13 Sep 2019, CROSSING Summer School on Sustainable Security & Privacy, Darmstadt, Germany

From Control-Flow to Data-Oriented Exploits

Hands-On Lab on Runtime Attacks

Lucas Davi, Oussama Draissi, Michael Rodler
University of Duisburg-Essen

Outline

Lecture
15:45 – 16:45

Hands-On Lab
16:45 – 17:45

Hands-On Lab

Download and install the VirtualBox VM:

http://bit.do/secsummer19

Motivating Problem

- Software increasingly sophisticated and complex

Motivating Problem

- Software increasingly sophisticated and complex

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved
- Native Code

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved
- Native Code
- Many program bugs

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved
- Native Code
- Many program bugs

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved
- Native Code
- Many program bugs

Large attack surface for run-time exploits on diverse platforms

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved
- Native Code
- Many program bugs

Large attack surface for run-time exploits on diverse platforms

Motivating Problem

- Software increasingly sophisticated and complex
- Various developers involved
- Native Code
- Many program bugs

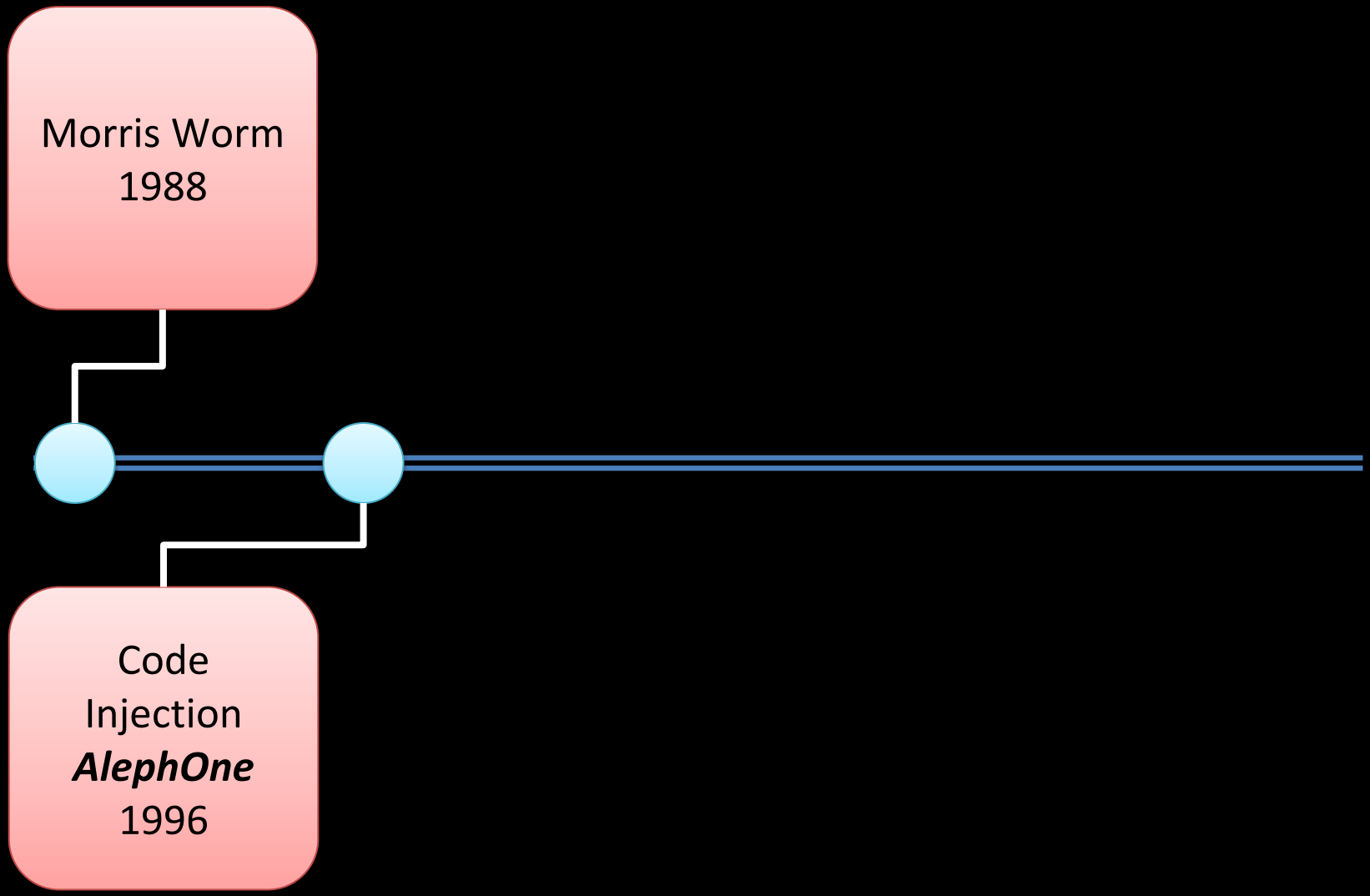
Large attack surface for run-time exploits on diverse platforms

