CSec15233
Malicious Software Analysis

Debugging

Qasem Abu Al-Haija

Overview of Debugging

Disassembler vs Debugger

- A disassembler shows the state of the program just before execution.
- A debugger examines the state of the program during execution.
- Debuggers allow you to see/change:
  - Every memory location, register, and argument to every function.
  - At any point during the processing.

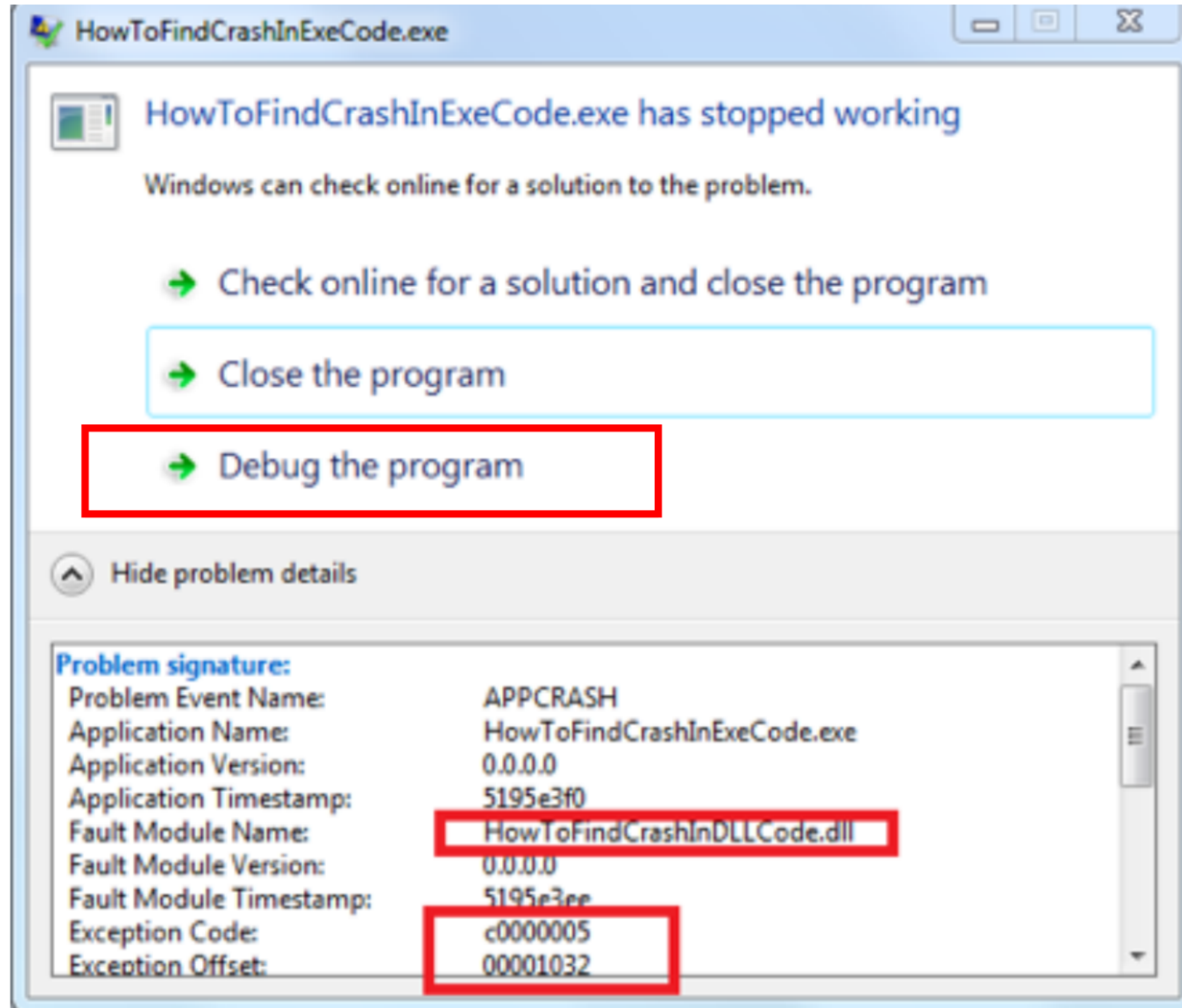
Source-level vs. Assembly-level debuggers

- Source-level debuggers.
  - Usually built into a development platform
  - Allow a programmer to debug while coding.

