Back To The Epilogue

Evading Control Flow Guard via Unaligned Targets

Andrea Biondo, Mauro Conti, Daniele Lain

University of Padua

NDSS Symposium 2018
San Diego, CA

Tuesday, 20 February 2018

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
MATEMATICA

Outline

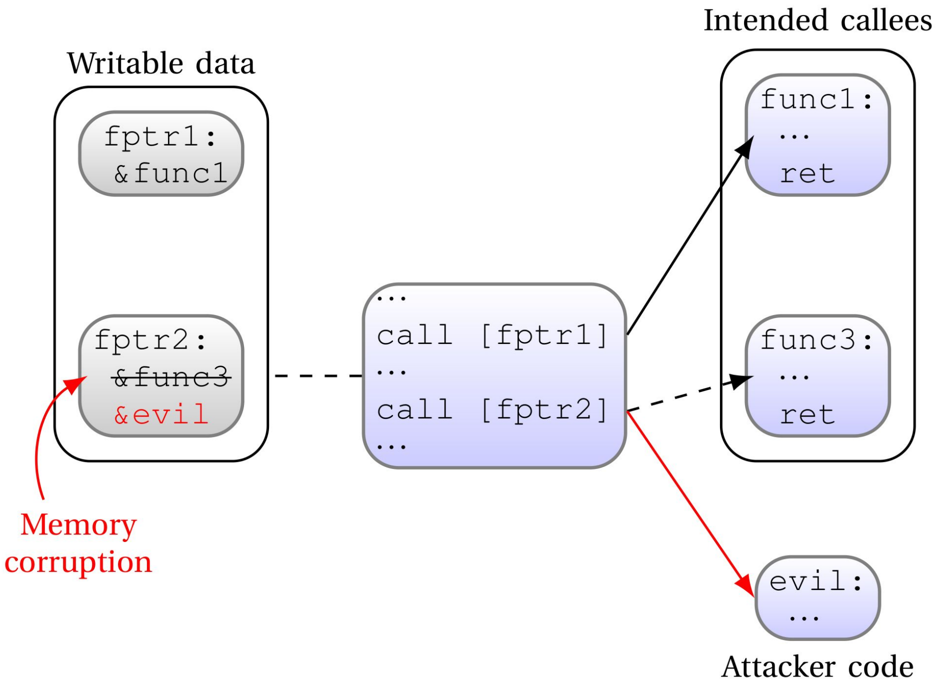
- Control Flow Integrity
- Microsoft Control Flow Guard
- BATE: Bypassing CFG
- Impact Evaluation
- Conclusions

Outline

- **<u>Control Flow Integrity</u>**
- Microsoft Control Flow Guard
- BATE: Bypassing CFG
- Impact Evaluation
- Conclusions

