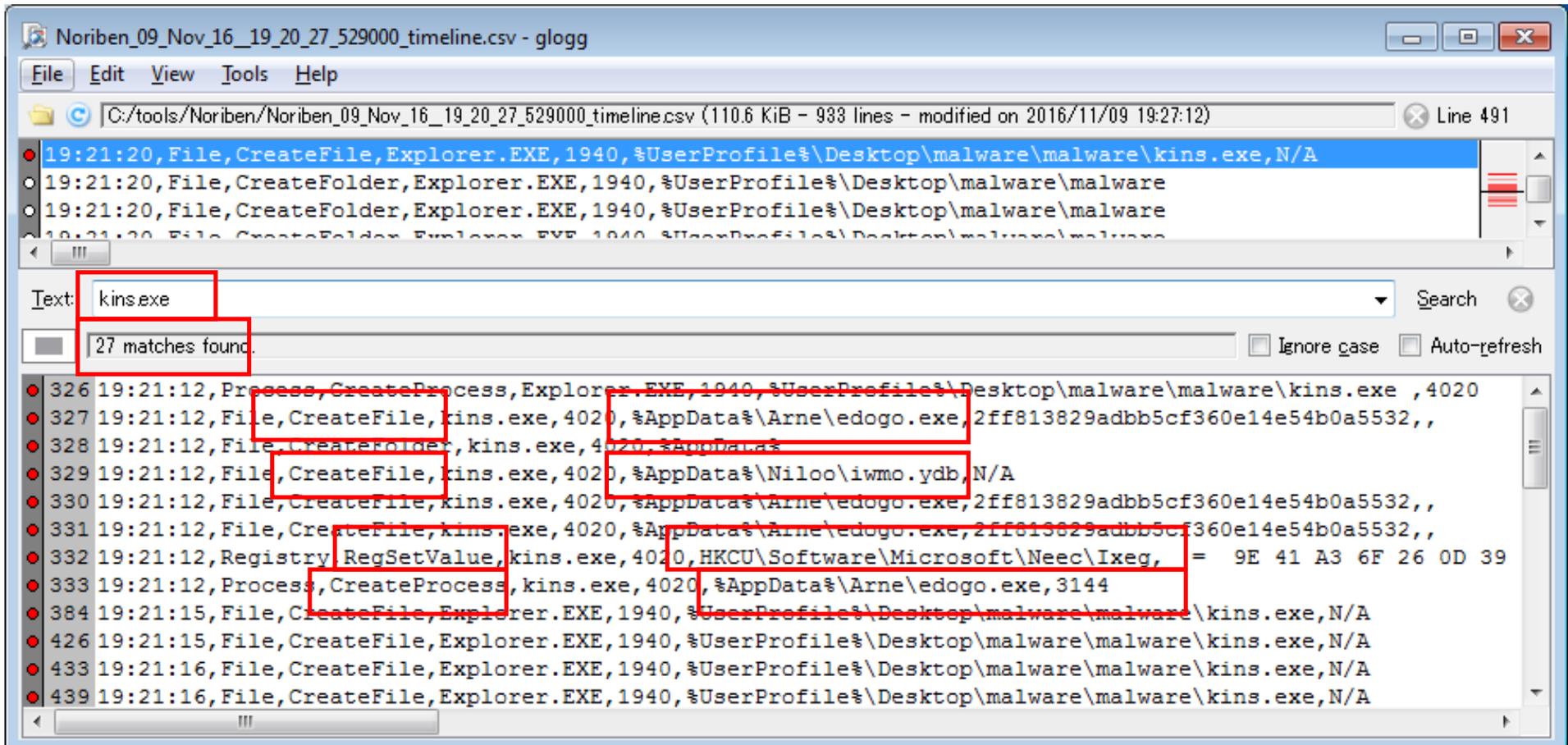
- Double-click Fakenet32.exe
 - Click “yes” when the UAC dialog shows up
- Double-click Noriben.py
 - Click “yes” when the UAC dialog shows up
- Double-click kins.exe (Banking Trojan)
 - Wait for approximately four minutes

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.



The screenshot shows the glogg application interface. The title bar reads "Noriben_09_Nov_16_19_20_27_529000_timeline.csv - glogg". The address bar shows the file path: "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)". The main window displays a list of events from a timeline report. A search bar at the bottom left contains the text "kins.exe", and a status bar below it indicates "27 matches found". Several lines in the event list are highlighted with red boxes, showing activities related to "kins.exe".

```
19:21:20, File, CreateFile, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, N/A
19:21:20, File, CreateFolder, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware
19:21:20, File, CreateFolder, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware
19:21:20, File, CreateFolder, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware
326 19:21:12, Process, CreateProcess, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, 4020
327 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Arne\edogo.exe, 2ff813829adbb5cf360e14e54b0a5532,,
328 19:21:12, File, CreateFolder, kins.exe, 4020, %AppData%
329 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Niloo\iwmo.ydb, N/A
330 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Arne\edogo.exe, 2ff813829adbb5cf360e14e54b0a5532,,
331 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Arne\edogo.exe, 2ff813829adbb5cf360e14e54b0a5532,,
332 19:21:12, Registry, RegSetValue, kins.exe, 4020, HKCU\Software\Microsoft\Nec\Ireg, = 9E 41 A3 6F 26 0D 39
333 19:21:12, Process, CreateProcess, kins.exe, 4020, %AppData%\Arne\edogo.exe, 3144
384 19:21:15, File, CreateFile, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, N/A
426 19:21:15, File, CreateFile, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, N/A
433 19:21:16, File, CreateFile, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, N/A
439 19:21:16, File, CreateFile, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, N/A
```

Exercise 1 (4)

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

The separator is “| (pipe)”

37 matches found

```

19:21:20, File, CreateFile, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, N/A
19:21:20, File, CreateFolder, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware
19:21:20, File, CreateFolder, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware
19:21:20, File, CreateFolder, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware
326 19:21:12, Process, CreateProcess, Explorer.EXE, 1940, %UserProfile%\Desktop\malware\malware\kins.exe, 4020
327 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Arne\edogo.exe, 2ff813829adbb5cf360e14e54b0a5532,,
328 19:21:12, File, CreateFolder, kins.exe, 4020, %AppData%\Arne\edogo.exe
329 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Niloo|iwmoydb, N/A
330 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Arne\edogo.exe, 2ff813829adbb5cf360e14e54b0a5532,,
331 19:21:12, File, CreateFile, kins.exe, 4020, %AppData%\Arne\edogo.exe, 2ff813829adbb5cf360e14e54b0a5532,,
332 19:21:12, Registry, RegSetValue, kins.exe, 4020, HKCU\Software\Microsoft\Nec\Ixeg, = 9E 41 A3 6F 26 0D 39
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\Nec\Ixeg, = 9C 41 A3 6F 26 0D
380 19:21:14, Registry, RegSetValue, Explorer.EXE, 1940, HKCU\Software\Microsoft\Nec\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
  
```

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|Nilooliwmo.ydb|NeeclIxe Search

37 matches found. Benign Explorer.exe never modifies registry values formerly created by malware.

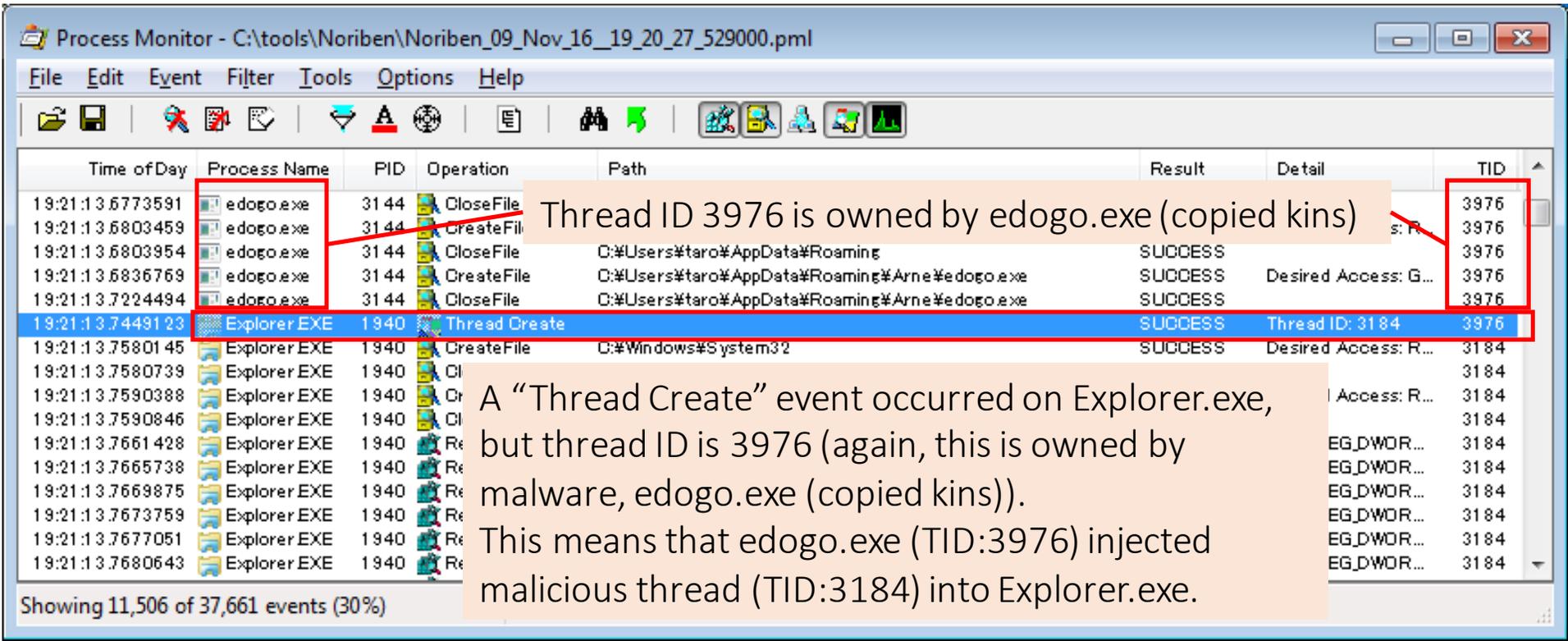
```
333 19:21:12, Process, CreateProcess, kins.exe, 4020, %AppData%\Arne\edogo.exe, 3144
368 19:21:13, Registry, RegSetValue, Explorer.EXE, 1940, HKCU\Software\Microsoft\NeeclIxe, = 9C 41 A3 6F 26 OD
380 19:21:14, Registry, RegSetValue, Explorer.EXE, 1940, HKCU\Software\Microsoft\NeeclIxe, = 9A 41 A3 6F 26 OD
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, Exp
433 19:21:16, File, CreateFile, Exp
439 19:21:16, File, CreateFile, Exp
445 19:21:17, File, CreateFile, Exp
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
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\NeeclIxe, = 9A 41 A3 6F 26 OD
639 19:21:24, Registry, RegSetValue, Explorer.EXE, 1940, HKCU\Software\Microsoft\NeeclIxe, = 9A 41 A3 6F 26 OD
640 19:21:24, Registry, RegSetValue, Explorer.EXE, 1940, HKCU\Software\Microsoft\NeeclIxe, = 92 41 A3 6F 26 OD
739 19:25:14, Registry, RegSetValue, Explorer.EXE, 1940, HKCU\Software\Microsoft\NeeclIxe, = 92 41 A3 6F 26 OD
917 19:25:30, Registry, RegSetValue, Explorer.EXE, 1940, HKCU\Software\Microsoft\NeeclIxe, = 92 41 A3 6F 26 OD
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
```

Benign Explorer.exe never register a run key to registry for starting malware automatically when a pc is booted.

Benign Explorer.exe never modifies registry values formerly created by malware.

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).



Process Monitor - C:\tools\Noriben\Noriben_09_Nov_16_19_20_27_529000.pml

Time of Day	Process Name	PID	Operation	Path	Result	Detail	TID
19:21:13.6773591	edogo.exe	3144	CloseFile				3976
19:21:13.6803459	edogo.exe	3144	CreateFile				3976
19:21:13.6803954	edogo.exe	3144	CloseFile	C:\Users\taro\AppData\Roaming	SUCCESS		3976
19:21:13.6836769	edogo.exe	3144	CreateFile	C:\Users\taro\AppData\Roaming\Arne\edogo.exe	SUCCESS	Desired Access: G...	3976
19:21:13.7224494	edogo.exe	3144	CloseFile	C:\Users\taro\AppData\Roaming\Arne\edogo.exe	SUCCESS		3976
19:21:13.7449123	Explorer.EXE	1940	Thread Create		SUCCESS	Thread ID: 3184	3976
19:21:13.7580145	Explorer.EXE	1940	CreateFile	C:\Windows\System32	SUCCESS	Desired Access: R...	3184
19:21:13.7580739	Explorer.EXE	1940	CloseFile				3184
19:21:13.7590388	Explorer.EXE	1940	CreateFile			Access: R...	3184
19:21:13.7590846	Explorer.EXE	1940	CloseFile				3184
19:21:13.7661428	Explorer.EXE	1940	ReadFile			EG_DWOR...	3184
19:21:13.7665738	Explorer.EXE	1940	ReadFile			EG_DWOR...	3184
19:21:13.7669875	Explorer.EXE	1940	ReadFile			EG_DWOR...	3184
19:21:13.7673759	Explorer.EXE	1940	ReadFile			EG_DWOR...	3184
19:21:13.7677051	Explorer.EXE	1940	ReadFile			EG_DWOR...	3184
19:21:13.7680643	Explorer.EXE	1940	ReadFile			EG_DWOR...	3184

Thread ID 3976 is owned by edogo.exe (copied kins)

A "Thread Create" event occurred on Explorer.exe, but thread ID is 3976 (again, this is owned by malware, edogo.exe (copied kins)). This means that edogo.exe (TID:3976) injected malicious thread (TID:3184) into Explorer.exe.

