Back To The Epilogue

How to Evade Windows' Control Flow Guard with Less than 16 Bytes

Andrea Biondo *
Prof. Mauro Conti
Daniele Lain *

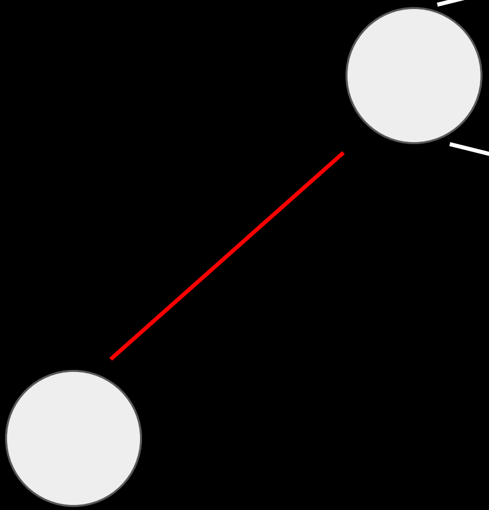
SPRITZ Group
University of Padua, IT

UNIVERSITÀ DEGLI STUDI DI PADOVA

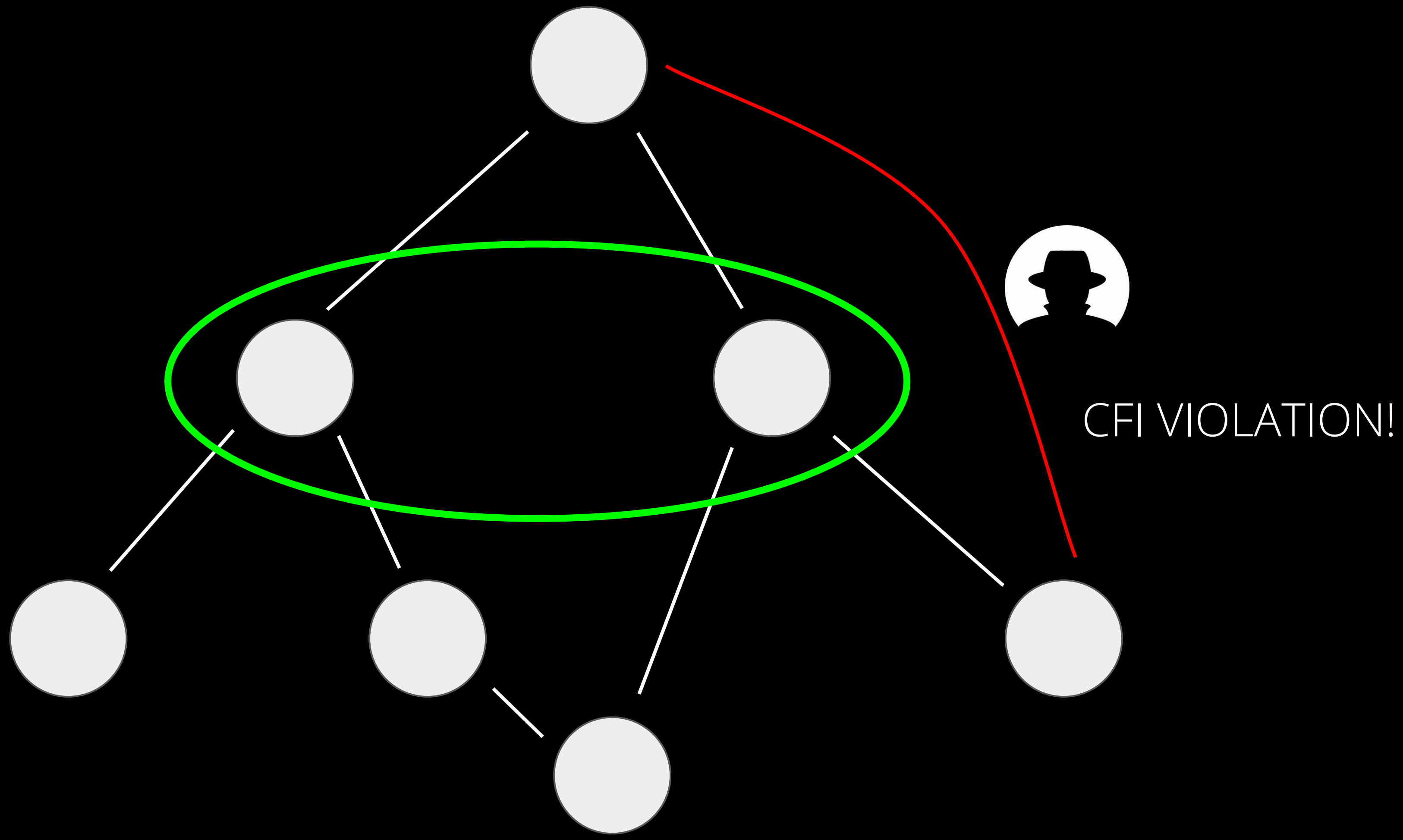
SPRITZ SECURITY & PRIVACY RESEARCH GROUP

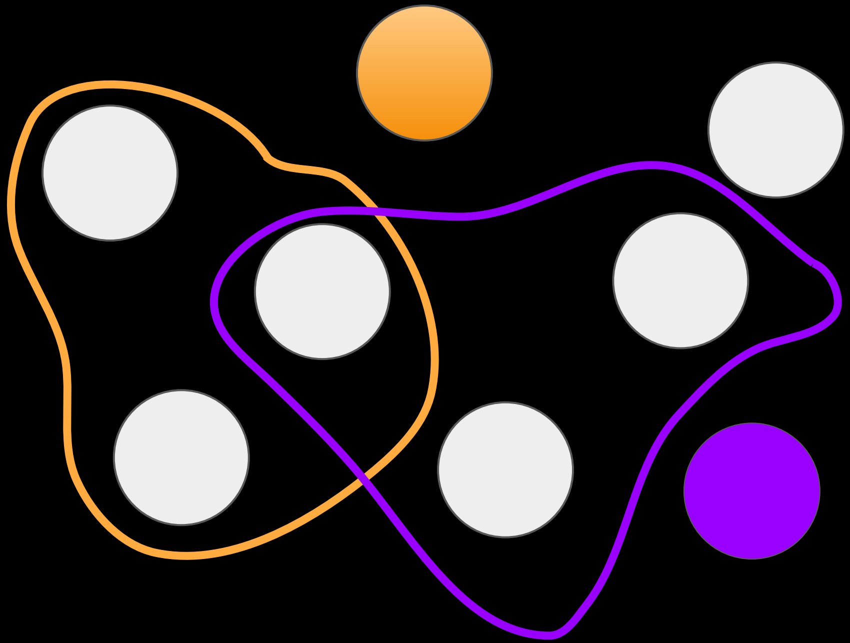
GOALS

- Return to function epilogue ❌
- Evade Windows' Control Flow Guard ❌
- With less than 16 bytes ❌

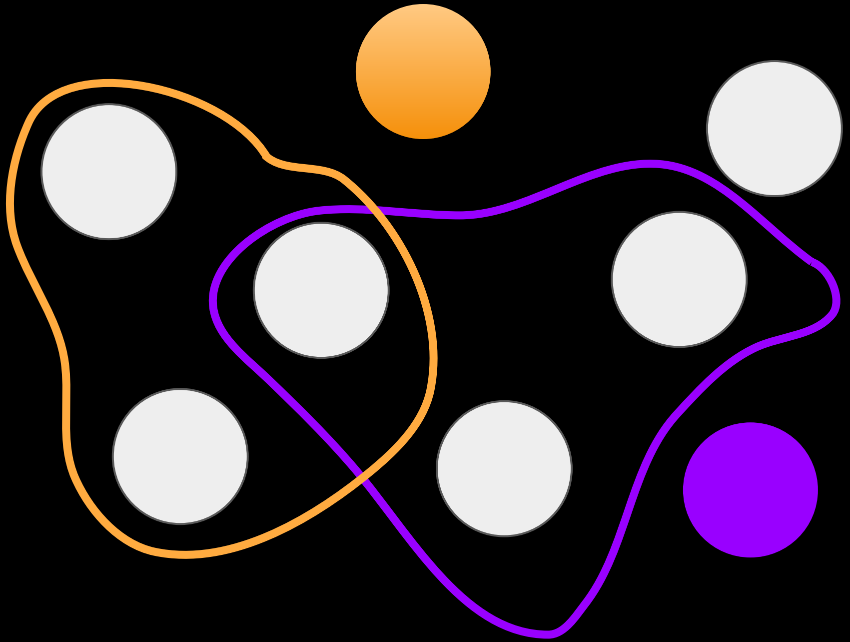


```
0000000000005000    xor     ebp, ebp
0000000000005002    mov     r9, rdx
0000000000005005    pop     rsi
0000000000005006    mov     rdx, rsp
0000000000005009    and     rsp, 0xfffffffffffffff0
000000000000500d    push    rax
000000000000500e    push    rsp
000000000000500f    lea     r8, qword [sub_15dc5+11]
0000000000005016    lea     rcx, qword [sub_15d59+7]
000000000000501d    lea     rdi, qword [0x35c0]
0000000000005024    call    qword [qword_21fca8+352]
```