- Assembly-level debuggers.
  - AKA ‘low-level debuggers’, operate on assembly code instead of source code
  - Heavily used by Malware analysts because they do not require access to a program’s source code.

Windows Crashes

- When an app crashes, Windows may offer to open it in a debugger
- Usually, it uses Windbg



Kernel vs. User-Mode Debugging

- User Mode Debugging
  - Debugger runs on the same system as the code being analyzed
  - Debugging a single executable
  - Separated from other executables by the OS.

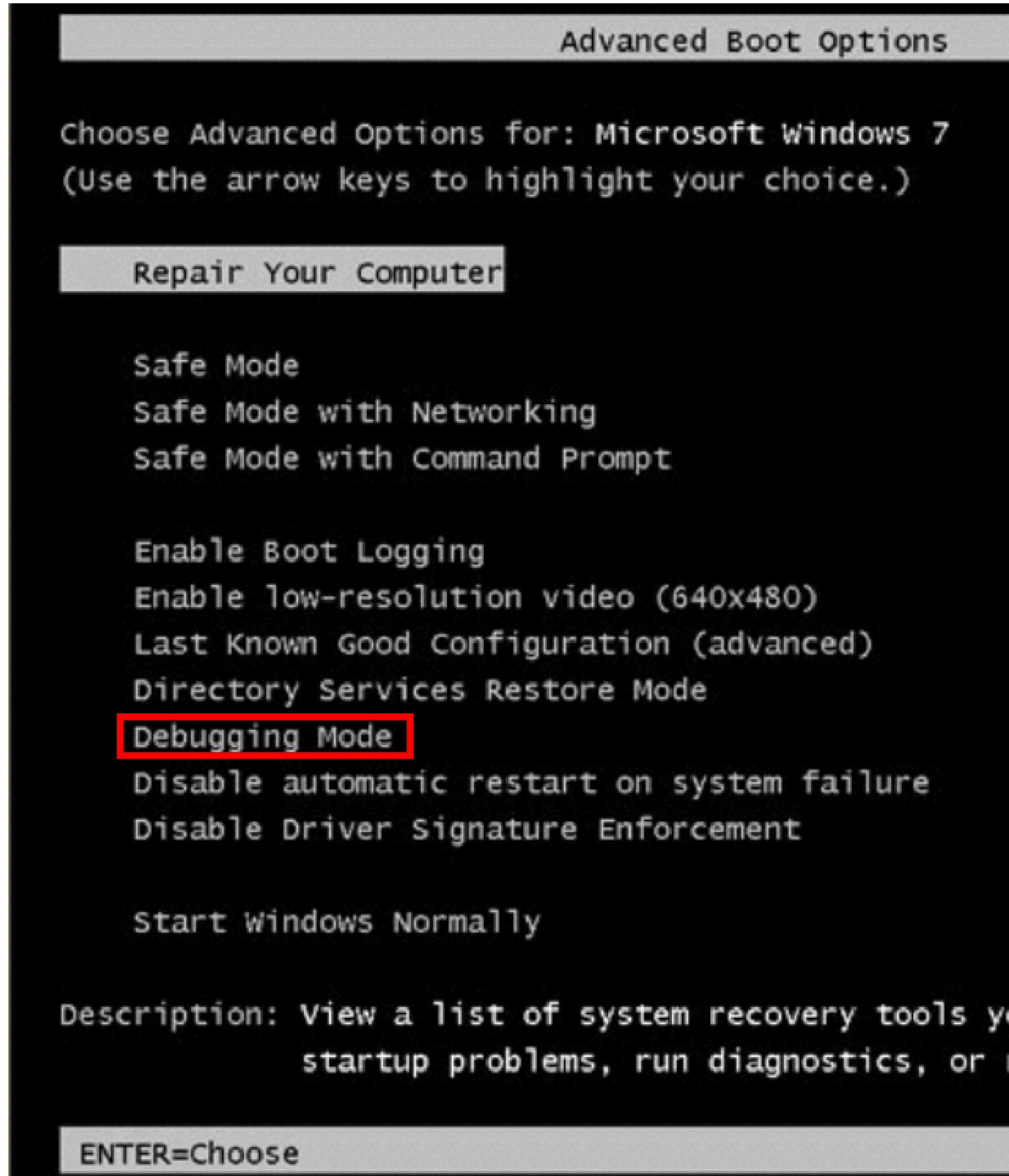
- Kernel Mode Debugging (Conventional way)
  - Needs 2 connected computers since only one kernel per computer
  - One runs the code being debugged, and another runs the debugger
  - OS must be configured to allow kernel debugging.
  - If the kernel is at a breakpoint, the system stops (no apps run).