Showing 11,506 of 37,661 events (30%)

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: <u>unknown binary</u> *2	Noriben/procmon
	Create	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{GUID} Value: <u>path to itself</u> (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

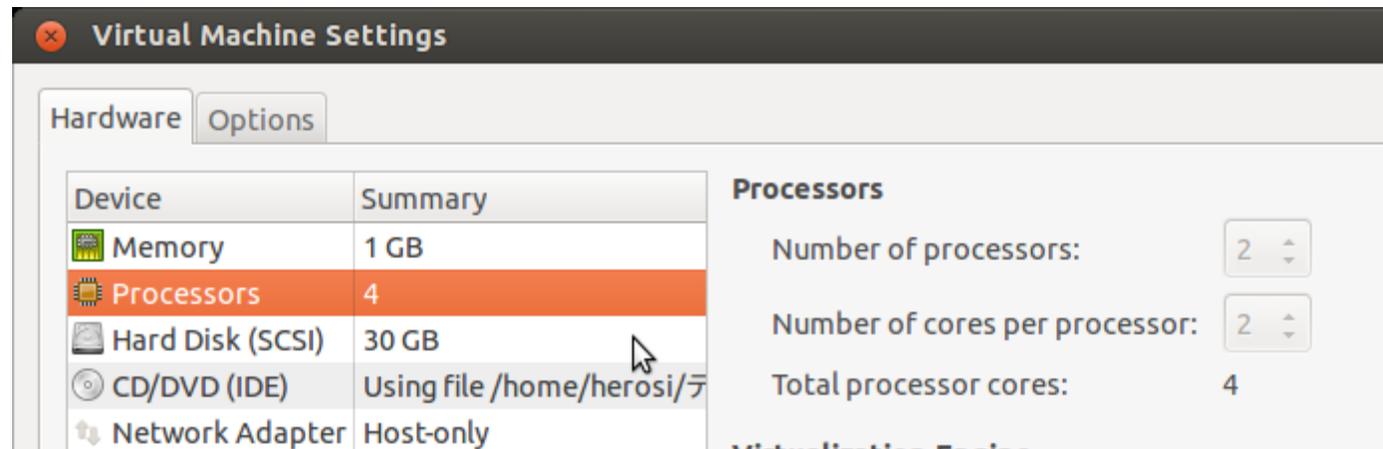
- One of the most likely causes of the problem is analysis environment detection.
 - There are many techniques to detect analysis environment.
 - VM detection
 - Detecting backdoor ports (Host-Guest communication)
 - Detecting differences between real PCs and VMs by executing some specific machine instructions
 - 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”.

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

The screenshot shows the x32dbg debugger interface. The main window displays assembly code for the module ntdll.dll, thread 468. The instruction pointer (EIP) is at address 7DE8FC52. The assembly code includes instructions such as `add esp,4`, `ret 28`, `mov eax,26`, `xor ecx,ecx`, `lea edx,dword ptr fs:...`, and `call dword ptr fs:...`. The registers window on the right shows the current state of the CPU registers, with EAX, EBX, ECX, EDX, EBP, ESP, ESI, and EDI all set to 00000000. The EIP register is at 7DE8FC52. The stack window at the bottom shows the current stack frame, with the return address at 7DE8FC52. The status bar at the bottom indicates the debugger is paused and shows the loaded DLLs and the time wasted debugging.

x32dbg - File: gozi_ursnif.exe - PID: 764 - Module: ntdll.dll - Thread: 468

File View Debug Plugins Favourites Options Help Jul 29 2016

CPU G... Log N... B... M... C... SEH S... S...

EIP → 7DE8FC52 83 C4 04 add esp,4
7DE8FC55 C2 28 00 ret 28
7DE8FC58 B8 26 00 00 00 mov eax,26
7DE8FC5D 33 C9 xor ecx,ecx
7DE8FC5F 8D 54 24 04 lea edx,dword ptr fs:00000000
7DE8FC63 64 FF 15 C0 00 00 00 call dword ptr fs:00000000
7DE8FC6A 83 C4 04 add esp,4
7DE8FC6D C2 2C 00 ret 2C
7DE8FC70 B8 27 00 00 00 mov eax,27
7DE8FC75 33 C9 xor ecx,ecx
7DE8FC77 8D 54 24 04 lea edx,dword ptr fs:00000000
7DE8FC7B 64 FF 15 C0 00 00 00 call dword ptr fs:00000000
7DE8FC82 83 C4 04 add esp,4
7DE8FC85 C2 08 00 ret 8
7DE8FC88 B8 28 00 00 00 mov eax,28
7DE8FC8D 33 C9 xor ecx,ecx
7DE8FC8F 8D 54 24 04 lea edx,dword ptr fs:00000000
7DE8FC93 64 FF 15 C0 00 00 00 call dword ptr fs:00000000
7DE8FC9A 83 C4 04 add esp,4

Hide FPU

EAX 00000000
EBX 00000000
ECX 00000000
EDX 00000000
EBP 0018F5EC
ESP 0018F598
ESI 7EFD0000
EDI 0018F6C4

EIP 7DE8FC52 ntdll.7DE8FC52

EFLAGS 00000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0

Default (stdcall) 5 [] Unlock

1: [esp+4] 7DEABECC ntdll.7DEABECC
2: [esp+8] 0000002C
3: [esp+C] FFFFFFFF
4: [esp+10] 0018F6C4

esp=18F598
.text:7DE8FC52 ntdll.dll:\$1FC52 #10052

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1

Address	Hex	ASCII
7DE80000	8B 44 24 04 CC C2 04 00 CC 90 C3 90 CC C3 90 90	.D\$.iA..i.A.iA
7DE80010	90 90 90 90 90 90 90 90 90 90 90 90 90 90
7DE80020	8B 4C 24 04 F6 41 04 06 74 05 E8 81 1D 01 00 B8	.L\$.oA..t.è...
7DE80030	01 00 00 00 C2 10 00 90 8D 84 24 DC 02 00 00 64	...A...\$Ù...
7DE80040	8B 0D 00 00 00 00 BA 20 00 E8 7D 89 08 89 50 04°.è}...

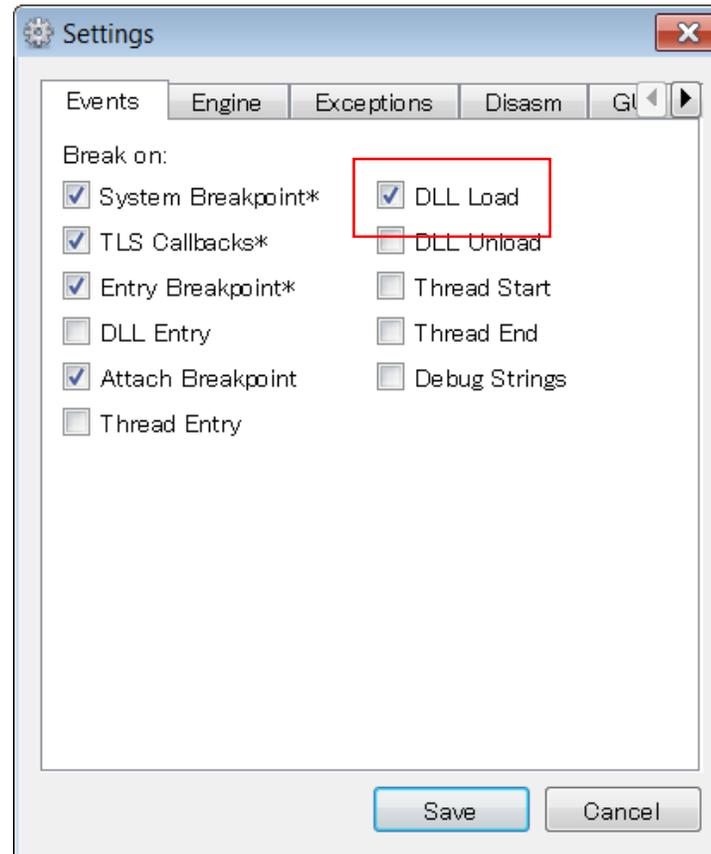
0018F598 7DE8FC52 return to
0018F59C 7DEABECC return to
0018F5A0 0000002C
0018F5A4 FFFFFFFF
0018F5A8 0018F6C4
0018F5AC 00000000
0018F5B0 00000000
0018F5B4 00000000
0018F5B8 0018F68C

Command: Default

Paused DLL Loaded: 7DAA0000 C:\Windows\SysWOW64\psapi.dll Time Wasted Debugging: 0:00:55:48

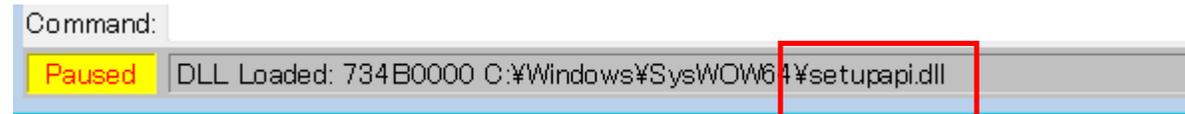
Exercise 2 (4)

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

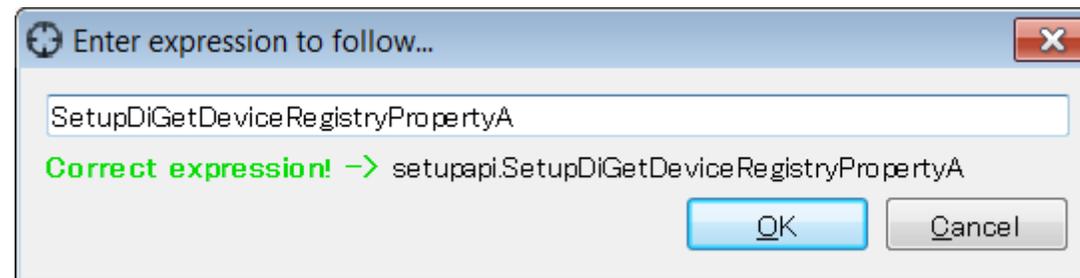


Exercise 2 (5)

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

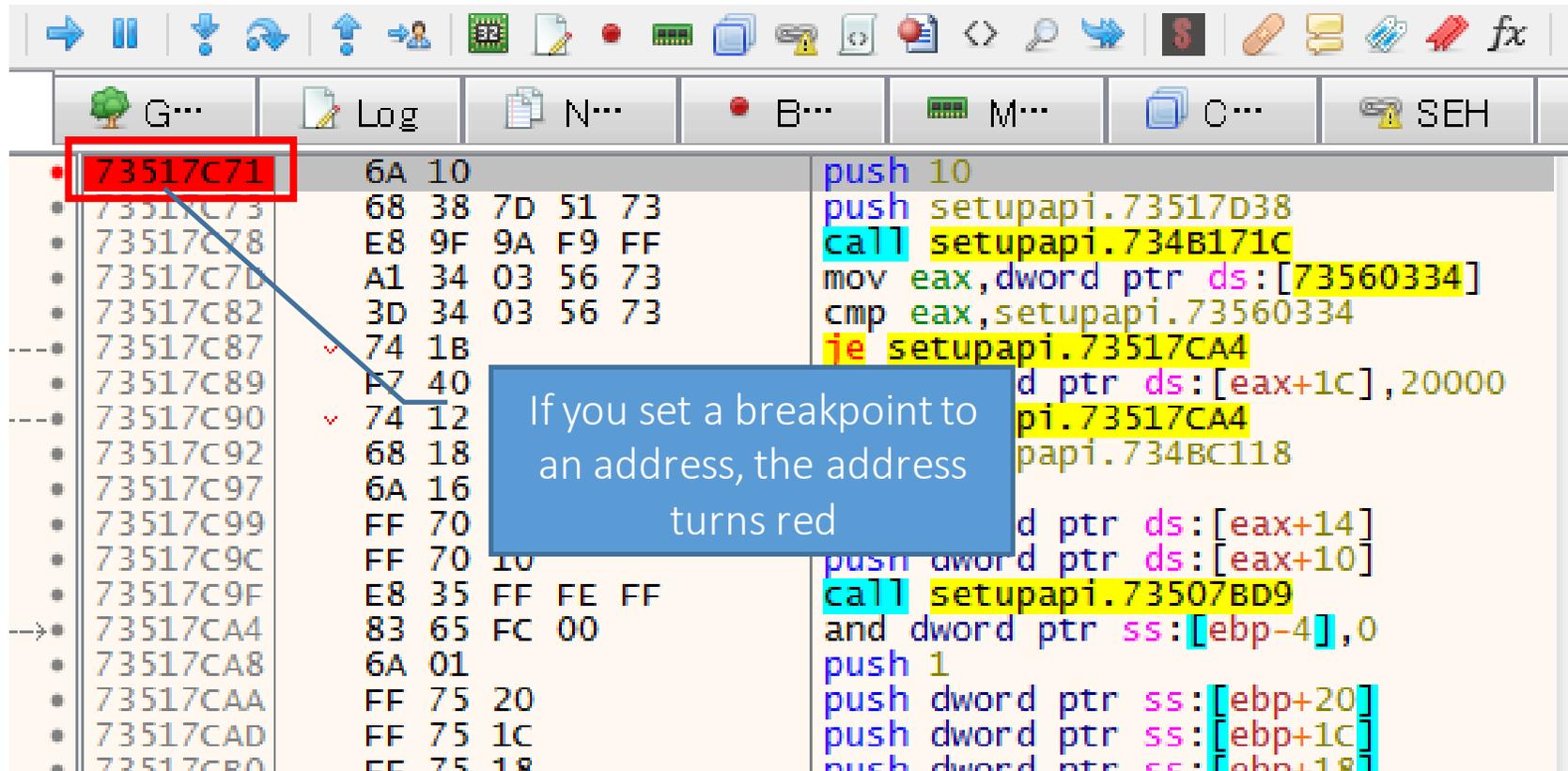


- Press “Ctrl + G” and type “SetupDiGetDeviceRegistryPropertyA” in the text box below. And then click “OK”.