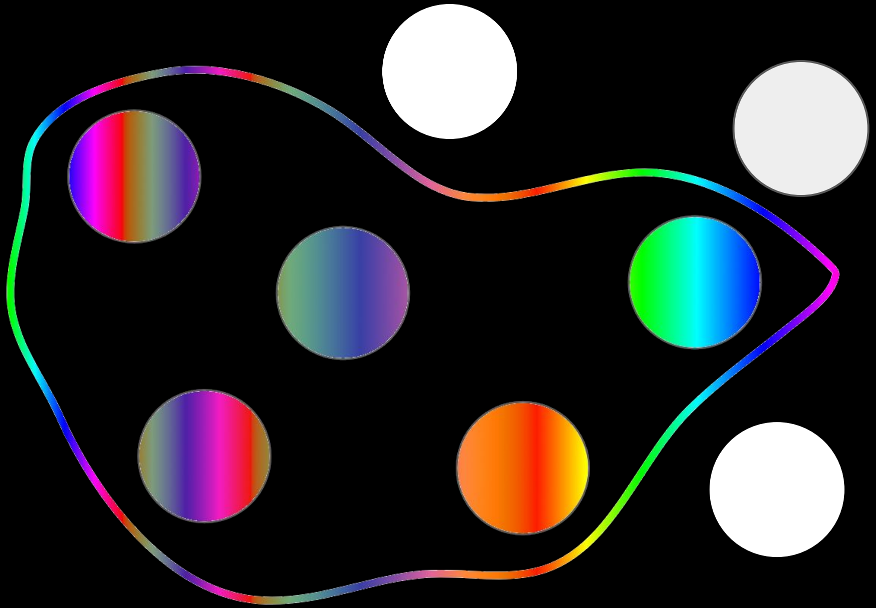

CFI VIOLATION!



FINE GRAINED



FINE GRAINED



COARSE GRAINED

CONTROL FLOW GUARD OVERVIEW

- Microsoft’s CFI implementation
- Deployed since Windows 8.1
- Coarse-grained (single target set)
- Forward-edge only

CONTROL FLOW GUARD INTERNALS

1. **Compile time:** instrument calls and build target set
    a. Check mode
    b. Dispatch mode

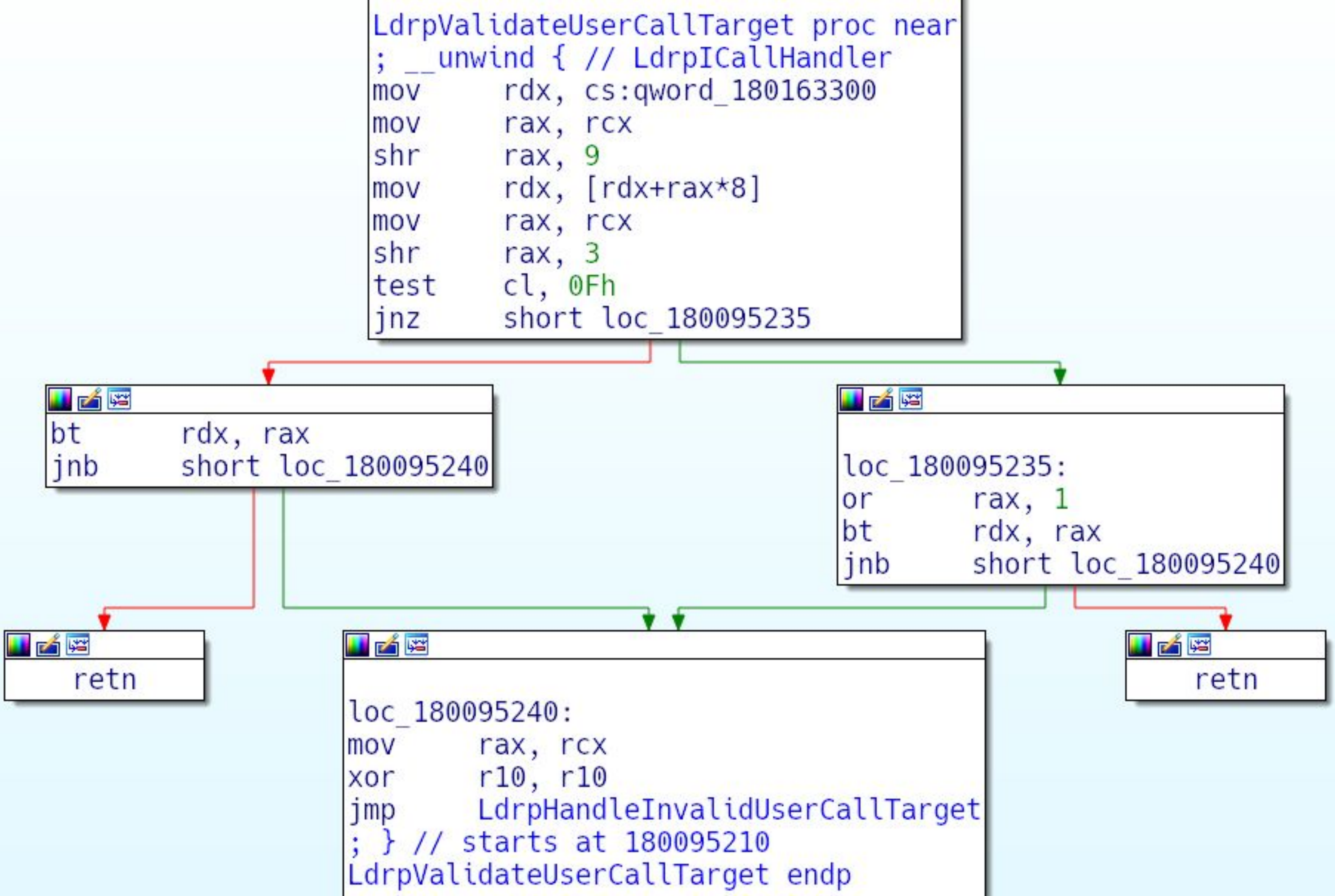
```
mov     [rsp+8], rbx
push    rdi
sub     rsp, 20h
mov     rbx, cs:qword_14004F960
mov     rdi, rcx
mov     rcx, rbx
call    cs:__guard_check_icall_fptr
lea     r8, sub_14000BC30
xor     edx, edx
mov     rcx, rdi
mov     rax, rbx
mov     rbx, [rsp+30h]
add     rsp, 20h
pop     rdi
jmp     rax
```

CONTROL FLOW GUARD INTERNALS

1. **Compile time:** instrument calls and build target set
    a. Check mode
    b. Dispatch mode

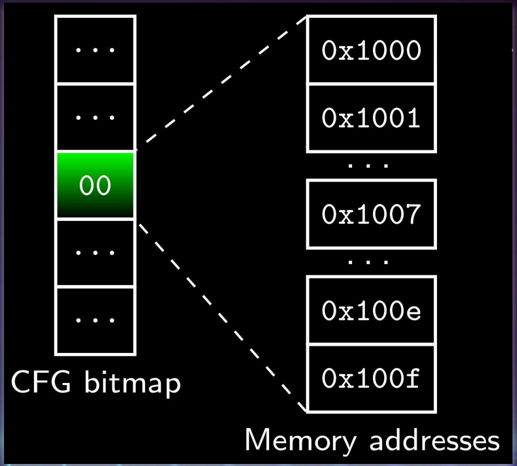
2. **Load time:** build bitmap, populate function pointers