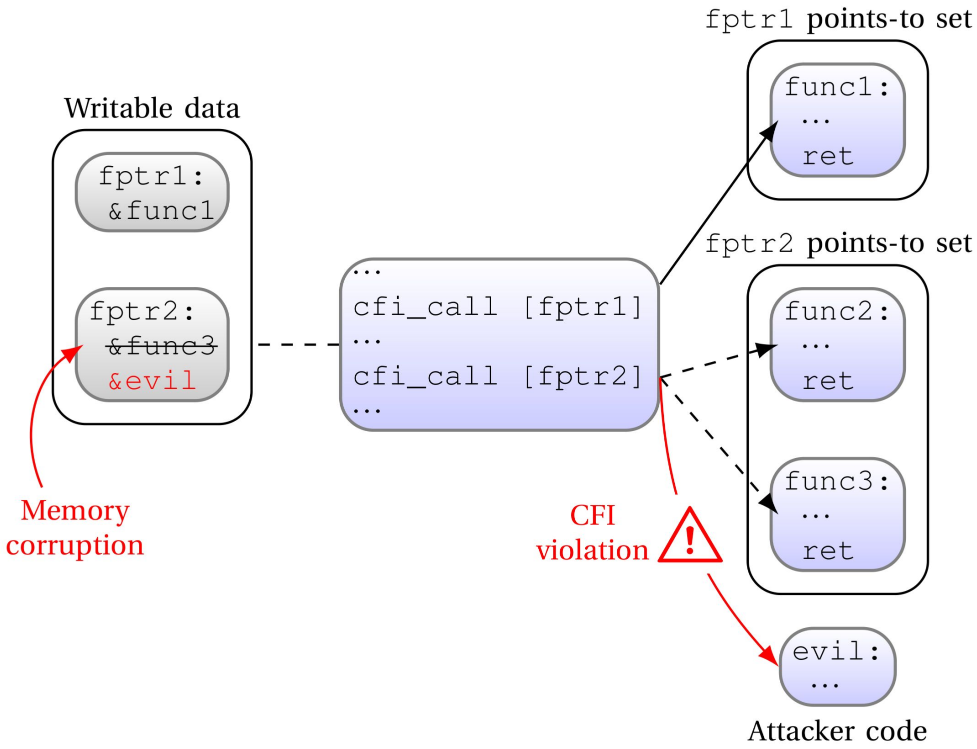
Control Flow Integrity

Memory corruption vulnerabilities lead to **Control Flow Hijacking**



Control Flow Integrity

CFIs prevent redirection of **control flow** to arbitrary locations



Control Flow Integrity

- CFIs can protect:
  - **Forward edges** *(calls, jumps)*
  - **Backward edges** *(return addresses)*

- Statically determined **set of valid targets** for a call

Control Flow Integrity

- CFIs can protect:
  - **Forward edges** *(calls, jumps)*
  - **Backward edges** *(return addresses)*

- Statically determined **set of valid targets** for a call

Undecidable!

- Resort to **approximations** of such sets:
  - **Coarse** grained *(single valid target set)*
  - **Fine** grained *(valid target set per call site)*

Outline

- Control Flow Integrity
- **Microsoft Control Flow Guard**
- BATE: Bypassing CFG
- Impact Evaluation
- Conclusions

Control Flow Guard - Overview

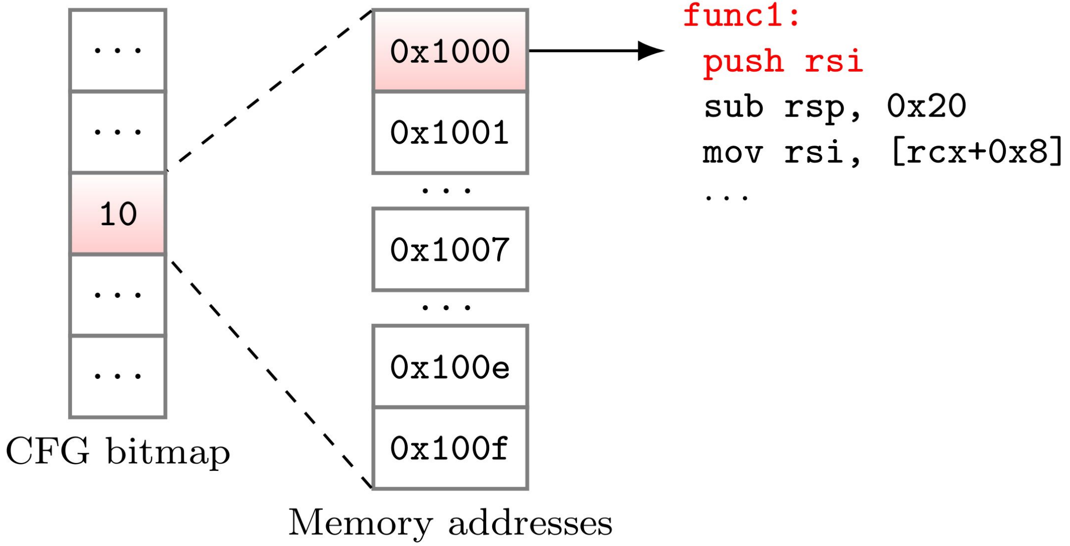
- **Coarse Grained** CFI mechanism
  - *Deployed in Microsoft Windows since Windows 8.1*
    (500 million machines worldwide)
  - Compile time → *valid target table* for **any** indirect branch

Control Flow Guard - Overview

- **Coarse Grained** CFI mechanism
    - *Deployed in Microsoft Windows since Windows 8.1*
      (500 million machines worldwide)
    - Compile time → *valid target table* for **any** indirect branch
    - Module loading → *CFG bitmap* **for 16-byte aligned ranges**

