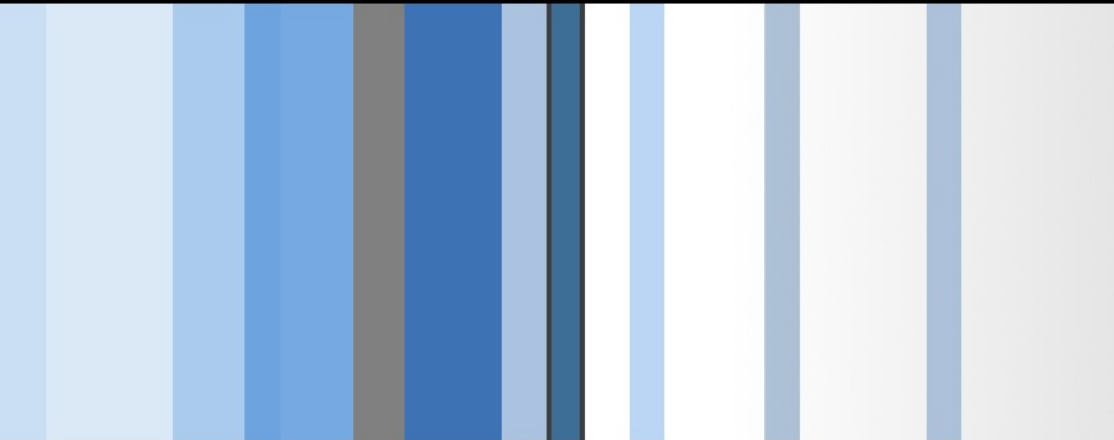


CSC 471 Modern Malware Analysis

Anti-Debugging Techniques (2): Dynamic Anti-Debugging

Si Chen (schen@wcupa.edu)



Anti-Debugging

- Malware authors have always looked for new techniques to **stay invisible**. This includes, of course, being invisible on the compromised machine, but it is even more important to hide malicious indicators and behavior during analysis.
- **Debugging** is the essential part of malware analysis. Every time we need to drill down into malware behavior, restore encryption methods or examine communication protocols, we use debuggers.
- To make the post-detection analysis more difficult, threat actors use various anti-analysis techniques, one of the more common ones is **Anti-Debugging**.



Static Anti-Debugging VS. Dynamic Anti-Debugging

	Static	Dynamic
Difficulty Level	Easy, Medium	Hard
Key idea	Use System Information	Reverse and exploit Debugger
Target	Detect Debugger	Hide it's own code and data
Time point	When debugging started	While debugger are running
Defend Method(s)	API hook, debugger plugin	API hook, Debugger Plugin, Other tools
Example(s)	PEB, TEB, Native API, TLS	SHE, Break Points (INT3), Timing Check

Dynamic Anti-Debugging

Dynamic Anti-Debugging techniques are trying to interfere with the debugger, so it cannot debug the binary program correctly (to hide its Original Entry Point (**OEP**)).

Dynamic Anti-Debugging -- Exception

- **Structured exception handling (SEH)** is a Microsoft extension to C to handle certain exceptional code situations, such as hardware faults, gracefully.

Microsoft-specific:

Grammar

`try-except-statement :`

`__try compound-statement __except (expression) compound-statement`

`try-finally-statement :`

`__try compound-statement __finally compound-statement`

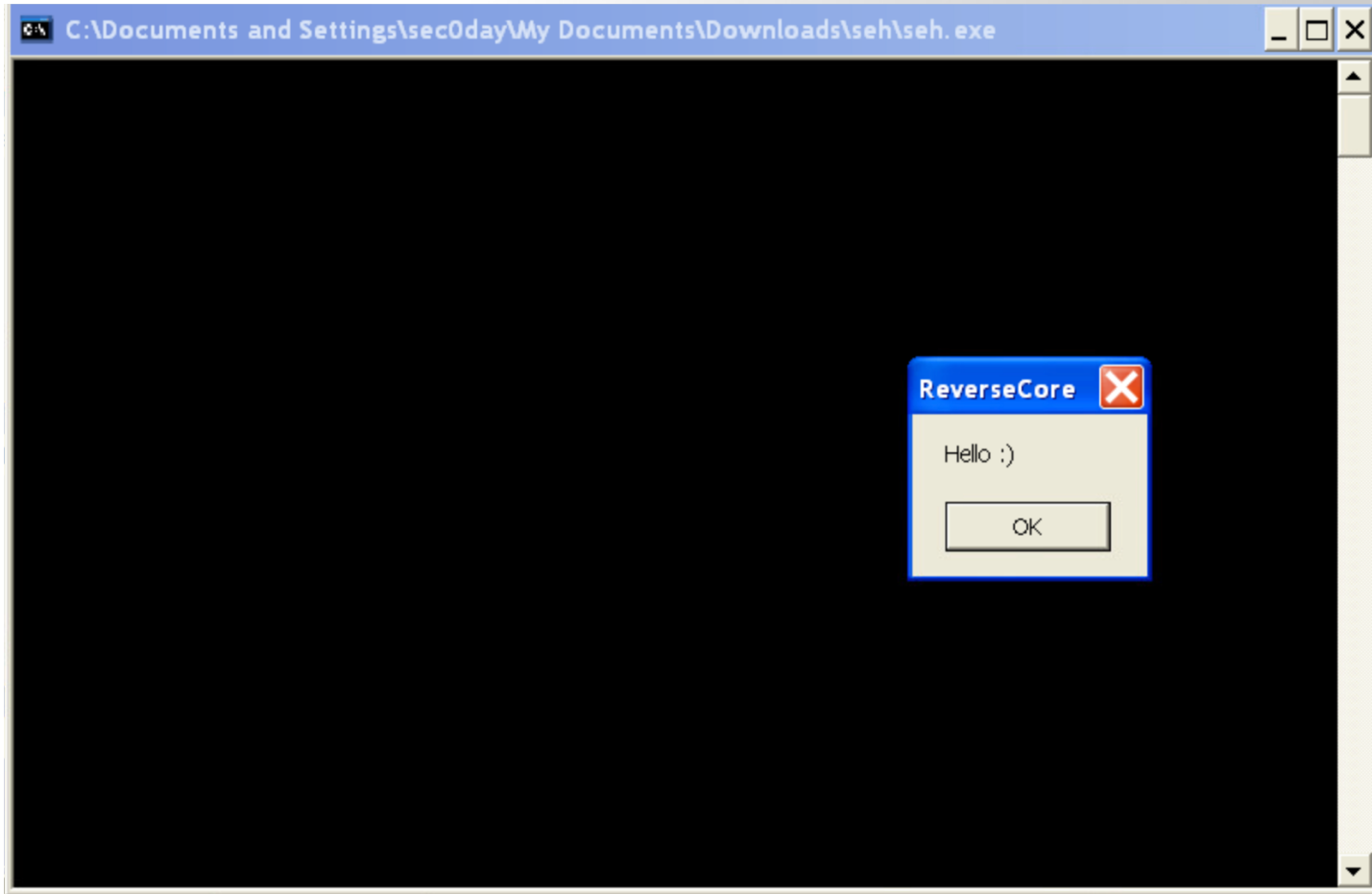
```
void fault_nocatch_fin(void) {
    __try
    {
        __try
        {
            sehR = *sehD;
        }
        __finally
        {
            TEST_STEP(17)
        }
    }
    __except (EvalFilter_dll(EXCEPTION_CONTINUE_SEARCH))
    {
        TEST_STEP(41)
    }
}
```

Although Windows and Microsoft C++ support SEH, we recommend that you use ISO-standard C++ exception handling. It makes your code more portable and flexible. -- MSDN

Typical Exceptions in Windows System

```
#define EXCEPTION_ACCESS_VIOLATION          0xC0000005u
#define EXCEPTION_DATATYPE_MISALIGNMENT     0x80000002u
#define EXCEPTION_BREAKPOINT                0x80000003u
#define EXCEPTION_SINGLE_STEP               0x80000004u
#define EXCEPTION_ARRAY_BOUNDS_EXCEEDED     0xC000008Cu
#define EXCEPTION_FLT_DENORMAL_OPERAND     0xC000008Du
#define EXCEPTION_FLT_DIVIDE_BY_ZERO        0xC000008Eu
#define EXCEPTION_FLT_INEXACT_RESULT        0xC000008Fu
#define EXCEPTION_FLT_INVALID_OPERATION     0xC0000090u
#define EXCEPTION_FLT_OVERFLOW              0xC0000091u
#define EXCEPTION_FLT_STACK_CHECK           0xC0000092u
#define EXCEPTION_FLT_UNDERFLOW             0xC0000093u
#define EXCEPTION_INT_DIVIDE_BY_ZERO        0xC0000094u
#define EXCEPTION_INT_OVERFLOW              0xC0000095u
#define EXCEPTION_PRIV_INSTRUCTION          0xC0000096u
#define EXCEPTION_IN_PAGE_ERROR             0xC0000006u
#define EXCEPTION_ILLEGAL_INSTRUCTION       0xC000001Du
#define EXCEPTION_NONCONTINUABLE_EXCEPTION  0xC0000025u
#define EXCEPTION_STACK_OVERFLOW            0xC00000FDu
#define EXCEPTION_INVALID_DISPOSITION       0xC0000026u
#define EXCEPTION_GUARD_PAGE                0x80000001u
#define EXCEPTION_INVALID_HANDLE            0xC0000008u
```