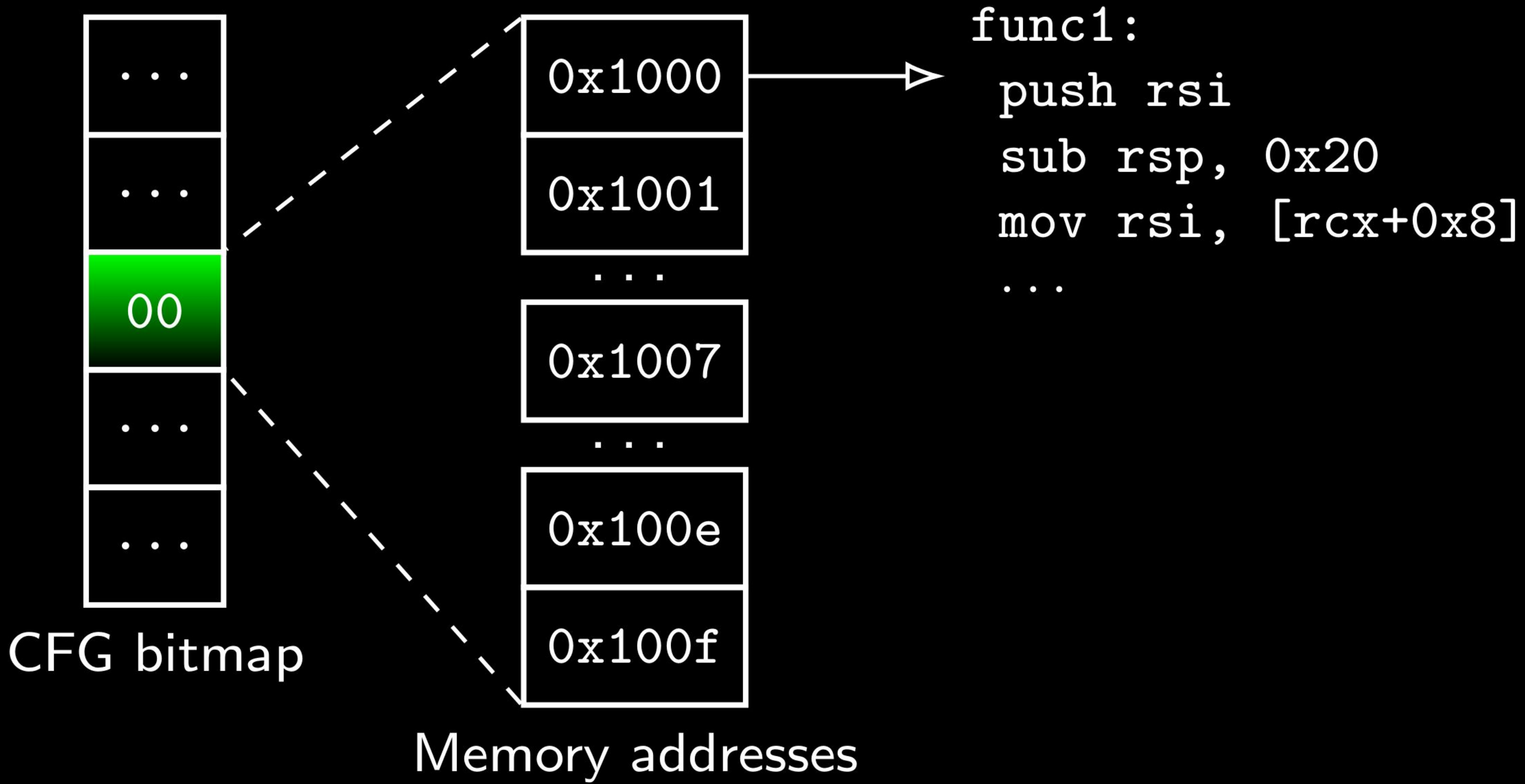
3. **Run time:** checks in `ntdll`


LdrpValidateUserCallTarget proc near
; __unwind { // LdrpICallHandler
mov rdx, cs:qword_180163300
mov rax, rcx
shr rax, 9
mov rdx, [rdx+rax*8]
mov rax, rcx
shr rax, 3
test cl, 0Fh
jnz short loc_180095235
bt rdx, rax
jnb short loc_180095240
loc_180095235:
or rax, 1
bt rdx, rax
jnb short loc_180095240
retn
loc_180095240:
mov rax, rcx
xor r10, r10
jmp LdrpHandleInvalidUserCallTarget
; } // starts at 180095210
LdrpValidateUserCallTarget endp
retn

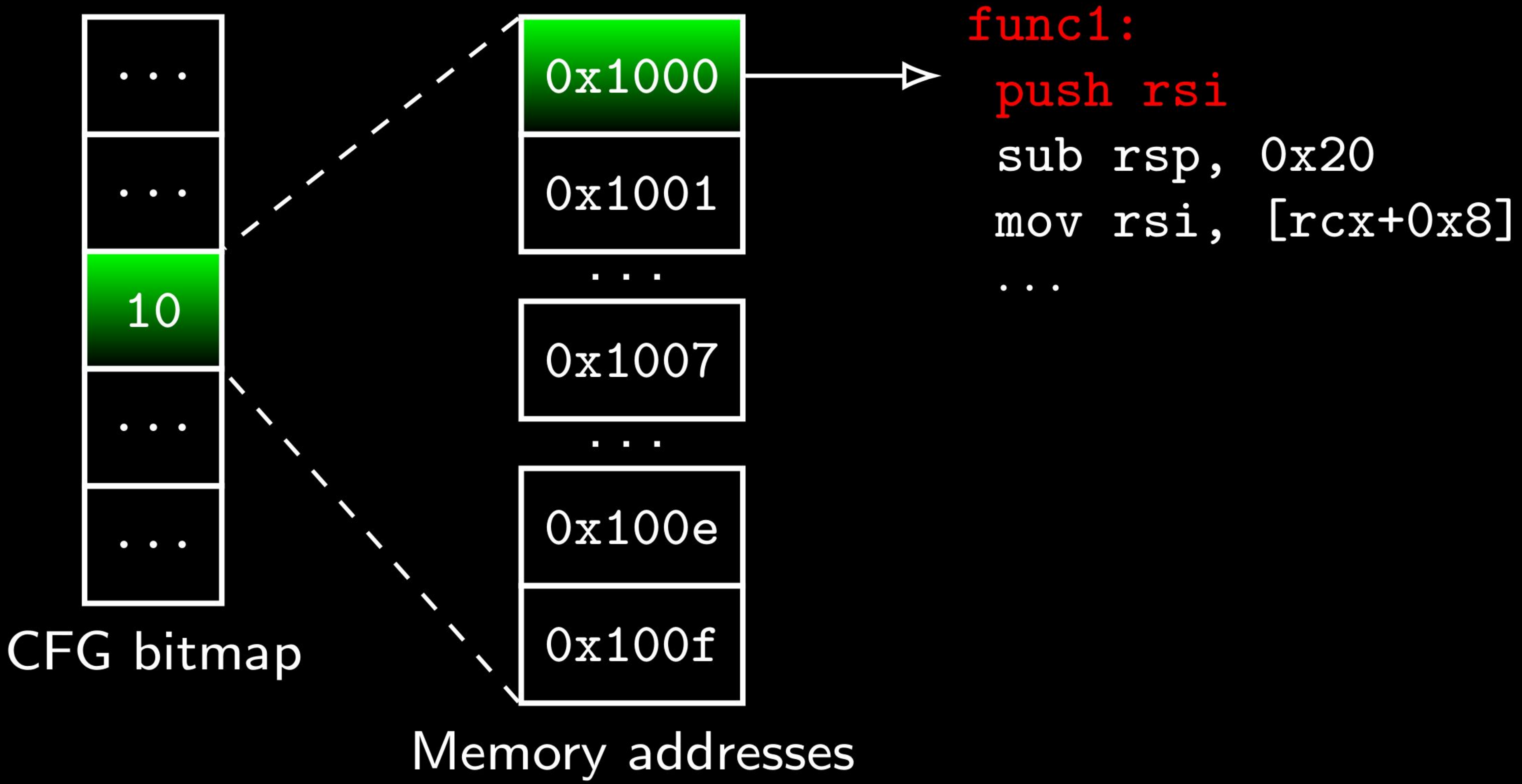
CONTROL FLOW GUARD INTERNALS

- Fast checking through a bitmap
- 2 bits map to 16 aligned bytes of target address space
  - `00`: No target allowed
  - `01`: Export suppression
  - `10`: Aligned allowed target
  - `11`: All targets allowed