Exercise 2 (6)

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

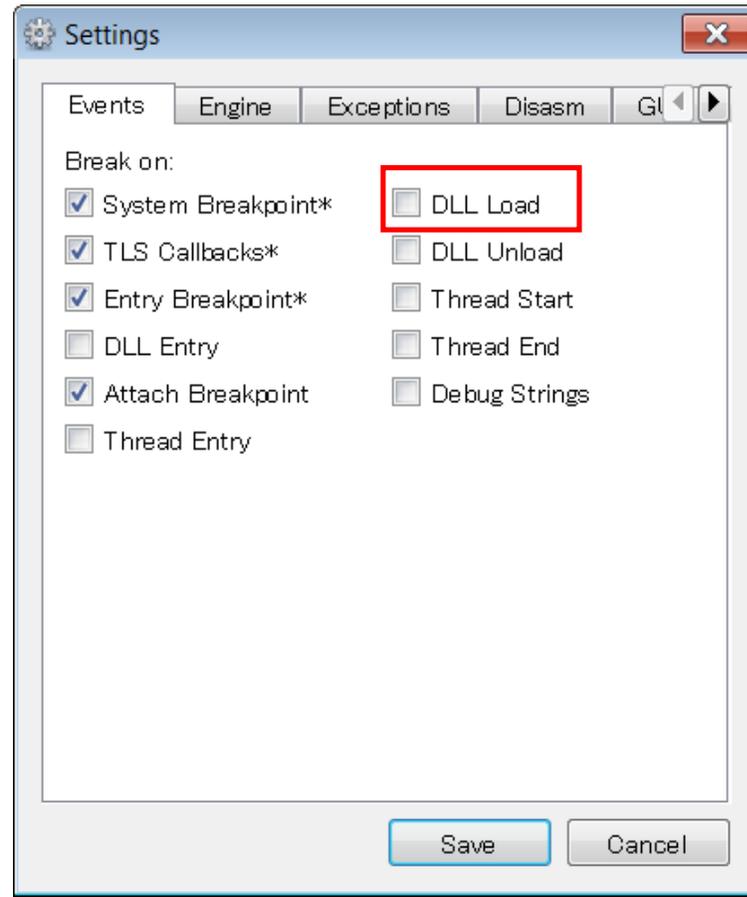


The screenshot shows a debugger window with the following assembly code:

Address	Disassembly
73517C71	push 10
73517C73	push setupapi.73517D38
73517C78	call setupapi.734B171C
73517C7D	mov eax,dword ptr ds:[73560334]
73517C82	cmp eax,setupapi.73560334
73517C87	je setupapi.73517CA4
73517C89	d ptr ds:[eax+1C],20000
73517C90	pi.73517CA4
73517C92	papi.734BC118
73517C97	d ptr ds:[eax+14]
73517C99	FF 70
73517C9C	FF 70 10
73517C9F	E8 35 FF FE FF
73517CA4	call setupapi.73507BD9
73517CA8	and dword ptr ss:[ebp-4],0
73517CAA	push 1
73517CAD	push dword ptr ss:[ebp+20]
73517CB0	push dword ptr ss:[ebp+1C]
73517CB0	push dword ptr ss:[ebp+18]

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.

x32dbg - File: gozi_ursnif.exe - PID: 764 - Module: setupapi.dll - Thread: 468

File View Debug Plugins Favourites Options Help Jul 29 2016

CPU Log N... B... M... C... SEH S... S... R... T...

EIP	ESI	Instruction
73517C71	6A 10	push 10
73517C73	68 38 7D 51 73	push setupapi.73517D38
73517C78	E8 9F 9A F9 FF	call setupapi.7348171C
73517C7D	A1 34 03 56 73	mov eax,dword ptr ds:[73560334]
73517C82	3D 34 03 56 73	cmp eax,setupapi.73560334
73517C87	74 1B	je setupapi.73517CA4
73517C89	F7 40 1C 00 00 02 00	test dword ptr ds:[eax+1C],20000
73517C90	74 12	je setupapi.73517CA4
73517C92	68 18 C1 4B 73	push setupapi.7348C118
73517C97	6A 16	push 16
73517C99	FF 70 14	push dword ptr ds:[eax+14]
73517C9C	FF 70 10	push dword ptr ds:[eax+10]
73517C9F	E8 35 FF FE FF	call setupapi.73507BD9
73517CA4	83 65 FC 00	and dword ptr ss:[ebp-4],0
73517CA8	6A 01	push 1

Hide FPU

EAX	0018FEC8
EBX	00000000
ECX	7DEA3CA3 ntdll.7DEA3CA3
EDX	02320178
EBP	0018FF00
ESP	0018FE9C
ESI	73517C71 <setupapi.SetupDiGetDeviceRe
EDI	027187D0
EIP	73517C71 <setupapi.SetupDiGetDeviceRe

Default (stdcall) 5 Unlocked

1: [esp+4]	005D1FA8
2: [esp+8]	0018FEC8
3: [esp+C]	0000000C
4: [esp+10]	0018FEC4

.text:73517C71 setupapi.dll:\$67C71 #67071 <SetupDiGetDeviceRegistryPropertyA>

Address	Hex	ASCII
7DE80000	8B 44 24 04 CC C2 04 00 CC 90 C3 90 CC C3 90 90	.D\$.i.A..i.A.i.A..
7DE80010	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
7DE80020	8B 4C 24 04 F6 41 04 06 74 05 E8 81 1D 01 00 B8	.L\$.o.A..t.e.....
7DE80030	01 00 00 00 C2 10 00 90 8D 84 24 DC 02 00 00 64	...A.....\$i...d

Command: Default

Paused INT3 breakpoint at <setupapi.SetupDiGetDeviceRegistryPropertyA> (73517C71) Time Wasted Debugging: 0:01:25:15

Exercise 2 (9)

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

You can see the result of the API in the stack (5th argument, this means "PropertyBuffer", of the API) Then, right-click and choose "Follow DWORD in Dump"

Address	Hex	AS
7DE80000	8B 44 24 04 CC C2 04 00 CC 90 C3 90 CC C3 90 90	.D
7DE80010	90 90 90 90 90 90 90 90 90 90 90 90 90 90	.L
7DE80020	8B 4C 24 04 F6 41 04 06 74 05 E8 81 1D 01 00 B8	.L
7DE80030	01 00 00 00 C7 10 00 90 8D 84 74 DC 07 00 00 64	.L

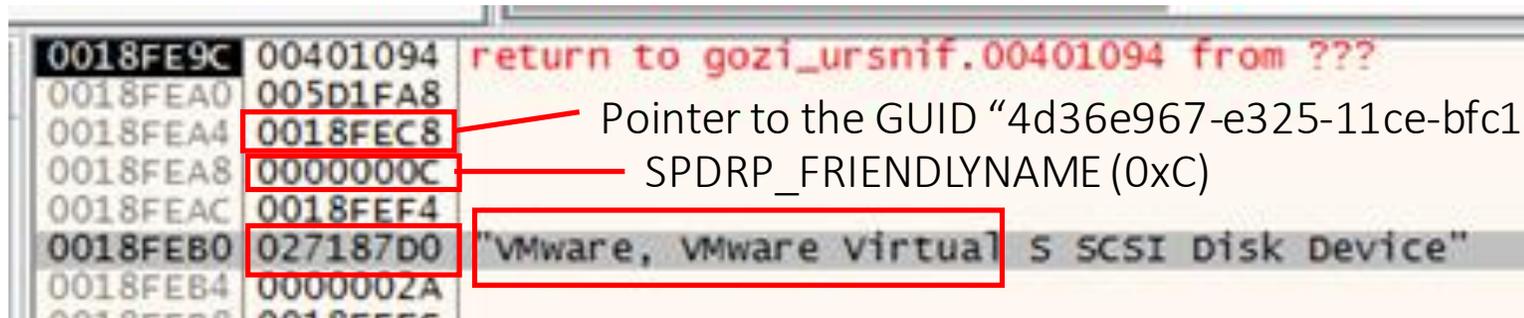
Address	Hex	Comment
0018FE9C	00401094	return to gozi_ursnif.00401094 from ???
0018FEA0	005D1FA8	
0018FEA4	0018FEA8	
0018FEA8	0000000C	
0018FEAC	0018FEF4	
0018FEB0	027187D0	"VMware, VMware virtual S SCSI Disk Device"
0018FEB4	0000002A	
0018FEB8	0018FEFC	

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

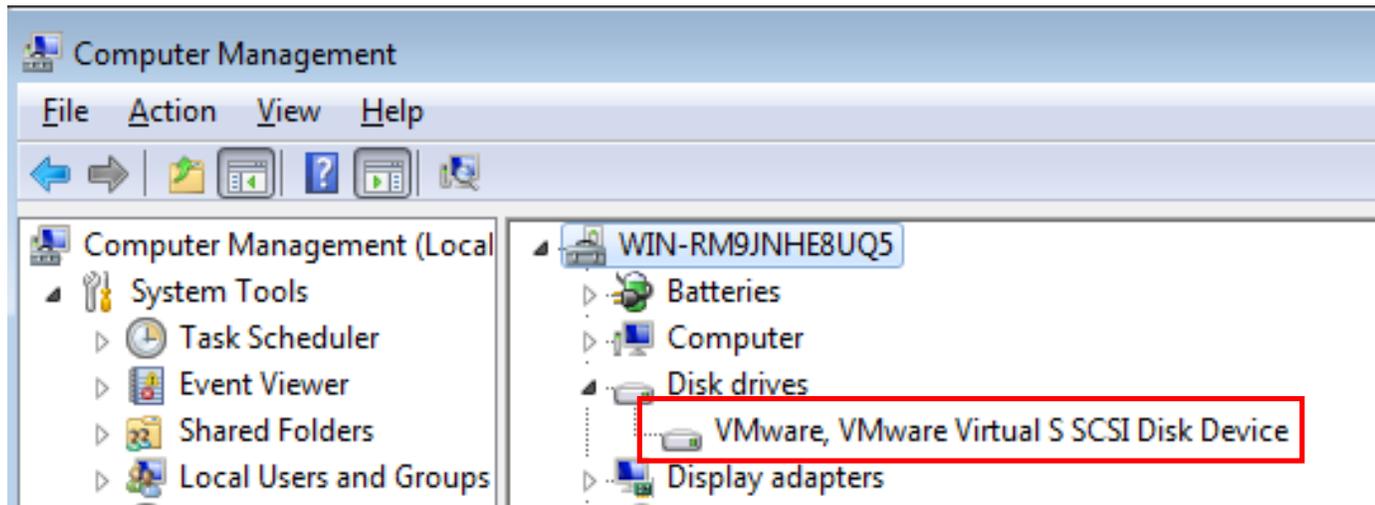
Disk Drives
 Class = **DiskDrive**
 ClassGuid = {4d36e967-e325-11ce-bfc1-08002be10318}
 This class includes hard disk drives. See also the HDC and SCSIAdapter classes.

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



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)

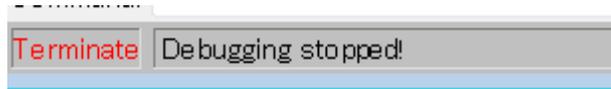
- Replace “PropertyBuffer” with arbitrary characters.

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.

Address	Hex	ASCII
027187D0	56 4D 77 61 72 65 2C 20 56 4D 77 61 72 65 20 56	VMware, VMware V
027187E0	69 72 74 75 61 6C 20 53 20 53 43 53 49 20 44 69	irtual s SCSI Di
027187F0	73 6B 20 44 65 76 69 63 65 00 5C 00 6D 00 61 00	sk Device.\.m. a.
02718800	3C AA 56 26 C0 B0 00 00 48 8A 71 02 C4 00 32 02	<=V&A'...H.n.A.?