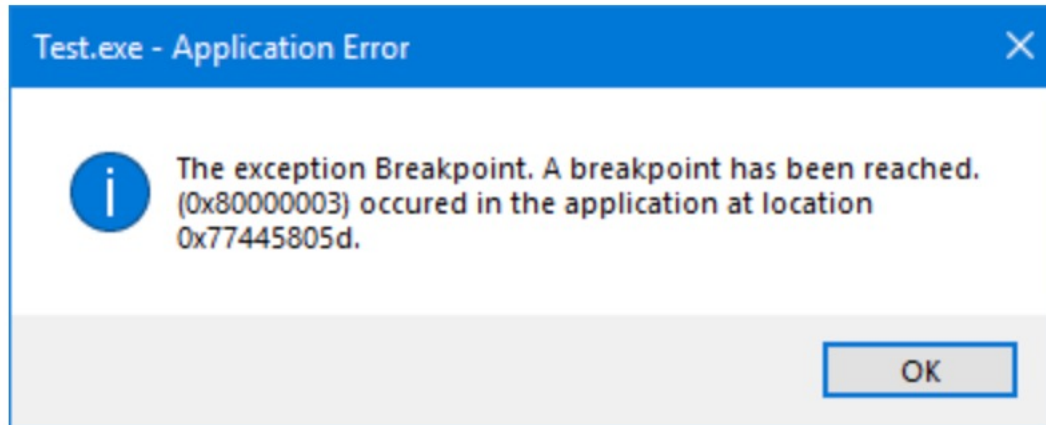
SEH Example – SEH.exe



EXCEPTION_BREAKPOINT

```
#define EXCEPTION_BREAKPOINT
```

```
0x80000003u
```



Program will automatically call the registered SEH. If the program is running under the Debug mode, it will stop the program and give the control back to the debugger.

SEH Example – DynAD_SEH.exe

OllyDbg - DynAD_SEH.exe - [CPU - main thread, module DynAD_SE]

File View Debug Plugins Options Window Help

LEMTW H C / K B R ... S

00401000 \$ 55 PUSH EBP
 00401001 . 8BEC MOV EBP,ESP
 00401003 . 53 PUSH EBX
 00401004 . 68 A0994000 PUSH DynAD_SE.004099A0
 00401009 . E8 69000000 CALL DynAD_SE.00401077
 0040100E . 83C4 04 ADD ESP,4
 00401011 . 68 2C104000 PUSH DynAD_SE.0040102C
 00401016 . 64:FF35 00000000 PUSH DWORD PTR FS:[0]
 0040101D . 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
 00401024 . CC INT3
 00401025 . B8 FFFFFFFF MOV EAX,-1
 0040102A . FFE0 JMP EAX
 0040102C \$ 36:8B4424 0C MOV EAX,DWORD PTR SS:[ESP+C]
 00401031 . BB 40104000 MOV EBX,DynAD_SE.00401040
 00401036 . 3E:8998 B8000000 MOV DWORD PTR DS:[EAX+B8],EBX
 0040103D . 33C0 XOR EAX,EAX
 0040103F . C3 RETN
 00401040 . 64:8F05 00000000 POP DWORD PTR FS:[0]
 00401047 . 83C4 04 ADD ESP,4
 0040104A . 68 B4994000 PUSH DynAD_SE.004099B4
 0040104F . E8 23000000 CALL DynAD_SE.00401077
 00401054 . 8B4424 0C MOV EAX,DWORD PTR SS:[ESP+C]
 0040105B . BB 40104000 MOV EBX,DynAD_SE.00401040
 00401060 . 3E:8998 B8000000 MOV DWORD PTR DS:[EAX+B8],EBX
 00401067 . 33C0 XOR EAX,EAX
 00401069 . C3 RETN

ASCII "SEH : Br
 SE handler inst
 Structured exce
 ASCII " => No

Registers (MMX)

00000000
 0012FFB0
 7C90E514 ntdll.KiFastSystemCallRet
 7FFD7000
 0012FFC4
 0012FFF0
 FFFFFFFF
 7C910228 ntdll.7C910228
 004012BE DynAD_SE.<ModuleEntryPoint>
 ES 0023 32bit 0(FFFFFFFF)
 CS 001B 32bit 0(FFFFFFFF)
 SS 0023 32bit 0(FFFFFFFF)
 DS 0023 32bit 0(FFFFFFFF)
 FS 003B 32bit 7FFDF000(FFF)
 GS 0000 NULL
 LastErr ERROR_SUCCESS (00000000)
 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
 0105 0104 0073 0064
 006F 005C 0030 0031
 0069 002E 0067 0062
 0000 0000 0000 0000
 0000 0000 0000 0000
 0012FFC4 7C81776F RETURN to kernel32.7C81776F
 0012FFC8 7C910228 ntdll.7C910228
 0012FFCC FFFFFFFF
 0012FFD0 7FFD7000
 0012FFD4 8054B6ED
 0012FFD8 0012FFC8
 0012FFDC 897A4520
 0012FFE0 FFFFFFFF End of SEH chain
 0012FFE4 7C839A90 SE handler
 0012FFEB 7C817778 kernel32.7C817778