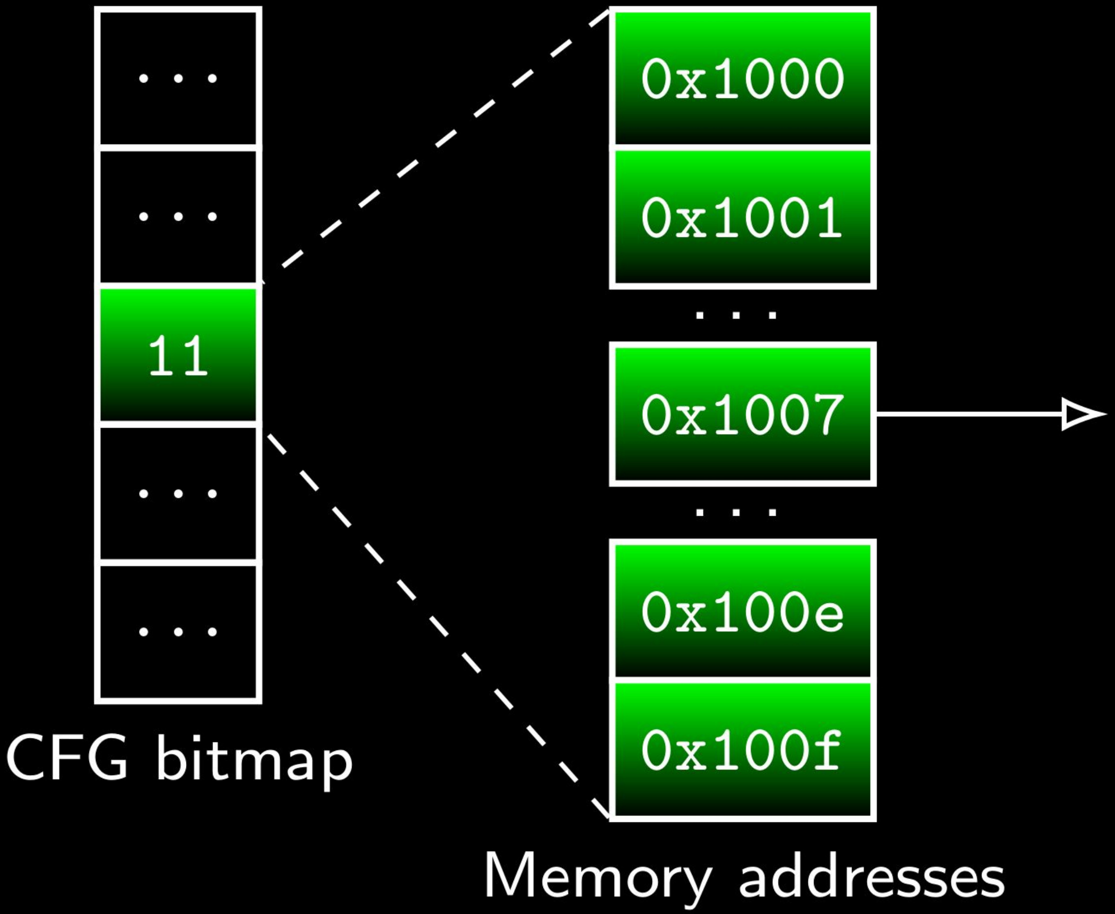




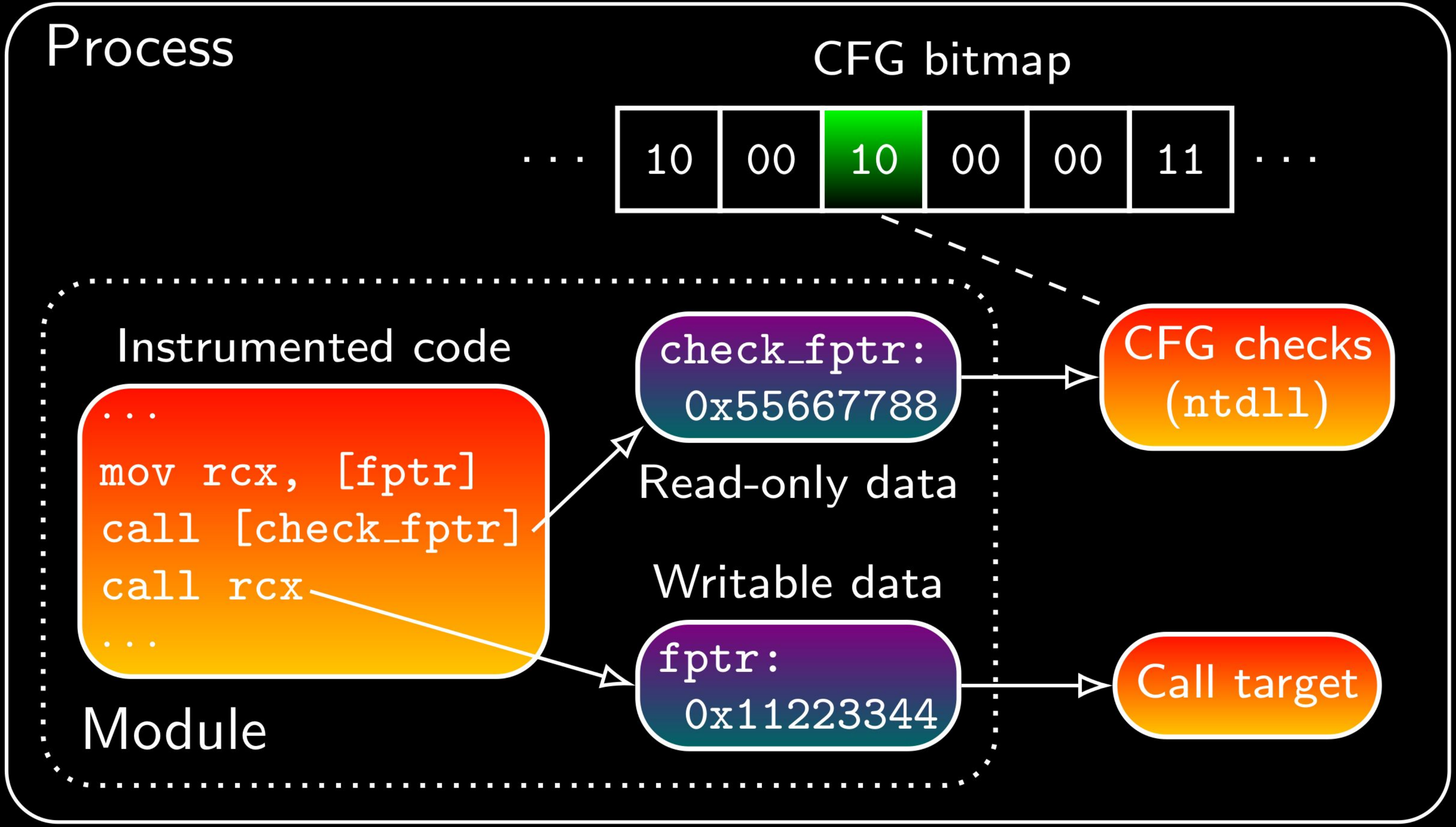
...
...
00
...
...
CFG bitmap
0x1000
0x1001
...
0x1007
...
0x100e
0x100f
Memory addresses
func1:
push rsi
sub rsp, 0x20
mov rsi, [rcx+0x8]
...



...
...
10
...
...
CFG bitmap
0x1000
0x1001
...
0x1007
...
0x100e
0x100f
Memory addresses
func1:
push rsi
sub rsp, 0x20
mov rsi, [rcx+0x8]
...




```
func1:
 ...
 add rsp, 0x40
 pop rdi
 pop rbx
 ret
func2:
 push rsi
 sub rsp, 0x20
 mov rsi, [rcx+0x8]
 ...
```


Process
CFG bitmap
... 10 00 10 00 00 11 ...
Instrumented code
...
mov rcx, [fptr]
call [check_fptr]
call rcx
...
check_fptr:
0x55667788
Read-only data
Writable data
fptr:
0x11223344
CFG checks
(ntdll)
Call target
Module

CONTROL FLOW GUARD
KNOWN ATTACKS

- Code reuse on modules built without CFG support
- Return address overwrite
- Improper protection of JITed code
  - *11 by default on memory mappings*
  - *Lack of instrumentation in JITed code*
- Unintended allowed calls (sensitive APIs)
- Making check/dispatch function pointers R/W
- Possibly R/W sections assumed to be RO
- I'M OUT OF SLIDE SPACE SEND HELP

