Captain Hook: Pirating AVS to Bypass Exploit Mitigations
WHO?

Udi Yavo
- CTO and Co-Founder, enSilo
- Former CTO, Rafael Cyber Security Division
- Researcher
- Author on BreakingMalware

Tomer Bitton
- VP Research and Co-Founder, enSilo
- Low Level Researcher, Rafael Advanced Defense Systems
- Malware Researcher
- Author on BreakingMalware
AGENDA

- Hooking In a Nutshell
- Scope of Research
- Inline Hooking – Under the hood
  - 32-bit function hooking
  - 64-bit function hooking
- Hooking Engine Injection Techniques
- The 6 Security Issues of Hooking
- Demo – Bypassing exploit mitigations
- 3rd Party Hooking Engines
- Affected Products
- Research Tools
- Summary
*Hooking is used to intercept function calls in order to alter or augment their behavior*

*Used in most endpoint security products:*
  * Anti-Exploitation – EMET, Palo-Alto Traps, ...
  * Anti-Virus – Almost all of them
  * Personal Firewalls – Comodo, Zone-Alarm,...
  * ...

*Also used in non-security products for various purposes:*
  * Application Performance Monitoring (APM)
  * Application Virtualization (Microsoft App-V)

*Used in Malware:*
  * Man-In-The-Browser (MITB)
Our research encompassed about a dozen security products

Focused on user-mode inline hooks – The most common hooking method in real-life products

Hooks are commonly set by an injected DLL. We’ll refer to this DLL as the “Hooking Engine”

Kernel-To-User DLL injection techniques

• Used by most vendors to inject their hooking engine
• Complex and leads security issues
Inline Hooking
INLINE HOOKING – 32-BIT FUNCTION HOOKING

Straight forward most of the time:

1. Disassemble Prolog
2. Allocate Code Stub
3. Copy Prolog Instructions
4. Patch the Prolog with a JMP
INLINE HOOKING – 32-BIT FUNCTION HOOKING

InternetConnectW before the hook is set:

```
0:000:x86> u WININET!InternetConnectW
WININET!InternetConnectW:
77090ec0 8bff     mov    edi,edi
77090ec2 55       push   ebp
77090ec3 8bec     mov    ebp,esp
77090ec5 83e4f8   and    esp,0FFFFFFFF8h
77090ec8 83ec7c   sub    esp,7Ch
77090ecb 53       push   ebx
77090ecc 56       push   esi
77090ecd 57       push   edi
```

InternetConnectW After the hook is set:

```
0:014:x86> u WININET!InternetConnectW
WININET!InternetConnectW:
77090ec0 e97b7a0e89 jmp    00178940
77090ec3 83e4f8   and    esp,0FFFFFFFF8h
77090ec8 83ec7c   sub    esp,7Ch
77090ecb 53       push   ebx
77090ecc 56       push   esi
77090ecd 57       push   edi
```
### The hooking function (0x178940)

<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>00178940 55</td>
<td>push ebp</td>
</tr>
<tr>
<td>00178941 8bec</td>
<td>mov ebp,esp</td>
</tr>
<tr>
<td>00178943 53</td>
<td>push ebx</td>
</tr>
<tr>
<td>00178944 8b5d1c</td>
<td>mov ebx,word ptr [ebp+1Ch]</td>
</tr>
<tr>
<td>00178947 56</td>
<td>push esi</td>
</tr>
<tr>
<td>00178948 57</td>
<td>push edi</td>
</tr>
<tr>
<td>00178949 ff7524</td>
<td>push dword ptr [ebp+24h]</td>
</tr>
<tr>
<td>0017894c 33f6</td>
<td>xor esi,esi</td>
</tr>
<tr>
<td>0017894e ff7520</td>
<td>push dword ptr [ebp+20h]</td>
</tr>
<tr>
<td>00178951 53</td>
<td>push ebx</td>
</tr>
<tr>
<td>00178952 ff7518</td>
<td>push dword ptr [ebp+18h]</td>
</tr>
<tr>
<td>00178955 ff7514</td>
<td>push dword ptr [ebp+14h]</td>
</tr>
<tr>
<td>00178958 ff7510</td>
<td>push dword ptr [ebp+10h]</td>
</tr>
<tr>
<td>0017895b ff750c</td>
<td>push dword ptr [ebp+0Ch]</td>
</tr>
<tr>
<td>0017895e ff7508</td>
<td>push dword ptr [ebp+8]</td>
</tr>
<tr>
<td>00178961 ff152cf21900</td>
<td>call dword ptr [0019f22c]</td>
</tr>
</tbody>
</table>