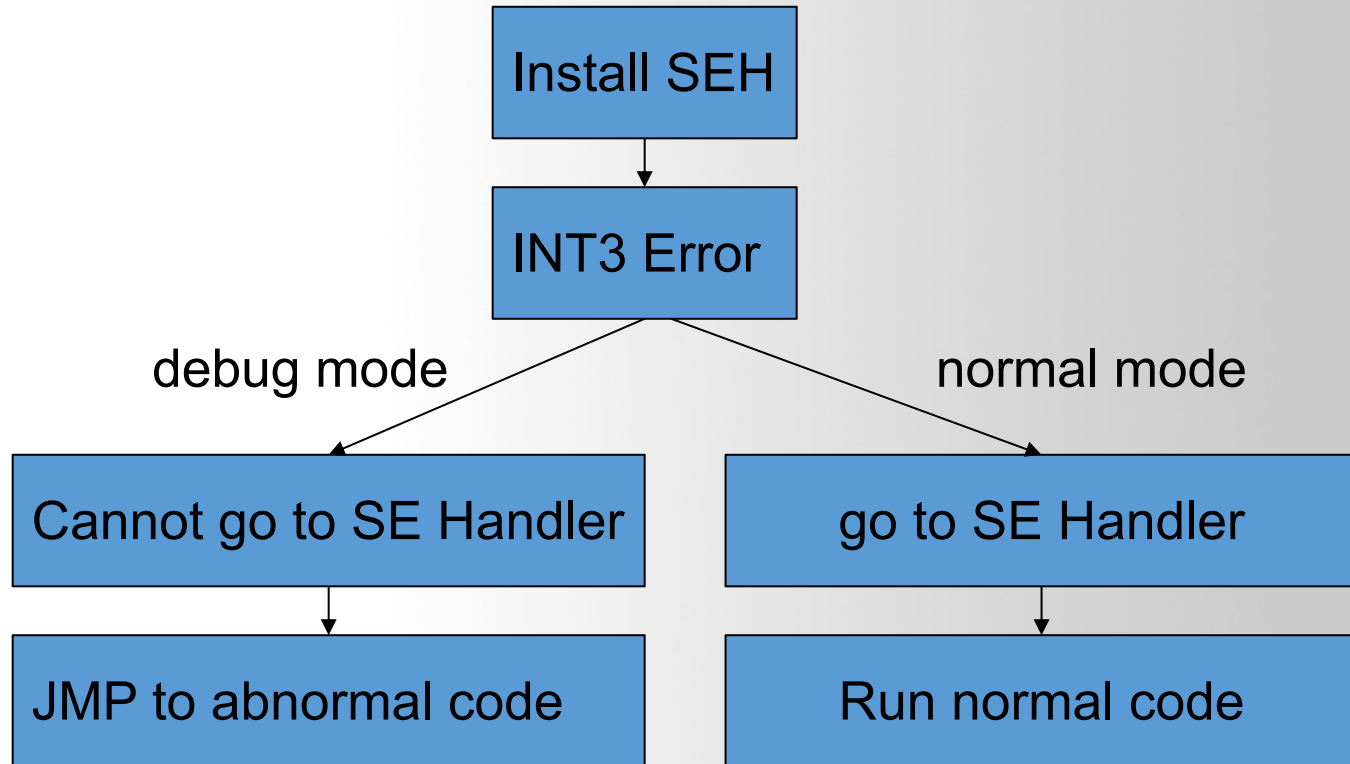
Local call from 00401060

Address	Hex dump	ASCII
0040B000	01 00 00 00 4E E6 40 BB	0...Np01
0040B008	B1 19 BF 44 00 00 00 00	1D....
0040B010	80 CC 40 00 00 00 00 00	CFe....
0040B018	80 CC 40 00 01 01 00 00	CFe.00..
0040B020	00 00 00 00 00 00 00 00
0040B028	00 10 00 00 00 00 00 00
0040B030	00 00 00 00 00 00 00 00
0040B038	00 00 00 00 02 00 00 000...
0040B040	01 00 00 00 00 00 00 00	0.....