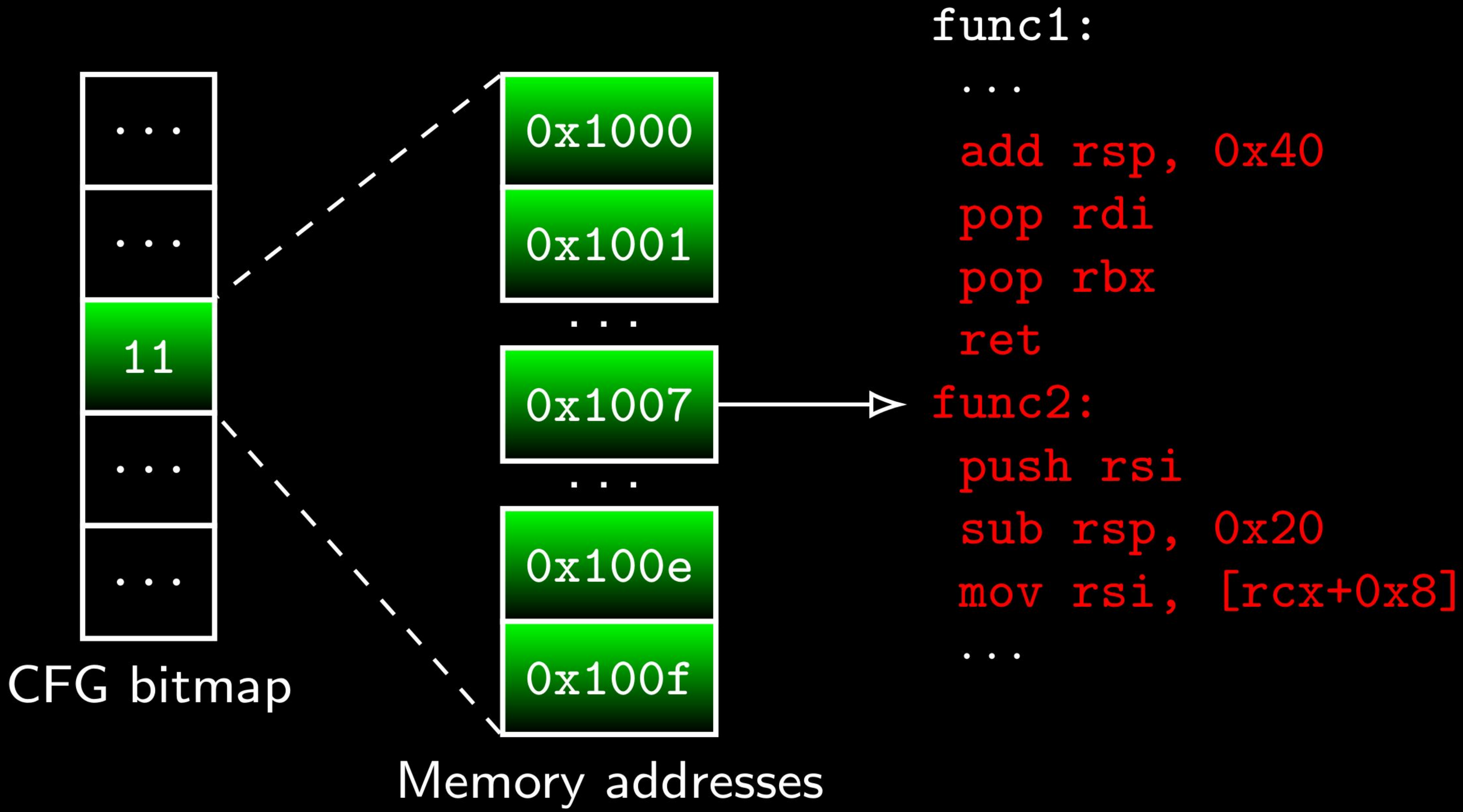

BACK
TO THE EPILOGUE

BACK TO THE EPILOGUE
THE IDEA

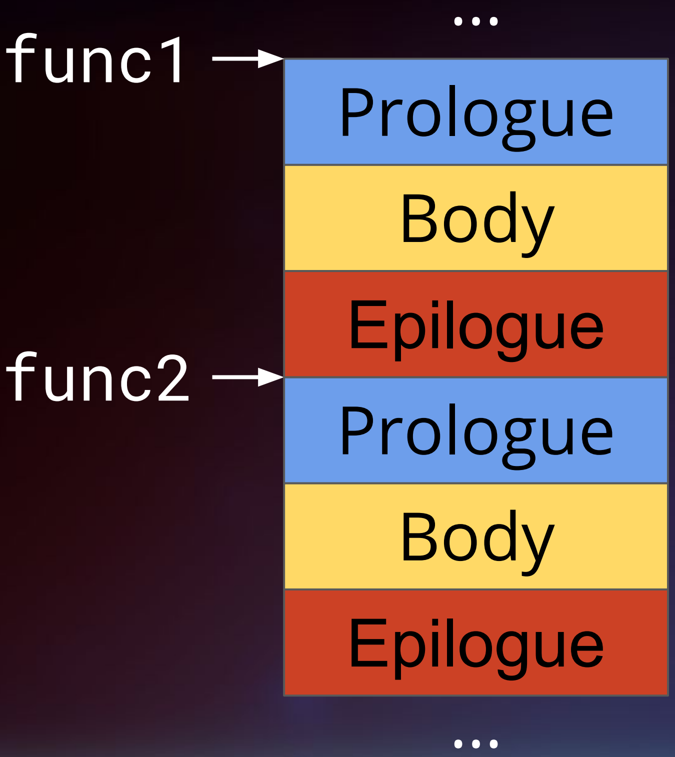
- What if an allowed target is not 16-byte aligned?

- Can’t be 10, must be 11 → unintended targets?
  - *(MJ0011 noted this back in 2014)*

- Unaligned targets are still there in system libraries


...
...
11
...
...
CFG bitmap
0x1000
0x1001
...
0x1007
...
0x100e
0x100f
Memory addresses
func1:
...
add rsp, 0x40
pop rdi
pop rbx
ret
func2:
push rsi
sub rsp, 0x20
mov rsi, [rcx+0x8]
...

ANATOMY OF A FUNCTION



Allocate stack frame

Business logic

Deallocate stack frame

BACK TO THE EPILOGUE
THE IDEA

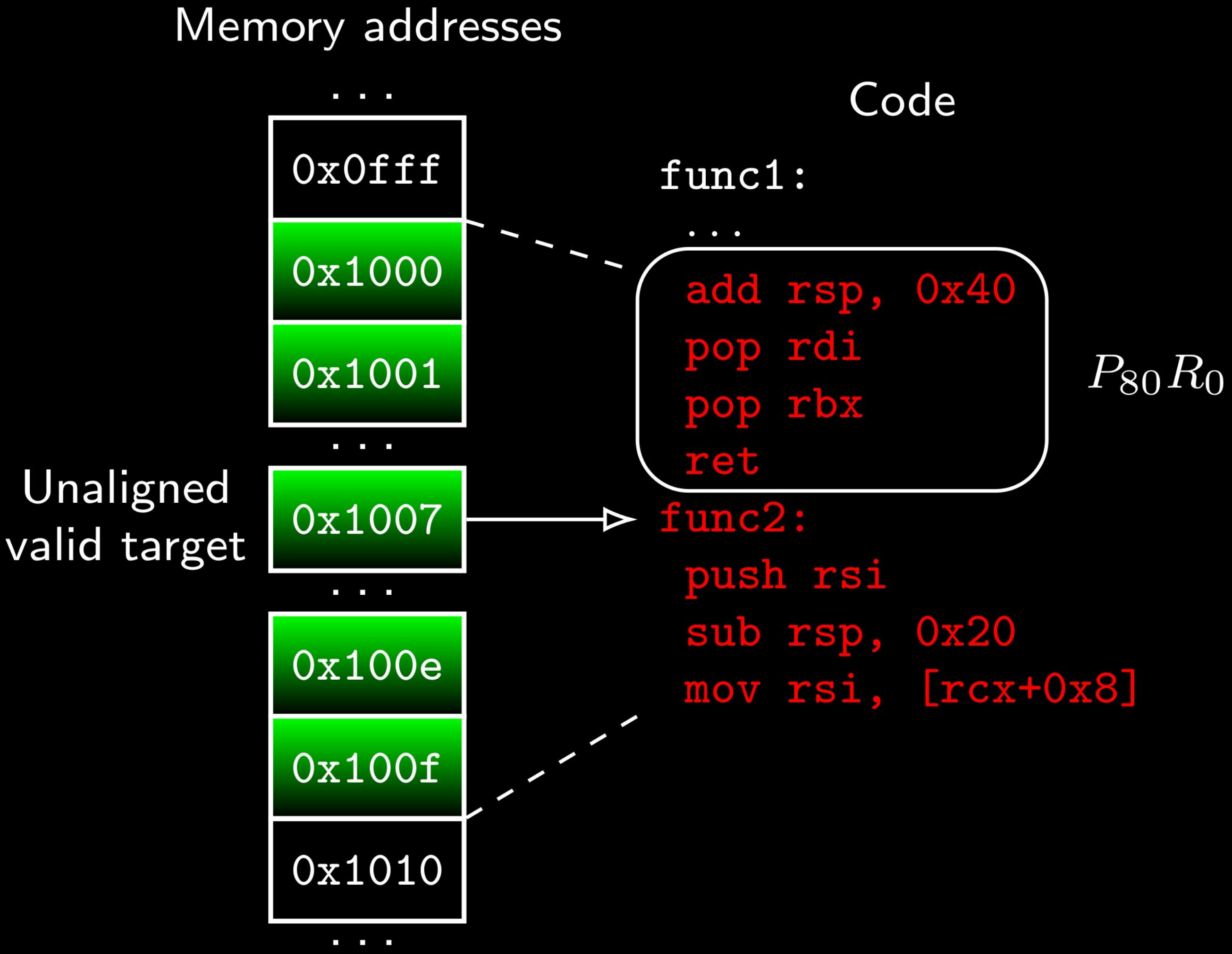
- We can reach instructions close to the entry point
- Prologues are boring
- Epilogues mess with the stack and return
  - Profit?