New (but uncommon) Kernel Mode Debugging tools

- Allows you to debug the kernel with only one computer
  - but it is very uncommon.
- SoftICE Program.
  - used to provide this functionality, but it has not been supported since early 2007.
- LiveKd Program.
  - Recent, MUCH easier, has some limitations.

Windows 7 Advanced Boot Options

- Press F8 during startup
- "Debugging Mode"



Different Debuggers

- WinDbg is the only popular tool that supports kernel debugging
  - It supports user-mode debugging

- OllyDbg is the most popular debugger for malware analysts,
  - But it does not support kernel debugging.

- IDA Pro has a built-in debugger.
  - But do not offer the same features as OllyDbg.

Using a Debugger

Using a Debugger

- ***Single-Stepping***
- ***Stepping-Over***
- ***Stepping-Into***

Single-Stepping

- run a single instruction and then return control to the debugger
- Simple but slow
- Don't get bogged down in details

Single-Stepping: Example

- This code decodes the string with XOR

Example 9-1. Stepping through code

```
mov     edi, DWORD_00406904
mov     ecx, 0x0d
LOC_040106B2
xor     [edi], 0x9C
inc     edi
loopw   LOC_040106B2
...
DWORD:00406904:    F8FDF3D01
```

Example 9-2. Single-stepping through a section of code to see how it changes memory

```
D0F3FDF8 D0F5FEEE FDEEE5DD 9C (.............)
4CF3FDF8 D0F5FEEE FDEEE5DD 9C (L............)
4C6FFDF8 D0F5FEEE FDEEE5DD 9C (Lo...........)
4C6F61F8 D0F5FEEE FDEEE5DD 9C (Loa..........)
. . . SNIP . . .
4C6F6164 4C696272 61727941 00 (LoadLibraryA.)
```

Stepping-Over

- When you step over call instruction, you bypass it.
  - Completes the call and returns without pausing
  - Decreases the amount of code you need to analyze
  - Might miss important functionality, especially if the function never returns

- For example, if you step over a call:
  - The next instruction you will see in your debugger will be the instruction after the function call returns

Stepping-Into

- Moves into the function and stops at its first command

- For example, if you step into a call instruction:
  - The next instruction you will see in the debugger is the call function's first instruction.

Breakpoints

Breakpoints

- *Breakpoints* ➔ pause execution to examine a program's state.
- Any program paused at a breakpoint is referred to as *broken*.
- Breakpoints are needed:
  - because you can't access registers or memory addresses while a program is running since these values are constantly changing.
- Example
  - You can't tell where this call is going
  - Set a breakpoint at the call and see what's in eax

```
00401008    mov     ecx, [ebp+arg_0]
0040100B    mov     eax, [edx]
0040100D    call    eax
```

Listing 8-3: Call to EAX

Breakpoints

- This code calculates a filename and then creates the file
- Set a breakpoint at CreateFileW and look at the stack to see the filename

```
0040100B  xor     eax, esp
0040100D  mov     [esp+0D0h+var_4], eax
00401014  mov     eax, edx
00401016  mov     [esp+0D0h+NumberOfBytesWritten], 0
0040101D  add     eax, 0FFFFFFFEh
00401020  mov     cx, [eax+2]
00401024  add     eax, 2
00401027  test    cx, cx
0040102A  jnz     short loc_401020
0040102C  mov     ecx, dword ptr ds:a_txt ; ".txt"
00401032  push    0                ; hTemplateFile
00401034  push    0                ; dwFlagsAndAttributes
00401036  push    2                ; dwCreationDisposition
00401038  mov     [eax], ecx
0040103A  mov     ecx, dword ptr ds:a_txt+4
00401040  push    0                ; lpSecurityAttributes
00401042  push    0                ; dwShareMode
00401044  mov     [eax+4], ecx
00401047  mov     cx, word ptr ds:a_txt+8
0040104E  push    0                ; dwDesiredAccess
00401050  push    edx              ; lpFileName
00401051  mov     [eax+8], cx
00401055❶call    CreateFileW ; CreateFileW(x,x,x,x,x,x,x)
```

