Class 21

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*

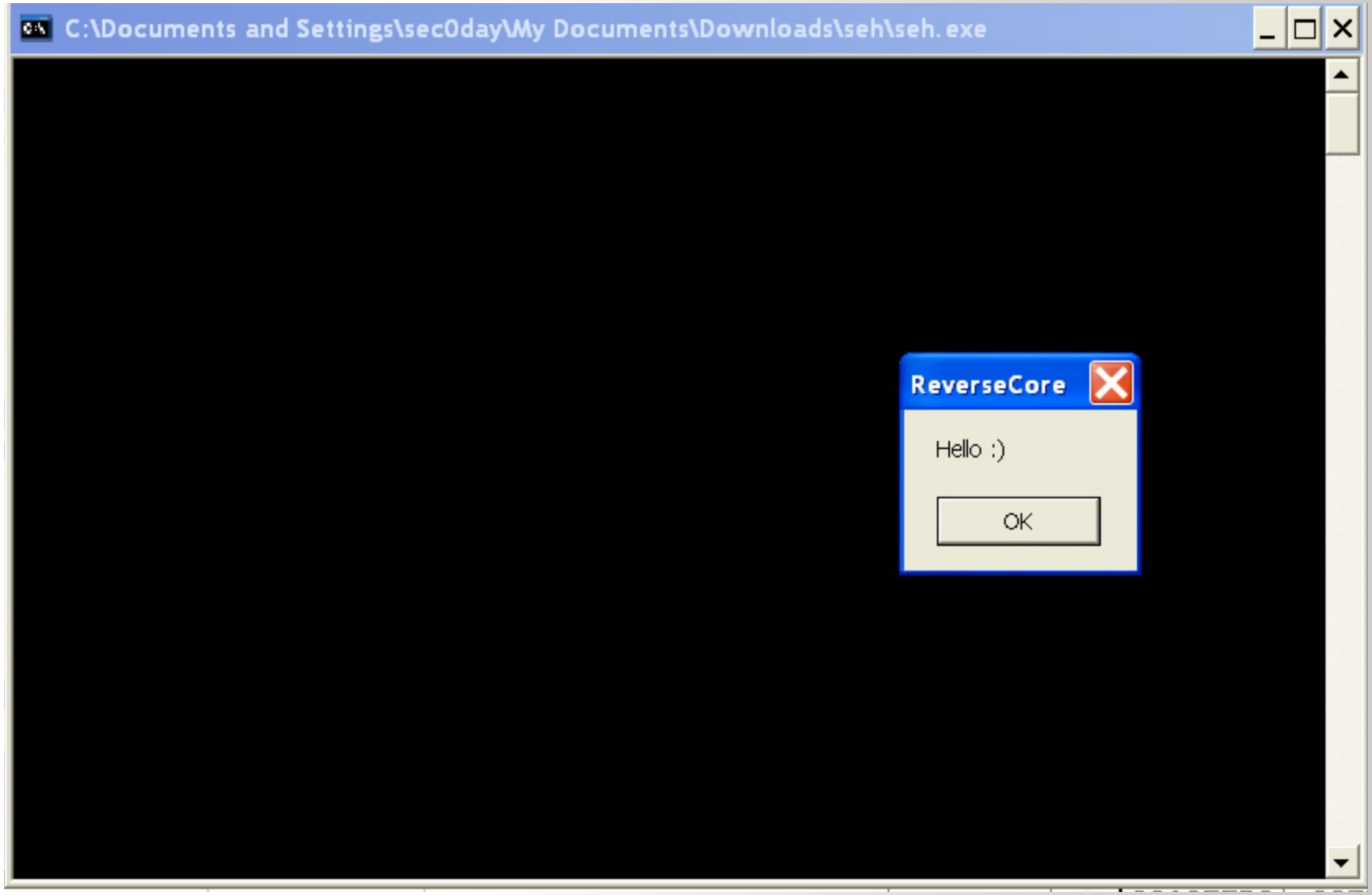
```
void fault_nocatch_fin(void) {
    __try
    {
        __try
        {
            sehR = *sehD;
        }
        __finally
        {
            TEST_STEP(17)
        }
    }
    __except (EvalFilter_dll(EXCEPTION_CONTINUE_SEARCH))
    {
        TEST_STEP(41)
    }
}
```