### The Copied Instructions

<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.014:x060 u poi(0019f22c)</td>
<td></td>
</tr>
<tr>
<td>03110000 0bff</td>
<td>mov    edi,edi</td>
</tr>
<tr>
<td>03110002 55</td>
<td>push   ebp</td>
</tr>
<tr>
<td>03110003 8b6c</td>
<td>mov    ebp,esp</td>
</tr>
<tr>
<td>03110007 e9bb0ef873</td>
<td>jmp WININET!InternetConnectW+0x5 (77090ec5)</td>
</tr>
<tr>
<td>03110016 90</td>
<td>nop</td>
</tr>
<tr>
<td>03110018 90</td>
<td>nop</td>
</tr>
<tr>
<td>0311001a 90</td>
<td>nop</td>
</tr>
</tbody>
</table>

#### Original Function Code

- Push ebp
- Move ebp, esp
- Push ebx
- Move ebx, word ptr [ebp+1Ch]
- Push esi
- Push edi
- Push dword ptr [ebp+24h]
- XOR esi, esi
- Push dword ptr [ebp+20h]
- Push ebx
- Push dword ptr [ebp+18h]
- Push dword ptr [ebp+14h]
- Push dword ptr [ebp+10h]
- Push dword ptr [ebp+0Ch]
- Push dword ptr [ebp+8]
- Call dword ptr [0019f22c]
Other Techniques:

- One Byte Patching (Malware) - Patch with an illegal instruction and catch in the exception handler
- Microsoft Hot Patching – Only 2 bytes function prolog overwrite

Some Possible Complications:

- Relative jmp/call in the prolog
- Very short functions/short prolog
- jmp/jxx to the middle of the prolog’s instruction
- ...
• More complex
• 5 bytes jmp instruction might not be enough (limited to a 2GB range)
- InternetConnectA before the hook is set:

```
0:000> u WININET!InternetConnectA
WININET!InternetConnectA:
0000007fe3b70a0 48935c2408 mov qword ptr [rsp+8],rbx
0000007fe3b70a5 4889e6c2410 mov qword ptr [rsp+10h],rbp
0000007fe3b70a8 4899742418 mov qword ptr [rsp+16h],rsi
0000007fe3b70af 57 push rdi
```

- InternetConnectA after the hook is set:

```
0:009> u WININET!InternetConnectA
WININET!InternetConnectA:
0000007fe3b70a0 e95b7fe4ff jmp 0000007fe3ff000
0000007fe3b70a5 58 pop rax
0000007fe3b70a6 90 nop
0000007fe3b70a7 90 nop
0000007fe3b70a8 90 nop
0000007fe3b70a9 90 nop
```

- Trampoline code:

```
0:009> u 000007fe3ff000
0000007fe3ff000 4889742418 mov qword ptr [rbp-18h],rsi
```

INLINE HOOKING – 64-BIT FUNCTION HOOKING
If we follow the hooking function we get:

```
00000000 03880000 48895c2460  mov  qword ptr [rsp+8],rbx
00000000 03880005 48899c2410  mov  qword ptr [rsp+10h],rbp
00000000 0388000a 50      push  rax
00000000 0388000b 482b0a5703bfe0000  mov  rax,offset UININET!InternetConnectA+0x5 (00000007'e'fe3b70a5)
00000000 03880015 ffe0      jmp  rax
```

Original Function Code
Other Techniques:
- 6 Bytes patching (requires hooks’ code stub to be in 32-bit address)
- Double Push (Nikolay Igotti) – Preserves all registers
- Possible Complications:
  - Similar to 32-bit hooks
  - More instruction pointer relative instructions:

```asm
MOV RAX, QWORD [RIP+0x15020]
```

Jumps to 0x7ffc00030000
Inline hooking is the most common hooking technique in real-life products
Rather intrusive – modifies the code of the hooking function
Used by most endpoint security products
More on hooking:
- Binary Hooking Problems - By Gil Dabah
- Trampolines in X64 - By Gil Dabah
- Powerful x86/x64 Mini Hook-Engine - Daniel Pistelli
- Inline Hooking for Programmers - Malware Tech
- ...
Kernel-To-User Code Injections
Mainly used for:
  • Injecting DLLs
  • Sandbox escapes – After exploiting privilege escalation vulnerability
  • Injecting to protected processes

Fewer techniques exist than user-mode

Less documented than user-mode techniques

Used by both Malware and Software/Security vendors
The most common Kernel-To-User injection method

Used by lots of malwares:
- TDL
- ZERO ACCESS
- Sandbox escape shellcodes
- ...

