
black hat
USA 2016
Captain Hook:
Pirating AVS to Bypass Exploit Mitigations
JULY 30 - AUGUST 4, 2016 / MANDALAY BAY / LAS VEGAS

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

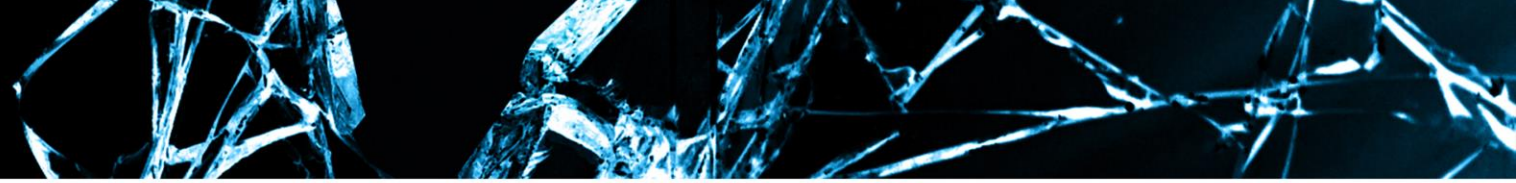
- Hooking In a Nutshell
- Scope of Research
- Inline Hooking – Under the hood
  - 32-bit function hooking
  - 64-bit function hooking
- Hooking Engine Injection Techniques
- The 6 Security Issues of Hooking
- Demo – Bypassing exploit mitigations
- 3rd Party Hooking Engines
- Affected Products
- Research Tools
- Summary

HOOKING IN A NUTSHELL

- 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)

SCOPE OF RESEARCH

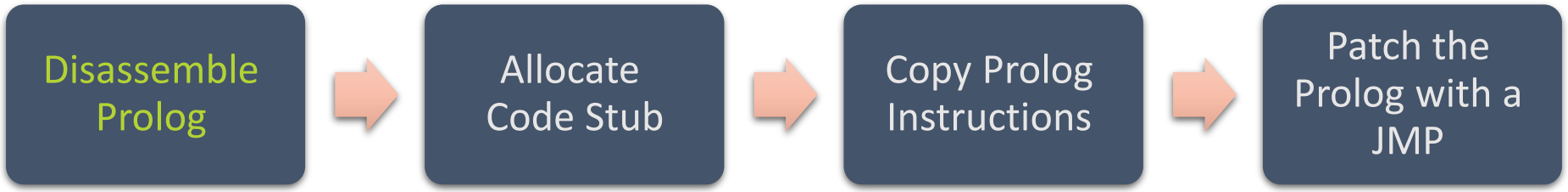
- Our research encompassed about a dozen security products
- Focused on user-mode inline hooks – The most common hooking method in real-life products
- Hooks are commonly set by an injected DLL. We'll refer to this DLL as the "Hooking Engine"
- Kernel-To-User DLL injection techniques
  - Used by most 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:



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,0FFFFFFF8h
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
77090ec5 83e4f8          and     esp,0FFFFFFF8h
77090ec8 83ec7c          sub     esp,7Ch
77090ecb 53              push    ebx
77090ecc 56              push    esi
77090ecd 57              push    edi
```

INLINE HOOKING – 32-BIT FUNCTION HOOKING

The hooking function (0x178940)

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

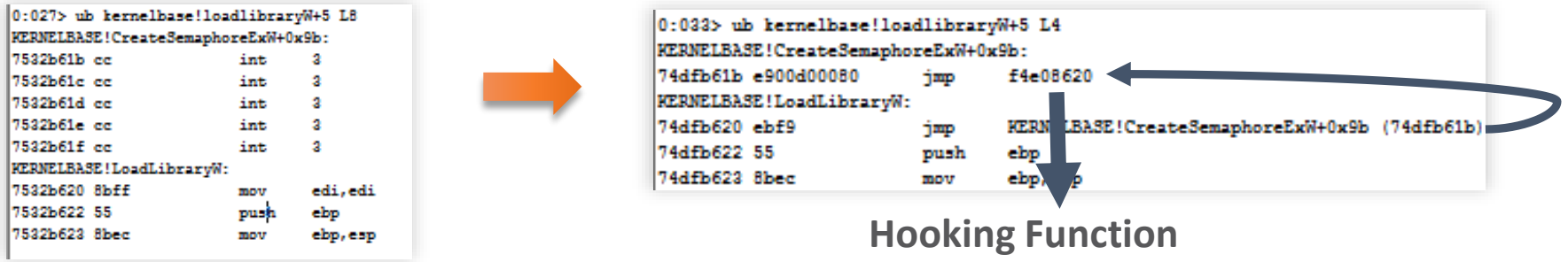
The Copied Instructions

```
0:014:x86> u poi(0019f22c)
03110000 8bff            mov     edi,edi
03110002 55              push    ebp
03110003 8bec            mov     ebp,esp
03110005 e9bb0ef873      jmp     WININET!InternetConnectW+0x5 (77090ec5)
0311000a 90              nop
0311000b 90              nop
0311000c 90              nop
```

Original Function Code

INLINE HOOKING – 32-BIT FUNCTION HOOKING

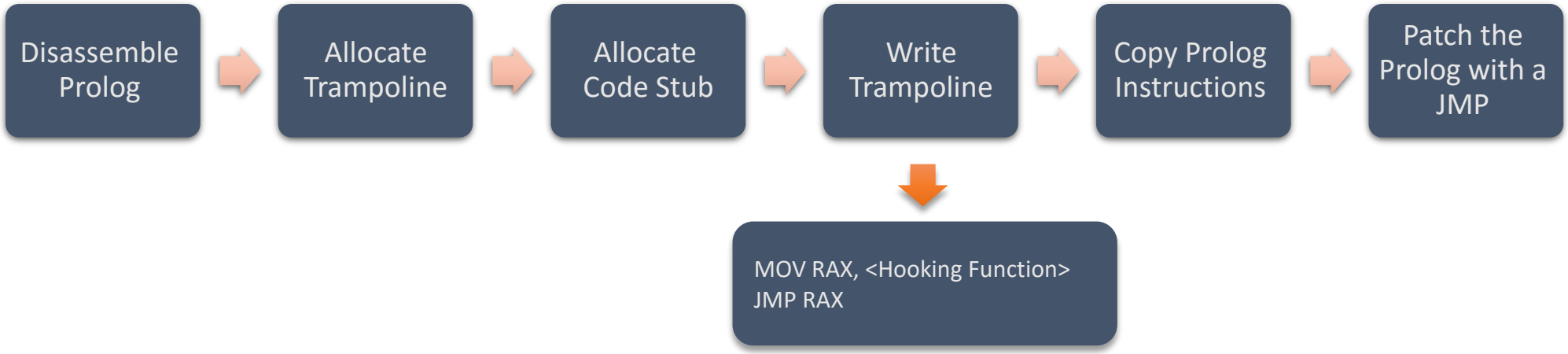
- Other Techniques:
  - One Byte Patching (Malware) - Patch with an illegal instruction and catch in the exception handler
  - Microsoft Hot Patching – Only 2 bytes function prolog overwrite



- Some Possible Complications:
  - Relative jmp/call in the prolog
  - Very short functions/short prolog
  - jmp/jxx to the middle of the prolog's instruction
  - …

INLINE HOOKING – 64-BIT FUNCTION HOOKING

- More complex
- 5 bytes jmp instruction might not be enough (limited to a 2GB range)



INLINE HOOKING – 64-BIT FUNCTION HOOKING

- InternetConnectA before the hook is set:

```
0:000> u WININET!InternetConnectA
WININET!InternetConnectA:
000007fe`fe3b70a0 48895c2408      mov     qword ptr [rsp+8],rbx
000007fe`fe3b70a5 48896c2410      mov     qword ptr [rsp+10h],rbp
000007fe`fe3b70aa 4889742418      mov     qword ptr [rsp+18h],rsi
000007fe`fe3b70af 57              push    rdi
```

- InternetConnectA after the hook is set:

```
0:009> u WININET!InternetConnectA
000007fe`fe3b70a0 e95b7fe4ff      jmp     000007fe`fe1ff000
000007fe`fe3b70a5 58              pop     rax
000007fe`fe3b70a6 90              nop
000007fe`fe3b70a7 90              nop
000007fe`fe3b70a8 90              nop
000007fe`fe3b70a9 90              nop
```

- Trampoline code:

```
0:009> u 00007fe`fe1ff000
000007fe`fe1ff000 48b8c094006800000000 mov rax,00000000`680094c0
000007fe`fe1ff00a ffe0            jmp     rax
000007fe`fe1ff00c 90              nop
000007fe`fe1ff00d 90              nop
```

INLINE HOOKING – 64-BIT FUNCTION HOOKING

If we follow the hooking function we get:

```
00000000`00380000 48895c2408      mov     qword ptr [rsp+8],rbx
00000000`00380005 48896c2410      mov     qword ptr [rsp+10h],rbp
00000000`0038000a 50              push    rax
00000000`0038000b 48b8a5703bfefe070000 mov rax,offset WININET!InternetConnectA+0x5 (000007fe`fe3b70a5)
00000000`00380015 ffe0            jmp     rax
```

Original Function Code

INLINE HOOKING – 64-BIT FUNCTION HOOKING

- Other Techniques:
  - 6 Bytes patching (requires hooks' code stub to be in 32-bit address)

```
0:004> u kernelbase!loadlibraryA
KERNELBASE!LoadLibraryA:
00007ffc`9c8d8760 6800000300      push    30000h
00007ffc`9c8d8765 c3              ret
00007ffc`9c8d8766 89742410        mov     dword ptr [rsp+10h],esi
00007ffc`9c8d876a 57              push    rdi
```

  - Double Push (Nikolay Igotti) – Preserves all registers

```
0:004> u kernelbase!loadlibrarya
KERNELBASE!LoadLibraryA:
00007ffc`9c8d8760 6800000300      push    30000h
00007ffc`9c8d8765 c7442404fc7f0000 mov     dword ptr [rsp+4],7FFCh
00007ffc`9c8d876d c3              ret
00007ffc`9c8d876e 20488b          and     byte ptr [rax-75h],cl
```

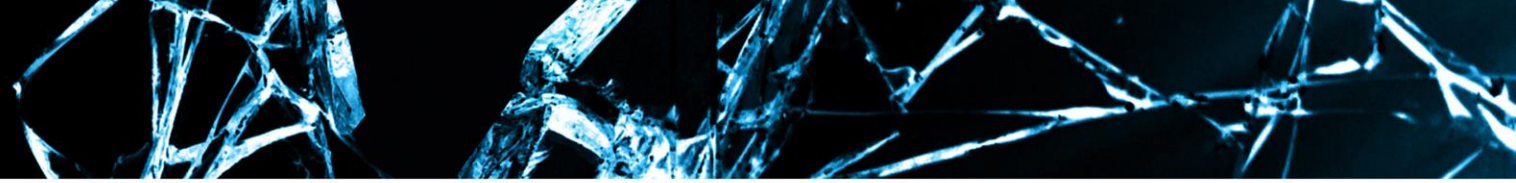
Jumps to 0x7ffc00030000

  - …
- Possible Complications:
  - Similar to 32-bit hooks
  - More instruction pointer relative instructions:

MOV RAX, QWORD [RIP+0x15020]