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

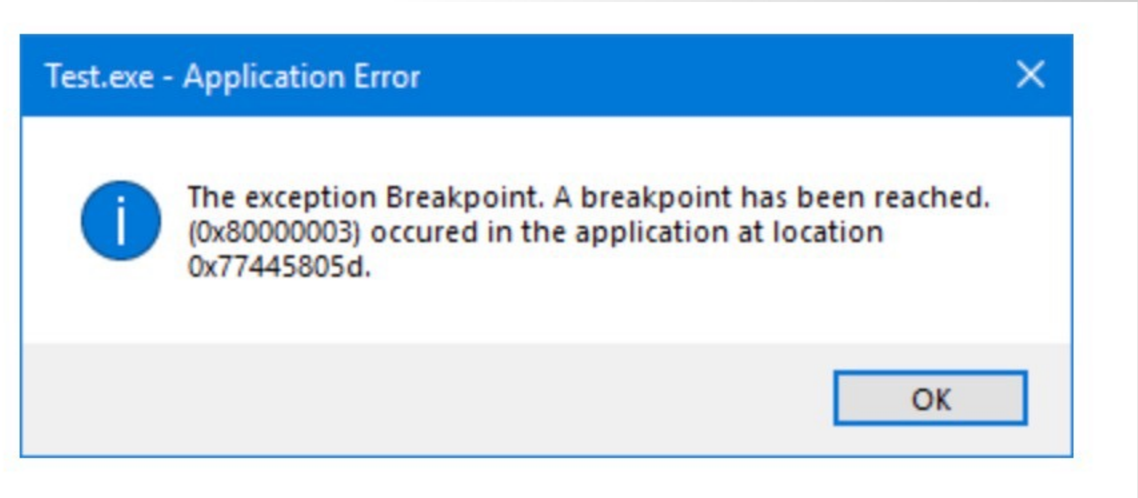
```c
#define EXCEPTION_ACCESS_VIOLATION          0xC0000005u
#define EXCEPTION_DATATYPE_MISALIGNMENT     0x80000002u
#define EXCEPTION_BREAKPOINT                0x80000003u
#define EXCEPTION_SINGLE_STEP               0x80000004u
#define EXCEPTION_ARRAY_BOUNDS_EXCEEDED     0xC000008Cu