Three Decades of Run-Time Attacks

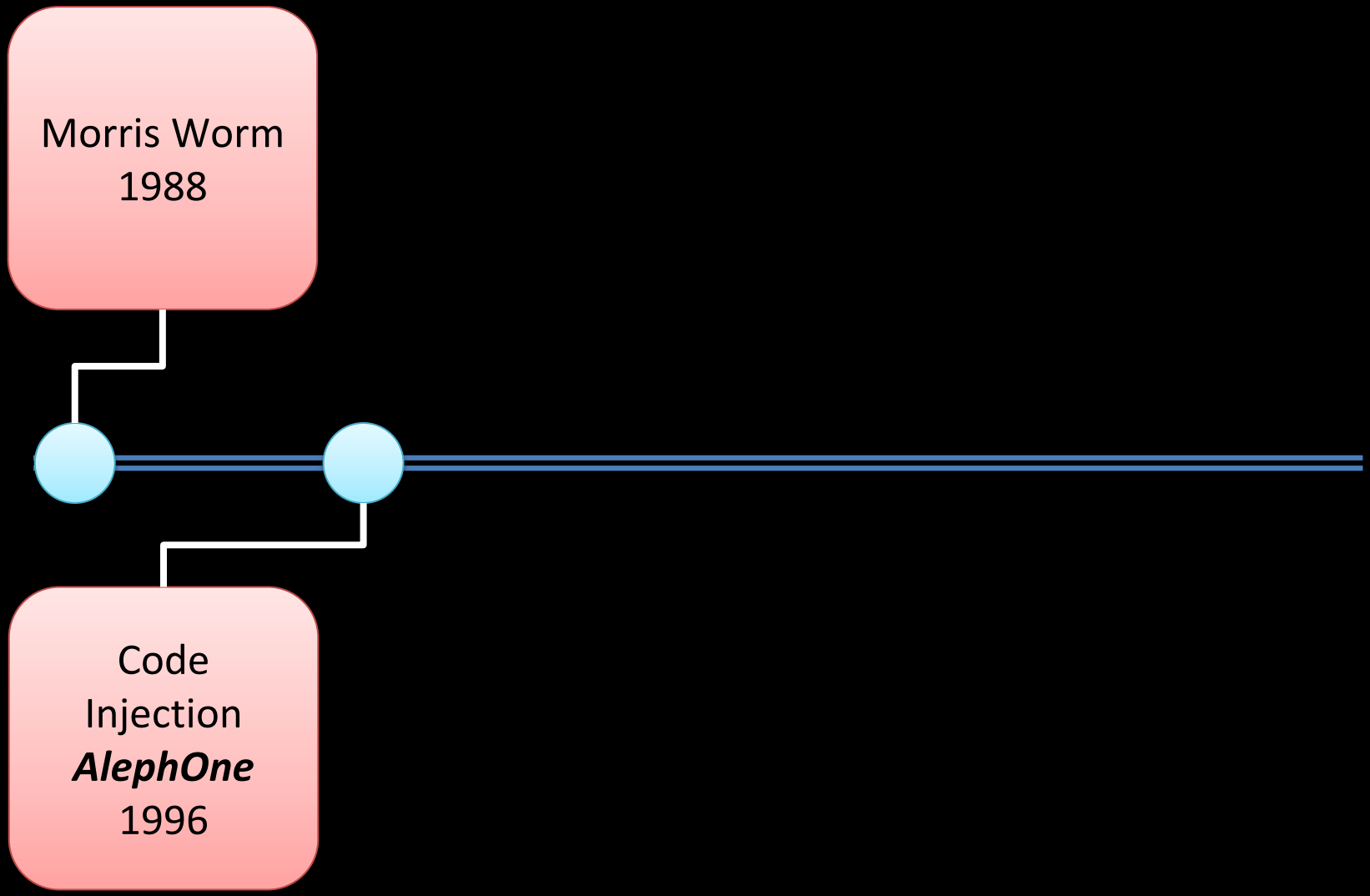
Three Decades of Run-Time Attacks

Three Decades of Run-Time Attacks

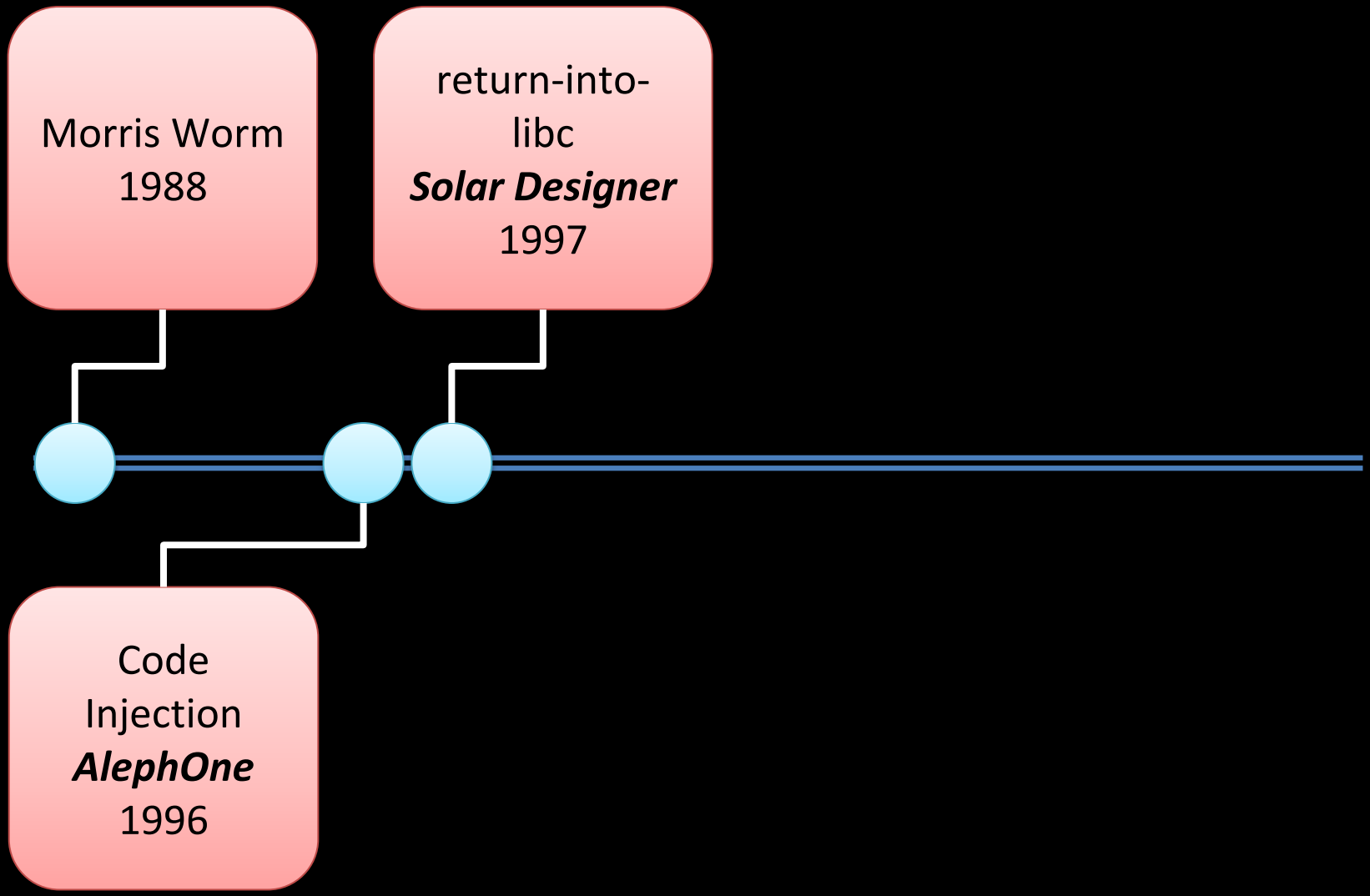
Three Decades of Run-Time Attacks



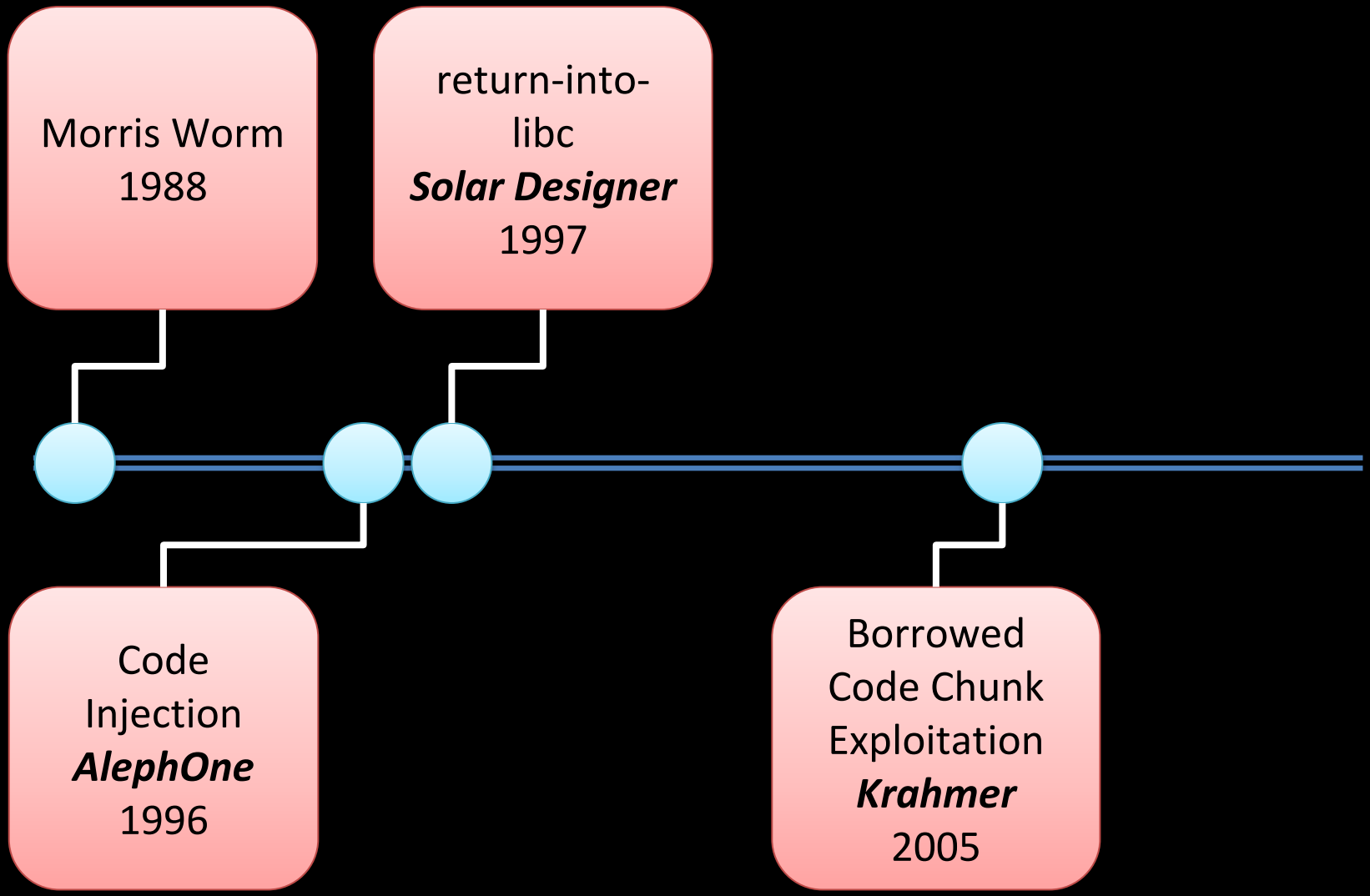
Three Decades of Run-Time Attacks



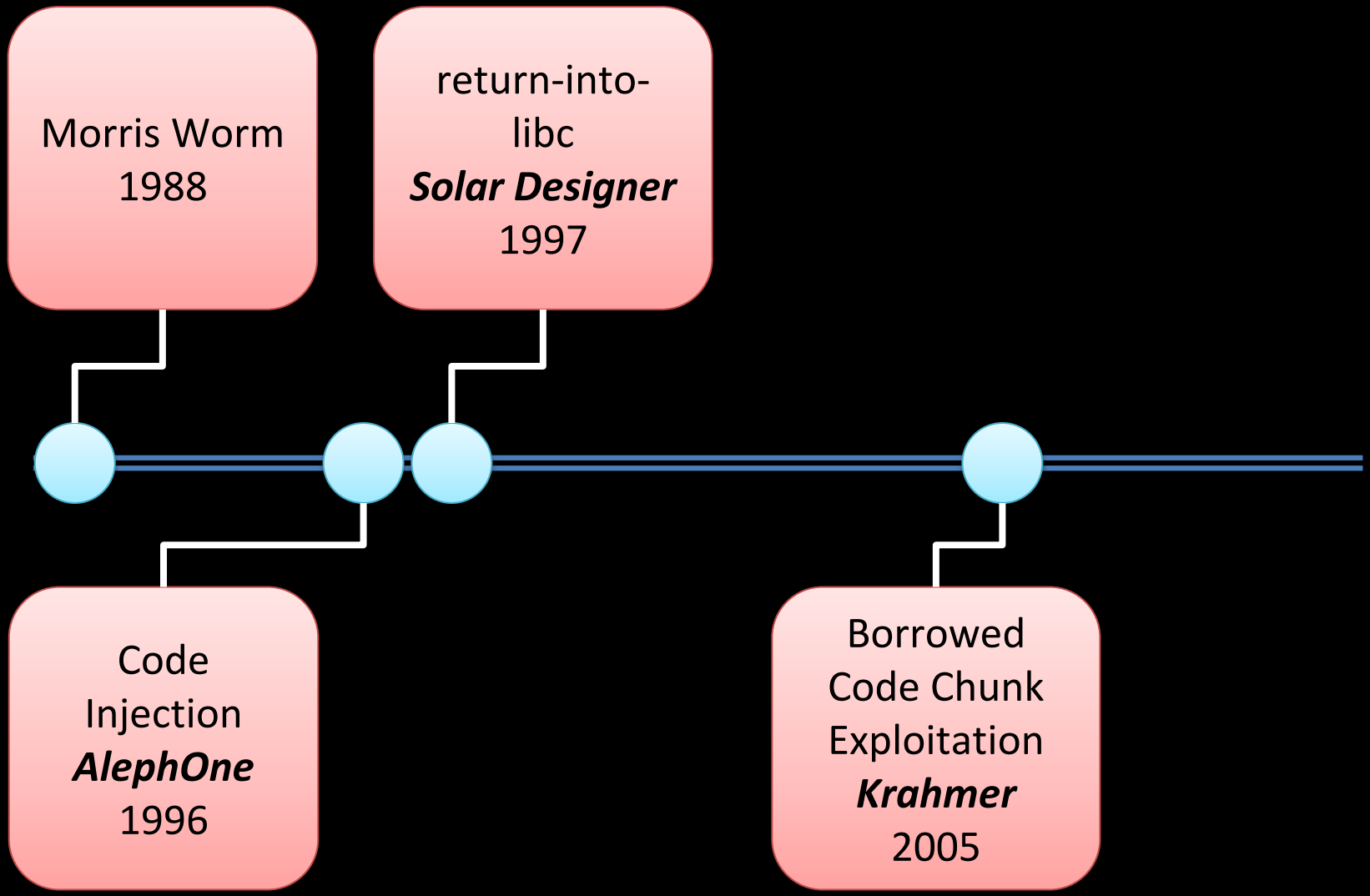
Three Decades of Run-Time Attacks



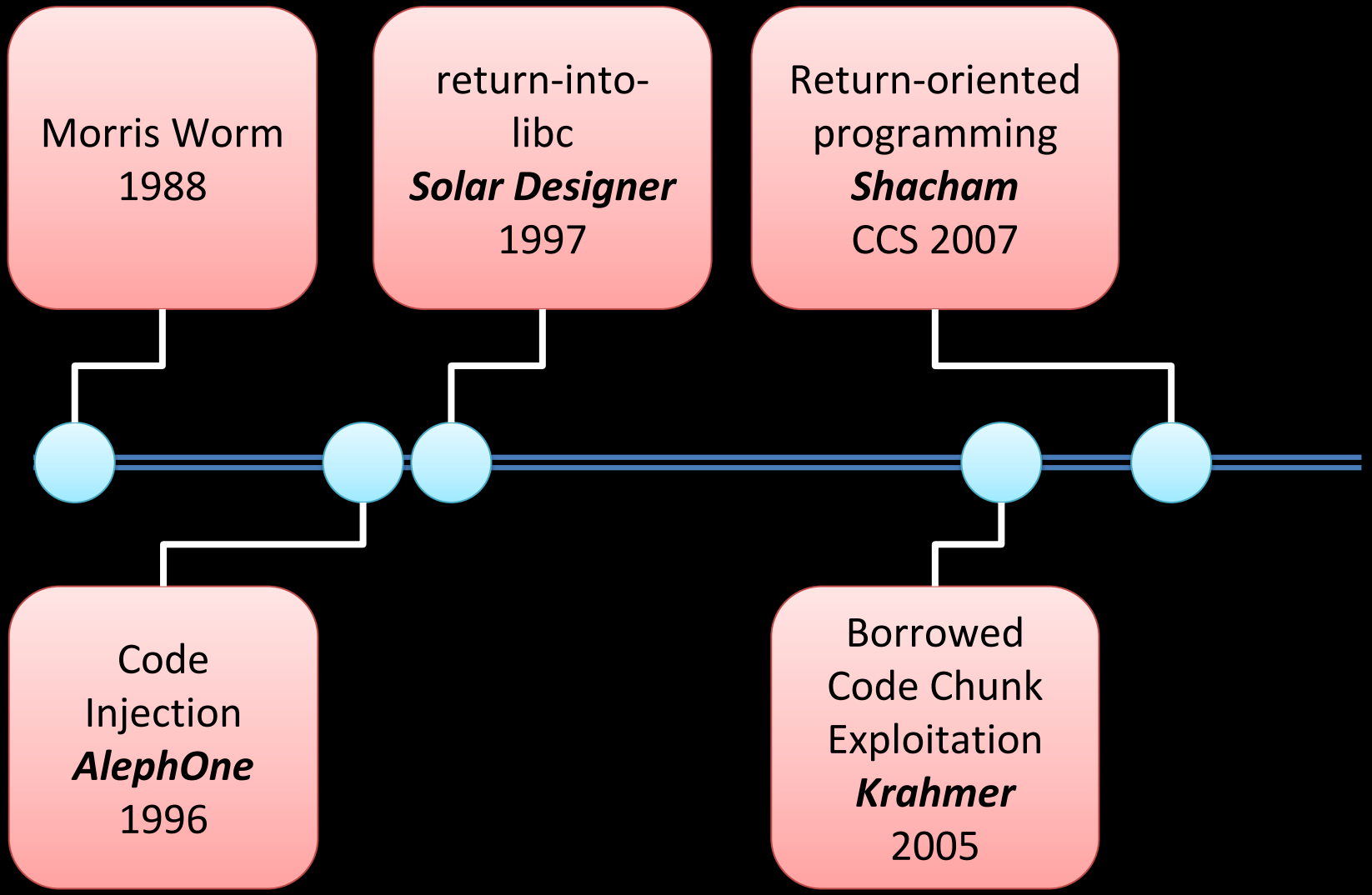
Three Decades of Run-Time Attacks



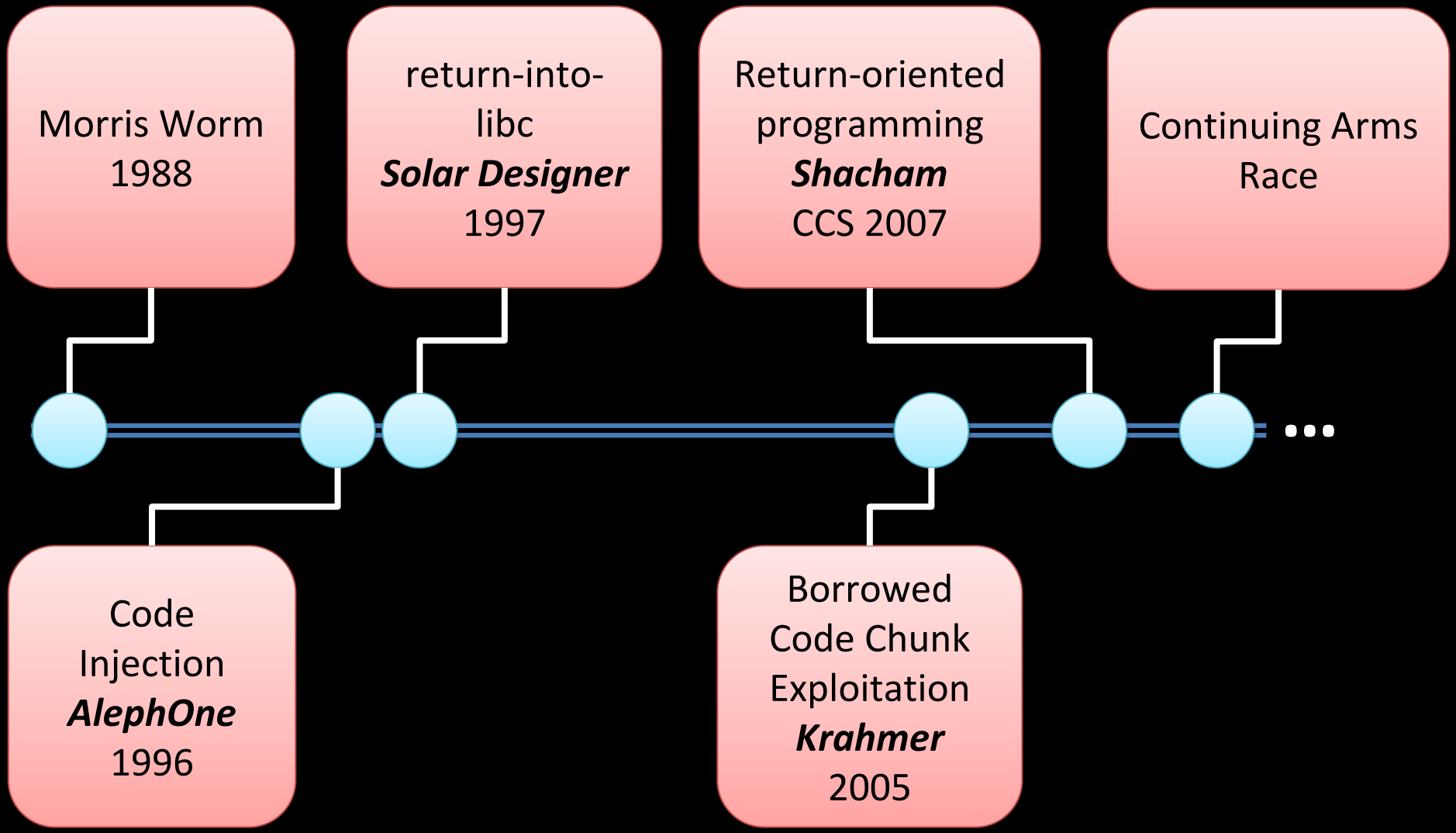
Three Decades of Run-Time Attacks



Three Decades of Run-Time Attacks



Three Decades of Run-Time Attacks



Relevance and Impact