INLINE HOOKING – RECAP

- Inline hooking is the most common hooking technique in real-life products
- Rather intrusive – modifies the code of the of 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

INTRODUCTION - 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

INJECTION METHODS – USER APC

- 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

INJECTION METHODS – USER APC

Basic Steps (There are several variations):

1. Register load image callback using PsSetLoadImageNotifyRoutine
2. Write payload that injects a dll using LdrLoadDll

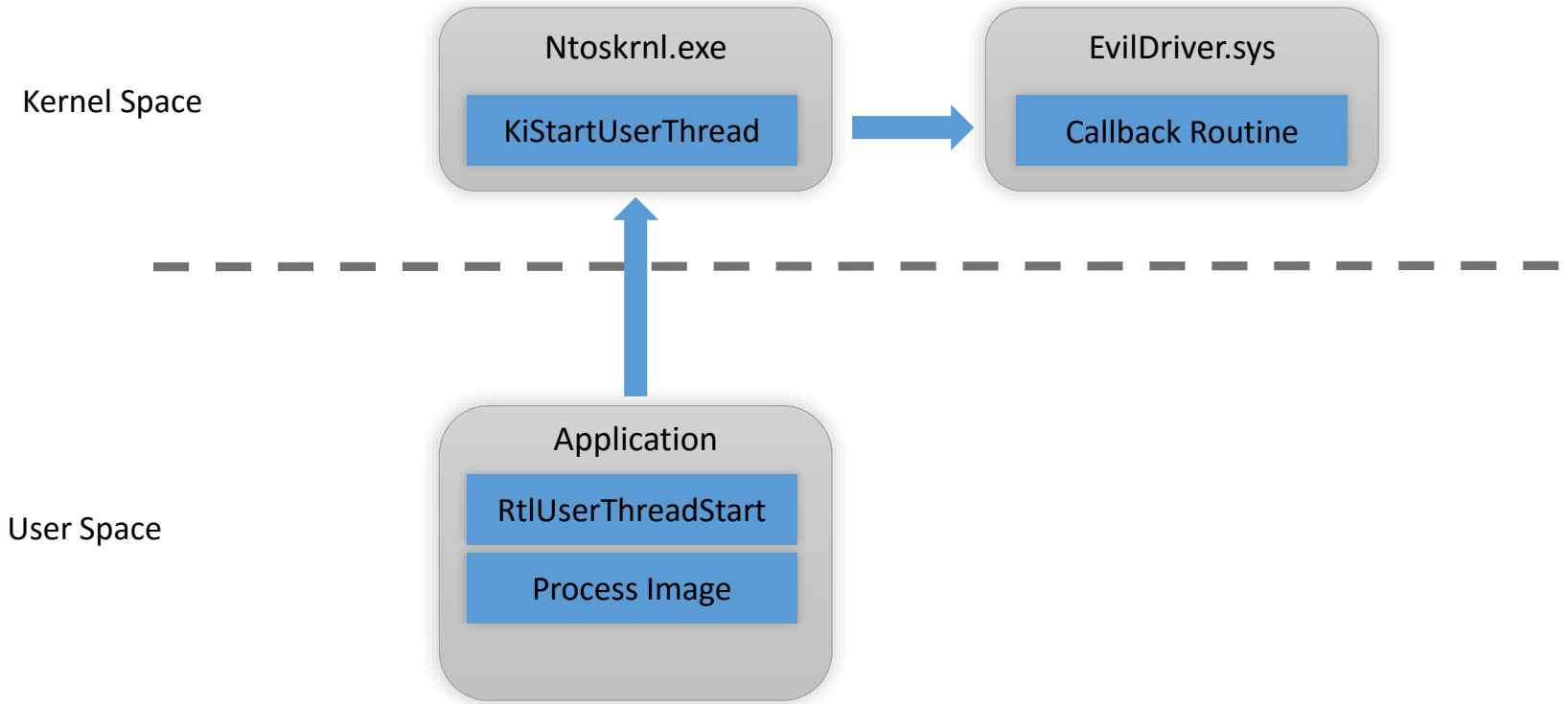
   (Other variations use LoadLibrary)
3. Insert User APC using KeInsertQueueApc

INJECTION METHODS – ENTRY POINT PATCHING

- Not really common but worth mentioning
- Used by Duqu
- Fully documented in:
  http://binsec.gforge.inria.fr/pdf/Malware2013-Analysis-Diversion-Duqu-paper.pdf

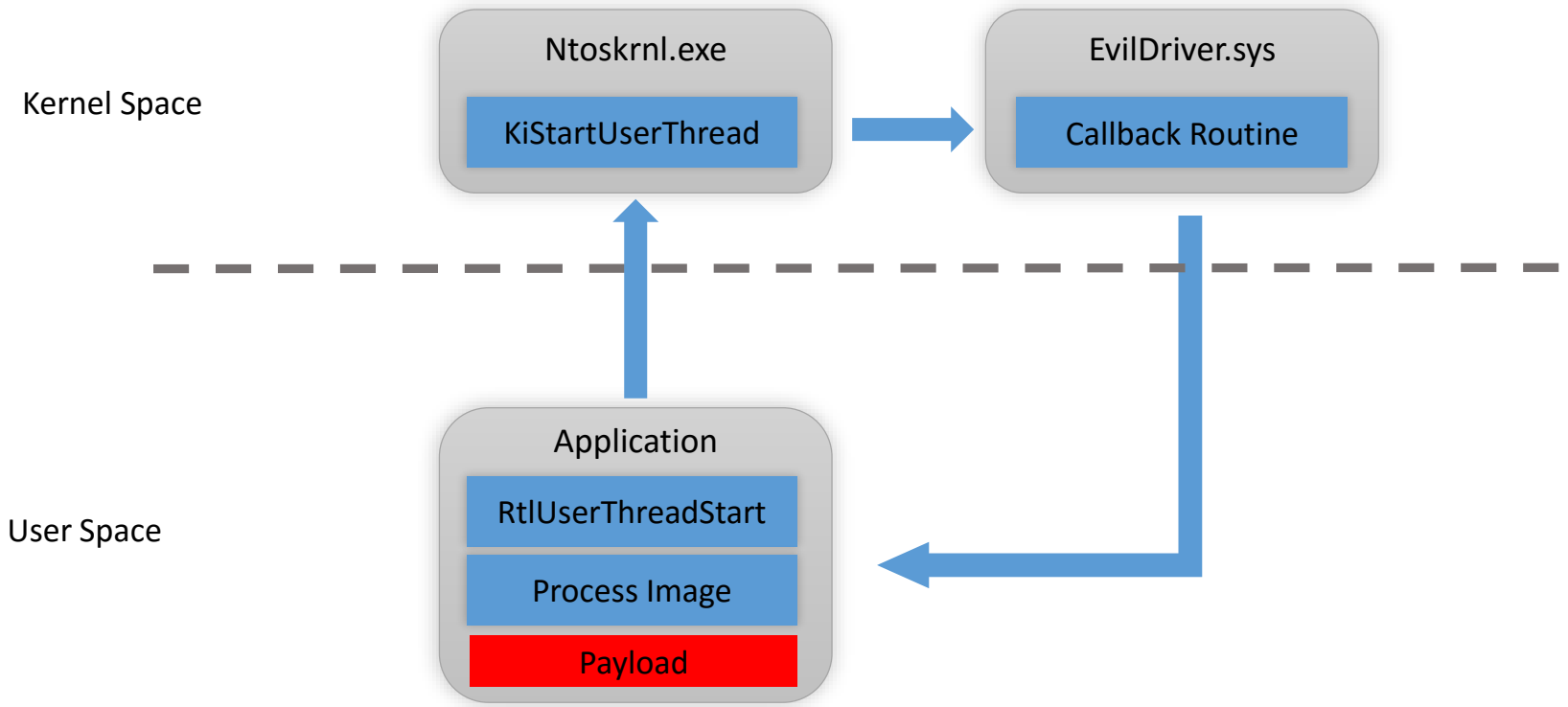
INJECTION METHODS – ENTRY POINT PATCHING

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



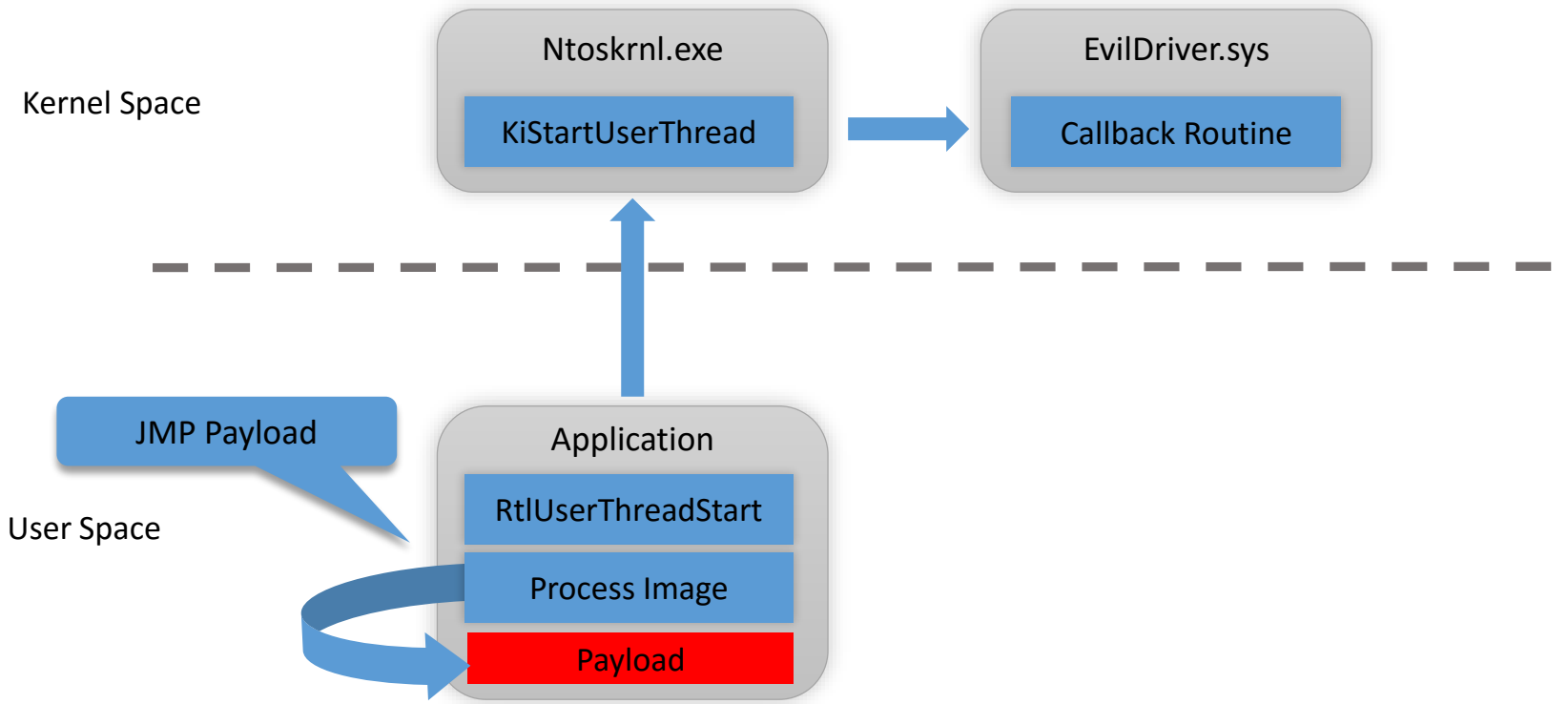
INJECTION METHODS – ENTRY POINT PATCHING

- Write the payload to the process address space



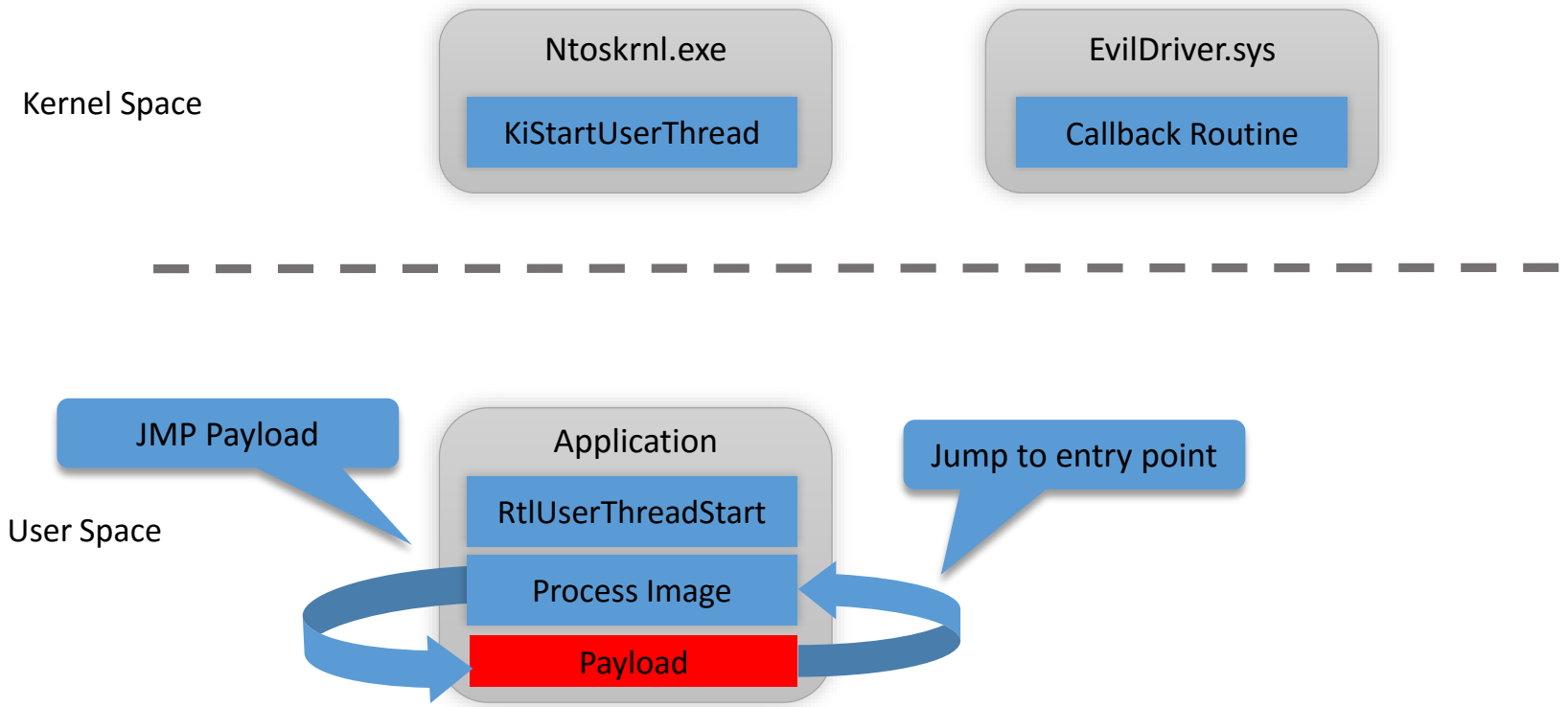
INJECTION METHODS – ENTRY POINT PATCHING

- 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