Control Flow Guard - Overview

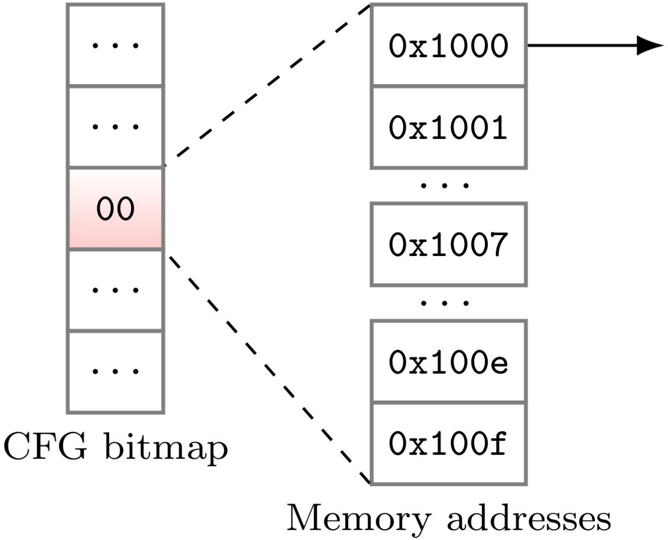
10: Aligned valid target



```
func1:
 push rsi
 sub rsp, 0x20
 mov rsi, [rcx+0x8]
 ...
```

Control Flow Guard - Overview

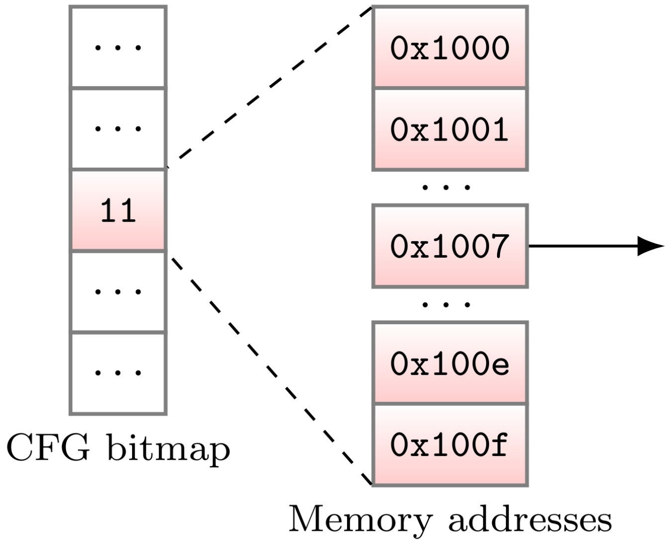
00: No valid target



```
func1:
 push rsi
 sub rsp, 0x20
 mov rsi, [rcx+0x8]
 ...
```