#define EXCEPTION_FLT_DENORMAL_OPERAND      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



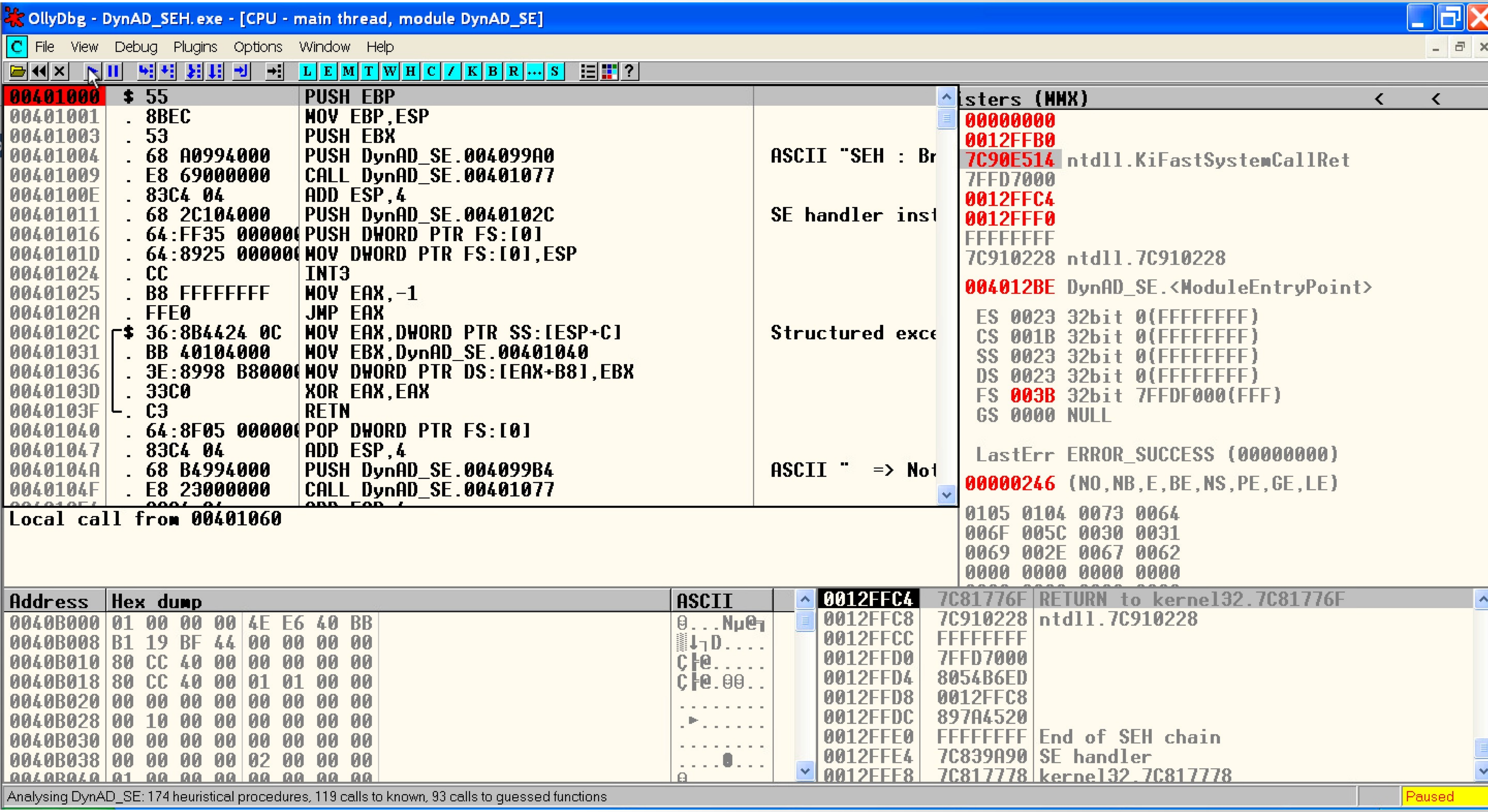
EXCEPTION_BREAKPOINT

```
#define EXCEPTION_BREAKPOINT                    0x80000003u
```

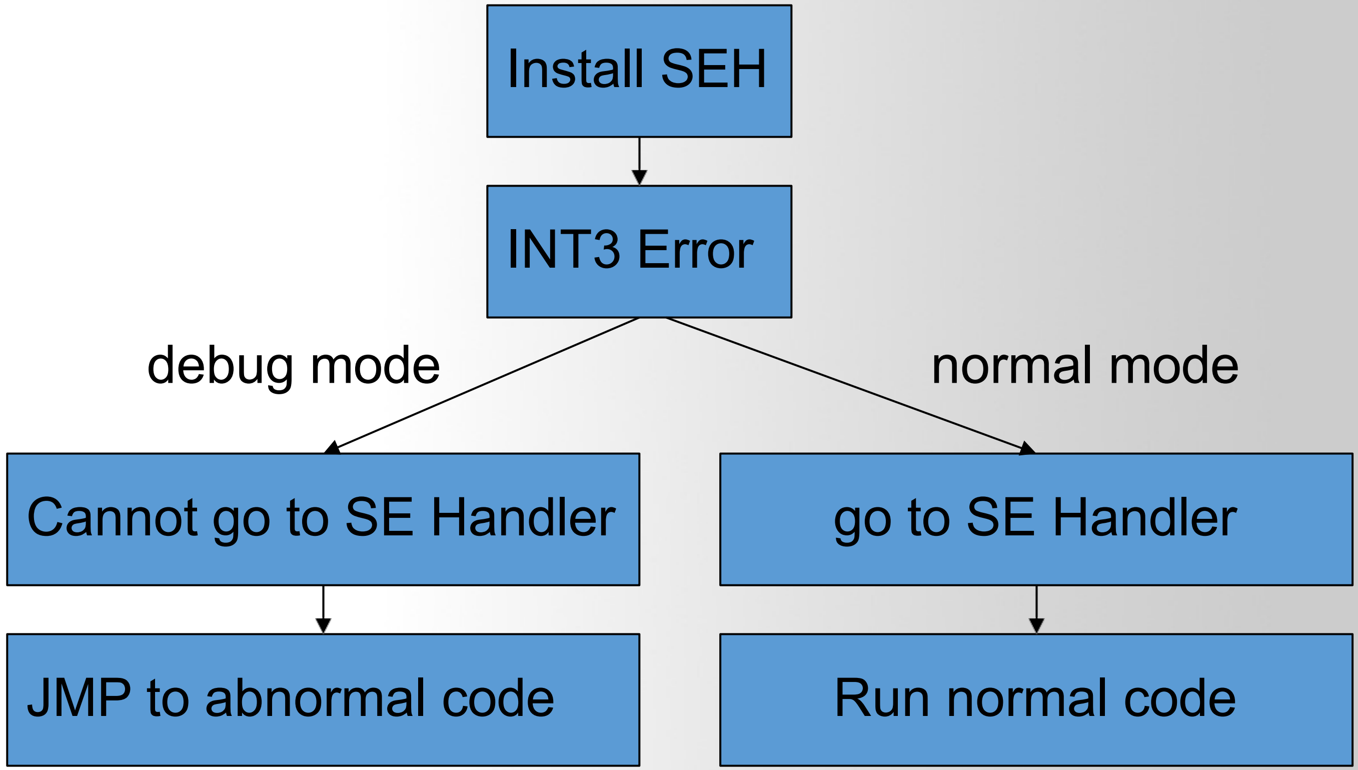


Program will automatically call the registered SEH. If the program is running under the Debug mode, it will stop the program and give the control back to the debugger.