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%\COL0snap\d3diound.exe %User
[CreateProcess] csrss.exe:400 > "???\WinDir\system32\conhost.exe" [Child PID: 2972]
[CreateProcess] cmd.exe:2880 > "cmd /C %AppData%\COL0snap\d3diound.exe %UserProfile%\Desktop\malware\GOZI U~1.EXE"
[CreateProcess] cmd.exe:2000 > "%AppData%\COL0snap\d3diound.exe %UserProfile%\Desktop\malware\GOZI U~1.EXE" [Cb
```

```
"cmd /c %LocalAppData%\Temp\4B61\2.bat %AppData%\COL0snap\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%\COL0snap\d3diound.exe [MD5: a780221be9d11249ea3845794714ba67]
[CreateFile] gozi_ursnif.exe:1892 > %AppData%\COL0snap\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-B540-9FD777574A20}
[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\apiIkmon = C:\Users\taro\AppData\Roaming\COL0snap\d3diound.exe
[RegDeleteValue] taskhost.exe:1772 > HKCU\Software\Microsoft\Windows\CurrentVersion\Run\apiIkmon
```

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\apiIkmon
```

```
= C:\Users\taro\AppData\Roaming\COL0snap\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)

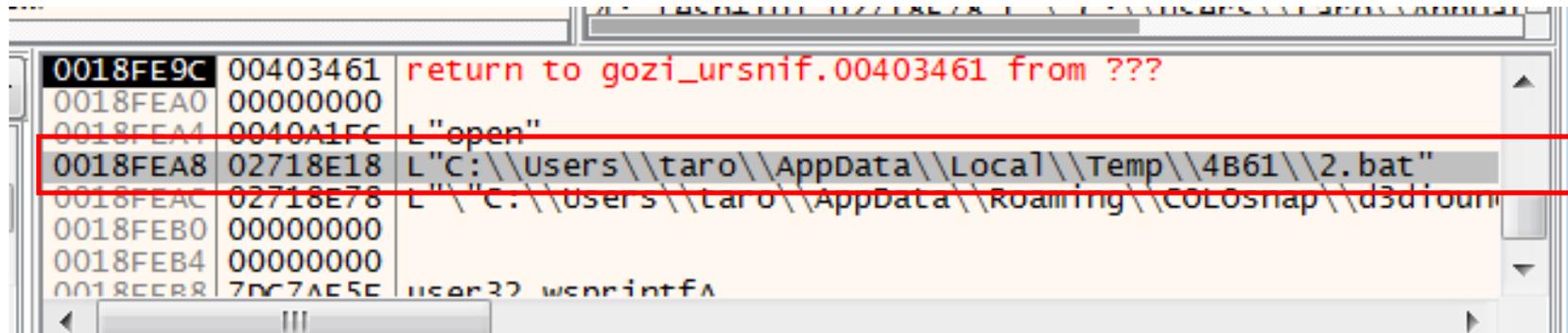
- What API does malware use to execute the batch file? (Cont.)
 - If you have finished setting the breakpoints, hit F9.
 - If you hit a breakpoint, you can get the detail of this activities.
 - If you see the process termination at the left bottom of the x64dbg window, it's sign that it was failed.
 - 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.

The image shows a Windows registry editor window with the following data:

Path	Value Name	Value Data
0018FE9C	00403461	return to gozi_ursnif.00403461 from ???
0018FEA0	00000000	
0018FEA4	0040A1FC	L"open"
0018FEA8	02718E18	L"C:\\Users\\taro\\AppData\\Local\\Temp\\4B61\\2.bat"
0018FEAC	02718E78	L"C:\\Users\\taro\\AppData\\Roaming\\COLOsnap\\d3droun
0018FEB0	00000000	
0018FEB4	00000000	
0018FEB8	70C7AE5E	user32 wsrprintfa

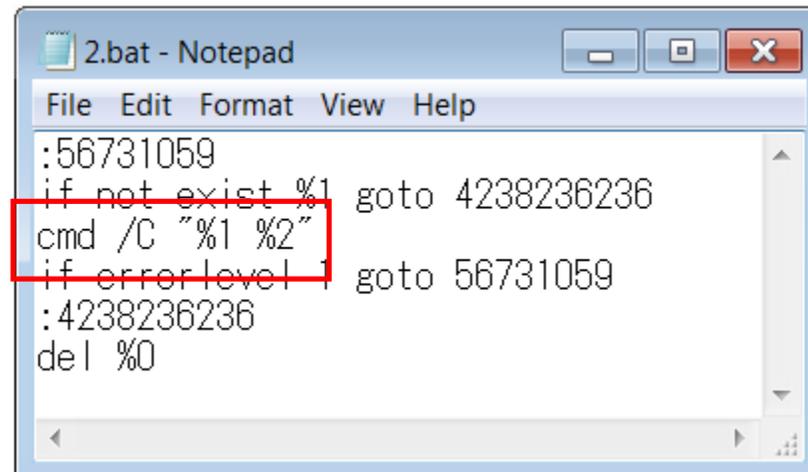
Below the registry editor is a File Explorer window showing the path: Computer > Local Disk (C:) > Users > taro > AppData > Local > Temp > 4B61. A file named 2.bat is listed with a date modified of 2016/11/24 20:21 and type Win.

The contents of the 2.bat file, viewed in Notepad, are:

```
:56731059
if not exist %1 goto 4238236236
cmd /C "%1 %2"
if errorlevel 1 goto 56731059
:4238236236
del %0
```

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.



```
2.bat - Notepad
File Edit Format View Help
:56731059
if not exist %1 goto 4238236236
cmd /C "%1 %2"
if errorlevel 1 goto 56731059
:4238236236
del %0
```

- 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.

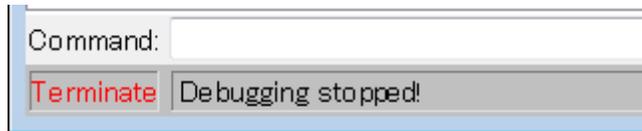
```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\apiIkmon
```