Also used by lots of security products:
- AVG
- Kaspersky Home Edition
- Avecto
- ...

Documented:
- Blackout: What Really Happened
- Much more in forums and leaked source codes
Basic Steps (There are several variations):

1. Register load image callback using PsSetLoadImageNotifyRoutine

2. Write payload that injects a dll using LdrLoadDll
   (Other variations use LoadLibrary)

3. Insert User APC using KeInsertQueueApc
• Not really common but worth mentioning
• Used by Duqu
• Fully documented in:
INJECTION METHODS – ENTRY POINT PATCHING

• Register load image callback using PsSetLoadImageNotifyRoutine and wait for main module to load
INJECTION METHODS – ENTRY POINT PATCHING

- Write the payload to the process address space

Kernel Space

User Space
INJECTION METHODS – ENTRY POINT PATCHING

- Replace the image entry point with **JMP** to the new code
INJECTION METHODS – ENTRY POINT PATCHING

- The payload executes, fixes the entry point and jumps to it
INJECTION METHODS – ENTRY POINT PATCHING

- Internet Explorer patched entrypoint

Load the hooking engine

Restore the code of the entrypoint

Jump back to the entrypoint
First published on Codeless-Code-Injections talk (to our knowledge)

Never been used by malware (to our knowledge)

Used by software and security vendors:
  • Symantec
  • Trusteer
  • Microsoft App-V

Similar method could probably use TLS data directory
INJECTION METHODS – IMPORT TABLE PATCHING

1. Register load image callback using PsSetLoadImageNotifyRoutine and wait for main module to load

2. Allocate memory for the new import table and copy old table with a new record for the injected DLL

3. Point the import data directory to the new table

4. When the DLL is loaded the original PE header is restored
INJECTION METHODS – IMPORT TABLE PATCHING

Internet Explorer patched import table

Import Directory RVA is out of image
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

- Register load image callback using PsSetLoadImageNotifyRoutine and wait for ntdll.dll module to load
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

- Write the payload to the process address space
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

- Replace the LdrLoadLibrary prolog with **JMP** (or equivalent) to the payload
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

• The payload loads a dll, fixes LdrLoadDll and jumps to it
• Kernel-To-User Injections are extensively used by both malware and security/software products
• Kernel injections are mainly used to inject a DLL to target processes
• In security products the injected DLL is commonly the hooking engine
• Prone to mistakes – due to its relative complexity
The 6 security issues of hooking
#1 – UNSAFE INJECTION

Severity: Very High  
Affected Systems: All Windows Versions

Occurs due to bad DLL injection implementation

- We found 2 types of unsafe injections:
  - LoadLibrary from a relative path – vulnerable to DLL Hijacking
  - Unprotected injected DLL file – placed in `%appdata%\Local\Vendor`  
    Can easily be replaced by the attacker
#2 – PREDICTABLE RWX CODE STUBS

Severity: Very High
Affected Systems: All Windows Versions

The Kernel-To-User DLL injection allocates RWX code in a predictable location

Functions pointers in constant addresses

• Implications:
  • ASLR Bypass – The code stubs normally contain addresses of critical OS functions
  • Great for shellcode – Allows writing malicious code to the allocated code-stub
Severity: Very High
Affected Systems: All Windows Versions

The Kernel-To-User DLL injection or hooking engine allocates R-X code in a predictable location

Implications:
- ASLR Bypass – The code stubs contain the addresses of critical OS functions
- Hooks Bypass – Calling the hook code stub effectively bypasses the hook
- Code Reuse – The code can also be useful for ROP
Severity: High
Affected Systems: Windows 7 and Below

The Kernel-To-User DLL injection allocates RWX code without specifying exact address

Implications:
• Similar to the first predictable RWX Code issue
Severity: Medium
Affected Systems: All Windows Versions

The most common issue: most hooking engines leave their hook code stubs as RWX

The implication - possible CFG bypass:
- Get arbitrary read/write in the target process
- Find the hook’s stub (R)
- Overwrite it (W)
- Trigger the execution of the hooked function (X)

* Note: Attacker with arbitrary read/write will probably succeed anyway
Some hooking engines leave the code of the hooked modules as RWX

The implication - possible CFG bypass
## SECURITY ISSUES OF HOOKING - RECAP

<table>
<thead>
<tr>
<th>Issue</th>
<th>Severity</th>
<th>Affected underlying systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unsafe injection</td>
<td>Very high</td>
</tr>
<tr>
<td>2</td>
<td>Predictable RWX code stubs</td>
<td>Very high</td>
</tr>
<tr>
<td>3</td>
<td>Predictable RX code stubs</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Predictable RWX code stubs</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>RWX hook code stubs</td>
<td>Medium</td>
</tr>
<tr>
<td>6</td>
<td>RWX hooked modules</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Demo

Bypassing Exploit Mitigations
3rd Party Hooking Engines
Developing a hooking engine is not an easy task

Using open-source* or commercial hooking engines has many advantages:
  • Easy API to work with
  • Supports many platforms
  • Saves development effort
  • Saves testing effort

3rd party hooking engines are also integrated into non-security products

A security issue in a hooking engine results in many patches...