Listing 8-4: Using a debugger to determine a filename

Breakpoints: WinDbg

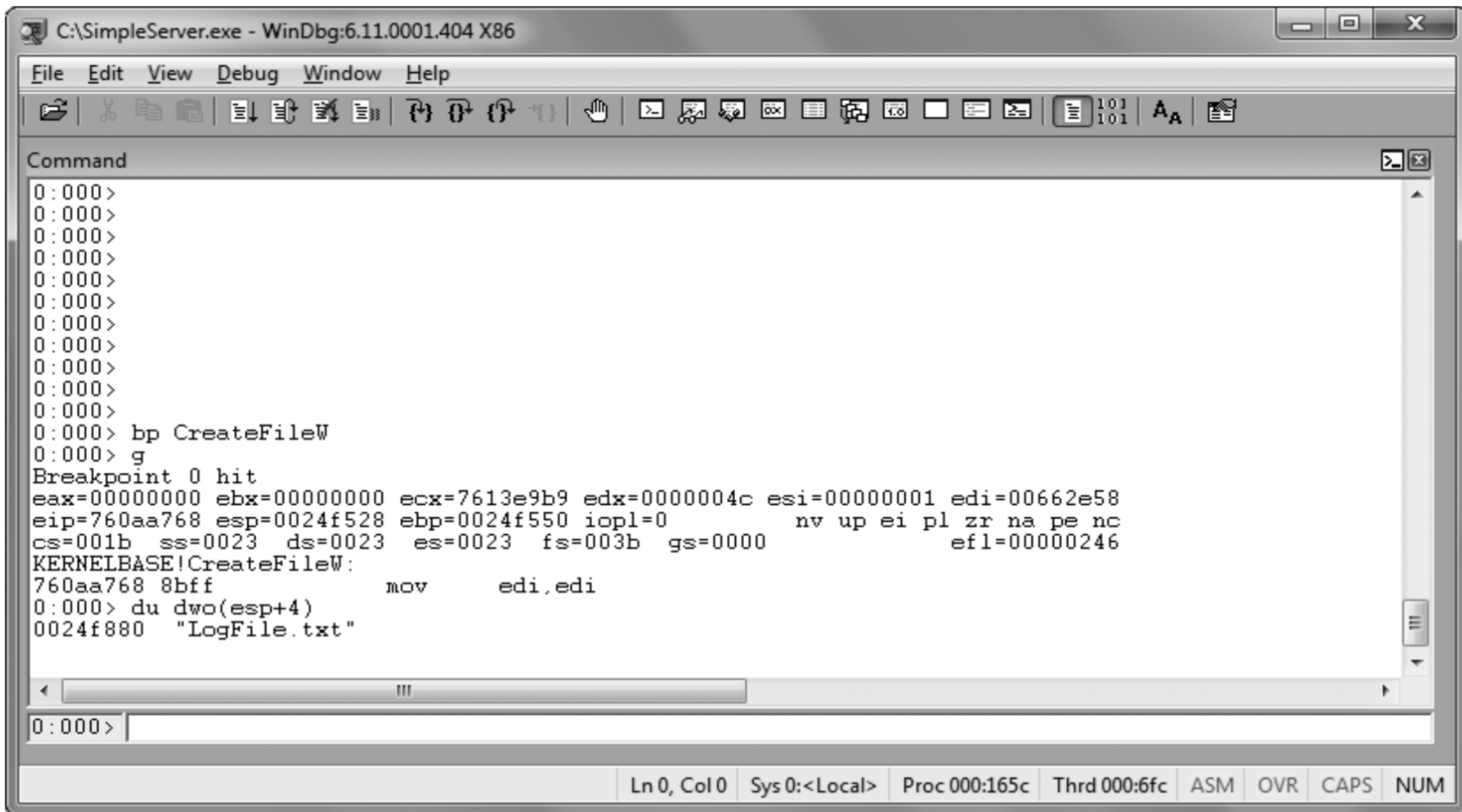


Figure 8-1: Using a breakpoint to see the parameters to a function call. We set a breakpoint on `CreateFileW` *and then examine the first parameter of the stack.*

Encrypted Data

- Suppose malware sends encrypted network data

- Set a breakpoint before the data is encrypted and view it

Encrypted Data

```
004010D0   sub      esp, 0CCh
004010D6   mov      eax, dword_403000
004010DB   xor      eax, esp
004010DD   mov      [esp+0CCh+var_4], eax
004010E4   lea      eax, [esp+0CCh+buf]
004010E7   call     GetData
004010EC   lea      eax, [esp+0CCh+buf]
004010EF ❶call     EncryptData
004010F4   mov      ecx, s
004010FA   push     0                  ; flags
004010FC   push     0C8h               ; len
00401101   lea      eax, [esp+0D4h+buf]
00401105   push     eax                ; buf
00401106   push     ecx                ; s
00401107   call     ds:Send
```