GOALS

- Return to function epilogue
- Evade Windows' Control Flow Guard
- With less than 16 bytes

BACK TO THE EPILOGUE
THE PLAN

- Epilogues increment stack pointer and return
    - **PR gadgets**

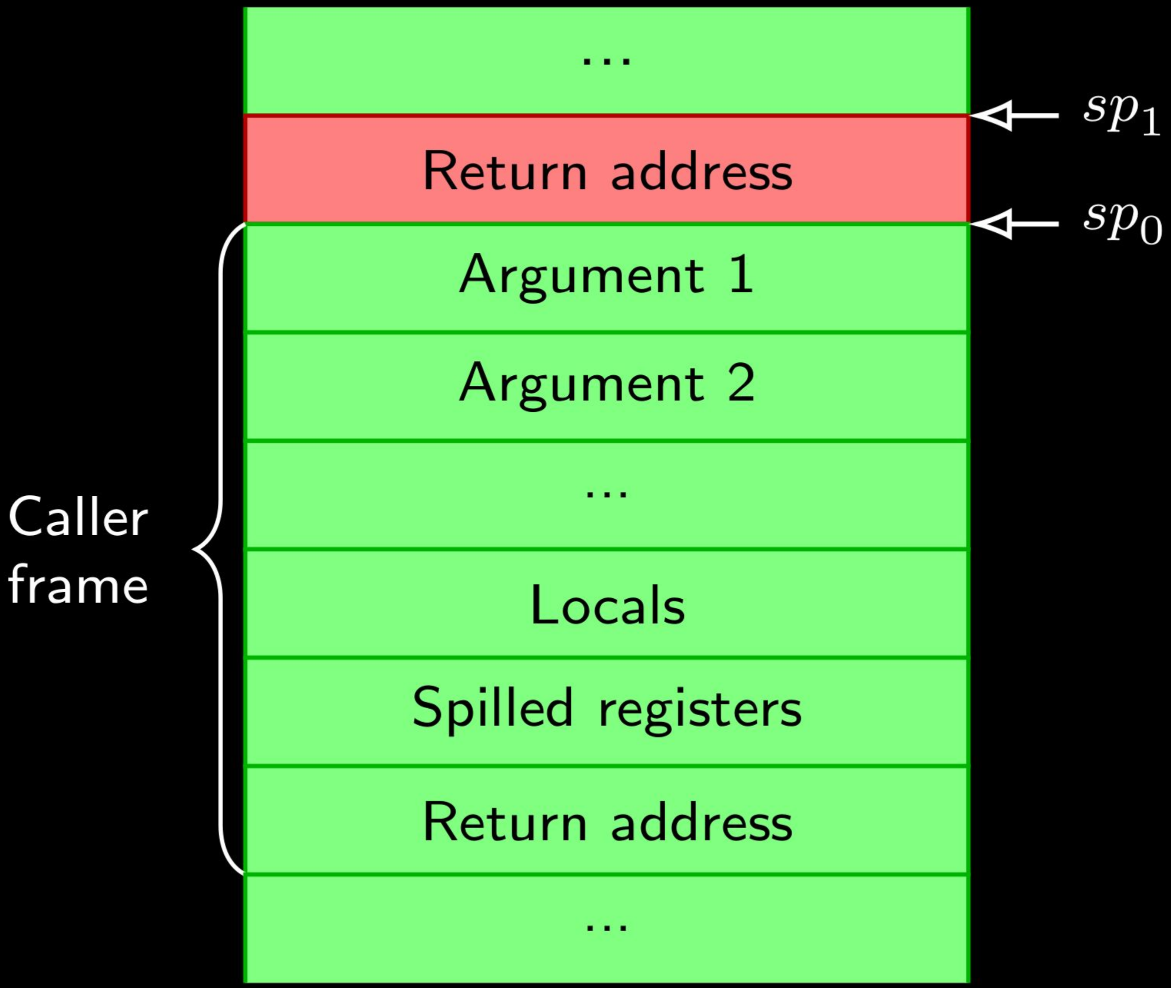

Memory addresses
Code
...
0x0fff
0x1000
0x1001
...
Unaligned valid target
0x1007
...
0x100e
0x100f
0x1010
...
func1:
...
add rsp, 0x40
pop rdi
pop rbx
ret
$P_{80} R_0$
func2:
push rsi
sub rsp, 0x20
mov rsi, [rcx+0x8]