```
iexplore+0x1ddd:
00c51ddd e91ee257ff      jmp     001d0000
00c51de2 e955f9ffff      jmp     iexplore+0x173c (00c5173c)
00c51de7 90              nop
00c51de8 90              nop
00c51de9 90              nop
00c51dea 90              nop
00c51deb 90              nop
00c51dec 8bff            mov     edi,edi
```

```
0:000:x86> uf 001d0000
001d0000 55              push    ebp
001d0001 8bec            mov     ebp,esp
001d0003 83ec48          sub     esp,48h
001d0006 eb50            jmp     001d0058

001d0058 6a40            push    40h
001d005a 6808001d00      push    1D0008h
001d005f 8d45b8          lea     eax,[ebp-48h]
001d0062 50              push    eax
001d0063 b840238077      mov     eax,offset ntdll32!memcpy (77802340)
001d0068 ffd0            call    eax
001d006a 8d45b8          lea     eax,[ebp-48h]
001d006d 50              push    eax
001d006e b8f3487276      mov     eax,offset kernel32!LoadLibraryW (767248f3)
001d0073 ffd0            call    eax
001d0075 8be5            mov     esp,ebp
001d0077 5d              pop     ebp
001d0078 55              push    ebp
001d0079 8bec            mov     ebp,esp
001d007b 83ec08          sub     esp,8
001d007e c745f800000000  mov     dword ptr [ebp-8],0
001d0085 c745fc02000000  mov     dword ptr [ebp-4],2
001d008c c745f8dd1dc500  mov     dword ptr [ebp-8],offset iexplore+0x1ddd (00c51ddd)
001d0093 8d45fc          lea     eax,[ebp-4]
001d0096 50              push    eax
001d0097 6a40            push    40h
001d0099 6805000000      push    5
001d009e 8b4df8          mov     ecx,dword ptr [ebp-8]
001d00a1 51              push    ecx
001d00a2 b827437276      mov     eax,offset kernel32!VirtualProtect (76724327)
001d00a7 ffd0            call    eax
001d00a9 6805000000      push    5
001d00ae 68df001d00      push    1D00DFh
001d00b3 68dd1dc500      push    offset iexplore+0x1ddd (00c51ddd)
001d00b8 b840238077      mov     eax,offset ntdll32!memcpy (77802340)
001d00bd ffd0            call    eax
001d00bf 8d55fc          lea     edx,[ebp-4]
001d00c2 52              push    edx
001d00c3 8b45fc          mov     eax,dword ptr [ebp-4]
001d00c6 50              push    eax
001d00c7 6805000000      push    5
001d00cc 8b4df8          mov     ecx,dword ptr [ebp-8]
001d00cf 51              push    ecx
001d00d0 b827437276      mov     eax,offset kernel32!VirtualProtect (76724327)
001d00d5 ffd0            call    eax
001d00d7 8be5            mov     esp,ebp
001d00d9 5d              pop     ebp
001d00da e9fe1ca800      jmp     iexplore+0x1ddd (00c51ddd)
```

Load the hooking engine

Restore the code of the entrypoint

Jump back to the entrypoint

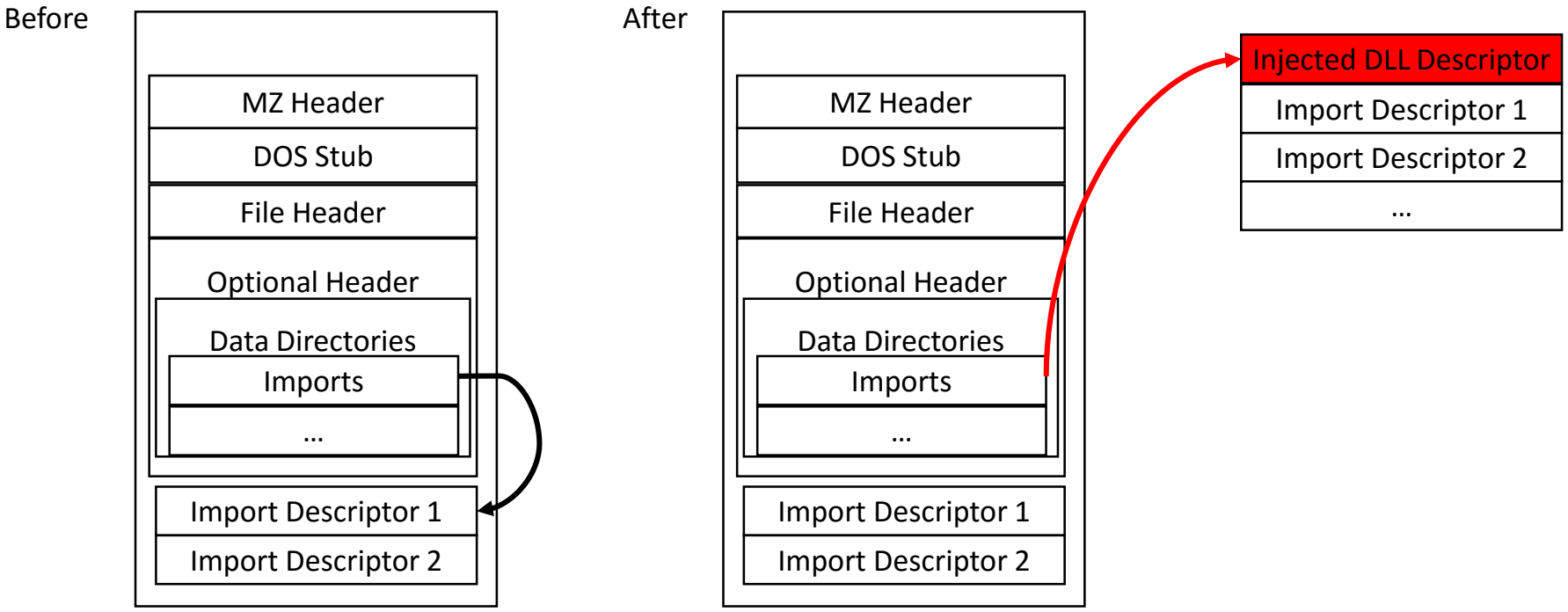
INJECTION METHODS – IMPORT TABLE PATCHING

- 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

```
0:000> !dh iexplore

File Type: EXECUTABLE IMAGE
FILE HEADER VALUES
     14C machine (i386)
       5 number of sections
53F262AC time date stamp Mon Aug 18 23:31:40 2014

       0 file pointer to symbol table
       0 number of symbols
      E0 size of optional header
     102 characteristics
           Executable
           32 bit word machine