Analysing DynAD_SE: 174 heuristical procedures, 119 calls to known, 93 calls to guessed functions

Paused

SEH Example – DynAD_SEH.exe



SEH Example – DynAD_SEH.exe

Install SEH

00401011	. 68 2C104000	PUSH DynAD_SE.0040102C	SE handler installation
00401016	. 64:FF35 00000000	PUSH DWORD PTR FS:[0]	
0040101D	. 64:8925 00000000	MOV DWORD PTR FS:[0],ESP	

00401024 . CC INT3

INT3 Error

debug mode

normal mode

Cannot go to SE Handler

go to SE Handler

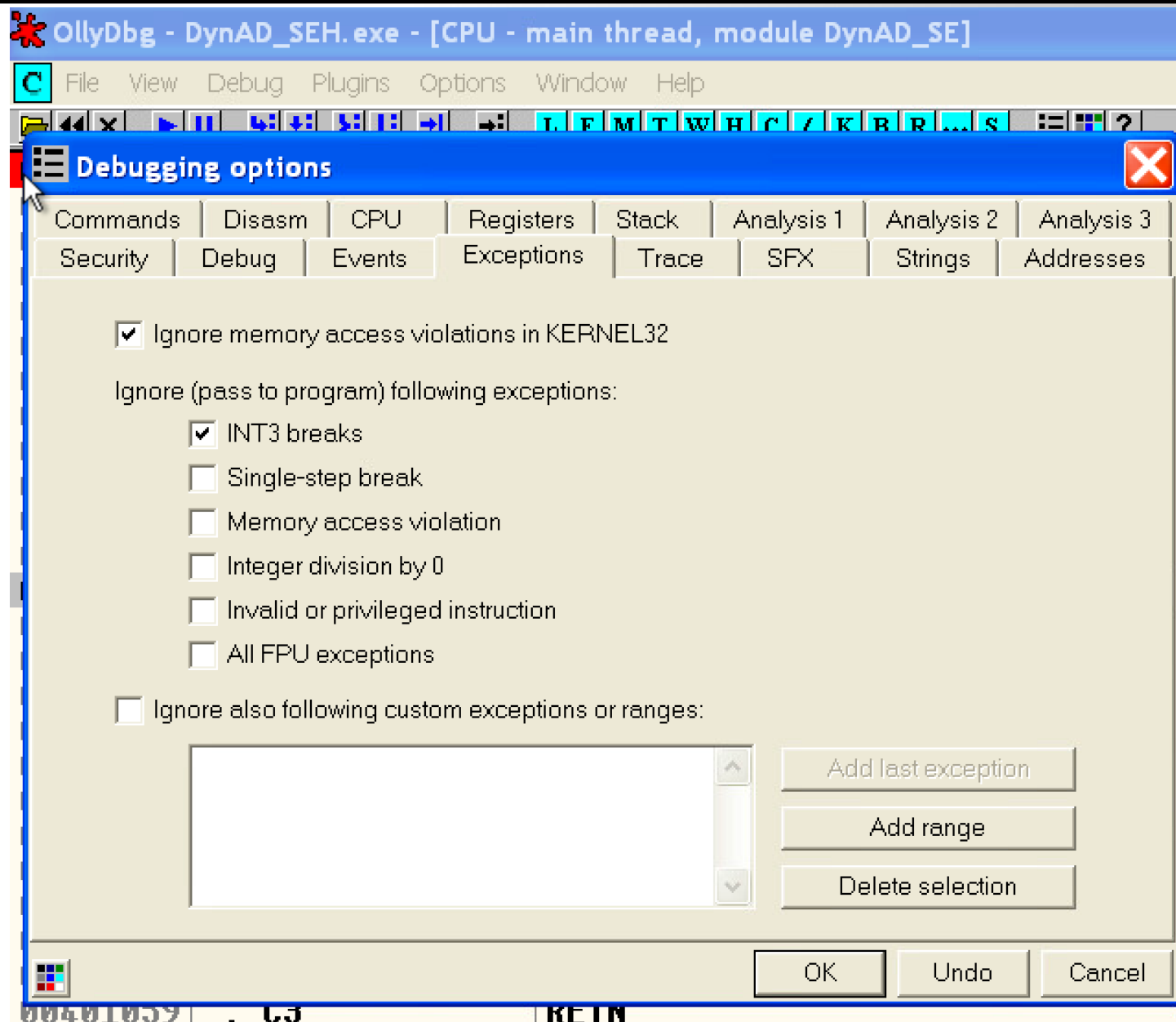
JMP to abnormal code

Run normal code

00401025	. B8 FFFFFFFF	MOV EAX,-1
0040102A	. FFE0	JMP EAX

0040102C	* 36:8B4424 0C	MOV EAX,DWORD PTR SS:[ESP+C]	Structured exception handler
00401031	. BB 40104000	MOV EBX,DynAD_SE.00401040	
00401036	. 3E:8998 B80000	MOV DWORD PTR DS:[EAX*B8],EBX	
0040103D	. 33C0	XOR EAX,EAX	
0040103F	. C3	RETN	