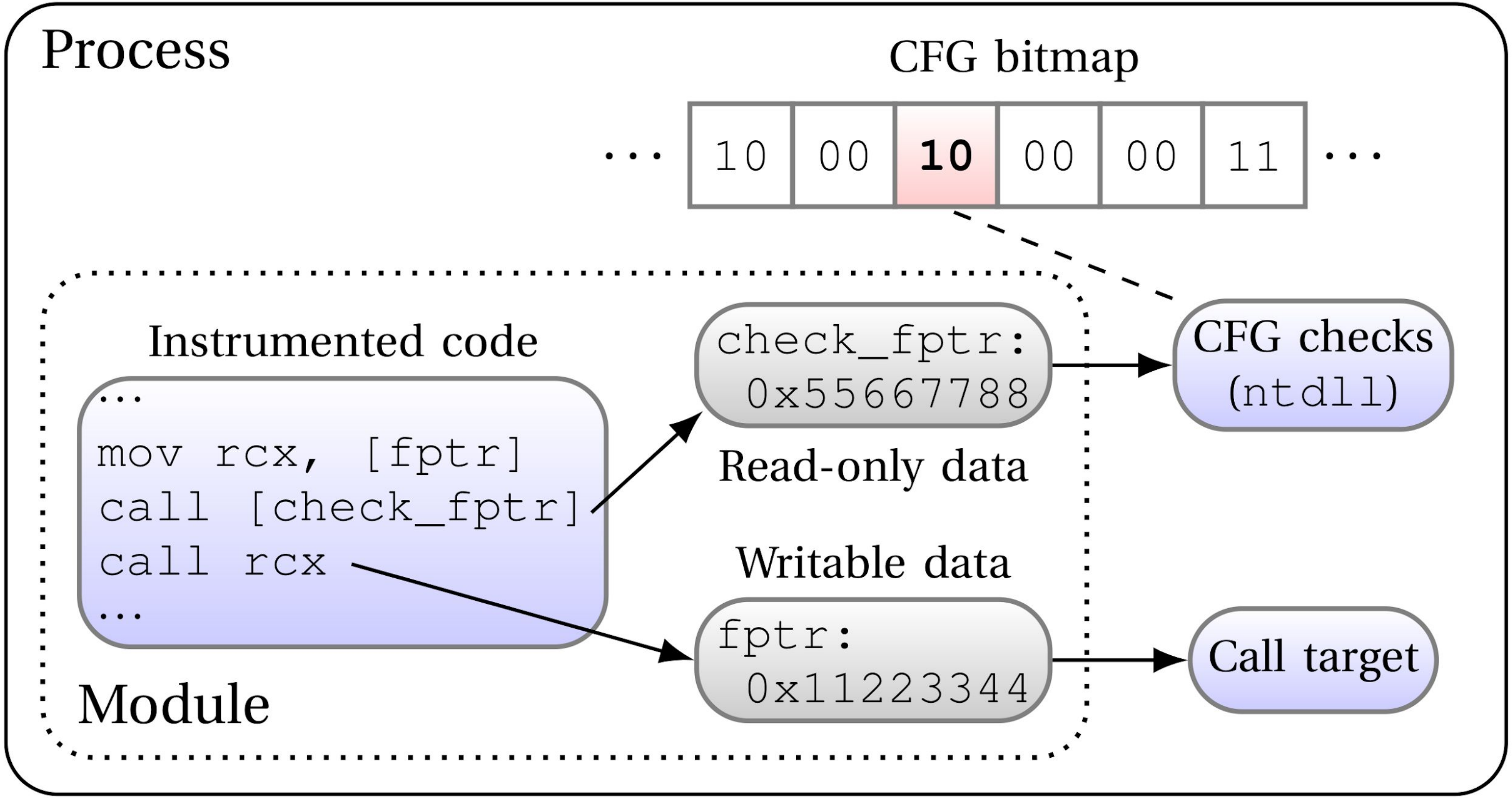
Control Flow Guard - Overview

11: Unaligned Valid Target



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

Control Flow Guard - Runtime



```
...
mov rcx, [fptr]
call [check_fptr]
call rcx
...
```

Outline

- Control Flow Integrity
- Microsoft Control Flow Guard
- **BATE: Bypassing CFG**
- Impact Evaluation
- Conclusions

Bypassing CFG

- Multiple issues
  - **Unaligned targets**
  - No backwards-edge CFI
  - Process-wide bitmap

Bypassing CFG

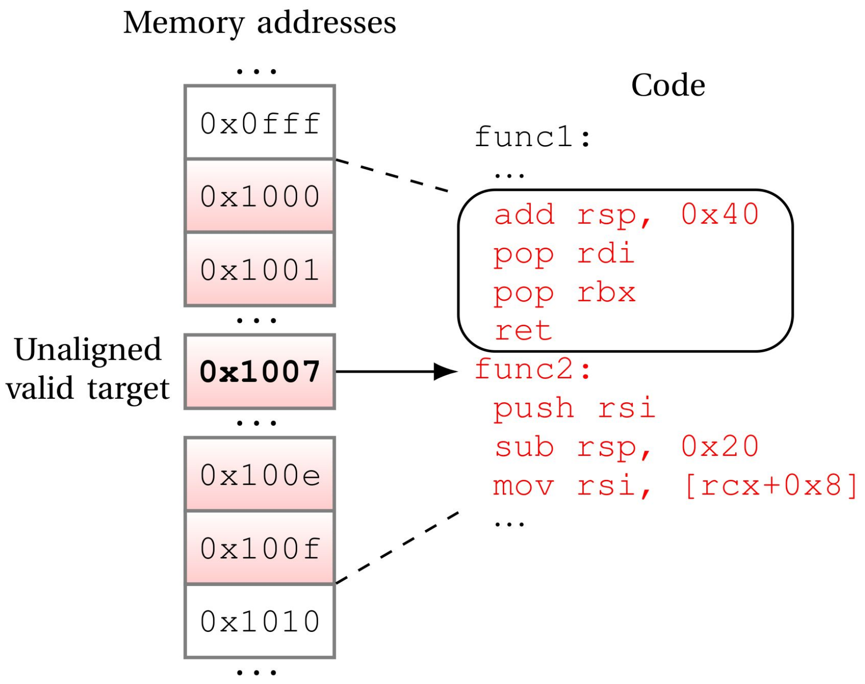
- Multiple issues
  - **Unaligned targets**
  - No backwards-edge CFI
  - Process-wide bitmap