Relevance and Impact

High Impact of Attacks

- Web browsers repeatedly exploited in pwn2own contests
- Zero-day issues exploited in Stuxnet/Duqu [Microsoft, BH 2012]
- iOS jailbreak

Relevance and Impact

High Impact of Attacks

- Web browsers repeatedly exploited in pwn2own contests
- Zero-day issues exploited in Stuxnet/Duqu [Microsoft, BH 2012]
- iOS jailbreak

Industry Efforts on Defenses

- Microsoft EMET (Enhanced Mitigation Experience Toolkit) includes a ROP detection engine
- Microsoft Control Flow Guard (CFG) in Windows 10
- Google‘s compiler extension VTV (virtual table verification)
- Intel Control-flow Enforcement Technology (CET)
- Kernel CFI (kCFI) since Android 9

But runtime exploits have also some “good” side-effects

Apple iPhone Jailbreak

Disable signature verification and escalate privileges to root

Apple iPhone Jailbreak

Disable signature verification and escalate privileges to root

Apple iPhone Jailbreak

Disable signature verification and escalate privileges to root

Request
http://www.jailbreakme.com/_/iPhone3,1_4.0.pdf

1) Exploit PDF Viewer Vulnerability by means of **Return-Oriented Programming**
2) Start Jailbreak

Apple iPhone Jailbreak

Disable signature verification and escalate privileges to root

Request
http://www.jailbreakme.com/_/iPhone3,1_4.0.pdf