Listing 8-5: Using a breakpoint to view data before the program encrypts it

Encrypted Data Example: OllyDbg

```
OllyDbg - SimpleServer.exe - [CPU - main thread, module SimpleSe]
C File View Debug Plugins Options Window Help

010410D0  $ 81EC CC000000  SUB ESP,0CC
010410D6  . A1 00300401    MOV EAX,DWORD PTR DS:[__security_cookie]
010410DB  . 33C4           XOR EAX,ESP
010410DD  . 898424 C800000 MOV DWORD PTR SS:[ESP+C8],EAX
010410E4  . 8D0424         LEA EAX,DWORD PTR SS:[ESP]
010410E7  . E8 A4FFFFFF    CALL SimpleSe.GetData
010410EC  . 8D0424         LEA EAX,DWORD PTR SS:[ESP]
010410EF  . E8 BCFFFFFF    CALL SimpleSe.EncryptData
010410F4  . 8B0D 70330401  MOV ECX,DWORD PTR DS:[s]
010410FA  . 6A 00          PUSH 0                        Flags
010410FC  . 68 C8000000    PUSH 0C8                      DataSi
01041101  . 8D4424 08      LEA EAX,DWORD PTR SS:[ESP+8]
01041105  . 50             PUSH EAX                      Data
01041106  . 51             PUSH ECX                      Socket

010410B0=SimpleSe.EncryptData
```

```
Address   Hex dump                                          ASCII
0012F9C8  53 65 63 72 65 74 20 4D 65 73 73 61 67 65 2E 00  Secret Message..
0012F9D8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0012F9E8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0012F9F8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

Breakpoint at SimpleSe.010410EF                                Paused
```

Figure 8-2: Viewing program data prior to the encryption function call

Types of Breakpoints

- Software execution
- Hardware execution
- Conditional

Software Execution Breakpoints

- The default option for most debuggers
- Debugger overwrites the first byte of the instruction with 0xCC
  - The instruction for INT 3
    - An interrupt designed for use with debuggers
  - When the breakpoint is executed:
    - The OS generates an exception and transfers control to the debugger

Memory Contents at a Breakpoint

- There's a breakpoint at the push instruction
- Debugger says it's 0x55, but it's really 0xCC

Table 8-1: Disassembly and Memory Dump of a Function with a Breakpoint Set

Disassembly view				Memory dump	
00401130	55	❶push	ebp	00401130	❷CC 8B EC 83
00401131	8B EC	mov	ebp, esp	00401134	E4 F8 81 EC
00401133	83 E4 F8	and	esp, 0FFFFFFF8h	00401138	A4 03 00 00
00401136	81 EC A4 03 00 00	sub	esp, 3A4h	0040113C	A1 00 30 40
0040113C	A1 00 30 40 00	mov	eax, dword_403000	00401140	00

Some issues of SW Execution Breakpoints

- When Software Execution Breakpoints Fail:
  - I.e., If 0xCC byte is changed during code execution:
  - The breakpoint won't occur
- If other code reads the memory containing breakpoint:
  - It will read 0xCC instead of the original byte

Hardware Execution Breakpoints

- x86 architecture supports HW execution BrkPts through dedicated debug registers.
  - DR0 through DR3 – addresses of breakpoints
  - DR7 stores control information (such as stop address)

- Can break on access or execution
  - Can set to break on read, write, or both
  - No change in code bytes

++ HW Execution Breakpoints

- No change in code bytes
  - It doesn't matter which bytes are stored at that location.
- Can break on access rather than execution.
  - break whenever a certain memory loc. is read or written

-- HW Execution Breakpoints

- Only 4 HW registers store breakpoint addresses.
  - While eight debug registers in the chipset,
- They are easy to modify by the running program.
  - Malicious programs can modify these registers.

Conditional Breakpoints (CN-BP)

- Break only if a certain condition is true.
  - *Ex: Set a breakpoint on the GetProcAddress function*
  - *Only if the parameter being passed in is RegSetValue*
- The debugger evaluates the condition;
  - *if not met, it continues execution without alerting the user.*
  - *Different debuggers support different condition*
- CN-BP takes much longer than ordinary instructions.