* We really like Gil Dabah’s distormx
Used by many open-source projects
Also used by a few security vendors. For example, Vera

Features:
- Kernel Hooking support
- Thread Deadlock Barrier
- RIP-relative address relocation for 64-bit
- ...

Security Issues:
- RWX Hook Code Stubs
- RWX Hooked Modules

Bad Practice:
- Uses Non-Executable heap and changes parts of it to code
Several Fortune 500 companies are using Deviare technology for application virtualization, packaging, and troubleshooting, and for computer security.

Dual License – Commercial or GPL for open-source
Fixed the issues quickly
From their web site:

“Several Fortune 500 companies are using Deviare technology for application virtualization, packaging, and troubleshooting, and for computer security.”

Features:
- Defer Hook – Set a hook only when and if a module is loaded
- .NET Function hooking
- Interface for many languages: (C++, VB, Python, C#,...)
- ...

Security Issues:
- RWX Hook Code Stubs
MADCODEHOOK – POWERFUL COMMERCIAL HOOKING

- Used by a lot for security vendors (75% of its users)
- Used by emsisoft
- Fixed the issues quickly

Features:
- Injection Driver – Used to perform kernel-injection into processes
- IPC API – Used to easily communicate with some main process
- IAT Hooking
- ...

Security Issues:
- RWX Hook Code Stubs
The most popular hooking engine in the world
Microsoft’s App-V uses Detours which is integrated into Office
We were surprised to find out that it has problems too...

Features:
- ARM support
- ...

Security Issues:
- Predictable RX (Universal).

* Details won’t be revealed until the patch is released (September)
Microsoft’s hooking engine Detours – via Microsoft.com:

“Under commercial release for over 10 years, Detours is licensed by over 100 ISVs and used within nearly every product team at Microsoft.”

- Could potentially affect millions of users
- Also used in security products
- Hard to patch - In most cases fixing this issue requires recompilation of each product individually which makes patching cumbersome
Affected Products
<table>
<thead>
<tr>
<th>Products/Vendors</th>
<th>UnSafe Injection</th>
<th>Predictable RWX(Universal)</th>
<th>Predictable RX(Universal)</th>
<th>Predictable RWX</th>
<th>RWX Hook code stubs</th>
<th>RWX Hooked Modules</th>
<th>Time To Fix (Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symantec</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>McAfee</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>Trend Micro</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>210</td>
</tr>
<tr>
<td>Kaspersky</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>90</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>30</td>
</tr>
<tr>
<td>BitDefender</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>30</td>
</tr>
<tr>
<td>WebRoot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>29</td>
</tr>
<tr>
<td>AVAST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>30</td>
</tr>
<tr>
<td>Emsisoft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>90</td>
</tr>
<tr>
<td>Citrix - Xen Desktop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>90</td>
</tr>
<tr>
<td>Microsoft Office*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>180</td>
</tr>
<tr>
<td>WebSense</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>30</td>
</tr>
<tr>
<td>Vera</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>Invincea</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>Anti-Exploitation*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>BeyondTrust</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fixed Independently</td>
</tr>
<tr>
<td><strong>TOTALS</strong></td>
<td><strong>2</strong></td>
<td><strong>2</strong></td>
<td><strong>6</strong></td>
<td><strong>8</strong></td>
<td><strong>7</strong></td>
<td><strong>5</strong></td>
<td><strong>79.9</strong></td>
</tr>
</tbody>
</table>
Research Tools
• Tool to detect predictable RWX code regions
• Can be found at https://github.com/BreakingMalware/AVulnerabilityChecker
• Compares memory maps of processes
• Tool for scanning hooks and checking their code permissions
• Compares code “On-Disk” with the code “In-Memory”
• Does best-effort to track hooks code stubs
• Code hooking is an important capability for security/software vendors
• Similar to other intrusive operations it has security implications
• Almost all the vendors we tested were vulnerable to at least one issue
• We worked closely with affected vendors to address all these issues – most are already patched
Thank You!

Contact Us:

Udi, udi@ensilo.com
Tomer, tomer@ensilo.com