How to bypass INT3 breaks



***Get 1st Time (T1)**

A bunch of code
-loop
-garbage code
-encryption/decryption

***Get 2nd Time (T2)**

**If $T2 - T1 > 1$ (sec)
Call ExitProcess()**

Aka **Anti-Emulating**

How to calculate time intervals

- Counter based method
 - RDTSC (Read Time Stamp Counter)
 - kernel32!QueryPerformanceCounter()/ntdll!NtQueryPerformanceCounter()
 - kernel32!GetTickCount()
- Time based method
 - timeGetTime()
 - _ftime()

Use CPU counter
Or system time

Timing Check Example – DynAD_RDTSC.exe

The **Time Stamp Counter (TSC)** is a 64-bit [register](#) present on all [x86](#) processors since the [Pentium](#). It counts the number of CPU [cycles](#) since its reset.

The instruction **RDTSC** returns the TSC in **EDX:EAX**. In [x86-64](#) mode, RDTSC also clears the upper 32 bits of [RAX](#) and [RDX](#). Its [opcode](#) is 0F 31

Timing Check Example – DynAD_RDTSC.exe

OllyDbg - DynAD_RDTSC.exe - [CPU - main thread, module DynAD_RD]

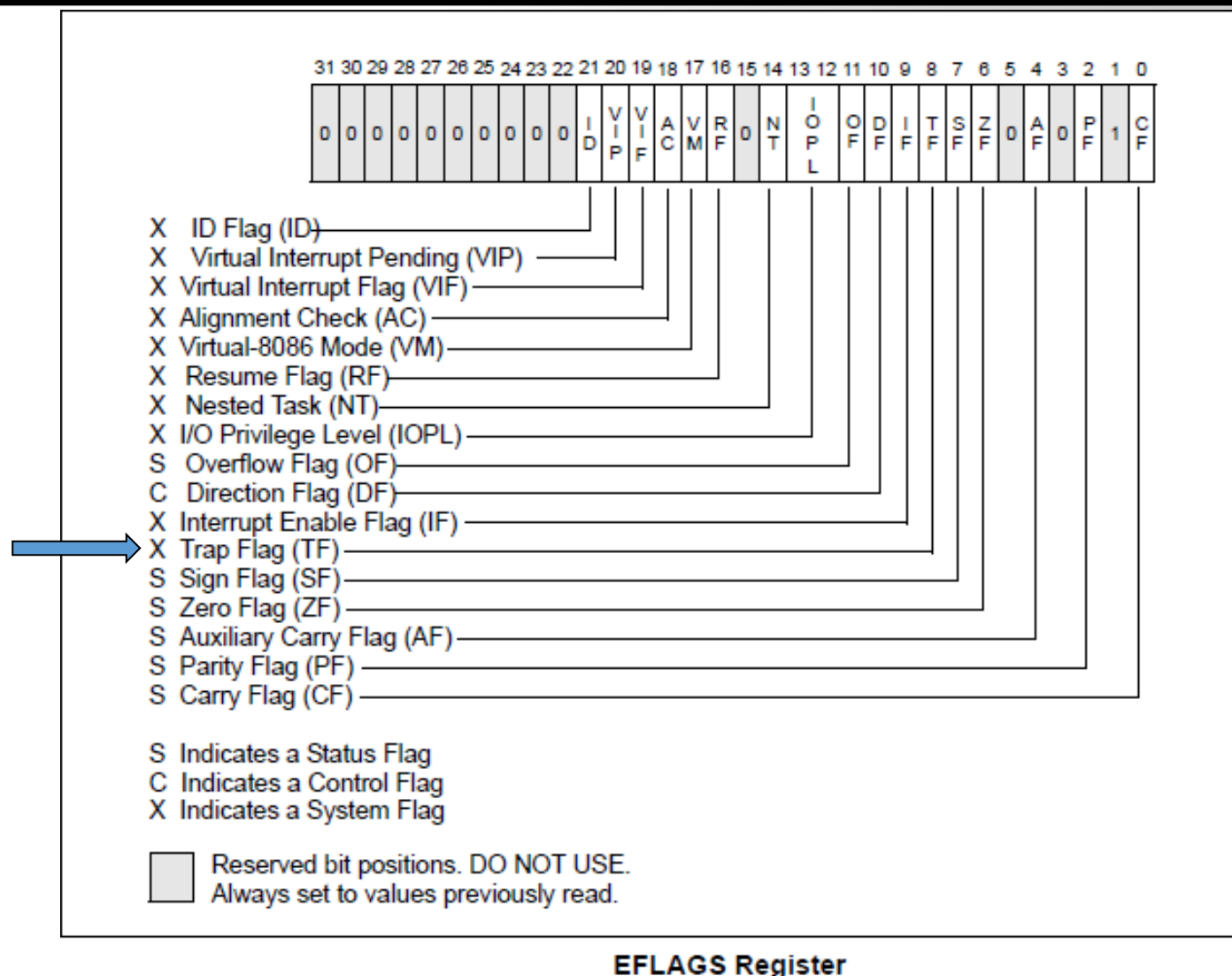
File View Debug Plugins Options Window Help