1) Exploit PDF Viewer Vulnerability by means of **Return-Oriented Programming**
2) Start Jailbreak
3) Download required system files
4) Jailbreak Done

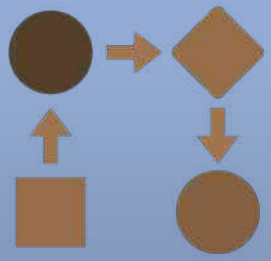
Overview

Background on Run-Time Attacks

Building Code Randomization Defenses

Code-Reuse Attacks

Building Control-Flow Integrity Defenses



Data-Oriented Exploits

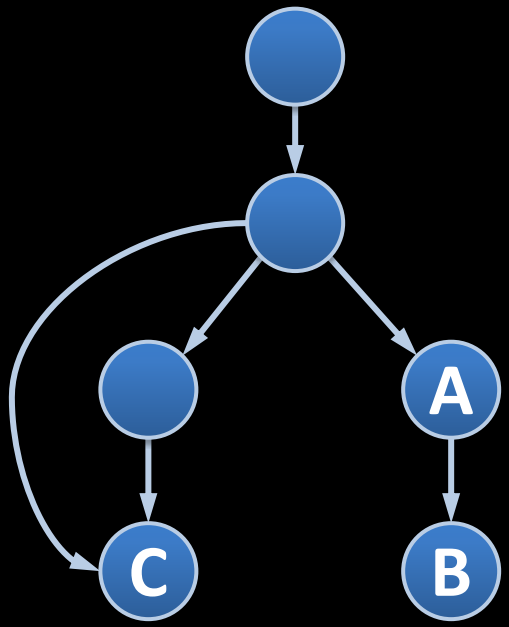
How can an attacker exploit vulnerabilities in a meaningful way?

Code Injection Attacks

Code-Reuse Attacks

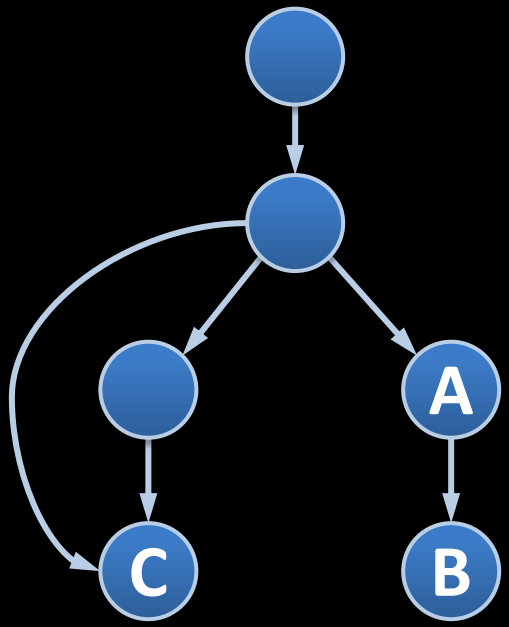
General Principle of Code Injection Attacks

Control-Flow Graph (CFG)