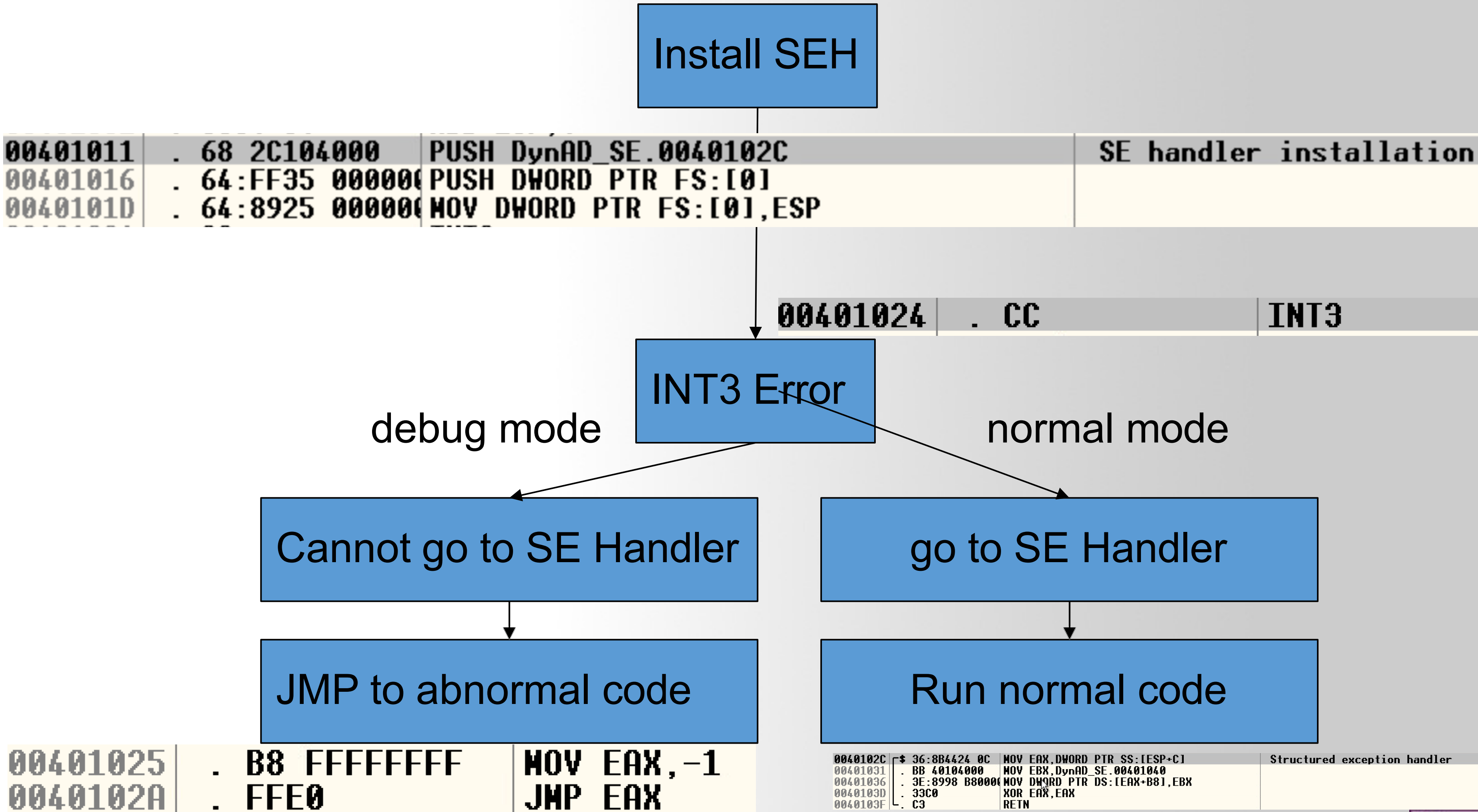
SEH Example – DynAD_SEH.exe



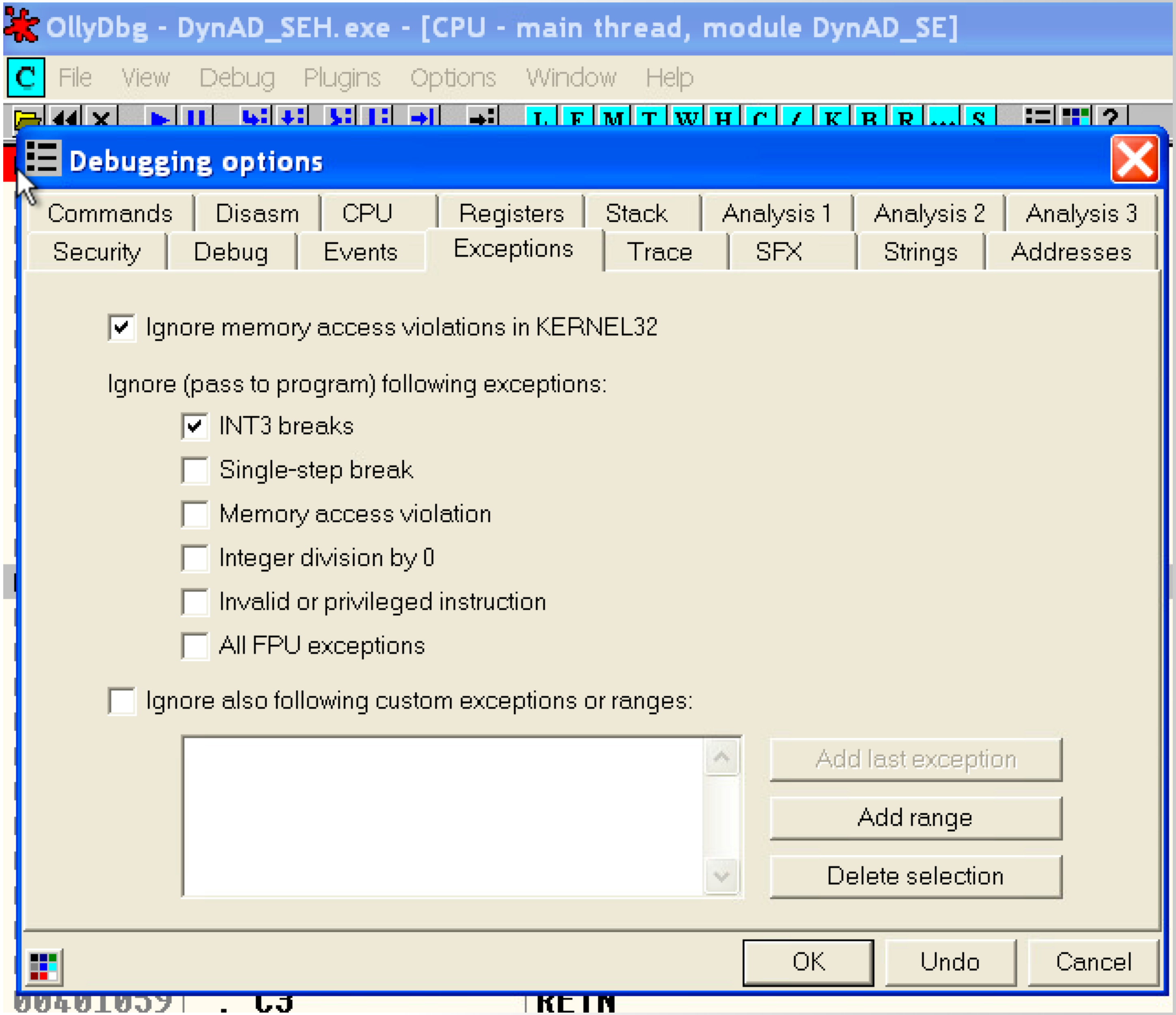
SEH Example – DynAD_SEH.exe



