Practical Malware Analysis

Ch 8: Debugging

Rev. 3-5-17
Disassemblers v. Debuggers

• A disassembler like IDA Pro shows the state of the program just before execution begins

• Debuggers show
  – Every memory location
  – Register
  – Argument to every function

• At any point during processing
  – And let you change them
Two Debuggers

• Ollydbg
  – Most popular for malware analysis
  – User-mode debugging only
  – IDA Pro has a built-in debugger, but it's not as easy to use or powerful as Ollydbg

• Windbg
  – Supports kernel-mode debugging
Source-Level v. Assembly-Level Debuggers

• Source-level debugger
  – Usually built into development platform
  – Can set breakpoints (which stop at lines of code)
  – Can step through program one line at a time

• Assembly-level debuggers (low-level)
  – Operate on assembly code rather than source code
  – Malware analysts are usually forced to use them, because they don't have source code
Windows Crashes

• When an app crashes, Windows may offer to open it in a debugger
• Usually it uses Windbg
• Links Ch 8c, 8d
Kernel v. 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
The Old Way

• Requires two computers, because there is only one kernel per computer
• If the kernel is at a breakpoint, the system stops
• One computer runs the code being debugged
• Other computer runs the debugger
• OS must be configured to allow kernel debugging
• Two machines must be connected
Kernel Mode Debugging
The New Way

• Mark Russinovich's Livekd tool allows you to debug the kernel with only one computer!
  • MUCH easier :)
  • Tool has some limitations (Link Ch 8e)
Windows 7 Advanced Boot Options

- Press F8 during startup
- "Debugging Mode"

Choose Advanced Options for: Microsoft Windows 7
(Use the arrow keys to highlight your choice.)

Repair Your Computer

Safe Mode
Safe Mode with Networking
Safe Mode with Command Prompt

Enable Boot Logging
Enable low-resolution video (640x480)
Last Known Good Configuration (advanced)
Directory Services Restore Mode
Debugging Mode
Disable automatic restart on system failure
Disable Driver Signature Enforcement

Start Windows Normally

Description: View a list of system recovery tools you can use to start Windows normally or repair startup problems, run diagnostics, or restore your computer...
Side-Effect of Debug Mode

• PrntScn key causes BSOD
• Please label machines in S214 that you place into debugging mode
  • Use Shoft+PrntScn instead
Good Intro to OllyDbg

• Link Ch 8a
Using a Debugger
Two Ways

• Start the program with the debugger
  – It stops running immediately prior to the execution of its entry point

• Attach a debugger to a program that is already running
  – All its threads are paused
  – Useful to debug a process that is affected by malware
Single-Stepping

- Simple, but slow
- Don't get bogged down in details
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: F8FD3D01
```

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 v. Stepping-Into

• Single step executes one instruction
• **Step-over** call instructions
  – 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
• **Step-into** a call
  – Moves into the function and stops at its first command
Pausing Execution with Breakpoints

• A program that is paused at a breakpoint is called broken

• Example
  – You can't tell where this call is going
  – Set a breakpoint at the call and see what's in eax

```
Example 9-3. Call to EAX
00401008    mov    ecx, [ebp+arg_0]
0040100B    mov    eax, [edx]
0040100D    call   eax
```
• This code calculates a filename and then creates the file
• Set a breakpoint at CreateFileW and look at the stack to see the filename

Example 9-4. Using a debugger to determine a 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
00401055 call CreateFileW ; CreateFileW(x,x,x,x,x,x,x)
```
Figure 9-1. 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
Example 9-5. Using a breakpoint to view data before the program encrypts it

```
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  lea   [call 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
```
Figure 9-2. 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>
<thead>
<tr>
<th>Disassembly view</th>
<th>Memory dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>00401130 55</td>
<td>00401130 FCC 8B EC 83</td>
</tr>
<tr>
<td>00401131 8B EC</td>
<td>00401134 E4 F8 81 EC</td>
</tr>
<tr>
<td>00401133 83 E4 F8</td>
<td>00401138 A4 03 00 00</td>
</tr>
<tr>
<td>00401136 81 EC A4 03 00 00</td>
<td>0040113C A1 00 30 40</td>
</tr>
<tr>
<td>0040113C A1 00 30 40 00</td>
<td>mov eax, dword_403000</td>
</tr>
</tbody>
</table>
When Software Execution Breakpoints Fail

• If the 0xCC byte is changed during code execution, the breakpoint won't occur
• If other code reads the memory containing the breakpoint, it will read 0xCC instead of the original byte
• Code that verifies integrity will notice the discrepancy
Hardware Execution Breakpoints

- Uses four hardware Debug Registers
  - DR0 through DR3 - addresses of breakpoints
  - DR7 stores control information
- The address to stop at is in a register
- Can break on access or execution
  - Can set to break on read, write, or both
- No change in code bytes
Hardware Execution Breakpoints

• Running code can change the DR registers, to interfere with debuggers
• General Detect flag in DR7
  – Causes a breakpoint prior to any mov instruction that would change the contents of a Debug Register
  – Does not detect other instructions, however
Conditional Breakpoints

• Breaks only if a condition is true
  – Ex: Set a breakpoint on the GetProcAddress function
  – Only if parameter being passed in is RegSetValue

• Implemented as software breakpoints
  – The debugger always receives the break
  – If the condition is not met, it resumes execution without alerting the user
Conditional Breakpoints

• Conditional breakpoints take much longer than ordinary instructions
• A conditional breakpoint on a frequently-accessed instruction can slow a program down
• Sometimes so much that it never finishes
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 a exception occurs while a debugger is attached
  – The program stops executing
  – The debugger is given first chance at control
  – Debugger can either handle the exception, or pass it on to the program
  – If it's passed on, the program's exception handler takes it
Second Chance

• 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, first-chance exceptions can usually be ignored
• Second-chance exceptions cannot be ignored
  – They usually mean that the malware doesn't like the environment in which it is running
Common Exceptions

• INT 3 (Software breakpoint)
• 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, either because the address is invalid or because of access-control protections
Common Exceptions

• Violating Privilege Rules
  – Attempt to execute privileged instruction with outside privileged mode
  – In other words, attempt to execute a kernel mode instruction in user mode
  – Or, attempt to execute Ring 0 instruction from Ring 3
## List of Exceptions

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

<table>
<thead>
<tr>
<th>Exception (dec/hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00h</td>
<td>Divide error: Occurs during a DIV or an IDIV instruction when the divisor is zero or a quotient overflow occurs.</td>
</tr>
<tr>
<td>1 01h</td>
<td>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</td>
</tr>
<tr>
<td>2 02h</td>
<td>Nonmaskable interrupt: Occurs because of a nonmaskable hardware interrupt.</td>
</tr>
<tr>
<td>3 03h</td>
<td>Breakpoint: Occurs when the processor encounters an INT 3 instruction.</td>
</tr>
</tbody>
</table>

- Link Ch 8b
Modifying Execution with a Debugger
Skipping a Function

• You can change control flags, the instruction pointer, or the code itself
• You could avoid a function call by setting a breakpoint where at the call, and then changing the instruction pointer to the instruction after it
  – This may cause the program to crash or malfunction, or course
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
Modifying Program Execution in Practice
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
Example 9-6. Assembly for differentiating between language settings

00411349 call GetSystemDefaultLCID
0041134F  mov [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