OPTIONAL HEADER VALUES
     10B magic #
   11.00 linker version
    3A00 size of code
    BEA00 size of initialized data
       0 size of uninitialized data
    1DDD address of entry point
    1000 base of code
           ----- new -----
00000000008c0000 image base
    1000 section alignment
     200 file alignment
       2 subsystem (Windows GUI)
    6.03 operating system version
    6.03 image version
    6.01 subsystem version
   C6000 size of image
     400 size of headers
   CAEE4 checksum
0000000000100000 size of stack reserve
000000000000e000 size of stack commit
0000000000100000 size of heap reserve
0000000000001000 size of heap commit
    8040  DLL characteristics
           Dynamic base
           Terminal server aware
       0 [       0] address [size] of Export Directory
FF7C0000 [      8C] address [size] of Import Directory
    7000 [    BD408] address [size] of Resource Directory
       0 [       0] address [size] of Exception Directory
   C2800 [    3CB8] address [size] of Security Directory
   C5000 [     328] address [size] of Base Relocation Directory
    4828 [      38] address [size] of Debug Directory
       0 [       0] address [size] of Description Directory
       0 [       0] address [size] of Special Directory
       0 [       0] address [size] of Thread Storage Directory
    2D88 [      40] address [size] of Load Configuration Directory
       0 [       0] address [size] of Bound Import Directory
    6000 [     138] address [size] of Import Address Table Directory
    45E0 [      A0] address [size] of Delay Import Directory
       0 [       0] address [size] of COR20 Header Directory
       0 [       0] address [size] of Reserved Directory
```

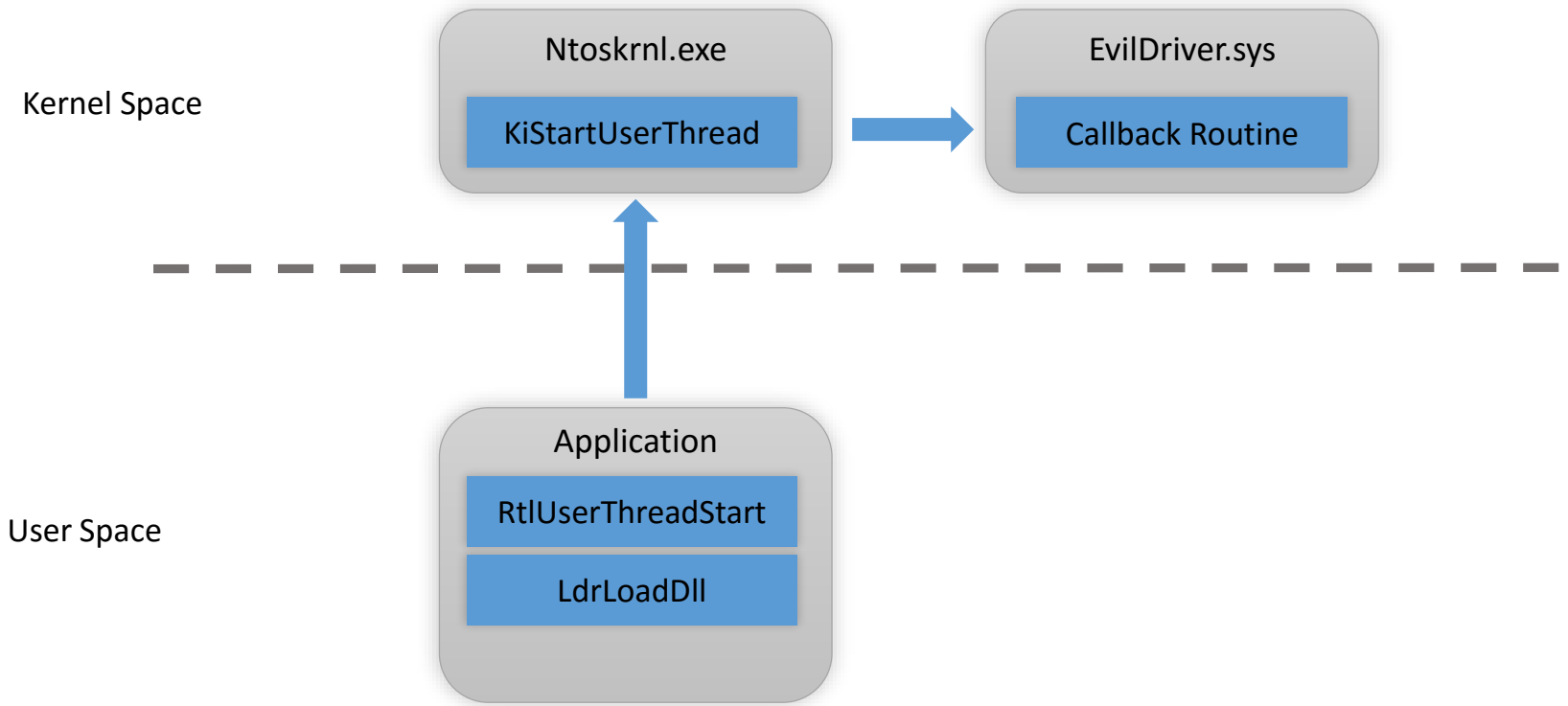
Import Directory RVA is out of image