SEH Example – DynAD_SEH.exe



How to bypass INT3 breaks



Timing Check

***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()

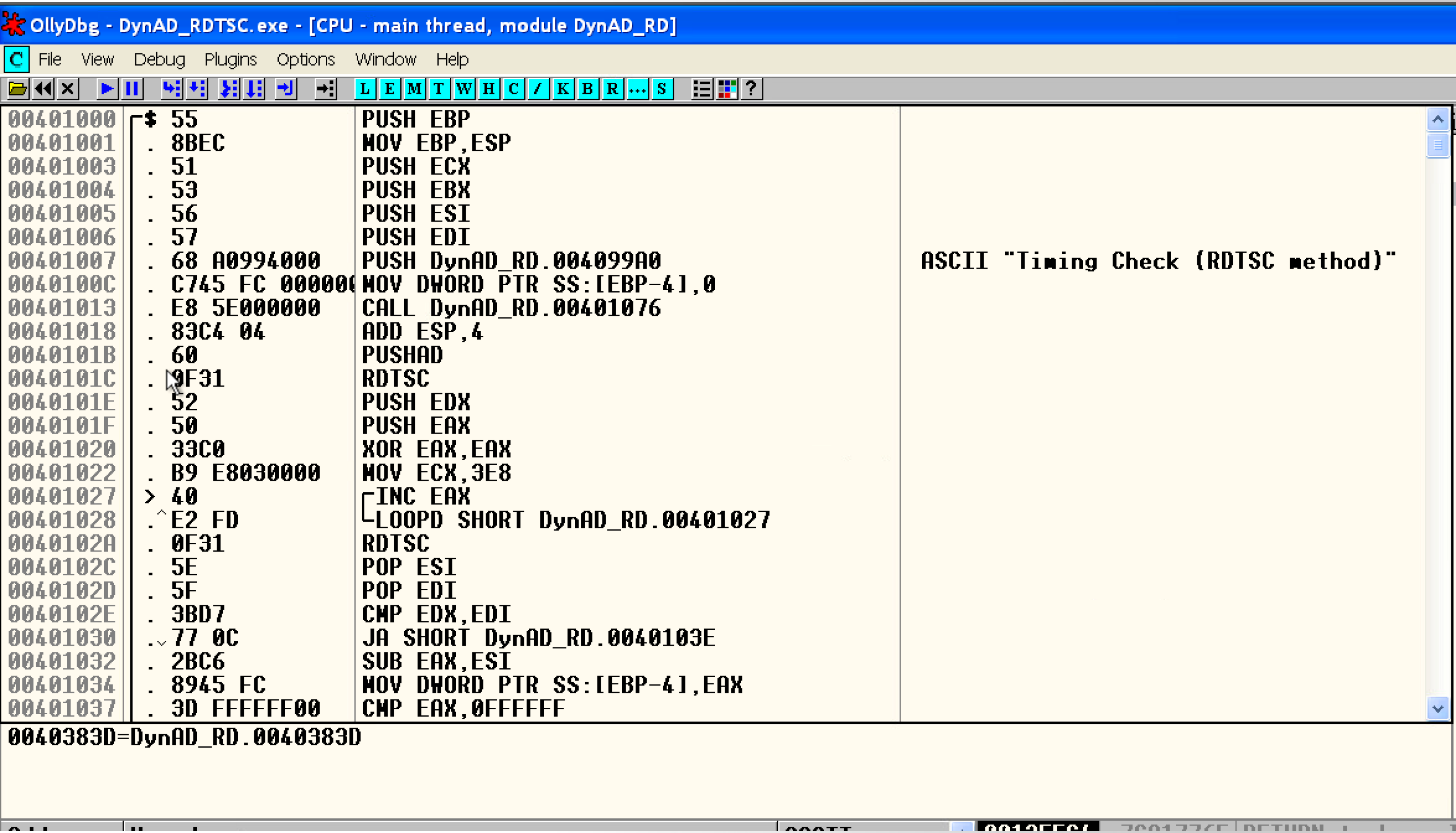
Use CPU counter
Or system time

Timing Check Example – DynAD_RDTSC.exe

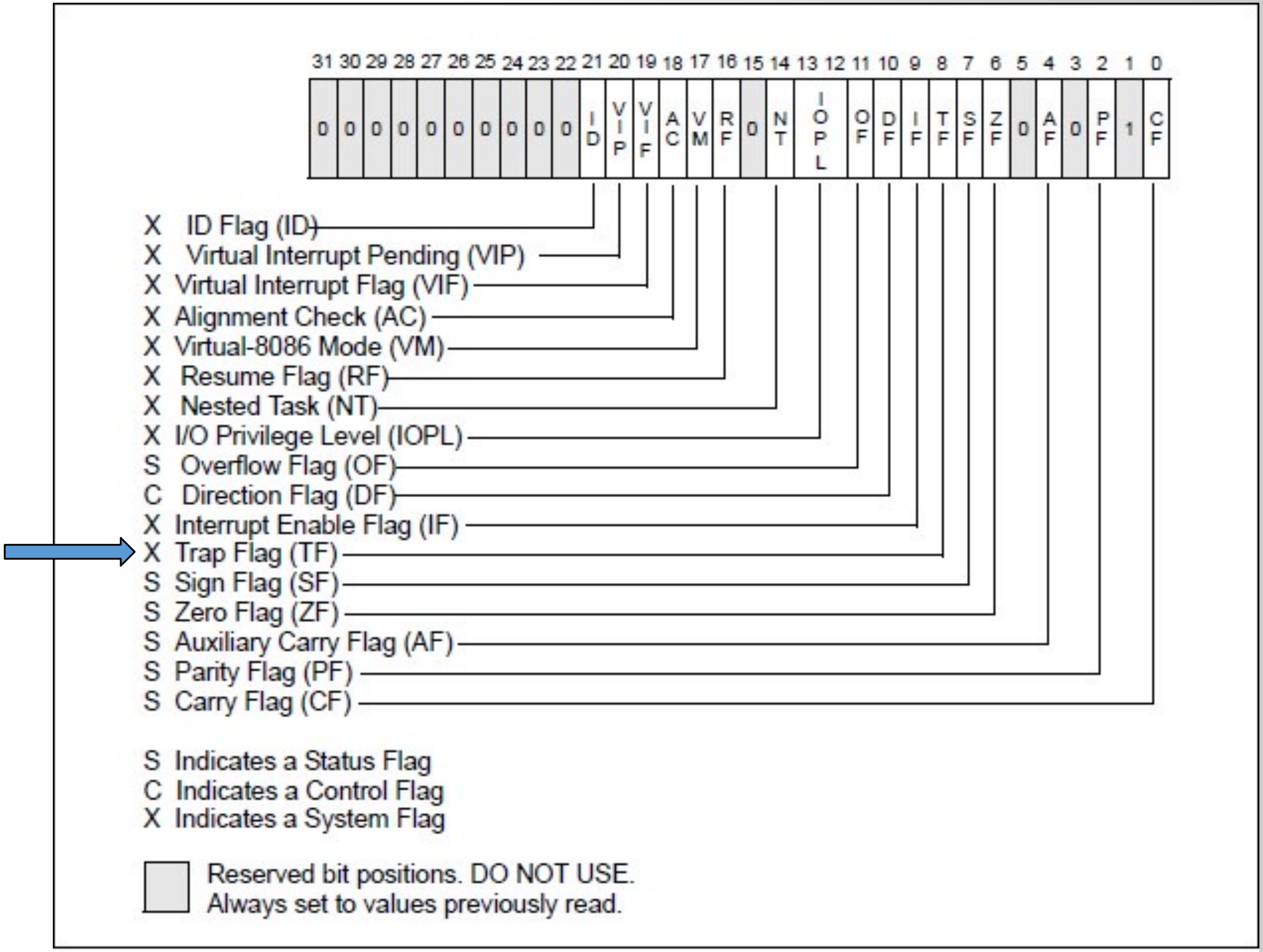
The **Time Stamp Counter** (**TSC**) is a 64-bit register present on all x86 processors since the Pentium. It counts the number of CPU cycles since its reset.