- Functions are made of three parts
  - Prologue *(allocate stack, save registers)*
  - Body
  - **Epilogue** *(deallocate stack, restore registers, return)*

Unaligned Function Epilogues

Unaligned targets allow us to reach **epilogues**

- **Increment** stack pointer

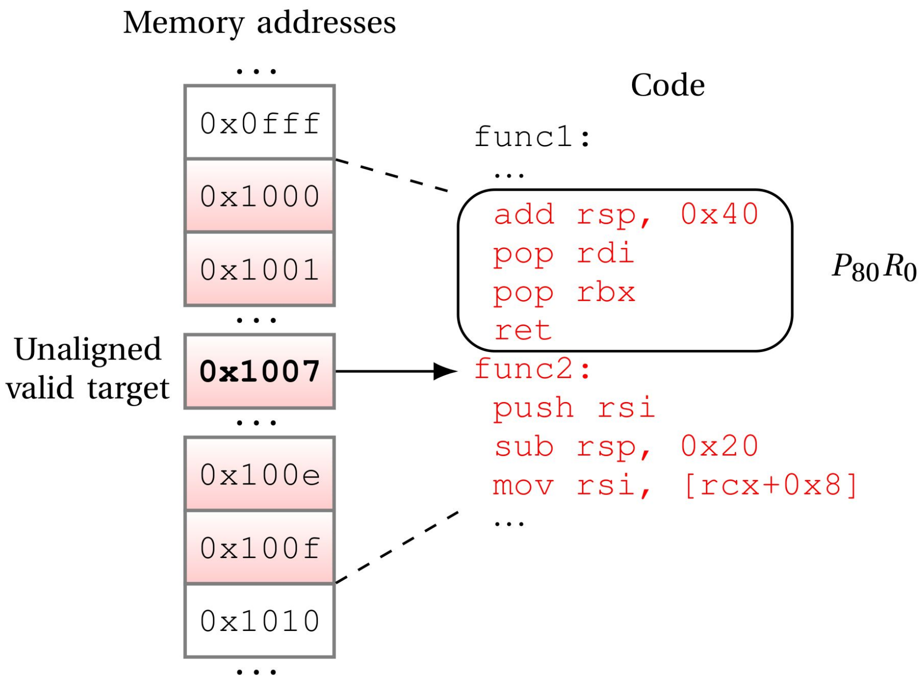


```
func1:
 ...
 add rsp, 0x40
 pop rdi
 pop rbx
 ret
func2:
 push rsi
 sub rsp, 0x20
 mov rsi, [rcx+0x8]
 ...
```