```
0:000:x86> dd /c5 80000
00080000  ff7c009c ffffffff ffffffff ff7c00b4 ff7c008c
00080014  00006230 00000000 00000000 00006224 00006000
00080028  00006294 00000000 00000000 00006214 00006064
0008003c  00006328 00000000 00000000 000061e8 000060f8
00080050  00006348 00000000 00000000 000061d8 00006118
00080064  00006360 00000000 00000000 000061b0 00006130
```

The new row

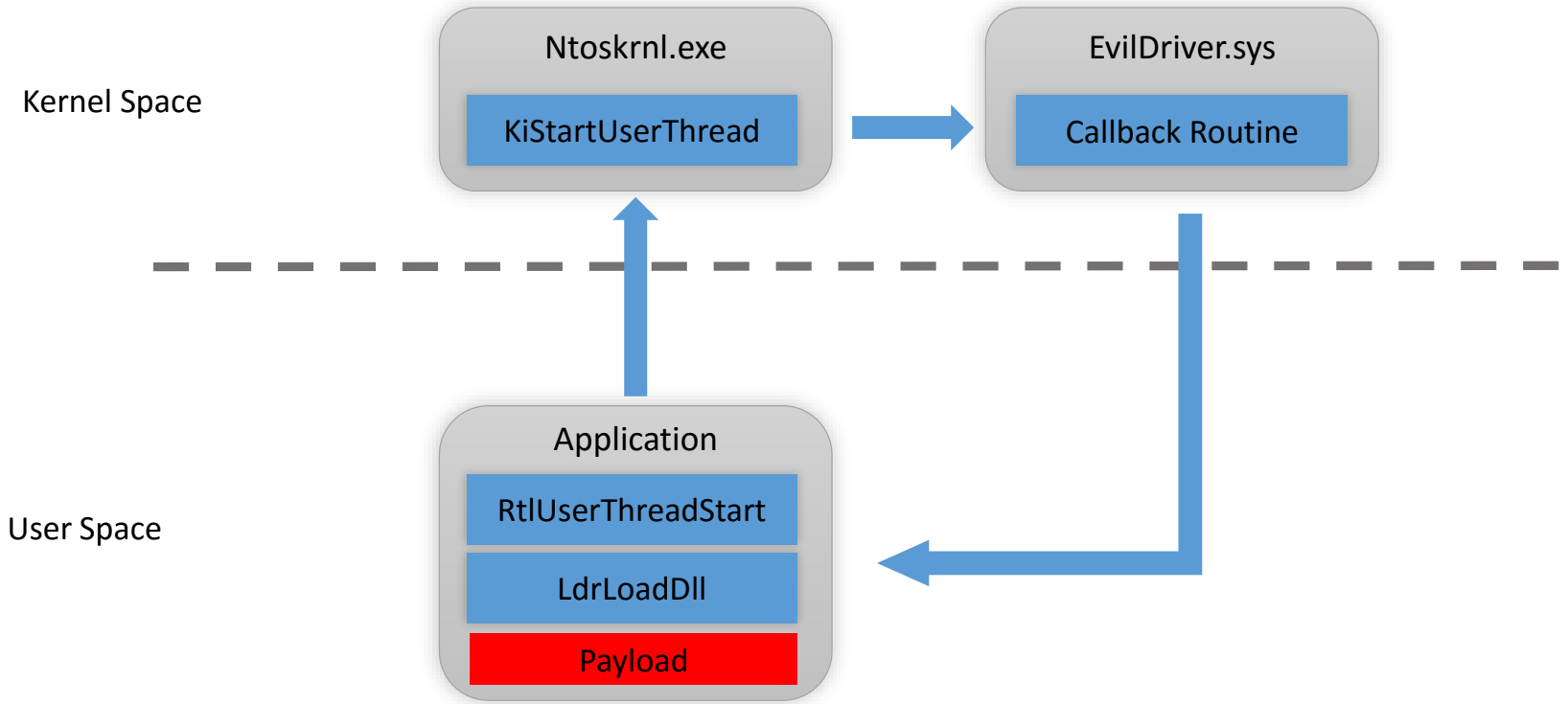
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

- Register load image callback using PsSetLoadImageNotifyRoutine and wait for ntdll.dll module to load



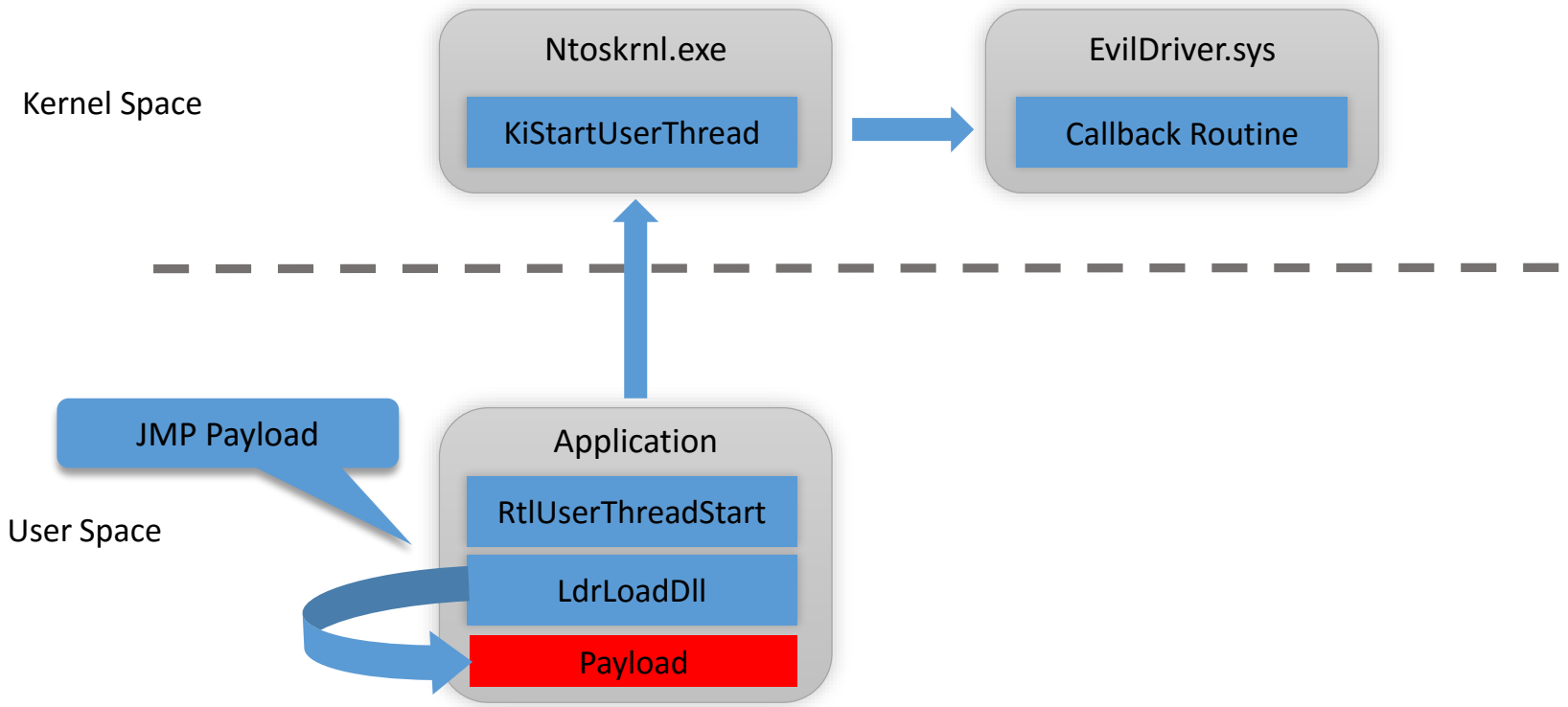
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

- Write the payload to the process address space



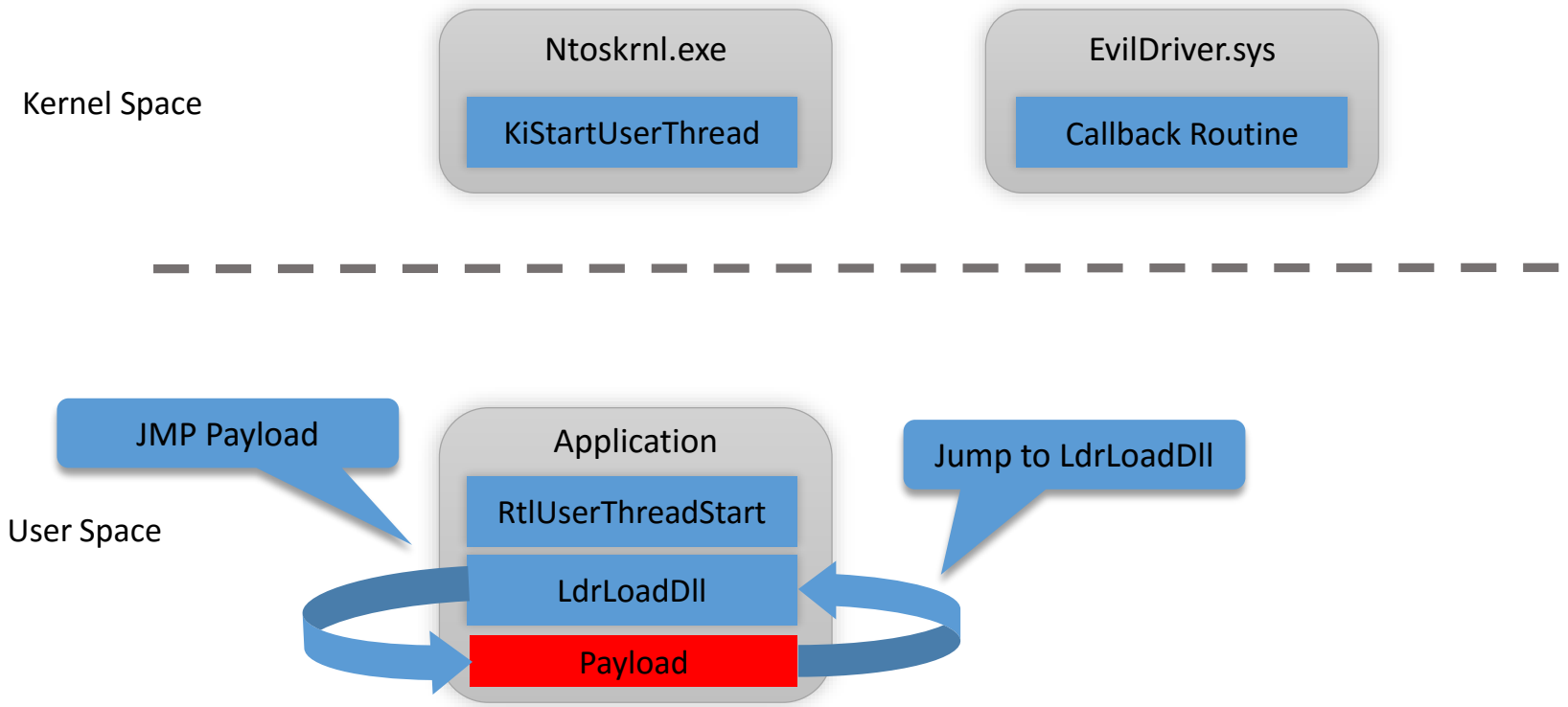
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

- Replace the LdrLoadLibrary prolog with **JMP** (or equivalent) to the payload



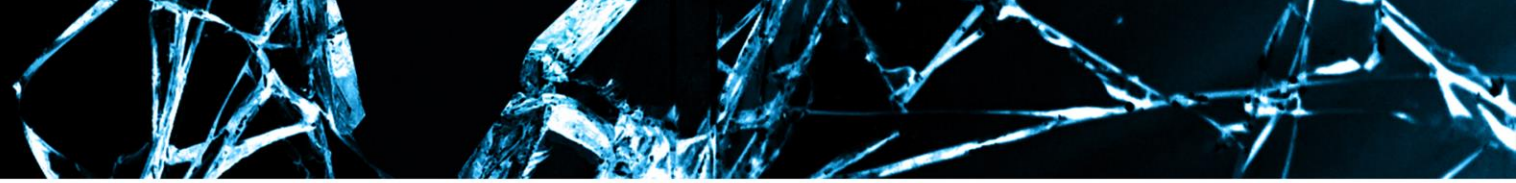
INJECTION METHODS – NTDLL.DLL/USER32.DLL PATCHING

- The payload loads a dll, fixes LdrLoadDll and jumps to it



INJECTION METHODS – QUICK SUMMARY

- 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

```
0:036> dds [redacted]410 L9
00000000`[redacted]410  77adc4dd ntdll_77aa0000!LdrLoadDll
00000000`[redacted]414  00000000
00000000`[redacted]418  00000068
00000000`[redacted]41c  cccc34c
00000000`[redacted]420  00000000
00000000`[redacted]424  00000000
00000000`[redacted]428  77adc4dd ntdll_77aa0000!LdrLoadDll
00000000`[redacted]42c  00000000
00000000`[redacted]430  77ace813 ntdll_77aa0000!RtlEqualUnicodeString
```

Functions pointers in constant addresses

```
0:036> !address [redacted]410
Usage:                  <unclassified>
Allocation Base:        00000000`[redacted]0000
Base Address:           00000000`[redacted]0000
End Address:            00000000`[redacted]1000
Region Size:            00000000`00001000
Type:                   00020000     MEM_PRIVATE
State:                  00001000     MEM_COMMIT
Protect:                00000040     PAGE_EXECUTE_READWRITE
```

RWX Permissions

- Implications:
  - ASLR Bypass – The code stubs normally contains addresses of critical OS functions
  - Great for shellcode – Allows writing malicious code to the allocated code-stub

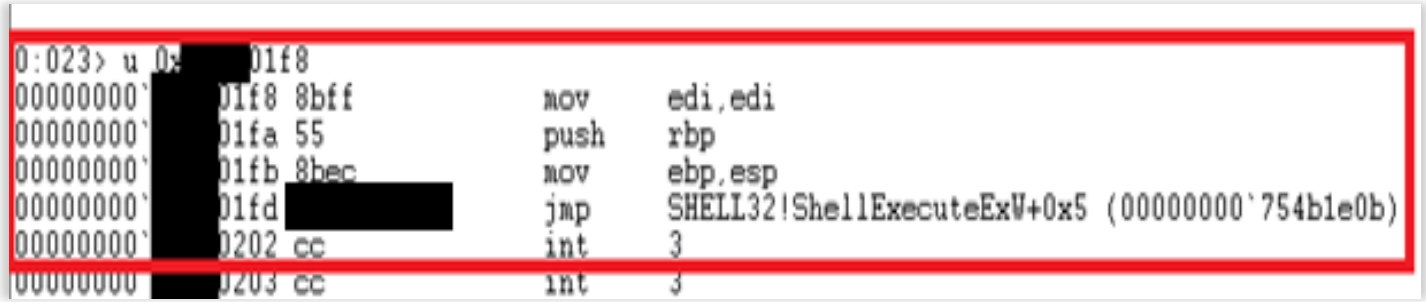
#3 – PREDICTABLE R-X CODE STUBS

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



#4 – PREDICTABLE RWX CODE STUBS 2

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

#5 –RWX CODE STUBS

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

#6 –RWX HOOKED MODULES

Severity: Medium
Affected Systems: All Windows Versions

Some hooking engines leave the code of the hooked modules as RWX

The implication - possible CFG bypass