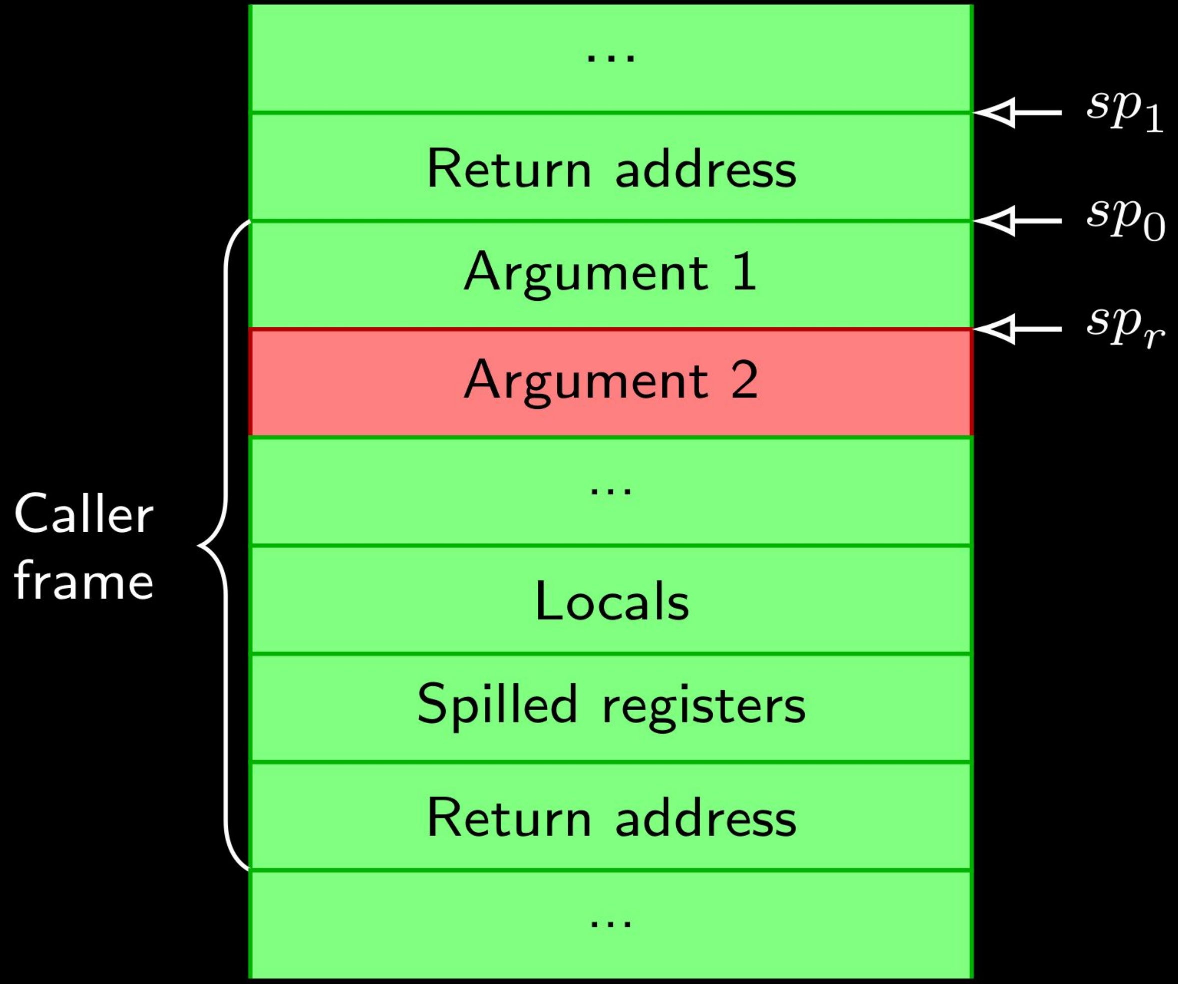
BACK TO THE EPILOGUE
THE PLAN

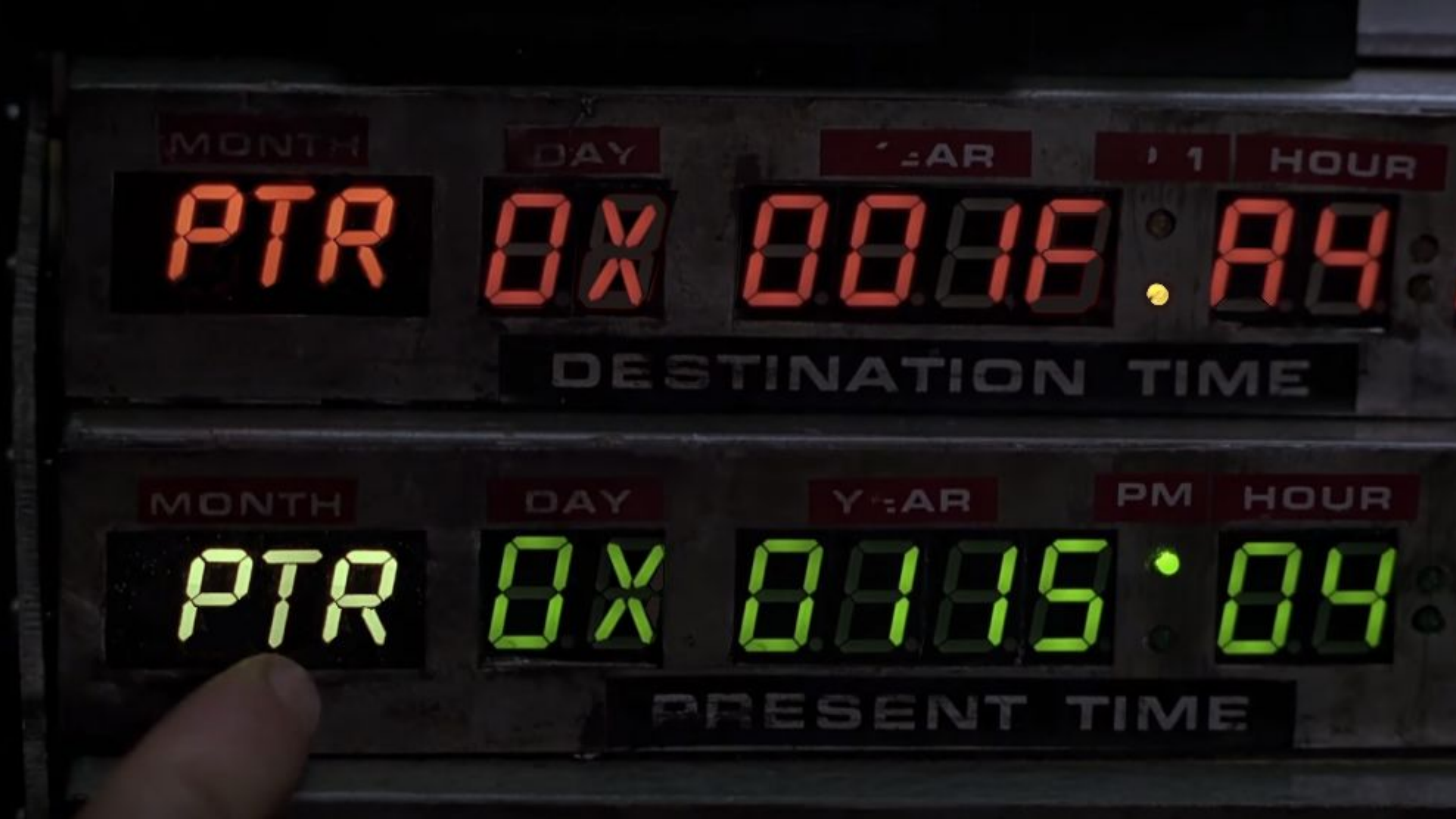
- Epilogues increment stack pointer and return
  - **PR gadgets**

- Pivot return address into attacker-controlled data

- No backward-edge CFI → profit!

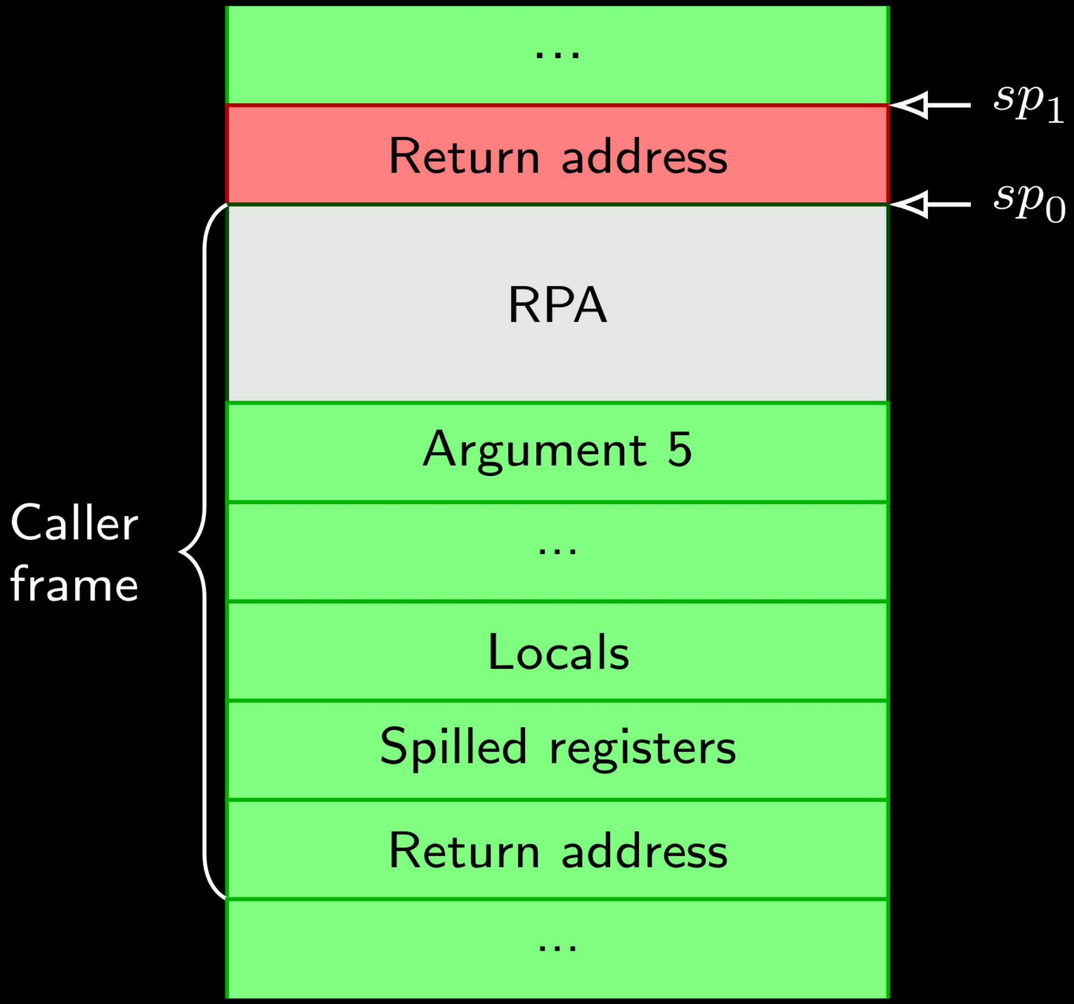

...
sp_1
Return address
sp_0
Argument 1
Argument 2
...
Caller frame
Locals
Spilled registers
Return address
...


...
sp_1
Return address
sp_0
Argument 1
sp_r
Argument 2
...
Caller frame
Locals
Spilled registers
Return address
...


MONTH
PTR
DAY
0X
0016
HOUR
A4
DESTINATION TIME
MONTH
PTR
DAY
0X
0115
PM
HOUR
04
PRESENT TIME

BACK TO THE EPILOGUE
64-BIT: THE PROBLEM

- First four arguments not on the stack
- Scumbag RPA foils our evil plan



...
sp_1
Return address
sp_0
RPA
Argument 5
...
Caller frame
Locals
Spilled registers
Return address
...

BACK TO THE EPILOGUE 64-BIT: THE IDEA

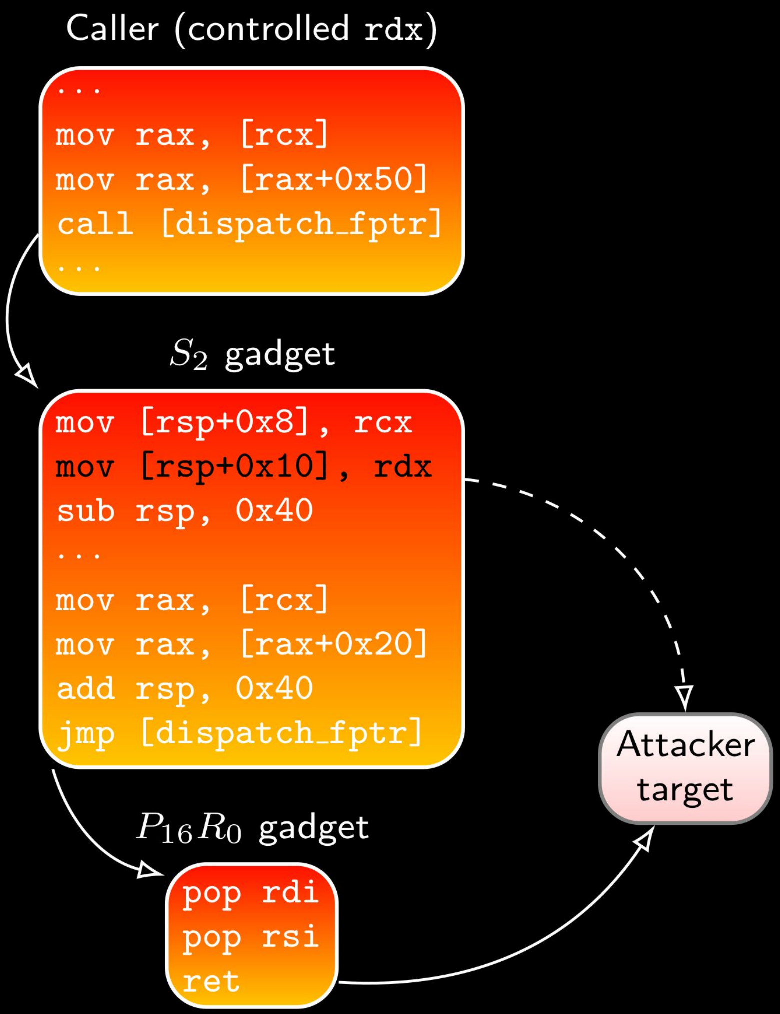
- Spill attacker-controlled values to RPA
- Need to call PR at the caller’s stack depth
  - Seems hard :(

Compiler optimizations to the rescue:

Tail jumps!