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

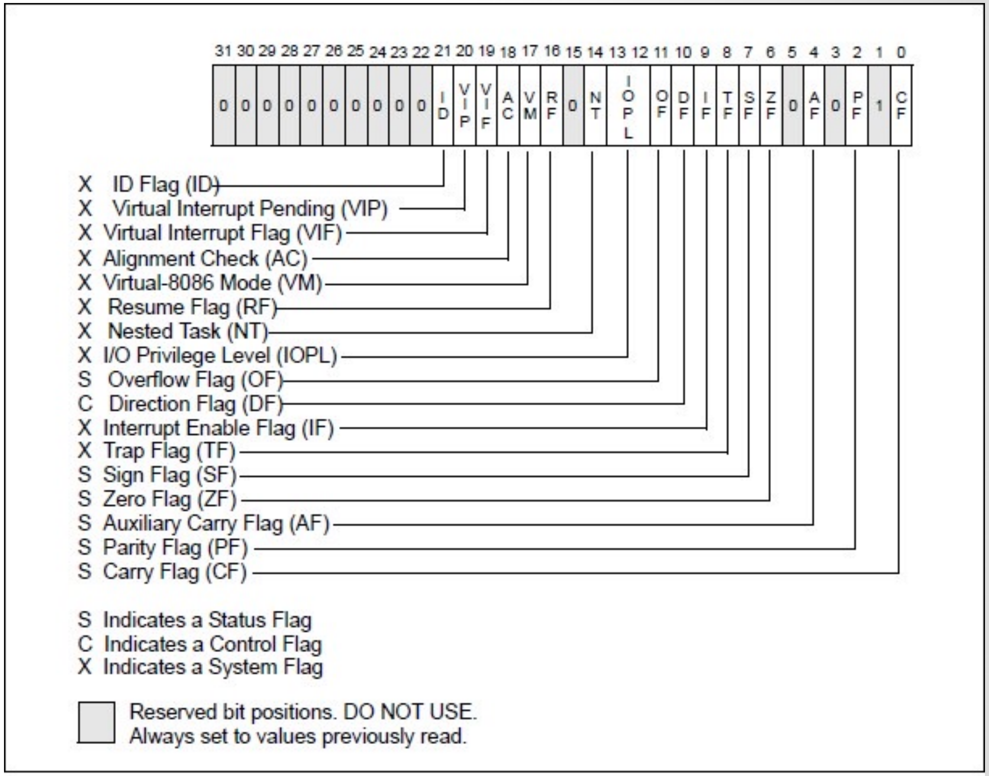


Trap Flag



EFLAGS Register

Compare Checksum – DynAD_SingleStep.exe



EFLAGS Register

- 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,CrateProcessAsUser, EnumProcess…
- **[Memory]:** ReadProcessMemory, WriteProcessMemory, VirtualAlloc, VirtualProtect, VirtualQuery…
- **[File]:** CreateFile, ReadFile, WriteFile, CopyFile, CreateDirectory, DeleteFile, MoveFile, GetFileSize…
- **[Register]:** RegCreateKeyEx, RegDeleteKey, RegSetValue
- **[Network]:** WSAStartup, 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,CrateProcessAsUser, EnumProcess…
  - **[Memory]:** ReadProcessMemory, WriteProcessMemory, VirtualAlloc, VirtualProtect, VirtualQuery…
  - **[File]:** CreateFile, ReadFile, WriteFile, CopyFile, CreateDirectory, DeleteFile, MoveFile, GetFileSize…
  - **[Register]:** RegCreateKeyEx, RegDeleteKey, RegSetValue
  - **[Network]:** WSAStartup, 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 |.v75 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 000040 |CMP WORD PTR DS:[400000],AX
0040119A |.v74 05          |JE SHORT DynAD_Si.004011A1
0040119C |> 8975 E4        |MOV DWORD PTR SS:[EBP-1C],ESI
0040119F |.vEB 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
```

→ 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 000040 |CMP WORD PTR DS:[400000],AX
0040119A |.v74 05          |JE SHORT DynAD_Si.004011A1
0040119C |> 8975 E4        |MOV DWORD PTR SS:[EBP-1C],ESI
0040119F |.vEB 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
```

→ 0xD71A5CA2

Compare Checksum – DynAD_Checksum.exe

Q & A