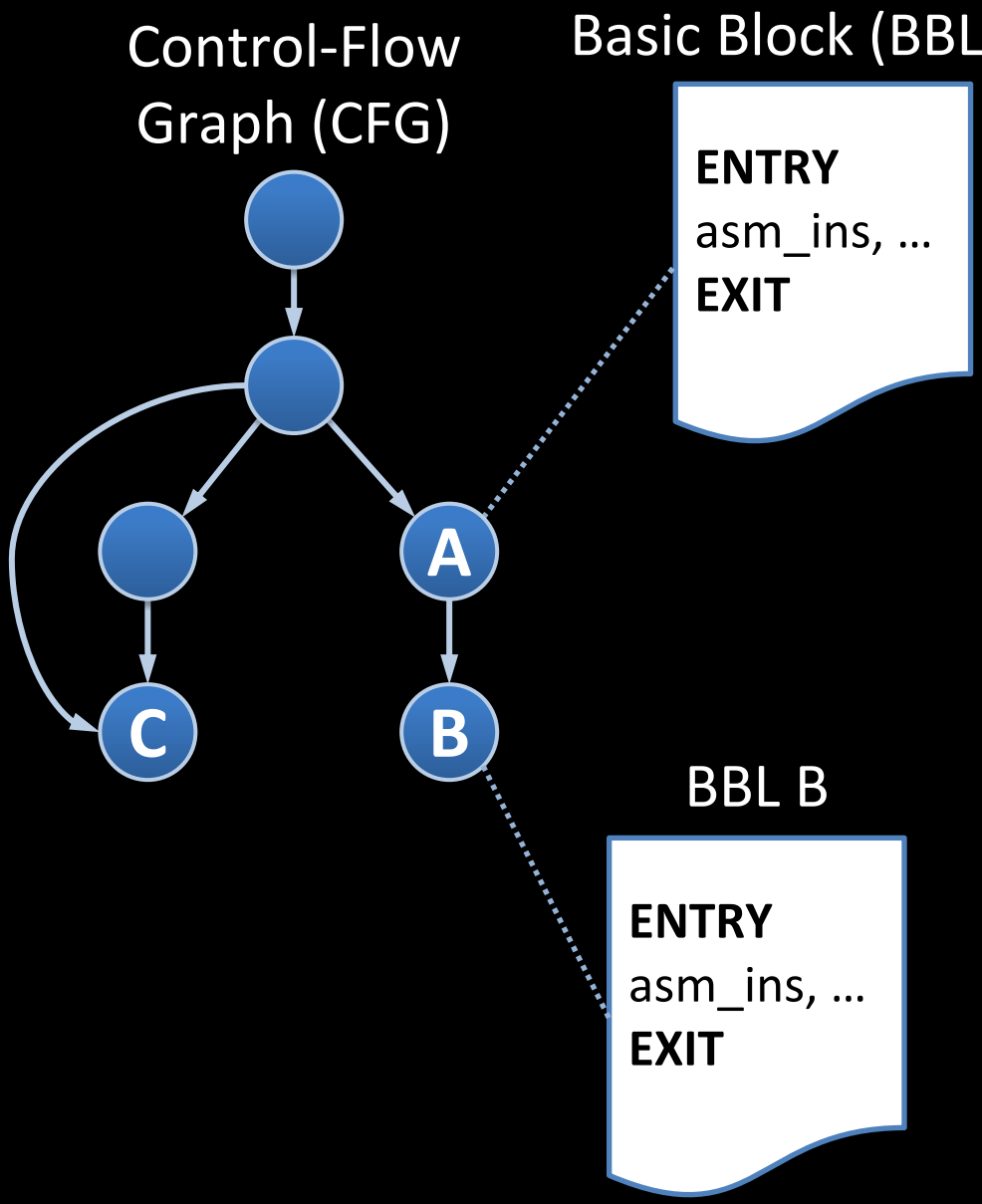


General Principle of Code Injection Attacks

Control-Flow Graph (CFG)

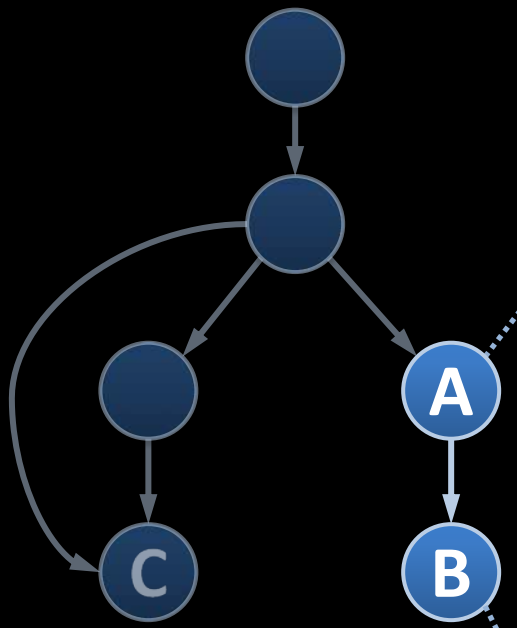


General Principle of Code Injection Attacks



General Principle of Code Injection Attacks

Control-Flow Graph (CFG)