```
= C:\Users\taro\AppData\Roaming\COL0snap\d3diound.exe
```

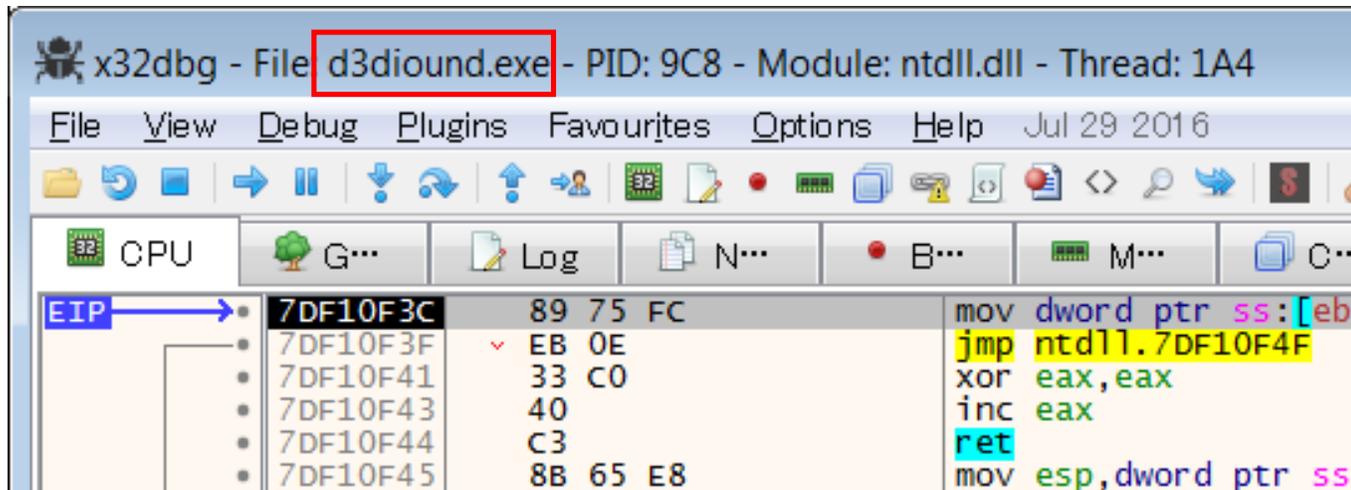
- Then we have two strategies here.
 - Execute copied gozi with original one as the argument in a debugger.
 - The batch file uses this method.
 - Execute copied gozi simply in a debugger.
 - If the installation task is finished, this method is used because of “Run” key.
- Let’s take 2nd method!

Exercise 2 (23)

- Hit F9 until the debugging process is terminated.

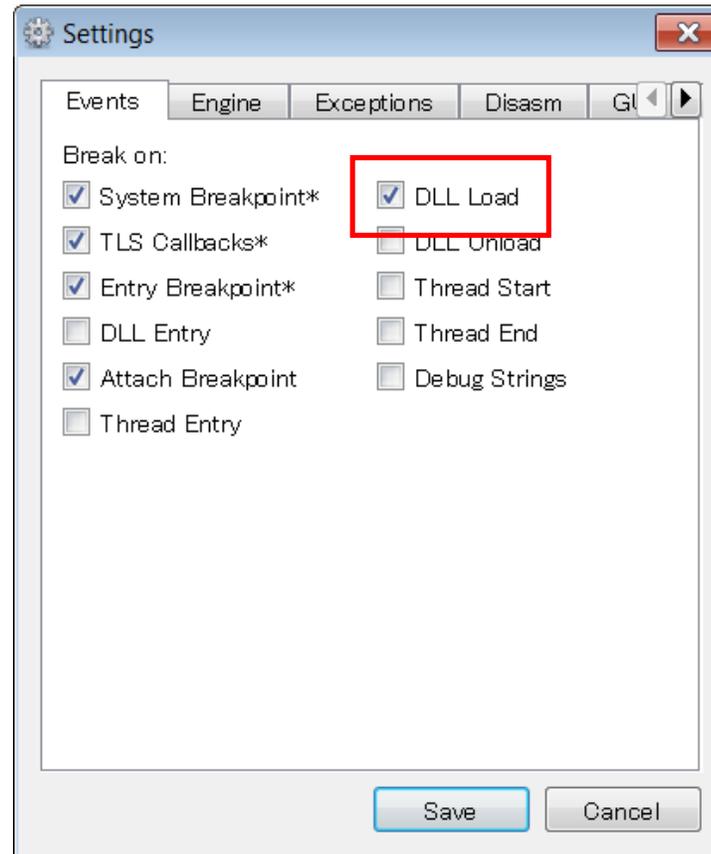


- Then load copied gozi into x32dbg.



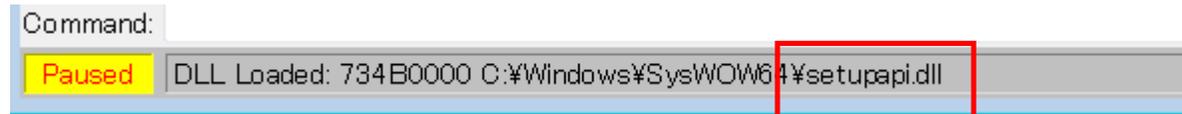
Exercise 2 (24)

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

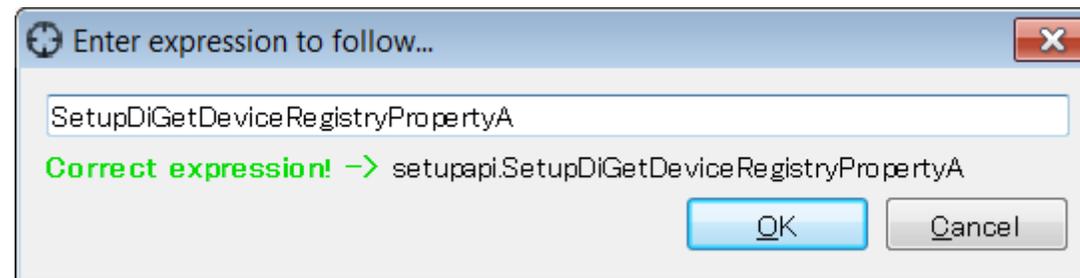


Exercise 2 (25)

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

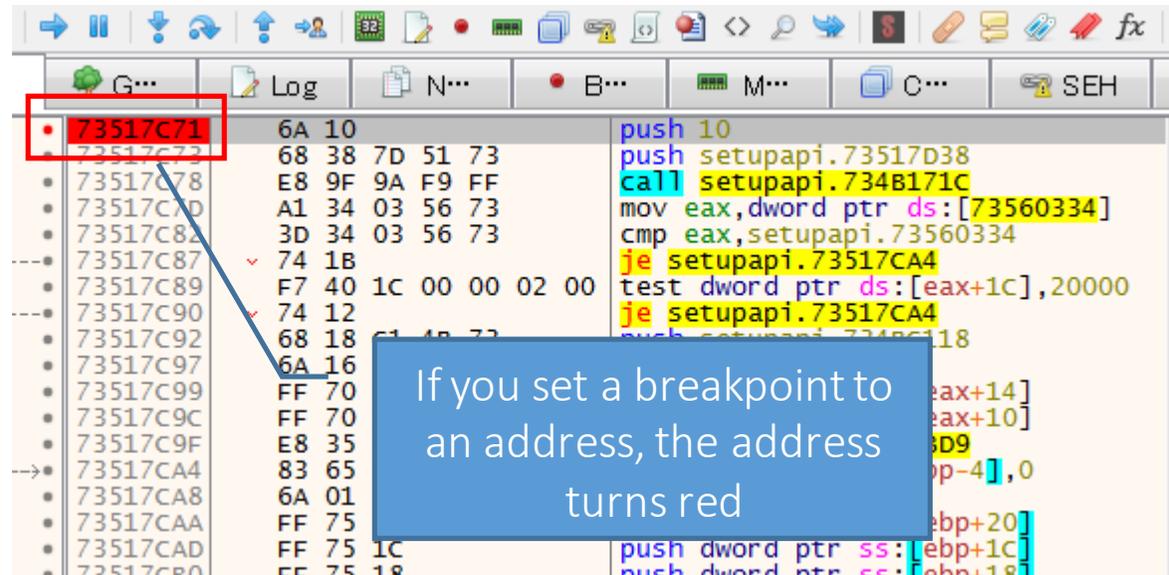


- Press “Ctrl + G” and type “SetupDiGetDeviceRegistryPropertyA” in the text box below. And then click “OK”.



Exercise 2 (26)

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



Address	Disassembly
73517C71	push 10
73517C73	push setupapi.73517D38
73517C78	call setupapi.734B171C
73517C7D	mov eax,dword ptr ds:[73560334]
73517C82	cmp eax,setupapi.73560334
73517C87	je setupapi.73517CA4
73517C89	test dword ptr ds:[eax+1c],20000
73517C90	je setupapi.73517CA4
73517C92	push setupapi.734B1118
73517C97	mov eax,dword ptr ds:[eax+14]
73517C99	mov eax,dword ptr ds:[eax+10]
73517C9C	mov eax,dword ptr ds:[eax+09]
73517C9F	mov eax,dword ptr ds:[eax-4],0
73517CA4	push dword ptr ss:[ebp+20]
73517CA8	push dword ptr ss:[ebp+1c]
73517CAA	push dword ptr ss:[ebp+18]
73517CAD	
73517CB0	

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”.

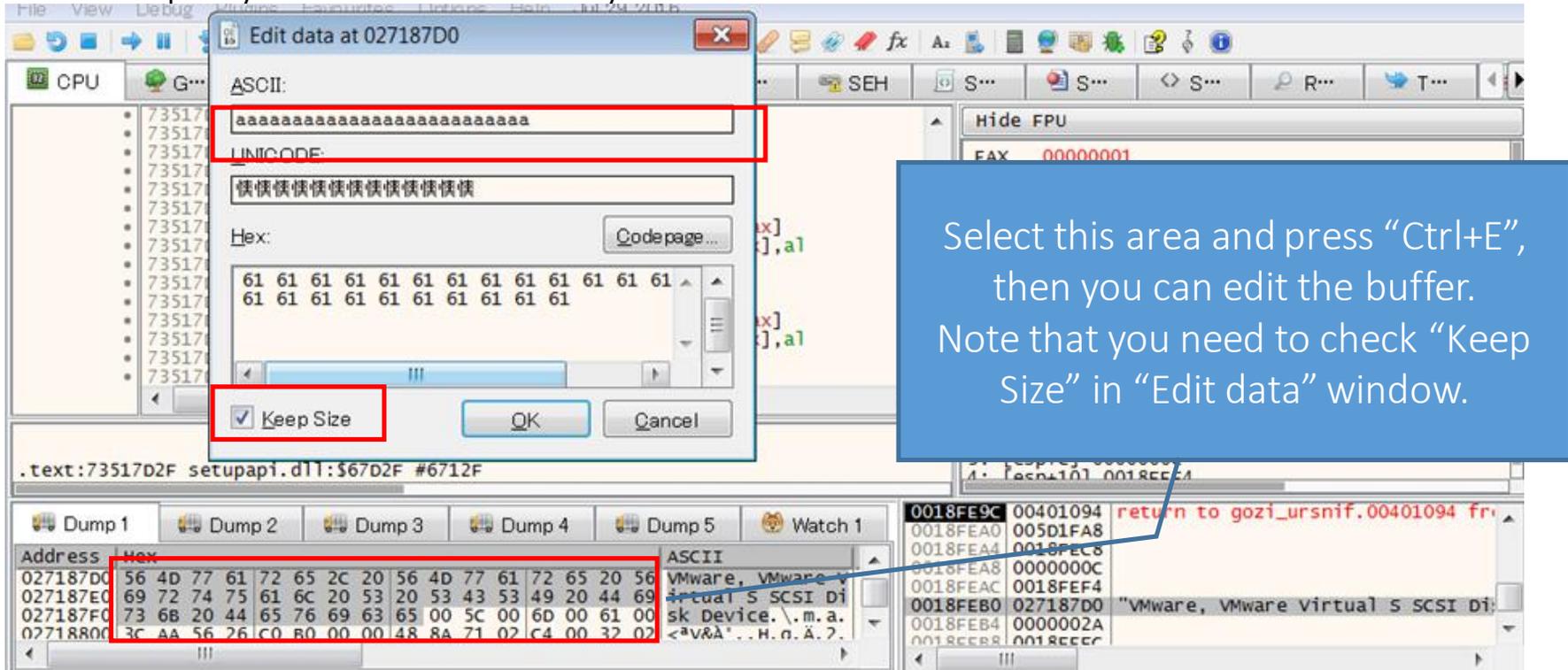
You can see the result of the API in the stack (the 5th argument, that is “PropertyBuffer”, of the API)
Then, right-click and choose “Follow DWORD in Dump”

Address	Hex	AS
7DE80000	8B 44 24 04 CC C2 04 00 CC 90 C3 90 CC C3 90 90	.C
7DE80010	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	.L
7DE80020	8B 4C 24 04 F6 41 04 06 74 05 E8 81 1D 01 00 88	.L
7DF80030	01 00 00 00 C2 10 00 90 8D 84 74 7C 02 00 00 64	.L

Address	Hex	Comment
0018FE9C	00401094	return to gozi_ursnif.00401094 from ???
0018FEA0	005D1FA8	
0018FEA4	0018FEC8	
0018FEA8	0000000C	
0018FEAC	0018FEP4	
0018FEB0	027187D0	"vmware, vmware virtual s SCSI Disk Device"
0018FEB4	0000002A	
0018FEB8	0018FEEC	

Exercise 2 (29)

- Replace “PropertyBuffer” with arbitrary characters.



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

Exercise 3

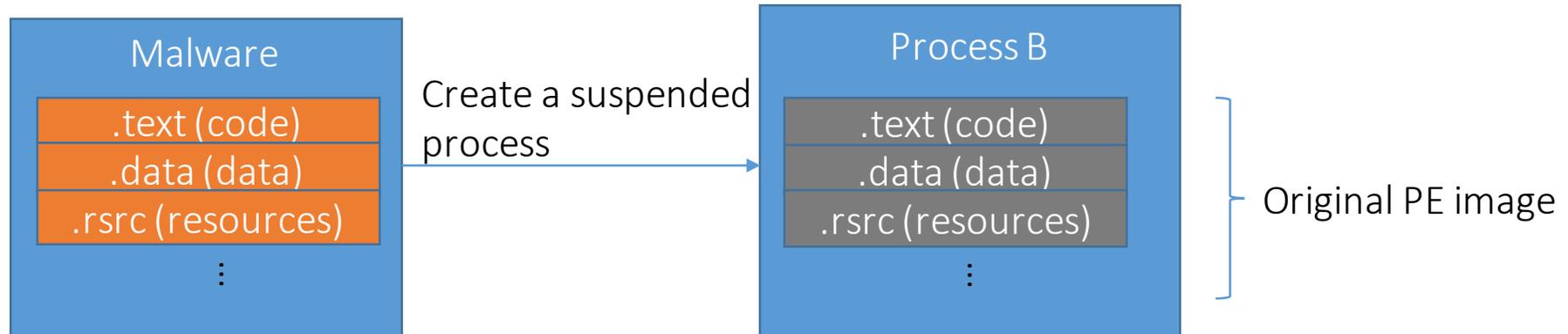
Dealing with the Process Hollowing / PE Reflective Injection Technique with Debuggers

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!

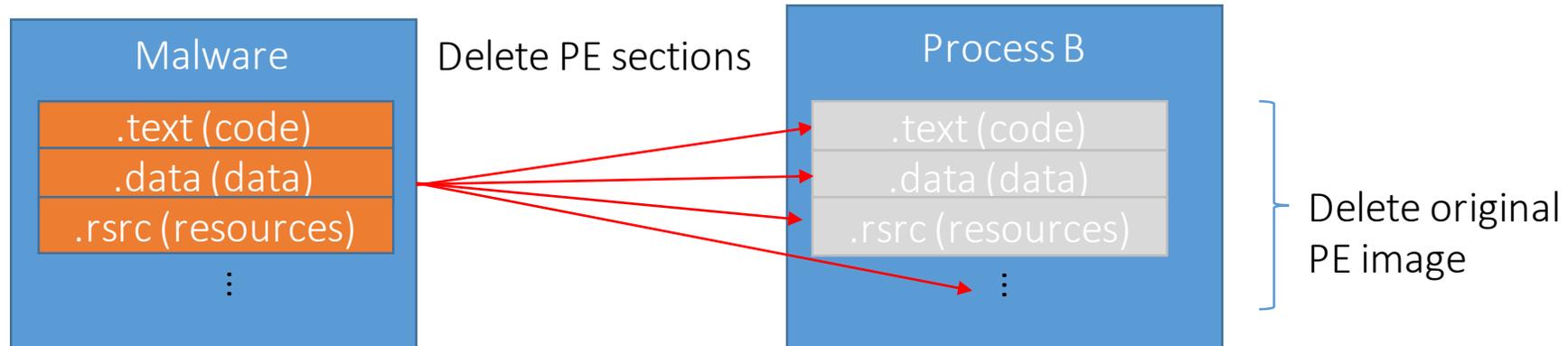
What is Process Hollowing / PE Reflective Injection?

- 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



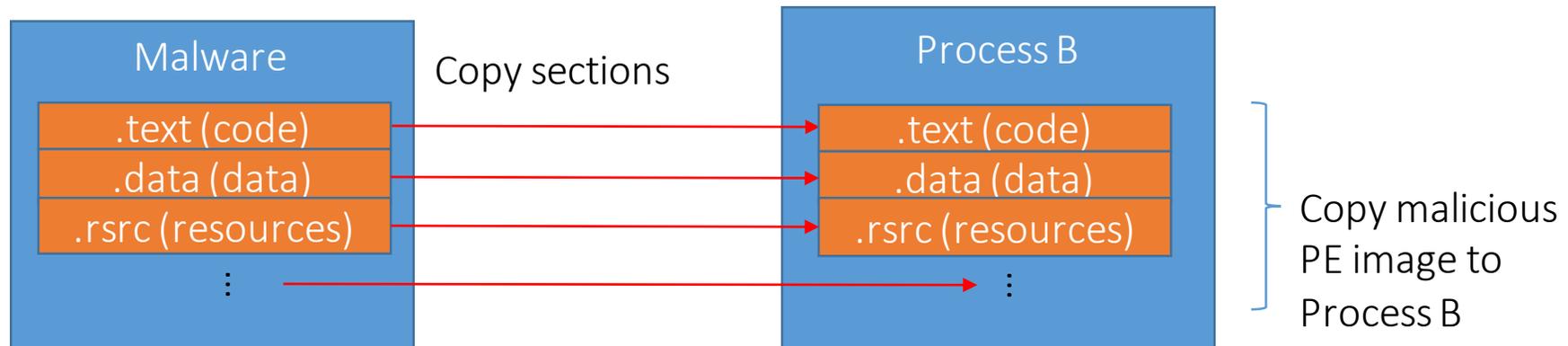
What is Process Hollowing / PE Reflective Injection?

- 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.



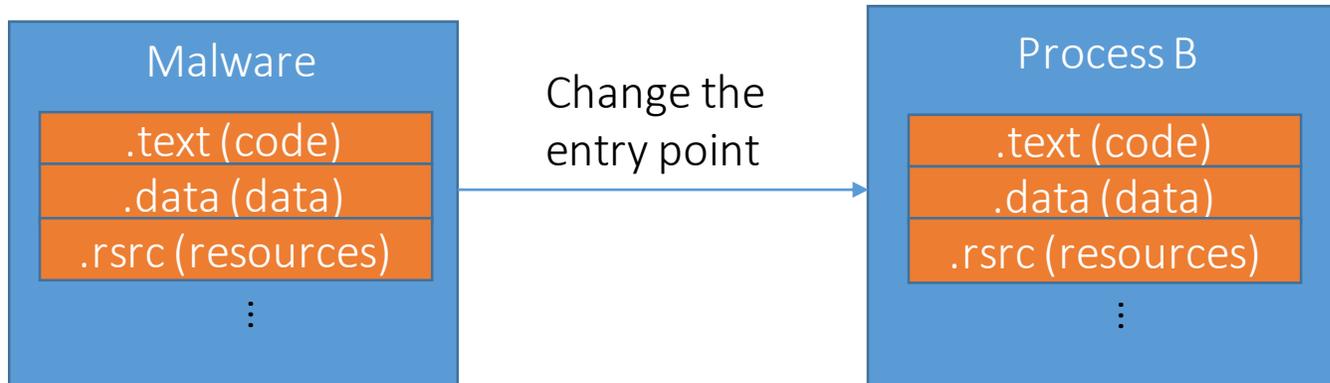
What is Process Hollowing / PE Reflective Injection?

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



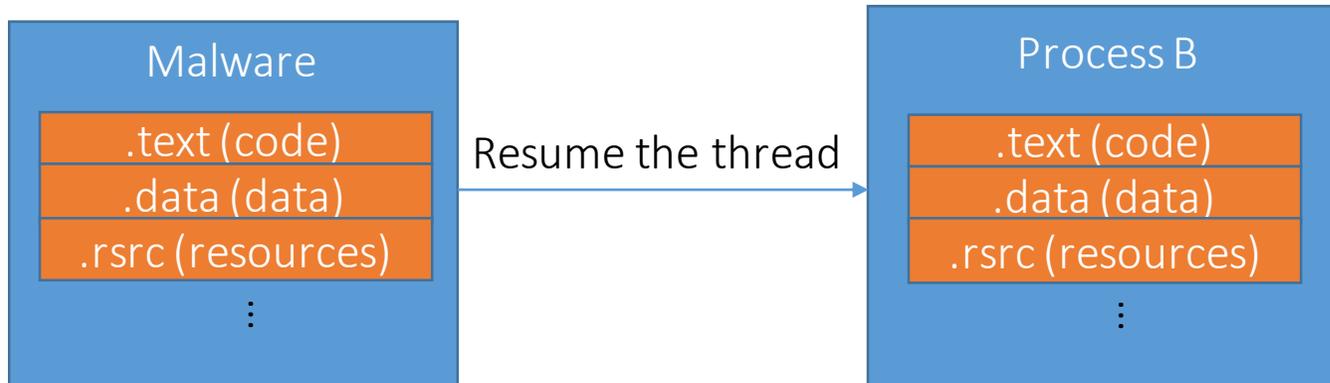
What is Process Hollowing / PE Reflective Injection?

- 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?

- Finally, it executes 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.

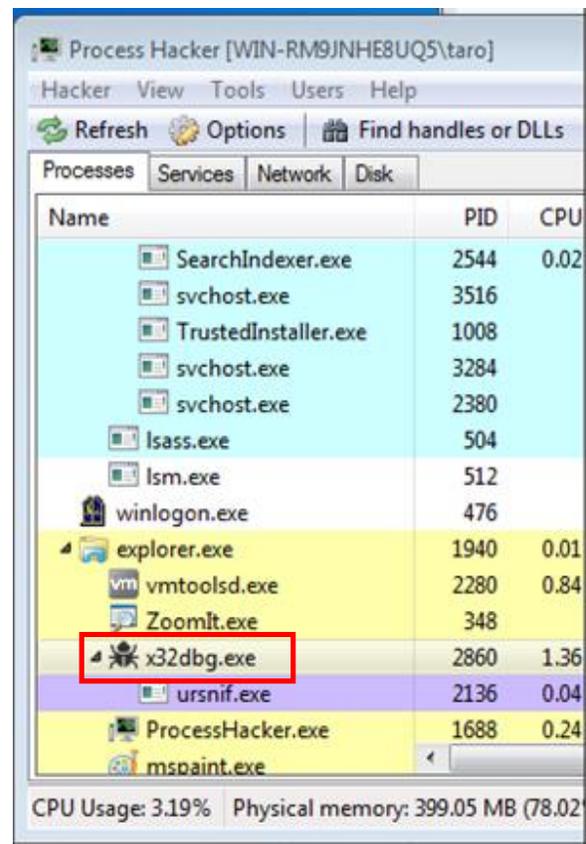
Exercise 3 (2)

- 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

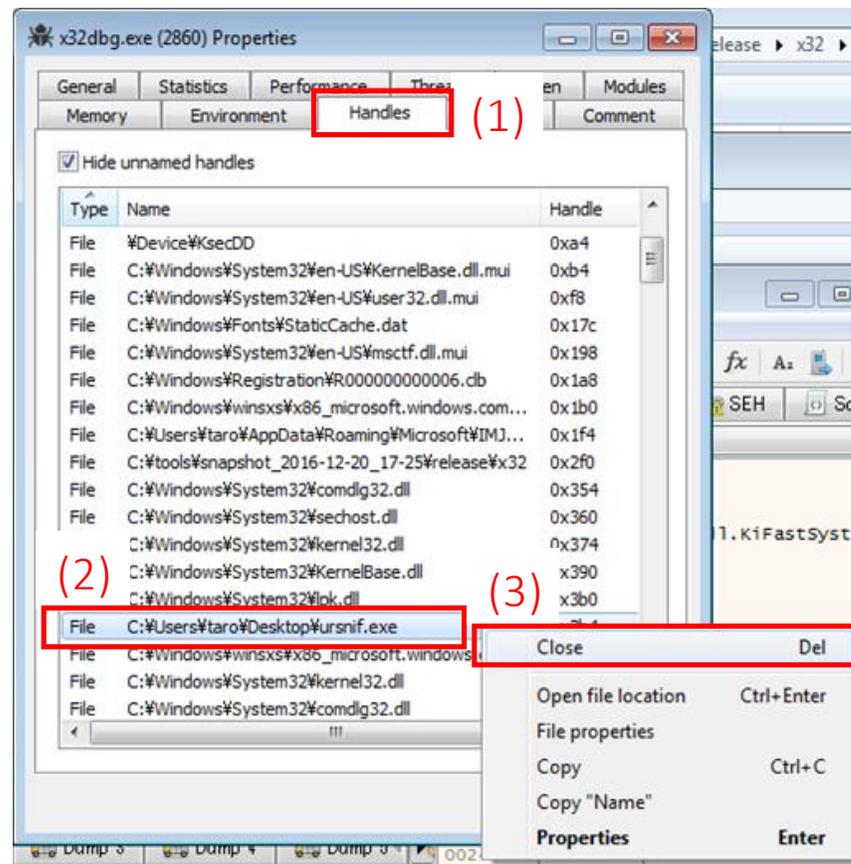
Exercise 3 (4)

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

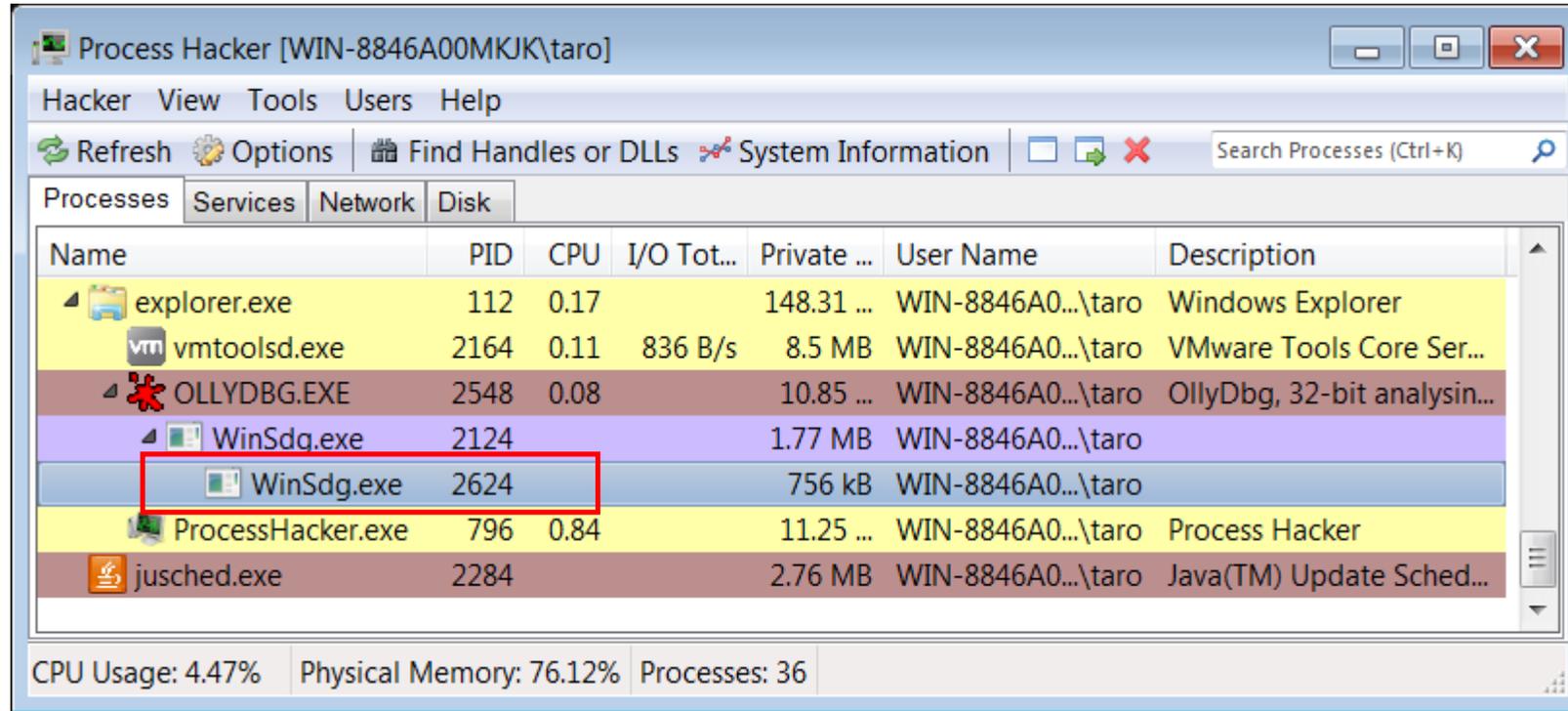
```