00401000 \$ 55 PUSH EBP
00401001 . 8BEC MOV EBP,ESP
00401003 . 51 PUSH ECX
00401004 . 53 PUSH EBX
00401005 . 56 PUSH ESI
00401006 . 57 PUSH EDI
00401007 . 68 A0994000 PUSH DynAD_RD.004099A0
0040100C . C745 FC 000000 MOV DWORD PTR SS:[EBP-4],0
00401013 . E8 5E000000 CALL DynAD_RD.00401076
00401018 . 83C4 04 ADD ESP,4
0040101B . 60 PUSHAD
0040101C . 9F31 RDTSC
0040101E . 52 PUSH EDX
0040101F . 50 PUSH EAX
00401020 . 33C0 XOR EAX,EAX
00401022 . B9 E8030000 MOV ECX,3E8
00401027 > 40 [INC EAX
00401028 . ^E2 FD [LOOPD SHORT DynAD_RD.00401027
0040102A . 0F31 RDTSC
0040102C . 5E POP ESI
0040102D . 5F POP EDI
0040102E . 3BD7 CMP EDX,EDI
00401030 . 77 0C JA SHORT DynAD_RD.0040103E
00401032 . 2BC6 SUB EAX,ESI
00401034 . 8945 FC MOV DWORD PTR SS:[EBP-4],EAX
00401037 . 3D FFFFFFF0 CMP EAX,0FFFFFFF

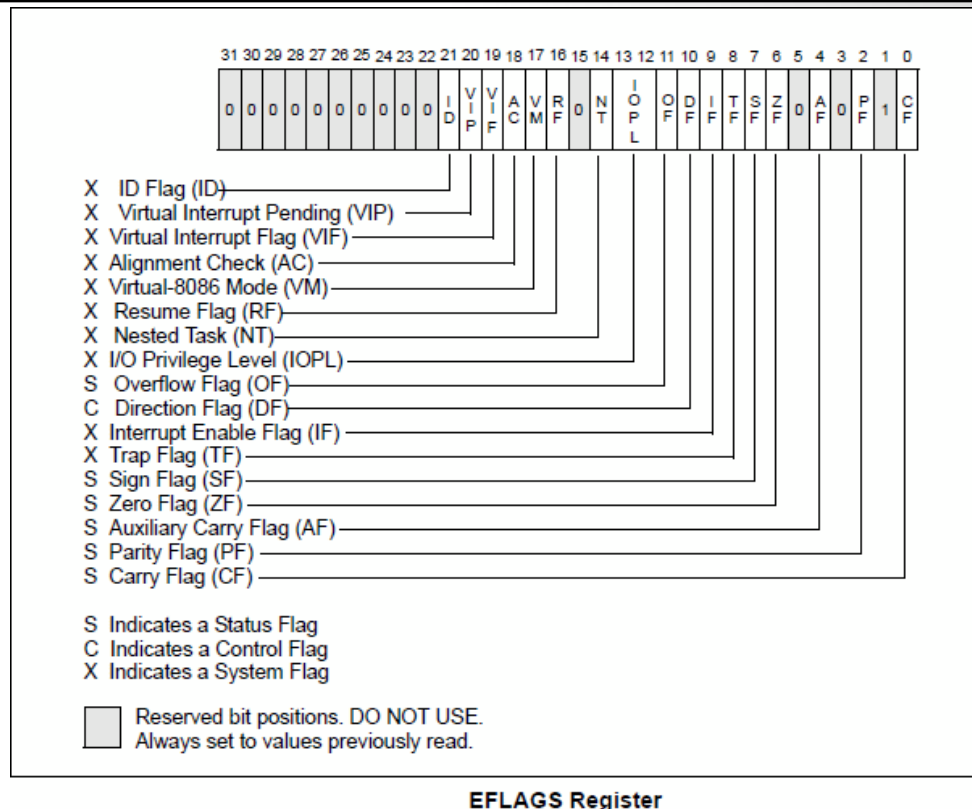
ASCII "Timing Check (RDTSC method)"

0040383D=DynAD_RD.0040383D

Trap Flag



Compare Checksum – DynAD_SingleStep.exe



- When TF is 1, CPU is switching to Single Step mode, each time CPU execute a command will trigger one EXCEPTION_SINGLE_STEP exception. And TF will reset to 0 automatically.