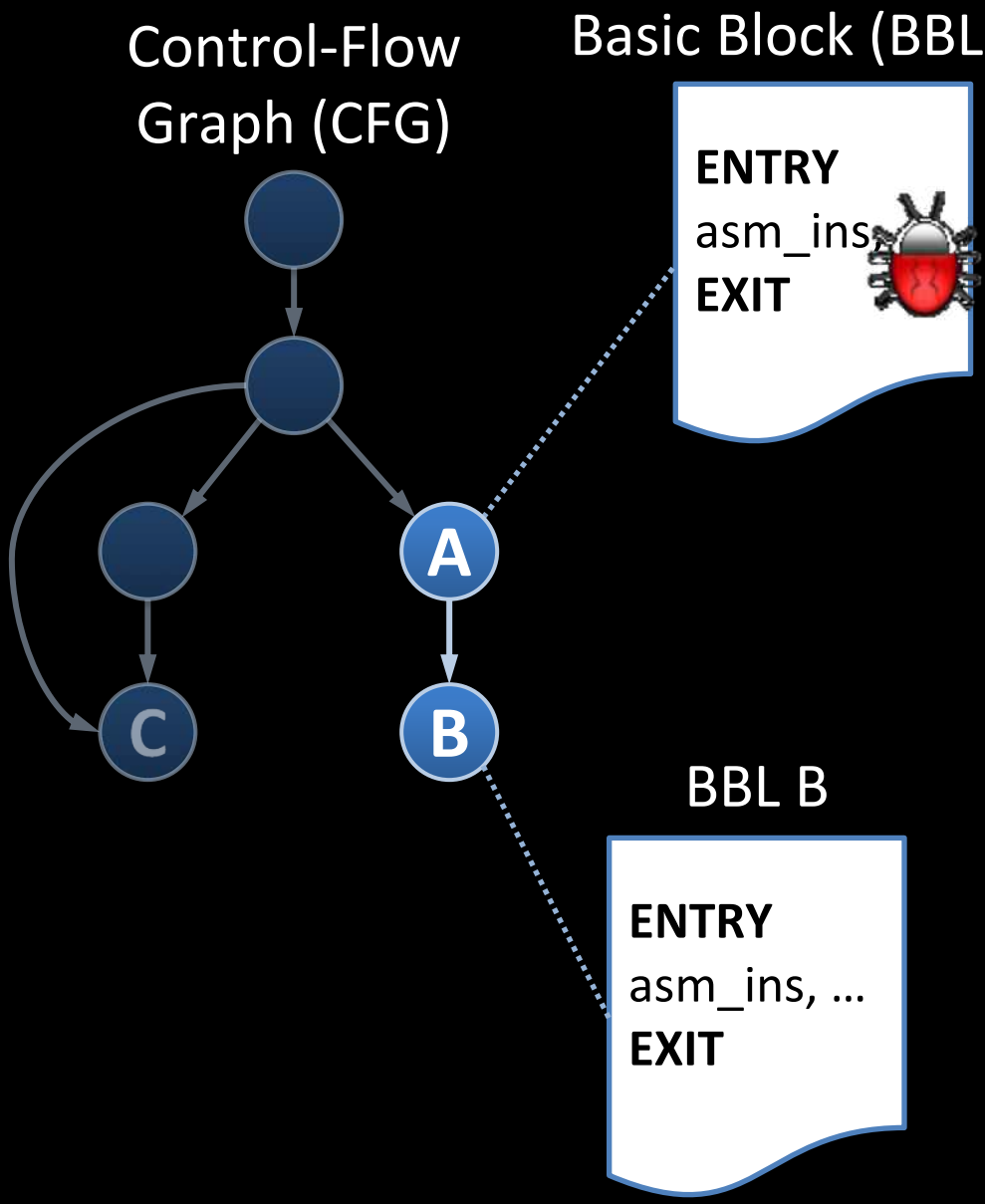
Basic Block (BBL) A

ENTRY
asm_ins, …
EXIT

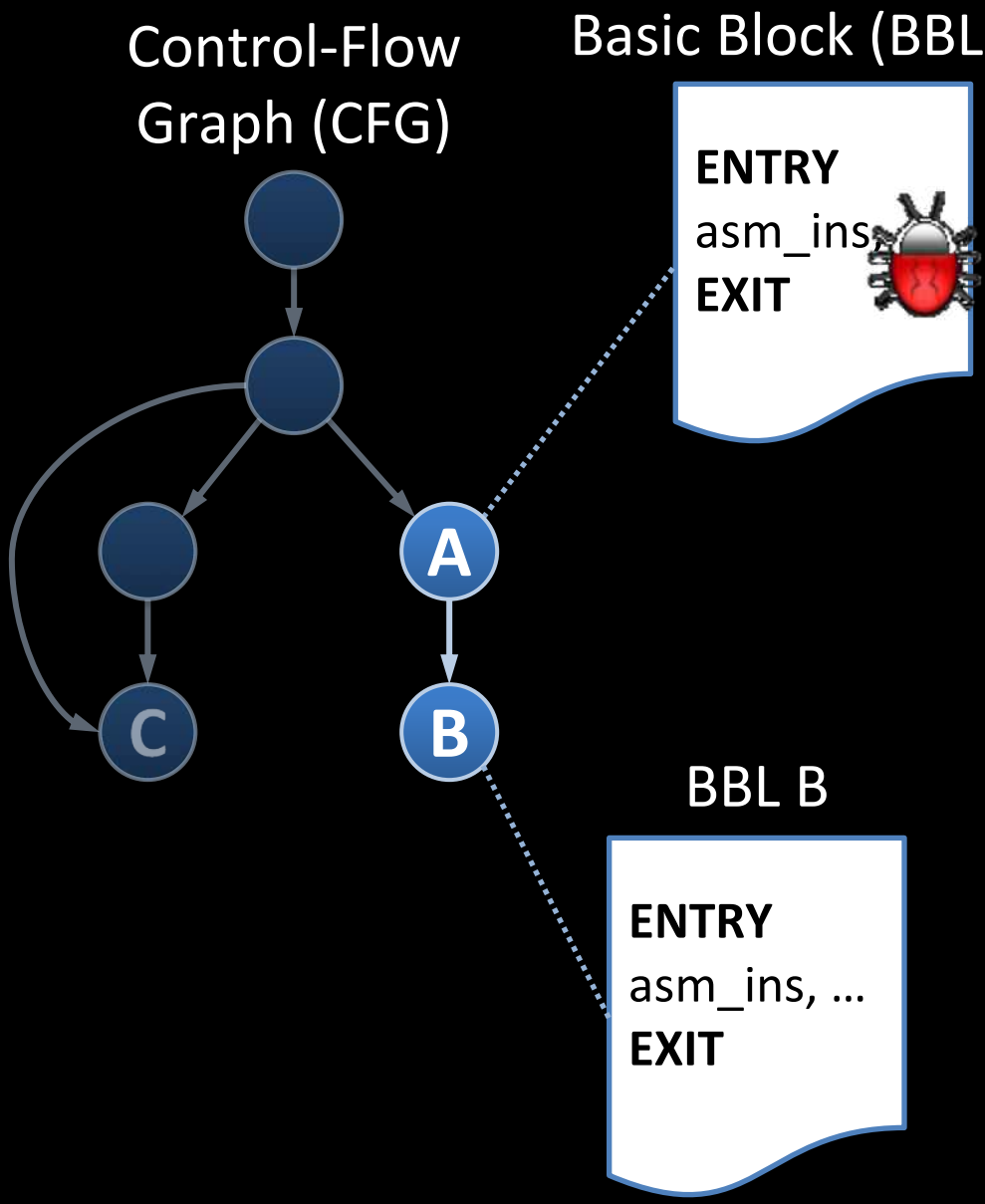
BBL B

ENTRY
asm_ins, …
EXIT

General Principle of Code Injection Attacks

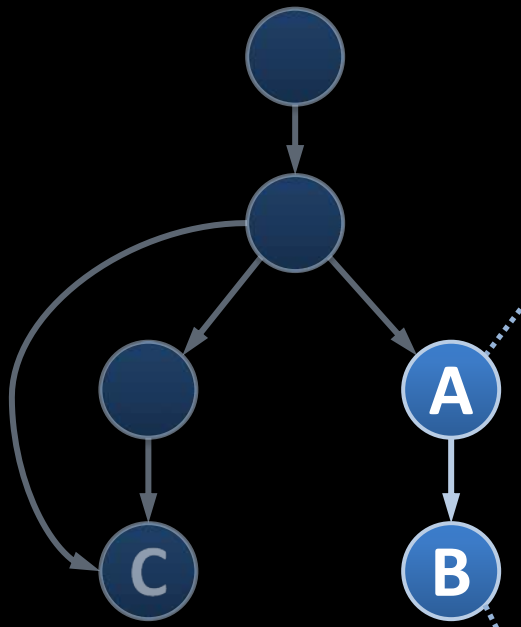


General Principle of Code Injection Attacks



General Principle of Code Injection Attacks

Control-Flow Graph (CFG)



