Chapter 16

Anti-Debugging
Anti Debugging

- Anti-analysis technique for malware to recognize when it is under the control of a debugger
  - Slow down analysis as much as possible to increase window of vulnerability
  - Once found in debugger, cause a crash
  - Popular techniques among hundreds of techniques
Windows Debugger Detection

Most common way: use the Windows API
- `IsDebuggerPresent()` returns 0 if no debugger attached by searching the Process Environment Block for field `IsDebuggerPresent`
- `CheckRemoteDebuggerPresent()` allows one to check the `IsDebuggerPresent` flag on other processes in local machine
- `NTQueryInformationProcess` using value `ProcessDebugPort` to tell if a process is currently being debugged
- `OutputDebugString` send string to debugger for display
  - `errorValue` set if no debugger attached (implicitly telling a debugger is in use)
Manually Checking Structures

- Manual checks of structures in memory
  - Most common way
  - Windows API may not be reliable
  - Bypass Windows API to check memory directly - preferred by malware

- **BeingDebugged flag (p. 353)**
  - Loading of PEB (Process Environment Block) structure address fs:[30h]
  - Followed by access of BeingDebugged flag at offset 0x2

```assembly
mov method
mov eax, dword ptr fs:[30h]
mov ebx, byte ptr [eax+2]
test ebx, ebx
jz NoDebuggerDetected
```

Check if EBX is zero, if so debugger is not attached>jmp

Hide Debugger in OllyDbg to disable the beingDebugged flag
Manually Checking Structures (undocumented)

- Debugger heap check – doing something to the heap
  - First heap -> has a header telling whether the heap was creating within a debugger (ForceFlags/Flags)
  - Get address of process’s first heap (ProcessHeap) by loading value at 0x18 into PEB structure, then accessing flag field at 0x10 (XP) or 0x44 (Win7) (Listing 16-3, p. 355)
  - Hide-debug plug-in/manual change of ProcessHeap flag

```
mov eax, large fs:30h
mov eax, dword ptr [eax+18h]
cmp dword ptr ds:[eax+10h], 0
jne DebuggerDetected
```
typedef struct _PEB {
    BYTE Reserved1[2];
    BYTE BeingDebugged;
    BYTE Reserved2[1];
    PVOID Reserved3[2];
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    BYTE Reserved4[104];
    PVOID Reserved5[52];
    PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
    BYTE Reserved6[128];
    PVOID Reserved7[1];
    ULONG SessionId;
} PEB, *PPEB;

Listing 16-2: Documented Process Environment Block (PEB) structure
Checking NTGlobalFlag

- NtGlobalFlag check
  - Process started with a debugger creates memory heaps differently
  - Specified at 0x68 offset in PEB. Set to 0x70 if debugger is running
  - Hide-debug plug-in/manual change of flag
Checking for System Residue

- Debugging tools also left residue on system
- Manual checks of system residuals
  - Registry values used by debuggers
    (HKLM\Base\AeDebug) – if set to OllyDbg (in use)
  - Perform `FindWindow` in search for debugger (e.g. Window Name `OLLYDBG`)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Default)</td>
<td>REG_SZ</td>
<td>(value not set)</td>
</tr>
<tr>
<td>Auto</td>
<td>REG_SZ</td>
<td>1</td>
</tr>
<tr>
<td>Debugger</td>
<td>REG_SZ</td>
<td>&quot;C:\Documents and Settings\My Documents\Tools\oldbg110\OLLYDBG.EXE&quot; -ADEBUG %Id %Id 0x00000000 (C)</td>
</tr>
</tbody>
</table>
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug`
Identifying Debugger Behavior

- INT scanning
  - INT 3 – software interrupt
  - Inserted by debugger to temporarily replace an instruction so that debug exception handler can run when software breakpoints are hit (Opcode 0xCC inserted)
  - Search for 0xCC in code – overcome by hardware interrupt

- Performing code checksums
  - Malware performs checksum (CRC/MD5) on its code pages and exits if tampering detected (if 0xCC inserted, code is changed)

- Timing checks
  - Slowdown while being debugged
  - Malware takes timestamps and exits if there is a lag
  - Especially effective when taken before & after an exception (human interventions to handle exception – delay)
Timing Checks

- Implemented via `rdtsc` instruction
  - Count the number of ticks since the last system reboot
  - Store value in EDX:EAX
  - Execute twice, compare difference two calls – whether too large
  - `QueryPerformanceCounter` – Can use this API as well
- GetTickCount – returns the number of milliseconds elapsed since last system reboot

```c
a = GetTickCount();
MaliciousActivityFunction();
b = GetTickCount();
```

Check to see the time increase whether you are single stepping