Breakpoint Detection

- When we debug the program, we usually set a breakpoint
 - breakpoint → x86 command is 0xCC
 - if malware detect 0xCC while running, then it will terminate itself
- How to detect 0xCC?

```
CC 3D CC100001 MOV EDI,DWORD PTR DS:[10010CC]
```

Can we just scanning for string 0xCC?

Breakpoint Detection – API Breakpoint Detection

■ Method 1: Detect API Breakpoint

- Most (experienced) code reverse engineer set a breakpoint for the following API:
 - **[Process]:** CreateProcess, CreateThread, EnumProcessModules, OpenProcess, TerminateProcess, ShellExecuteA, CreateRemoteThread, CreateProcessAsUser, EnumProcess...
 - **[Memory]:** ReadProcessMemory, WriteProcessMemory, VirtualAlloc, VirtualProtect, VirtualQuery...
 - **[File]:** CreateFile, ReadFile, WriteFile, CopyFile, CreateDirectory, DeleteFile, MoveFile, GetFileSize...
 - **[Register]:** RegCreateKeyEx, RegDeleteKey, RegSetValue
 - **[Network]:** WSASStartup, socket, inet_addr, recv, send, HttpOpenRequest

Malware just need to check if the first byte of these functions is changed to 0XCC

Breakpoint Detection

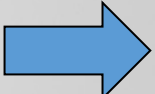
■ Method 1: Detect API Breakpoint

- Most (experienced) code reverse engineer set a breakpoint for the following API:
 - **[Process]:** CreateProcess, CreateThread, EnumProcessModules, OpenProcess, TerminateProcess, ShellExecuteA, CreateRemoteThread, CreateProcessAsUser, EnumProcess...
 - **[Memory]:** ReadProcessMemory, WriteProcessMemory, VirtualAlloc, VirtualProtect, VirtualQuery...
 - **[File]:** CreateFile, ReadFile, WriteFile, CopyFile, CreateDirectory, DeleteFile, MoveFile, GetFileSize...
 - **[Register]:** RegCreateKeyEx, RegDeleteKey, RegSetValue
 - **[Network]:** WSASStartup, socket, inet_addr, recv, send, HttpOpenRequest

Malware just need to check if the first byte of these functions is changed to 0XCC

Breakpoint Detection – Code Checksum Comparison

00401181	. 75 0B	JNZ SHORT DynAD_Si.0040118E
00401183	. 56	PUSH ESI
00401184	. 56	PUSH ESI
00401185	. 6A 01	PUSH 1
00401187	. 56	PUSH ESI
00401188	. FF15 04804000	CALL DWORD PTR DS:[<&KERNEL32.HeapSetInf
0040118E	> B8 4D5A0000	MOV EAX,5A4D
00401193	. 66:3905 00004000	CMP WORD PTR DS:[400000],AX
0040119A	. 74 05	JE SHORT DynAD_Si.004011A1
0040119C	> 8975 E4	MOV DWORD PTR SS:[EBP-1C],ESI
0040119F	. EB 36	JMP SHORT DynAD_Si.004011D7
004011A1	> A1 3C004000	MOV EAX,DWORD PTR DS:[40003C]
004011A6	. 81B8 00004000	CMP DWORD PTR DS:[EAX+400000],4550
004011B0	. ^75 EA	JNZ SHORT DynAD_Si.0040119C
004011B2	. 80 00010000	MOV ECX,100

 0x12345678

Checksum

00401181	CC 0B	JNZ SHORT DynAD_Si.0040118E
00401183	. 56	PUSH ESI
00401184	. 56	PUSH ESI
00401185	. 6A 01	PUSH 1
00401187	. 56	PUSH ESI
00401188	. FF15 04804000	CALL DWORD PTR DS:[<&KERNEL32.HeapSetInf
0040118E	> B8 4D5A0000	MOV EAX,5A4D
00401193	. 66:3905 00004000	CMP WORD PTR DS:[400000],AX
0040119A	. 74 05	JE SHORT DynAD_Si.004011A1
0040119C	> 8975 E4	MOV DWORD PTR SS:[EBP-1C],ESI
0040119F	. EB 36	JMP SHORT DynAD_Si.004011D7
004011A1	> A1 3C004000	MOV EAX,DWORD PTR DS:[40003C]
004011A6	. 81B8 00004000	CMP DWORD PTR DS:[EAX+400000],4550
004011B0	. ^75 EA	JNZ SHORT DynAD_Si.0040119C
004011B2	. 80 00010000	MOV ECX,100

 0xD71A5CA2

Compare Checksum – DynAD_Checksum.exe

Q & A