> 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 (8)

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



Exerci

WinSdg.exe (2624) Properties

General Statistics Performance Threads Token Modules
 Memory GPU Comment

Strings... Refresh

Name	Address	Size	Protec...
Free	0x0	64 kB	NA
Private (Commit)	0x10000	128 kB	RW
Private (Commit)	0x30000	8 kB	RW
Free	0x32000	56 kB	NA
apisetschema.dll: I...	0x40000	4 kB	R
Free	0x41000	60 kB	NA
Private (Reserve)	0x50000	228 kB	
Private (Commit)	0x89000	12 kB	RW+G
Thread 1728 Stack: ...	0x8c000	16 kB	RW
Private (Reserve)			
Private (Commit)			
Thread 1728 32-bit .			
Mapped (Commit)			
Free			
Mapped (Commit)			
Free			
Private (Commit)			
Free			
Private (Commit)	0x400000	408 kB	RWX
Free	0x466000	1.88 GB	NA

Close

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

The screenshot shows the WinSdg.exe memory editor interface. The main window displays a hex dump of memory with corresponding ASCII characters. A 'Goto Offset' dialog box is open, prompting the user to enter an offset. The offset '0x10e7' is entered in the text field. The 'OK' button is highlighted with a red box. Below the dialog box, the 'Go To...' button in the main window is also highlighted with a red box. The main window has tabs for 'General', 'Strings...', and 'Memory'. The 'Memory' tab is active, showing a list of memory regions with their names, addresses, sizes, and permissions.

Name	Address	Size	Permissions
Free	0x1a0000	4 kB	R
Free	0x1a1000	60 kB	NA
Private (Commit)	0x1b0000	4 kB	RW
Free	0x1b1000	2.31 MB	NA
Private (Commit)	0x400000	408 kB	RWX
Free	0x466000	1.88 GB	NA

2. input "0x10e7" and click "OK".

