Practical Malware Analysis

Lecture 8 | Debugging

Debuggers

- View and modify a program as it runs
- Source- v assembly-level
- Kernel- v user-mode
- Run, or attach to a running program
- Modify execution
  - Break before a function, modify instruction ptr to skip it
  - Run functions manually

While a disassembler like IDA allows us to view a snapshot of a program's state as it exists prior to being run, a debugger lets us analyze and modify things such as memory locations, registers, and parameters as a program is running.

Source-level debuggers are heavily used by developers, and often exist within an integrated development environment (IDE). Assembly- or low-level debuggers have the same functionality as a source-level debugger, but operate on the assembly instead of source code, which is often not available to a malware analyst.

Since programs operating at the kernel level share a single process context, any kernel-level debugging that required execution to pause (breakpoints) would pause the entire system, effectively bricking the environment. Therefore debugging in kernel-mode is often performed remotely from a second host. We will use WinDbg for kernel debugging and OllyDbg for everything else. IDA has a debugger but it's not as good.

Running a program with a debugger will cause execution to stop directly prior to the entry point and pass control to the debugger. You can also attach a debugger to a running process, pausing all of the threads.

Debuggers also allow you to change the program as it runs, enabling you to try things like skipping functions entirely or running functions manually with your own inputs to see what they do.

Stepping

- Single-stepping
  - Break after every instruction
- Step-over
  - Break after next function returns
- Step-into
  - Break on first instruction of called function

```
mov     edi, DWORD_00406904
mov     ecx, 0x0d
LOC_040106B2
xor     [edi], 0x9C
inc     edi
loopw   LOC_040106B2
...
DWORD:00406904:   F8FDF3D0❶
```

```
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.)
```

Single-stepping
    Break after every instruction
    Slow, use only on small chunks of code
Step-over
    Break after next function returns
    Quick, risk missing something
Step-into
    Break on first instruction of called function
    Can get lost down a rabbit hole, step-out or restart if you go too deep

Breakpoints

- Stop execution to examine the state of memory, registers, etc
  - View data before it gets encoded, encrypted, etc
  - View the value of a parameter before a function is called
- Software (default)
  - Overwrites 1st byte of instruction w/ 0xCC (INT 3), OS will throw exception
  - Breakpoint may be (unintentionally) accessed or modified after set
- Hardware
  - X86 - dedicated hardware registers check if instruction ptr = breakpoint address
  - Can break on access (r, w, both) instead of execution
  - Only 4 (DR0 - DR3) registers, can be modified by program (intentionally)
    - General Detect flag (x86 feature) can detect any *mov* modification of a debug register
  - Debug control register (DR7) holds metadata on DR0-3

Stop execution to examine the state of memory, registers, etc
View data before it gets encoded, encrypted, etc
View the value of a parameter before a function is called
Software (default)
Overwrites 1st byte of instruction w/ 0xCC (INT 3), OS will throw exception
Breakpoint may be (unintentionally) accessed or modified after set
Hardware
X86 - dedicated hardware registers check if instruction ptr = breakpoint address
Can break on access (r, w, both) instead of execution
Only 4 (DR0 - DR3) registers, can be modified by program (intentionally)
General Detect flag (x86 feature) can detect any mov modification of a debug register
Debug control register (DR7) holds metadata on DR0-3

Conditional Breakpoints

Break if <condition> is true

- Ex. *GetProcAddress* is called AND param *RegSetValue*
- Debugger receives exception, checks for condition
  - Can really slow down execution

Break if <condition> is true
Ex. GetProcAddress is called AND param RegSetValue
Debugger receives exception, checks for condition
Can really slow down execution

Exceptions

Breakpoints, invalid memory access, divide by zero, etc

1. Debugger gets **first-chance** at handling exception
2. Program gets to handle w/ registered exception handler
3. Debugger gets **second-chance** (instead of program crash)

Common exceptions:

- INT 3 exception for debuggers
- Trap processor flag for single-stepping
- Memory-access violation (invalid, permissions)

Breakpoints, invalid memory access, divide by zero, etc

1. Debugger gets first-chance at handling exception
2. Program gets to handle w/ registered exception handler
3. Debugger gets second-chance (instead of program crash)

Common exceptions:

INT 3 exception for debuggers
Trap processor flag for single-stepping
Memory-access violation (invalid, permissions)

Resources

WinDbg kernel debugger

OllyDbg debugger