BACK TO THE EPILOGUE 64-BIT: THE PLAN

- Find CFG-valid functions that:
  a. Spill attacker-controlled registers to the RPA
  b. Have manageable side effects
  c. End with an attacker-controlled indirect tail jump
- We call them **S gadgets**
- Symbolic execution + taint tracking
  - *<insert jankiest taint tracking ever>*
  - *<insert more analysis buzzwords>*


Caller (controlled rdx)
...
mov rax, [rcx]
mov rax, [rax+0x50]
call [dispatch_fptr]
...
S_2 gadget
mov [rsp+0x8], rcx
mov [rsp+0x10], rdx
sub rsp, 0x40
...
mov rax, [rcx]
mov rax, [rax+0x20]
add rsp, 0x40
jmp [dispatch_fptr]
Attacker target
$P_{16}R_0$ gadget
pop rdi
pop rsi
ret

GOALS

- Return to function epilogue
- Evade Windows' Control Flow Guard
- With less than 16 bytes

CONTROL FLOW GUARD
KNOWN ATTACKS

- Code reuse on modules built with ... G support
- Return address overwrite ...
- Improper protect... ...ed code
  - *11 by d... ...mappi...*
  - *... ...ation ... ...code*
- ... all... ...alls (sensitive APIs)
- Making ... dispatch function pointers R/W
- Po...bly R/W sections assumed to be RO
- I'M OUT OF SLIDE SPACE SEND HELP

CONTROL ONLY AN ARGUMENT TO A CORRUPTED CALL

LOAD MODULE WITH GADGETS

EVALUATION

- Systematically evaluated Windows’ system libraries
  - *Loaded by a large number of processes*

- Pattern matching PR gadgets


GADGETS
GADGETS EVERYWHERE

EVALUATION

32-bit	57 PR GADGETS	`msvcrt.dll` (!!!) `MSVP9DEC.dll`
64-bit	22 PR GADGETS	`jscript9.dll` `msmpeg2vdec.dll`

Load vuln lib → whole program vulnerable

EVALUATION

- S gadgets via symex

- **985** different ones
  - *IE & Edge JS engines* - `jscript9.dll, Chakra.dll`
  - *IE & EDGE HTML parsers* - `mshtml.dll, edgehtml.dll`
  - *Skype codecs*
  - ...

EDGE EXPLOIT

- CVE-2016-7200
  - `Array.filter` Infoleak
  - Leak address of object
- CVE-2016-7201
  - `FillFromPrototypes` type confusion
  - Arbitrary memory R/W

EDGE EXPLOIT
GADGET SELECTION

- $P_{16}R_0$ from `msmpeg2vdec.dll`
- S_2 from `chakra.dll`
  - Spills `rdx` (2nd arg) to `rsp+16`
  - Calls fptr @ +0x50 in vtable of object in `rcx` (1st arg)

EDGE EXPLOIT
ASLR BYPASS (`chakra.dll`)

1. Leak address of JavaScript object
2. Read vtable pointer from object
3. Read function pointer from vtable

Now we have a code pointer in `chakra.dll`.

EDGE EXPLOIT
ASLR BYPASS (`msmpeg2vdec.dll`)

1. Derandomize `msvcrt.dll` from `chakra.dll`'s IAT
2. Derandomize `ntdll.dll` from `msvcrt.dll`'s IAT
3. Look up `msmpeg2vdec.dll` in `ntdll`'s loaded modules hash table

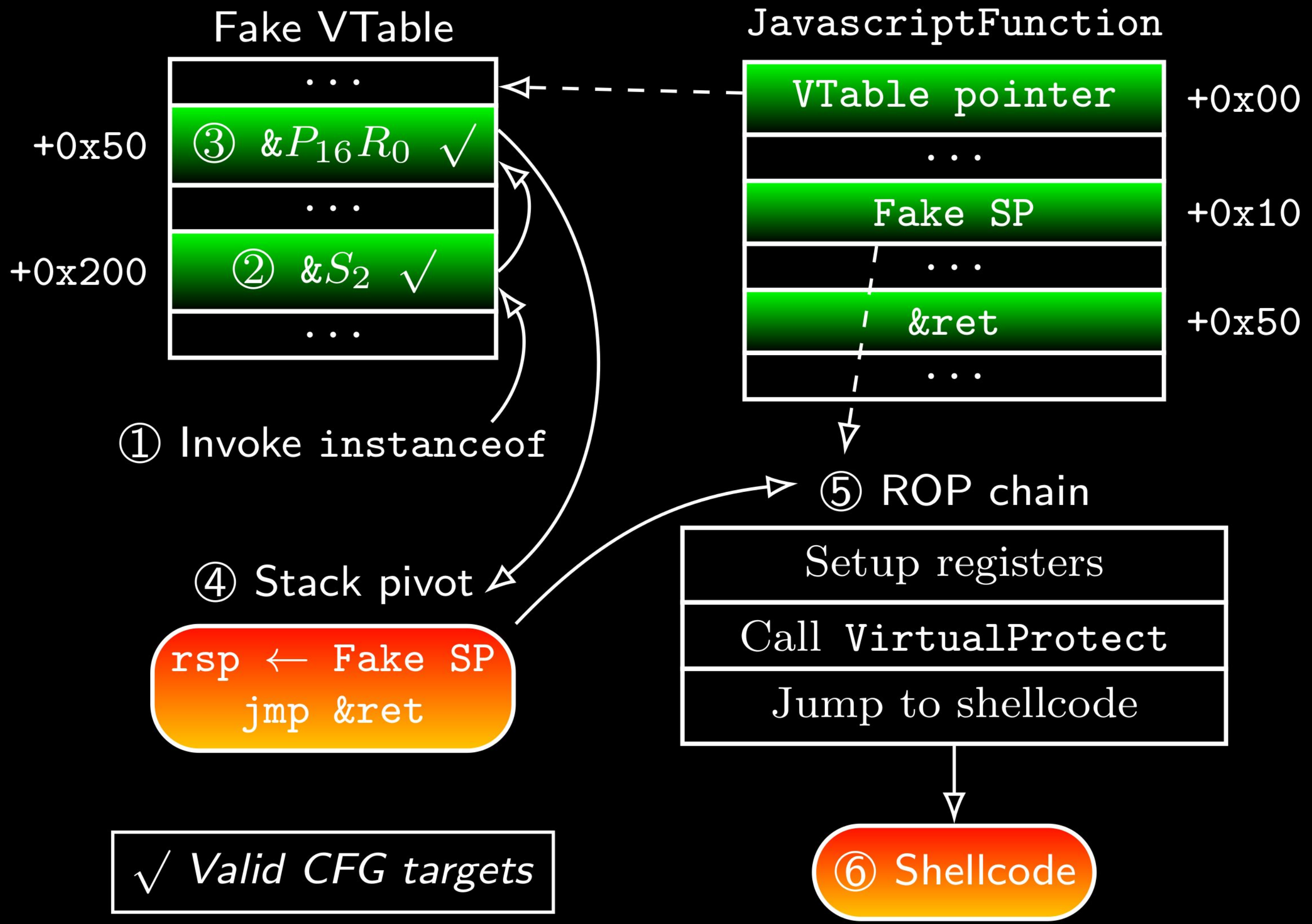
EDGE EXPLOIT
CONTROLLING ARGUMENTS

- Most functions accept `Var` arguments
- `Var` is either a pointer to object or a double

1. Create array → elements will be `Var`s
2. Corrupt array element via write primitive
3. Use corrupted element as argument

EDGE EXPLOIT
CONTROL FLOW HIJACKING

1. Hijack `JavascriptFunction` vtable
   a. `HasInstance` @ +0x200 → S gadget
   b. @ +0x50 → PR gadget
2. Call instanceof
   a. LHS: `JavascriptFunction` (1st arg to `HasInstance`)
   b. RHS: controlled `Var` (2nd arg to `HasInstance`)



Fake VTable
...
+0x50
③ &$P_{16}R_0$ √
...
+0x200
② &S_2 √
...
JavascriptFunction
VTable pointer
+0x00
...
Fake SP
+0x10
...
&ret
+0x50
...
① Invoke instanceof
④ Stack pivot
rsp ← Fake SP
jmp &ret
⑤ ROP chain
Setup registers
Call VirtualProtect
Jump to shellcode
⑥ Shellcode
√ Valid CFG targets

GOALS

- Return to function epilogue
- Evade Windows' Control Flow Guard
- With less than 16 bytes

DEMO! TO THE EPILOGUE

BLACK HAT SOUND BYTES

- Attack your mitigations!
- Be careful in what you shrug off as *not dangerous*
- Seemingly small issues might not be so small after all

BACK TO THE EPILOGUE

an attack by

ANDREA BIONDO

MAURO CONTI

DANIELE LAIN

UNIVERSITY OF PADUA - SPRITZ GROUP