Bypassing CFG

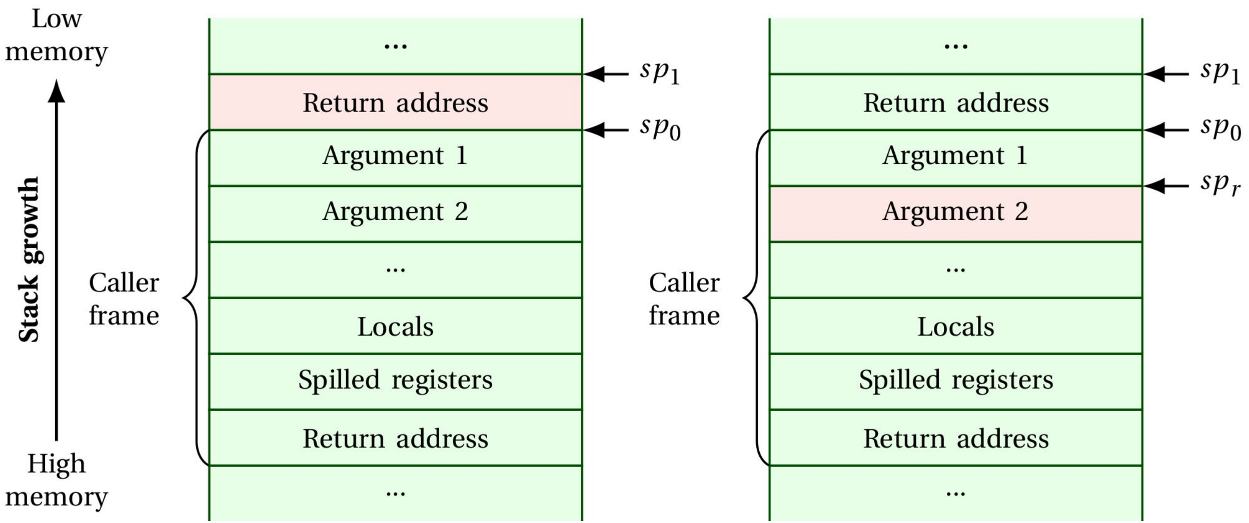
Define **PR gadgets**

- Increment stack pointer by *P* bytes **before** returning
- Increment stack pointer by *R* bytes **after** returning



Bypassing CFG

Hijack execution to a PR gadget to **pivot** the stack



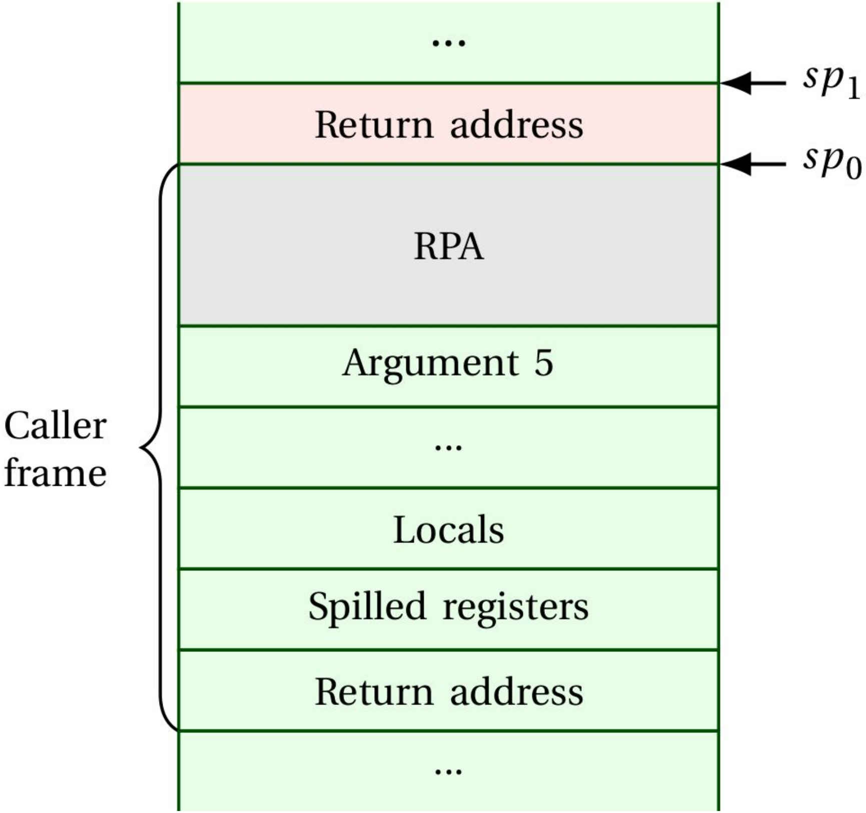
Return address into attacker-controlled data