```
0:000> u ntdll!ldrloaddll
ntdll!LdrLoadDll:
77be2576 6813040178      push    78010413h
77be257b c3              ret
77be257c cc              int     3
77be257d 90              nop
77be257e 48              dec     eax
77be257f 78bd            js      ntdll!RtlLengthRequiredSid+0x16 (77be253e)
77be2581 7753            ja      ntdll!LdrLoadDll+0x60 (77be25d6)
77be2583 56              push    esi
```

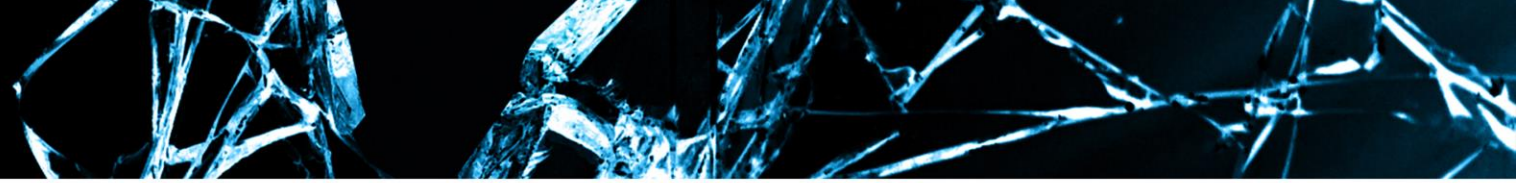
LdrLoadDll Hook