1. Click "Go To..." button.

Exerci

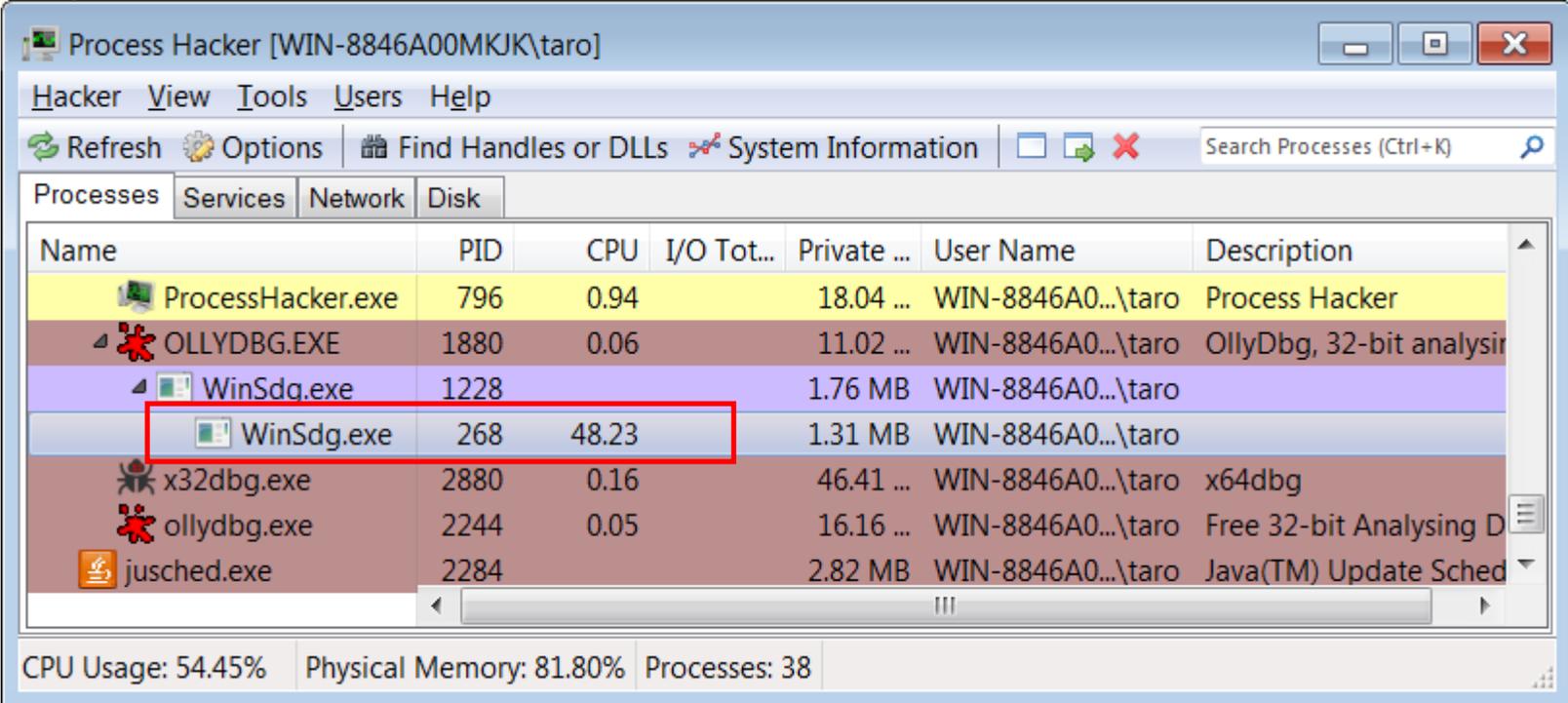
00001060 50 56 8b 35 90 71 40 00 ff d6 39 5d fc 74 68 ff e3 5 a8 91 th
 00001070 75 fc e8 b7 00 00 00
 00001080 50 ff 75 fc 8d 45 f4
 00001090 75 f8 ff d6 85 c0 74
 000010a0 a5 40 00 57 ff d6 85
 000010b0 ff d6 85 c0 75 18 68
 000010c0 75 0c 68 44 a6 40 00
 000010d0 43 57 e8 6c 00 00 00 ff 75 f8 ff 15 88 71 40 00 CW.1....u....q@.
 000010e0 5f 5e 8b c3 5b c9 c3 56 33 f6 56 68 00 00 40 00 _^..[..V3.Vh..@.
 000010f0 56 ff 15 90 70 40 00 a3 08 98 40 00 3b c6 74 26 V...p@....@.;.ts
 00001100 56 ff 15 8c 70 40 00 a3 28 98 40 00 ff 15 98 70 V...p@..(.@....p
 00001110 40 00 50 e8 ce 04 00 00 ff 35 08 98 40 00 8b f0 @.P.....5..@...
 00001120 ff 15 94 70 40 00 56 ff 15 9c 70 40 00 5e ff 74 ...p@.V...p@.^.t
 00001130 24 04 6a 00 ff 35 08 98 40 00 ff 15 88 70 40 00 \$.j..5..@....p@.
 00001140 c2 04 00 ff 74 24 04 6a 00 ff 35 08 98 40 00 fft\$.j..5..@..
 00001150 15 50 70 40 00 c2 04 00 55 8b ec 83 ec 14 a1 38 .Pp@....U.....8
 00001160 98 40 00 53 56 33 db 57 89 5d f4 89 45 fc 89 45 .@.SV3.W.]..E..E
 00001170 f0 be 00 99 40 00 33 ff 8d 45 f8 50 8d 45 f0 e8@.3..E.P.E..
 00001180 94 0d 00 00 3b c3 74 2c ff 75 f8 6a 0d 58 e8 1a;.t,.u.j.X..
 00001190 0c 00 00 3b c3 74 0d 89 06 83 c6 04 47 83 ff 03 ...;.t.....G...
 000011a0 72 d6 eb 10 ff 75 f8 53 ff 35 08 98 40 00 ff 15 r....u.S.5..@...
 000011b0 50 70 40 00 83 ff 03 75 7f 68 04 a5 40 00 ff 35 Fp@....u.h..@..5
 000011c0 04 99 40 00 ff 15 74 70 40 00 53 68 c6 a1 40 00 ..@...tp@.Sh..@.
 000011d0 8d 45 fc e8 b7 28 00 00 a3 fc 98 40 00 3b c3 74 F / @ . +
 000011e0
 000011f0

2. Click "Write" button. 3. Click "Close" button.

Mapped (Commit)	0x1a0000	4 kB	R
Free	0x1a1000	60 kB	NA
Private (Commit)	0x1b0000	4 kB	RW
Free	0x1b1000	2.31 MB	NA
Private (Commit)	0x400000	408 kB	RWX
Free	0x466000	1.88 GB	NA

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.



Process Hacker [WIN-8846A00MKJK\taro]

Hacker View Tools Users Help

Refresh Options Find Handles or DLLs System Information Search Processes (Ctrl+K)

Processes Services Network Disk

Name	PID	CPU	I/O Tot...	Private ...	User Name	Description
ProcessHacker.exe	796	0.94		18.04 ...	WIN-8846A0...\taro	Process Hacker
OLLYDBG.EXE	1880	0.06		11.02 ...	WIN-8846A0...\taro	OllyDbg, 32-bit analysir
WinSdg.exe	1228			1.76 MB	WIN-8846A0...\taro	
WinSdg.exe	268	48.23		1.31 MB	WIN-8846A0...\taro	
x32dbg.exe	2880	0.16		46.41 ...	WIN-8846A0...\taro	x64dbg
ollydbg.exe	2244	0.05		16.16 ...	WIN-8846A0...\taro	Free 32-bit Analysing D
jusched.exe	2284			2.82 MB	WIN-8846A0...\taro	Java(TM) Update Sched

CPU Usage: 54.45% Physical Memory: 81.80% Processes: 38

Exercise 3 (13)

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

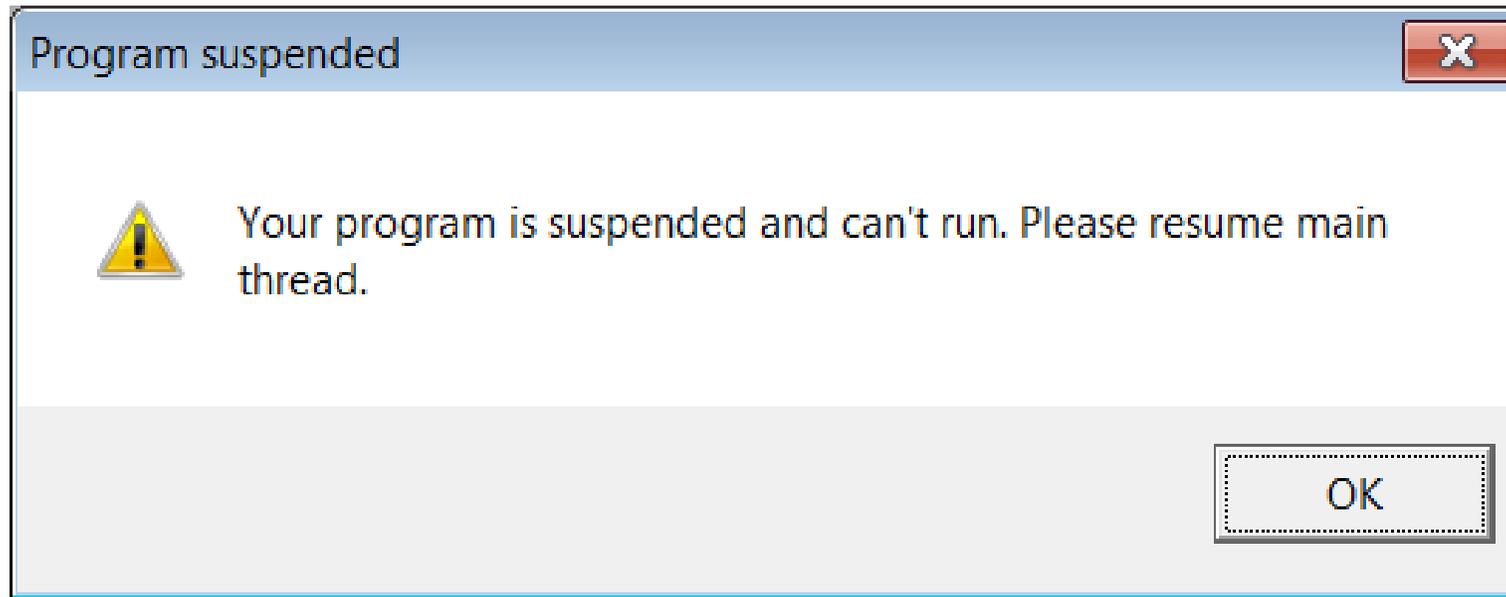
The screenshot shows the 'Attach' dialog box in Process Hacker. The dialog box contains a table with the following data:

PID	Path	Command Line
0000010C	C:\Users\taro\Desktop\20161011_gozi_ursnif\20161011_gozi_ursnif\winSdg.exe	
000004CC	C:\Users\taro\Desktop\20161011_gozi_ursnif\20161011_gozi_ursnif\winSdg.exe	
00000758	C:\Users\taro\Desktop\OllyDbg\odbg110\OLLYDBG.EXE	
000008EC	C:\Program Files (x86)\Common Files\Java\Java update\jusched.exe	

Below the dialog box, the main Process Hacker window shows a list of processes. The PID 268 is highlighted in a red box. A text box with a red arrow pointing to the PID 268 in the process list contains the text: "These have the same process ID (268 is equal to 0x10c in hex)." The status bar at the bottom shows: CPU Usage: 54.45% Physical Memory: 81.80% Processes: 38

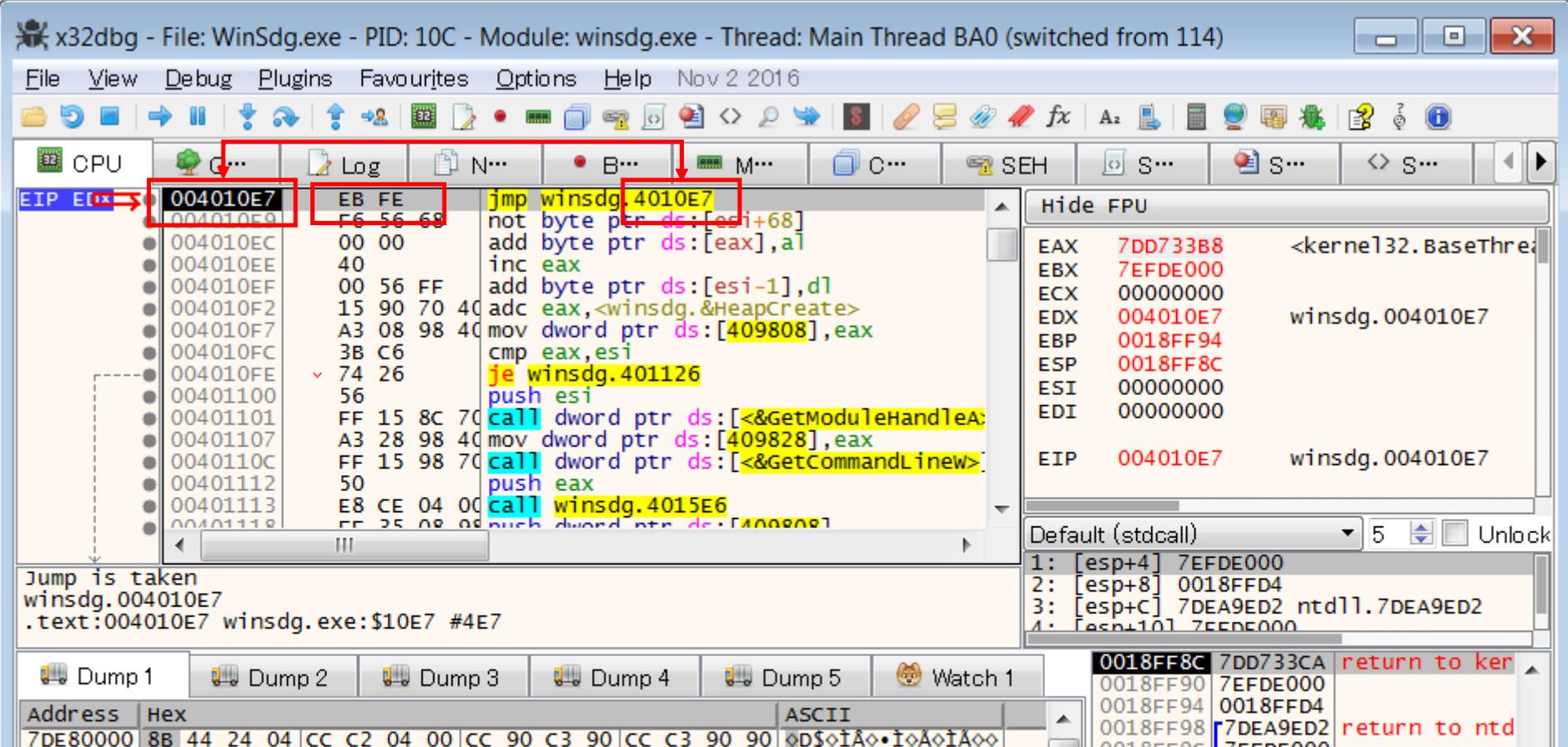
Exercise 3 (14)

- 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”)

The screenshot shows the x32dbg interface with a memory dump window open. The address 004010E7 is selected, and the bytes EB FE are highlighted. A dialog box titled 'Edit code at 004010E7' is open, showing the ASCII and Hex views. The Hex view shows 56 33, which is highlighted with a red box. A red arrow points from the text 'Replace "EB FE" to "56 33"' to the 56 33 in the Hex view.