No **backwards-edge** CFI

Bypassing CFG

Problem: on **64-bit**, stack control is harder

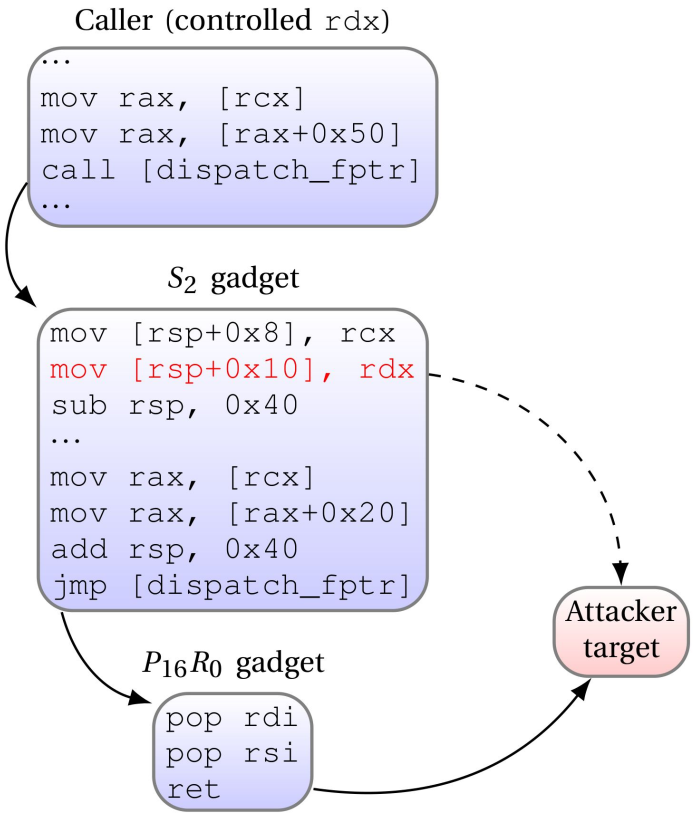
- First 4 **arguments** passed in registers
- *Register Parameter Area* at stack top



Bypassing CFG

Solution: **spill** argument registers to stack

- **S gadgets**
- Chain S gadget - PR gadget



```
...
mov rax, [rcx]
mov rax, [rax+0x50]
call [dispatch_fptr]
...
```

```
mov [rsp+0x8], rcx
mov [rsp+0x10], rdx
sub rsp, 0x40
...
mov rax, [rcx]
mov rax, [rax+0x20]
add rsp, 0x40
jmp [dispatch_fptr]
```

```
pop rdi
pop rsi
ret
```

Outline

- Control Flow Integrity
- Microsoft Control Flow Guard
- BATE: Bypassing CFG
- **Impact Evaluation**
- Conclusions

Impact Evaluation

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

	PR	S
32-bit	57	-
64-bit	22	985

Impact Evaluation

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

- Found PR and S gadgets in **high-risk** libraries
  - **C runtime** (32-bit)
  - Media codecs
  - Script engines

	PR	S
32-bit	57	-
64-bit	22	985

Outline

- Control Flow Integrity
- Microsoft Control Flow Guard
- BATE: Bypassing CFG
- Impact Evaluation
- **Conclusions**

Conclusions

- **Coarse grained** 16-byte approximation by CFG
    - Well-performing practical design
    - **Very strong assumptions** (→ *alignment)* do not hold

- BATE: High impact attack
    - **Widespread** gadgets
    - General, allows us to **bypass CFG entirely**
    - Feasible in practice

- **Disclosed** to Microsoft
    - Will be mitigated in RS4 (March/April)
    - We have permission to present this work

Thanks!

And align your code :-)

Backup Slides

Related Work

- Gadget Stitching *(Davi et al., 2014)*
  - Chains of CFI-allowed gadgets

- Counterfeit Object-Oriented Programming *(Schuster et al., 2015)*
  - Chains of CFI-allowed virtual methods

Both draw from **restricted gadget sets**

- Writing chains is harder
- BATE enables unrestricted code reuse

More gadgets?!

- Systematically evaluated Microsoft **Office 2016 Suite**
    - Exposed to attacks (e.g., macros on received documents)
    - 64-bit version

- **123 PR gadgets**

- Of which 101 are interesting: $P_{40}R_{0}$

Countermeasures

Aligning targets

- Simple
- May be difficult in corner cases (e.g., handwritten assembly)
- May impact certain optimizations

Making CFG more precise

- Virtual addressing space limitations
- CFG redesign?

Proof-of-Concept

PoC exploit for 64-bit **Edge** on Windows 10

- Based on CVE-2017-720{0,1}
- **Remote code execution** from JavaScript
- MPEG-2 media codec by embedding a video