```
Usage:                  Image
Allocation Base:        77b80000
Base Address:           77be2000
End Address:            77be3000
Region Size:            00001000
Type:                   01000000    MEM_IMAGE
State:                  00001000    MEM_COMMIT
Protect:                00000040    PAGE_EXECUTE_READWRITE
More info:              lmv m ntdll
More info:              !lmi ntdll
More info:              ln 0x77be2576
```

RWX Permissions

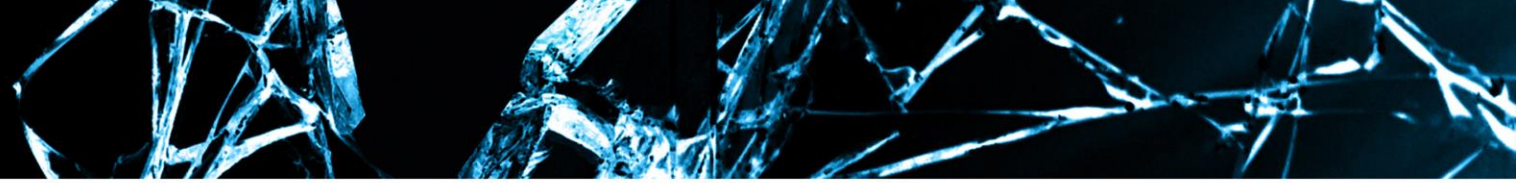
SECURITY ISSUES OF HOOKING - RECAP

Issue		Severity	Affected underlying systems
1	Unsafe injection	Very high	All windows versions
2	Predictable RWX code stubs	Very high	All windows versions
3	Predictable RX code stubs	High	All windows versions
4	Predictable RWX code stubs	High	Windows 7 and below
5	RWX hook code stubs	Medium	All windows versions
6	RWX hooked modules	Medium	All windows versions



Demo

Bypassing Exploit Mitigations



3rd Party Hooking Engines

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

EASYHOOK – OPEN-SOURCE HOOKING ENGINE

- 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

DEVIARE2 - OPEN-SOURCE HOOKING ENGINE

- 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

MICROSOFT DETOURS

- 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).

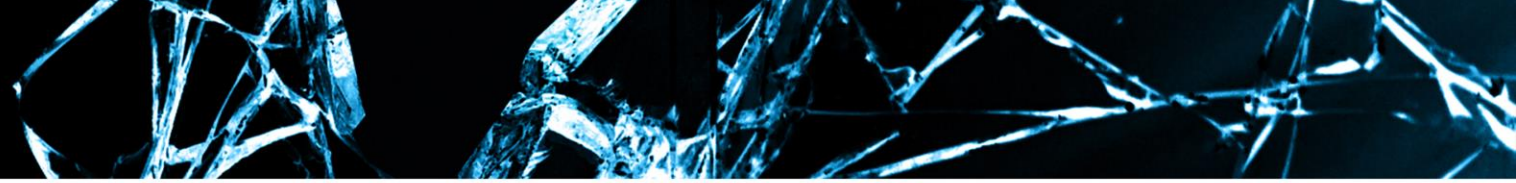
* Details won't be revealed until the patch is released (September)

MICROSOFT DETOURS VULNERABILITY - IMPLICATIONS

- 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

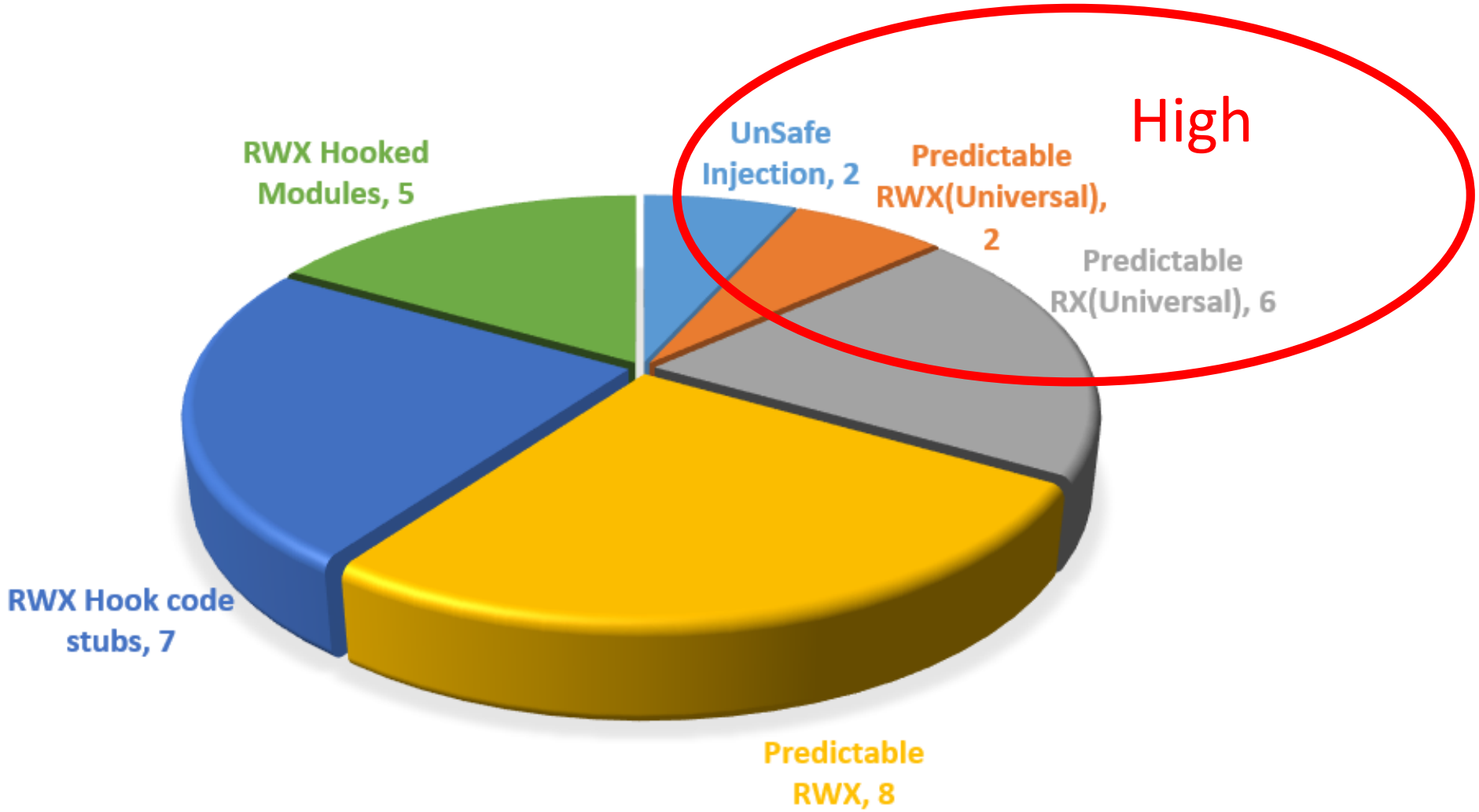


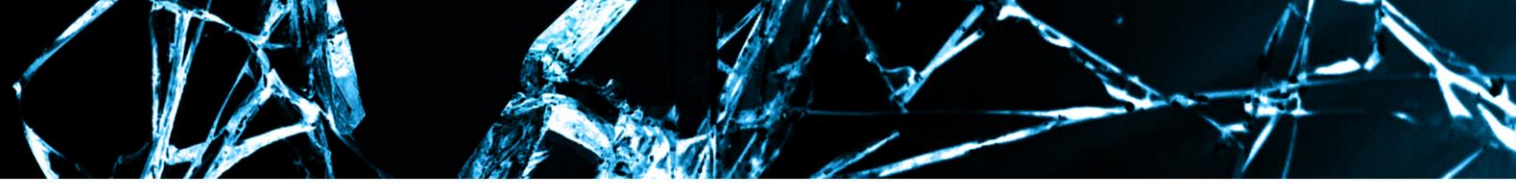
Affected Products

AFFECTED PRODUCTS

Products/Vendors	UnSafe Injection	Predictable RWX(Universal)	Predictable RX(Universal)	Predictable RWX	RWX Hook code stubs	RWX Hooked Modules	Time To Fix (Days)
Symantec				X			90
McAfee				X	X		90
Trend Micro		X	X (Initial Fix)		X		210
Kaspersky			X	X			90
AVG				X			30
BitDefender					X	X	30
WebRoot			X			X	29
AVAST			X		X		30
Emsisoft					X		90
Citrix - Xen Desktop					X	X	90
Microsoft Office*			X				180
WebSense	X			X		X	30
Vera	X			X			?
Invincea		X(64-bit)			X	X	?
Anti-Exploitation*				X			?
BeyondTrust			X	X			Fixed Independently
TOTALS	**2**	**2**	**6**	**8**	**7**	**5**	**79.9**

* Patch wasn't released yet

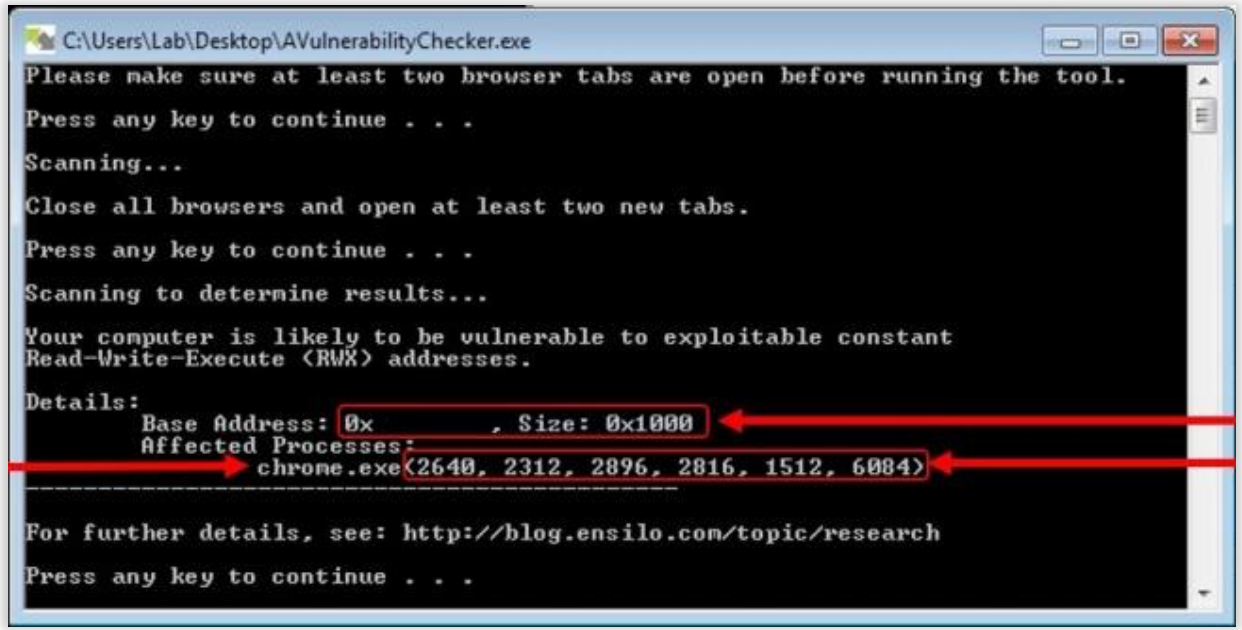

High
UnSafe Injection, 2
Predictable RWX(Universal), 2
Predictable RX(Universal), 6
RWX Hooked Modules, 5
RWX Hook code stubs, 7
Predictable RWX, 8



Research Tools

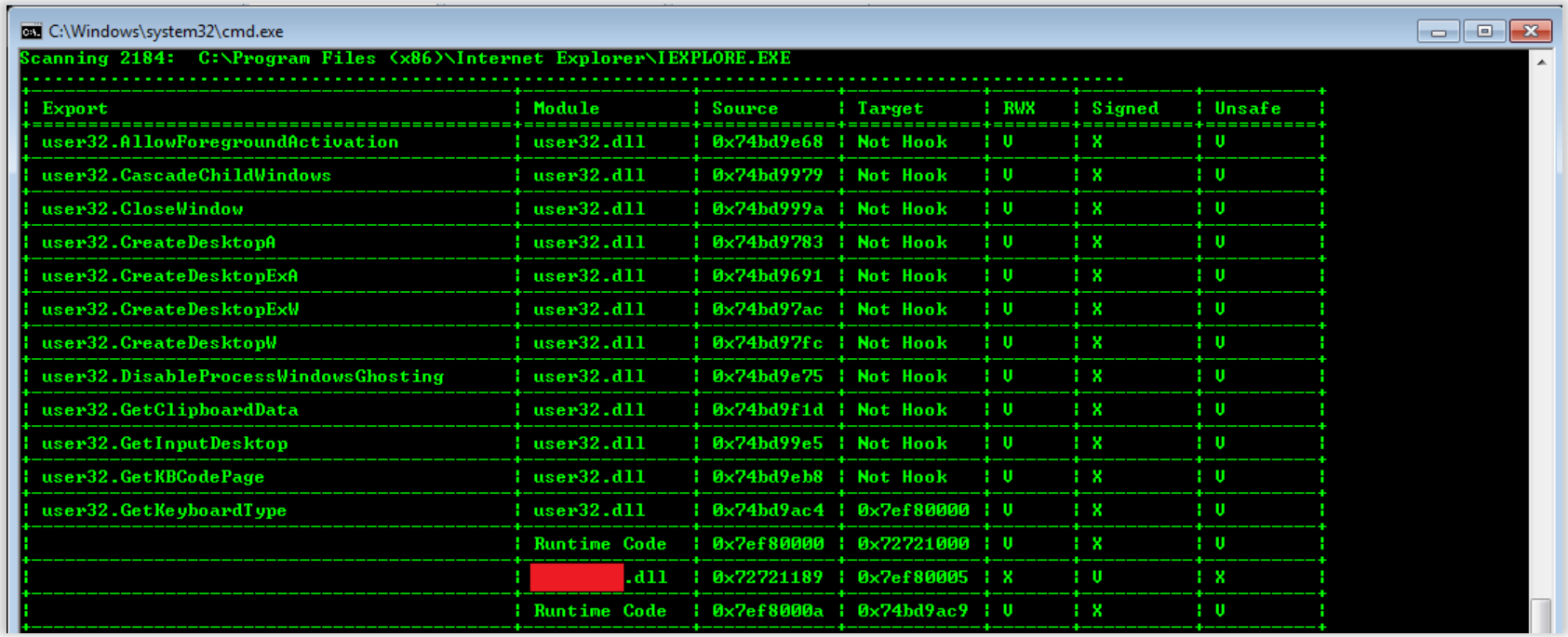
RESEARCH TOOLS – AVULNERABILITY

- Tool to detect predictable RWX code regions
- Can be found at https://github.com/BreakingMalware/AVulnerabilityChecker
- Compares memory maps of processes



RESEARCH TOOLS – HOOKS SCAN

- 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



SUMMARY

- 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

ENSILO

SILO YOUR DATA FROM THREAT ACTORS