Jump is taken
winsdg.004010E7
.text:004010E7 winsdg.exe:\$10E7 #4

Address	Hex
004010E7	EB FE
004010E9	F6 56 68
004010EC	00 00
004010EE	40
004010EF	00 56 FF
004010F2	15 90 70 40
004010F7	A3 08 98 40
004010FC	3B C6
004010FE	74 26
00401100	56
00401101	FF 15 8C 70
00401107	A3 28 98 40
0040110C	FF 15 98 70
00401112	50
00401113	E8 CE 04 00
00401118	FF 25 08 00

Replace "EB FE" to "56 33".

Exercise 3 (17)

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

The screenshot shows the x32dbg debugger interface. The main window displays assembly code for the process WinSdg.exe. A red box highlights the instructions at addresses 004010E7 and 004010E8, which are labeled as "Restored instructions". The assembly code includes instructions like `push esi`, `xor esi,esi`, `push esi`, `push winstdg.400000`, `push esi`, `call dword ptr ds:[<&HeapCreate>]`, `mov dword ptr ds:[409808],eax`, `cmp eax,esi`, `je winstdg.401126`, `push esi`, `call dword ptr ds:[<&GetModuleHandleA>]`, `mov dword ptr ds:[409828],eax`, `call dword ptr ds:[<&GetCommandLineA>]`, `push eax`, and `call winstdg.4015E6`.

The CPU registers window on the right shows the following values:

Register	Value	Comment
EAX	7DD733B8	<kernel32.BaseThread
EBX	7EFDE000	
ECX	00000000	
EDX	004010E7	winstdg.004010E7
EBP	0018FF94	
ESP	0018FF8C	
ESI	00000000	
EDI	00000000	
EIP	004010E7	winstdg.004010E7

The stack window shows the following values:

Index	Address	Value
1	[esp+4]	7EFDE000
2	[esp+8]	0018FFD4
3	[esp+C]	7DEA9ED2 ntdll.7DEA9ED2
4	[esp+10]	7EFDE000

The bottom window shows a memory dump with the following data:

Address	Hex	ASCII
7DE80000	8B 44 24 04 CC C2 04 00 CC 90 C3 90 CC C3 90 90	oD\$oiA•i•AoiA•
7DE80010	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	oooooooooooooooo
7DE80020	8B 4C 24 04 F6 41 04 06 74 05 E8 81 1D 01 00 B8	oL\$•oA•o•e•o••
7DE80030	01 00 00 00 C2 10 00 90 8D 84 24 DC 02 00 00 64	o•••&o•••o\$ii•o••

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.

Exec

x32dbg - File: WinSdg.exe - PID: 10C - Module: winsdg.exe - Thread: Main Thread BA0

File View Debug Plugins Favourites Options Help Nov 2 2016

CPU G... Lo M... C... SEH S... S...

Sandbox Evasion Code

```

00401683 8D 44 1ea eax,dword ptr ss:[esp+2C]
00401687 50      push eax
00401688 C7 44 44 mov dword ptr ss:[esp+30],14
00401690 FF 15 3a call dword ptr ds:[<&GetCursorInfo>]
00401696 8B 44 44 mov eax,dword ptr ss:[esp+3C]
0040169A 2B 06 sub eax,esi
0040169C 2B 07 sub eax,edi
0040169E 03 44 add eax,dword ptr ss:[esp+38]
004016A2 50      push eax
004016A3 E8 80 call winsdg.403894
004016A8 8B 7C mov edi,dword ptr ss:[esp+38]
004016AC 8B 74 mov esi,dword ptr ss:[esp+3C]
004016B0 83 F8 cmp eax,C
004016B3 74 CE je winsdg.401683
004016B5 3B C0 cmp eax,ebx
004016B7 0F 85 jne winsdg.401A9B
004016BD A1 18 mov eax,dword ptr ds:[409818]
004016C2 53      push ebx
004016C3 E8 47 call winsdg.402A0F
004016C8 85 C0 test eax,eax
  
```

EIP → 004016A3

Hide FPU

EAX	
EBX	
ECX	
EDX	
EBP	
ESP	
ESI	00000179
EDI	
EIP	004016A3 WINSdg.004016A3
EFLAGS	00000247
ZF	1
PF	1
AF	0
OF	0
SF	0
DF	0
CF	1
TF	0
IF	1

Default (stdcall) 5 Unlocked

1:	[esp]	00000000
2:	[esp+4]	00000000
3:	[esp+8]	00000000
4:	[esp+C]	7EFD5000

winsdg.00403894

.text:004016A3 winsdg.exe:\$16A3 #AA3

Address	Hex	ASCII
7DE80000	8B 44 24 04 CC C2 04 00 CC 90 C3 90 CC C3 90 90	0D\$0iA0.i0A0iA
7DE80010	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	0000000000000000
7DE80020	8B 4C 24 04 F6 41 04 06 74 05 E8 81 1D 01 00 B8	0L\$00A00t0è000
7DE80030	01 00 00 00 C2 10 00 90 8D 84 24 DC 02 00 00 64	0...A0...00\$ü0
7DE80040	8B 0D 00 00 00 00 BA 20 00 E8 7D 89 08 89 50 04	00...0 .è}000

Command: Default

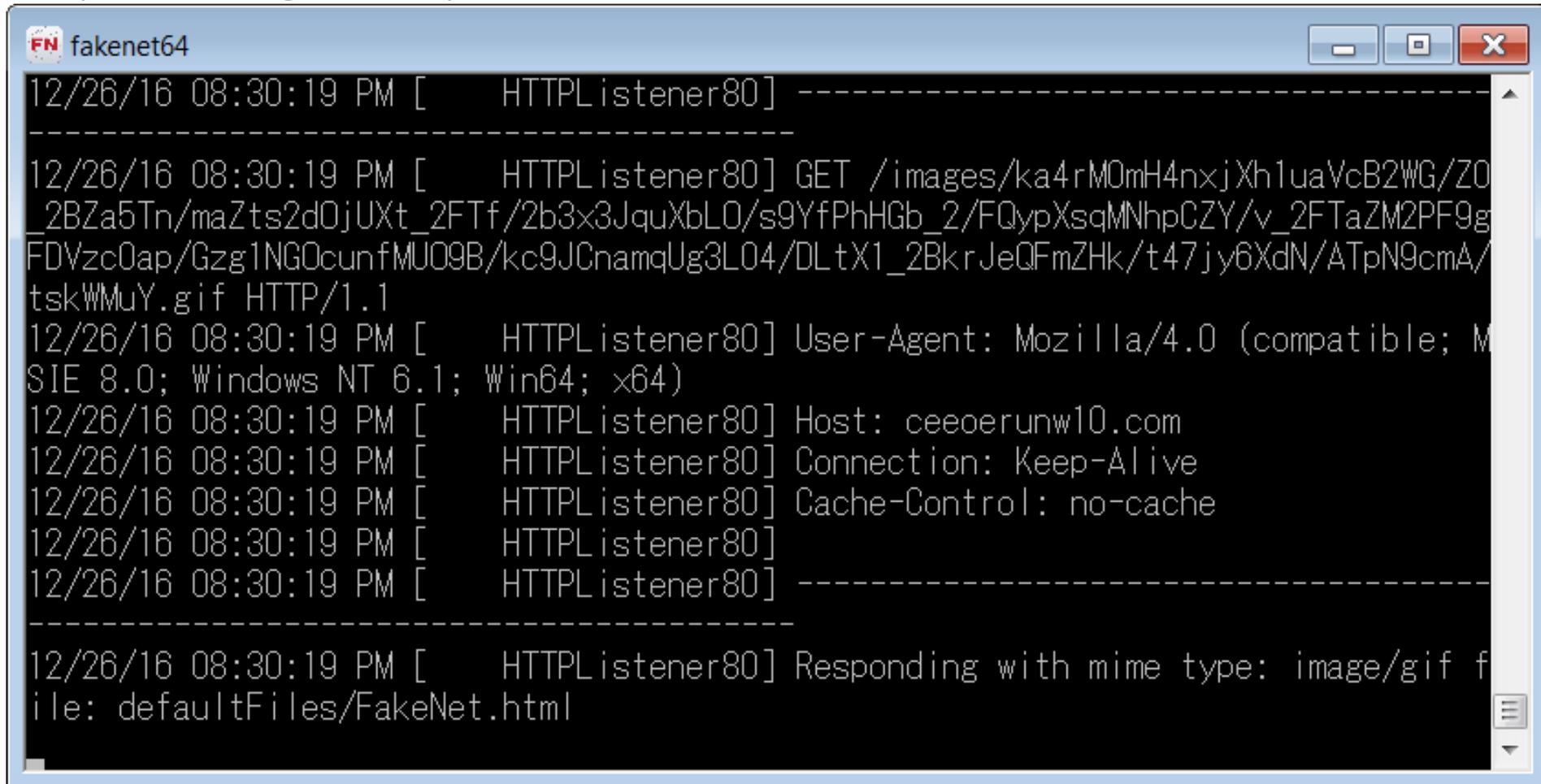
Paused INT3 breakpoint at winsdg00401696 (00401696) Time Wasted Debugging: 0:00:34:41

Difference between last and current mouse positions

.bss section decode routine

Exercise 3 (21)

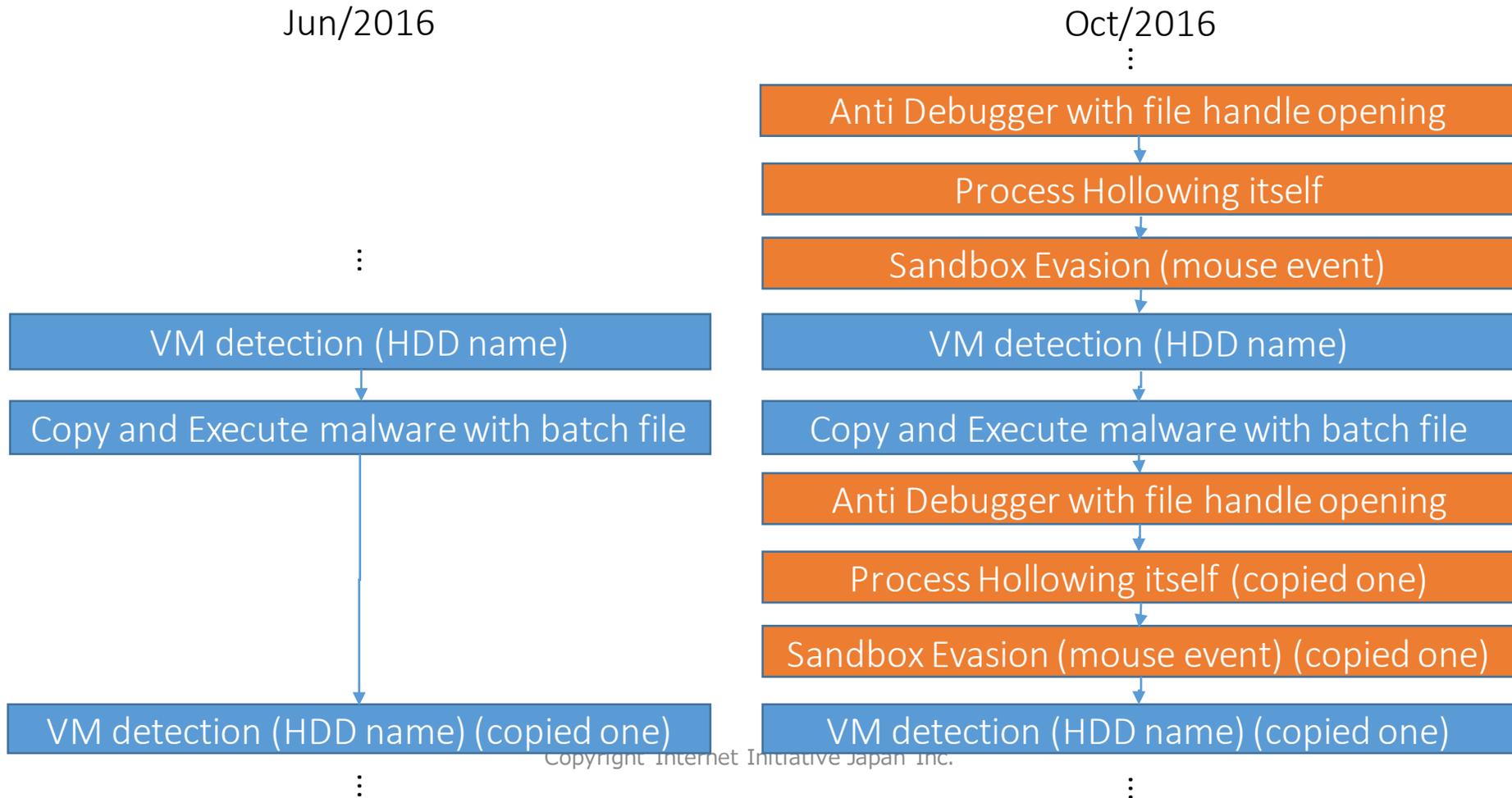
- Then you can get suspicious HTTP communications.



```
FN fakenet64
12/26/16 08:30:19 PM [ HTTPListener80 ] -----
12/26/16 08:30:19 PM [ HTTPListener80 ] GET /images/ka4rM0mH4nxjXh1uaVcB2WG/Z0
_2BZa5Tn/maZts2d0jUXt_2FTf/2b3x3JquXbL0/s9YfPhHGb_2/FQypXsqMNhpCZY/v_2FTaZM2PF9g
FDVzc0ap/Gzg1NGOcunfMU09B/kc9JCnamqUg3L04/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/>

To be continued...