Exceptions

Exceptions

- Used by debuggers to gain control of a running program
- Breakpoints generate exceptions
- Exceptions are also caused by
  - Invalid memory access
  - Division by zero
  - Other conditions

First- and Second-Chance Exceptions

- When an exception occurs while a debugger is attached
  - The program stops executing
  - The debugger is given the first chance at control
  - Debugger either handle the exception or pass it on to the program
  - If it's passed on, the program's exception handler takes it

First- and Second-Chance Exceptions

- If the application doesn't handle the exception
  - The debugger is given a second chance to handle it
  - This means the program would have crashed if the debugger were not attached
- In malware analysis, the first-chance exception can usually be ignored
- Second-chance exceptions cannot be ignored
  - This means that malware doesn't like the environment in which it is running

Common Exceptions

- INT 3 (Software breakpoint).
  - Programs may include their handlers for INT 3 exc., but when a debugger is attached, it will get the first chance.
- Single-stepping in a debugger is implemented as an exception
  - If the trap flag in the flags register is set, the processor executes one instruction and then generates an exception
- Memory-access violation exception
  - Code tries to access a location that it cannot access.
  - Because: address is invalid or access-control protections
- Violating Privilege Rules
  - Attempt to execute a privileged instruction (kernel mode) with an outside privileged mode (user mode)

Common Exceptions

The following chart lists the exceptions that can be generated by the Intel 80286, 80386, 80486, and Pentium processors:

```
Exception | Description
(dec/hex) |
----------|-------------------------------------------------
  0  00h  | Divide error:
          | Occurs during a DIV or an IDIV instruction when the
          | divisor is zero or a quotient overflow occurs.

  1  01h  | Single-step/debug exception:
          | Occurs for any of a number of conditions:
          | - Instruction address breakpoint fault
          | - Data address breakpoint trap
          | - General detect fault
          | - Single-step trap
          | - Task-switch breakpoint trap

  2  02h  | Nonmaskable interrupt:
          | Occurs because of a nonmaskable hardware interrupt.

  3  03h  | Breakpoint:
          | Occurs when the processor encounters an INT 3 instruction.
```

Modifying Execution with a Debugger

Skipping a Function

- Debuggers can change program execution.
  - You can change the control flags, instruction pointer, or the code itself.

- You could avoid a function call by:
  - Setting a breakpoint where at the call and then
  - Changing instruction pointer to the instruction after it
  - This may cause the program to crash or malfunction

Testing a Function

- You could run a function directly without waiting for the main code to use it
  - You will have to set the parameters
  - This destroys a program's stack
  - The program won't run properly when the function completes

Practical Example: Real Virus

Operation depends on language setting of a computer

- Simplified Chinese
  - Uninstalls itself & does no harm
- English
  - Display pop-up "Your luck's no good“
- Japanese or Indonesian
  - Overwrite the hard drive with random data

Practical Example: Real Virus

```
00411349    call    GetSystemDefaultLCID
0041134F   ❶mov     [ebp+var_4], eax
00411352    cmp     [ebp+var_4], 409h
00411359    jnz     short loc_411360
0041135B    call    sub_411037
00411360    cmp     [ebp+var_4], 411h
00411367    jz      short loc_411372
00411369    cmp     [ebp+var_4], 421h
00411370    jnz     short loc_411377
00411372    call    sub_41100F
00411377    cmp     [ebp+var_4], 0C04h
0041137E    jnz     short loc_411385
00411380    call    sub_41100A
```

Break at 1; Change Return Value

Listing 8-6: Assembly for differentiating between language settings

Example:

Debugging with IDA Pro

Solving "CrackMe" Challenge

- Here, we will use IDA Pro debugging to solve the "FindMySerial" Challenge
- "FindMySerial" Challenge is an executable file protected with a serial number (or password).
  - If you enter the password correctly, the program will let you in
  - If you enter the password incorrectly, the program will not allow you.
- Therefore, our target in this example is to use IDA Pro debugging trying to solve this challenge.
- Let's go ahead and get done of this mission.

Main Sources for these slides

- *Michael Sikorski and Andrew Honig, "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software"; ISBN-10: 1593272901.*
- *Xinwen Fu, "Introduction to Malware Analysis," University of Central Florida*
- *Sam Bowne, "Practical Malware Analysis," City College San Francisco*
- *Abhijit Mohanta and Anoop Saldanha, "Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware," ISBN: 1484261925.*

Thank you