Basic Block (BBL) A

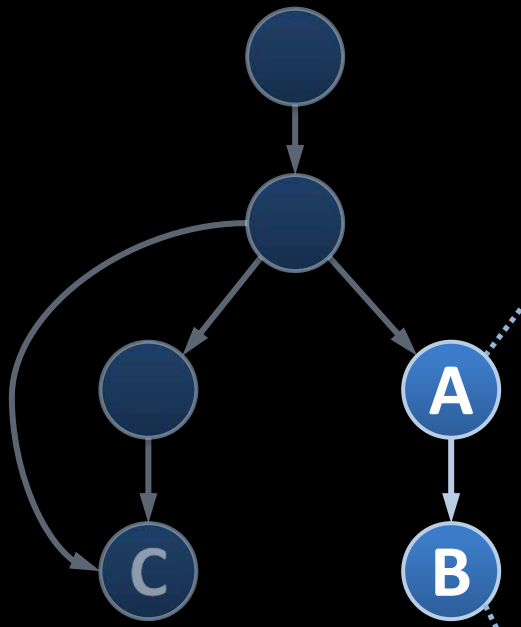
ENTRY
asm_ins
EXIT

BBL B

ENTRY
asm_ins, …
EXIT

General Principle of Code Injection Attacks

Control-Flow Graph (CFG)



Basic Block (BBL) A

ENTRY
asm_ins
EXIT

BBL B

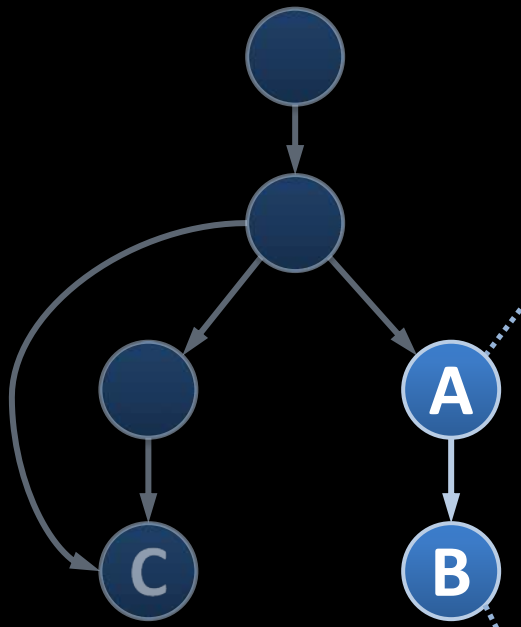
ENTRY
asm_ins, …
EXIT

Data flows

Program flows

General Principle of Code Injection Attacks

Control-Flow Graph (CFG)



Basic Block (BBL) A

ENTRY
asm_ins
EXIT

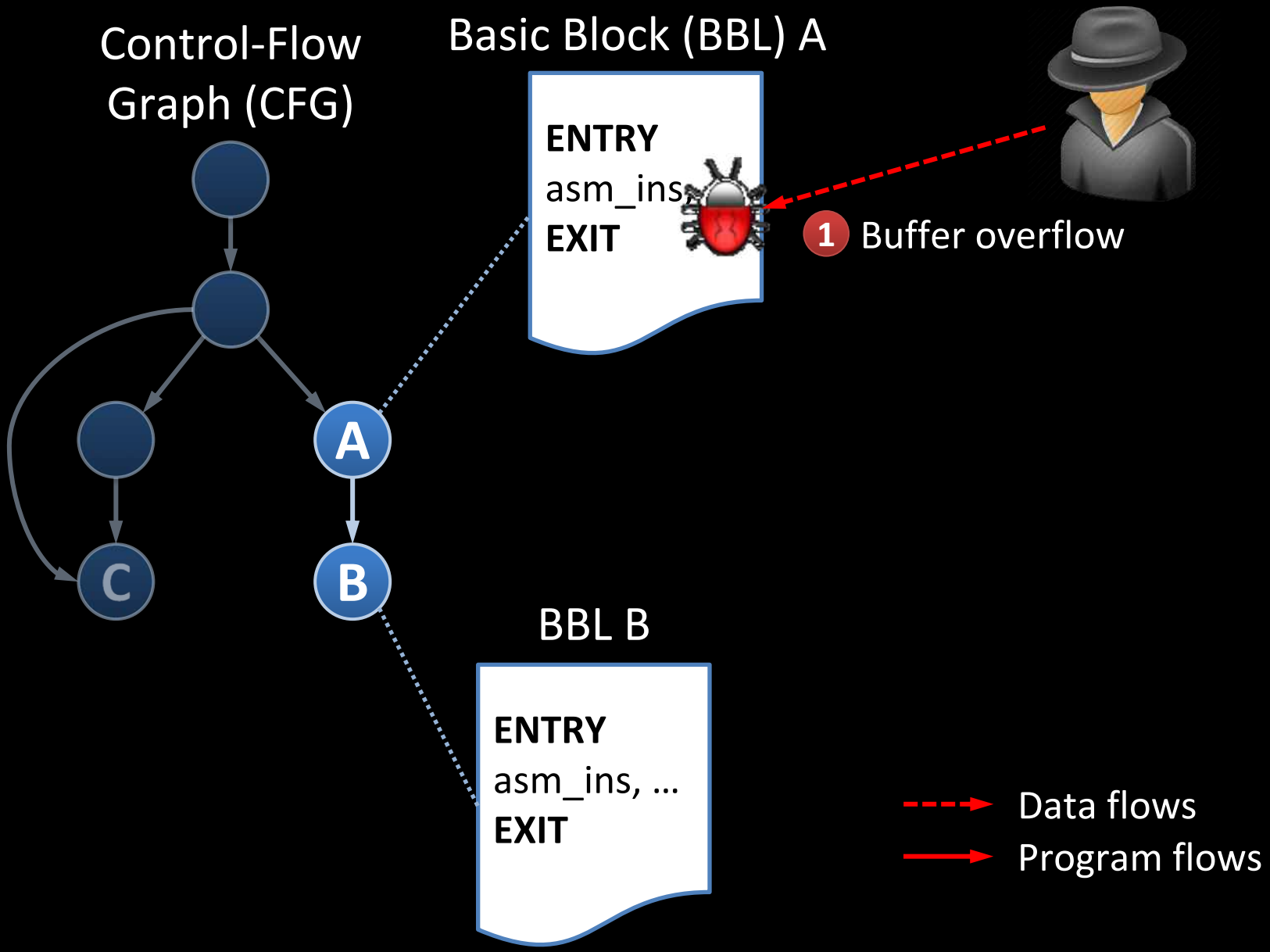
BBL B

ENTRY
asm_ins, …
EXIT

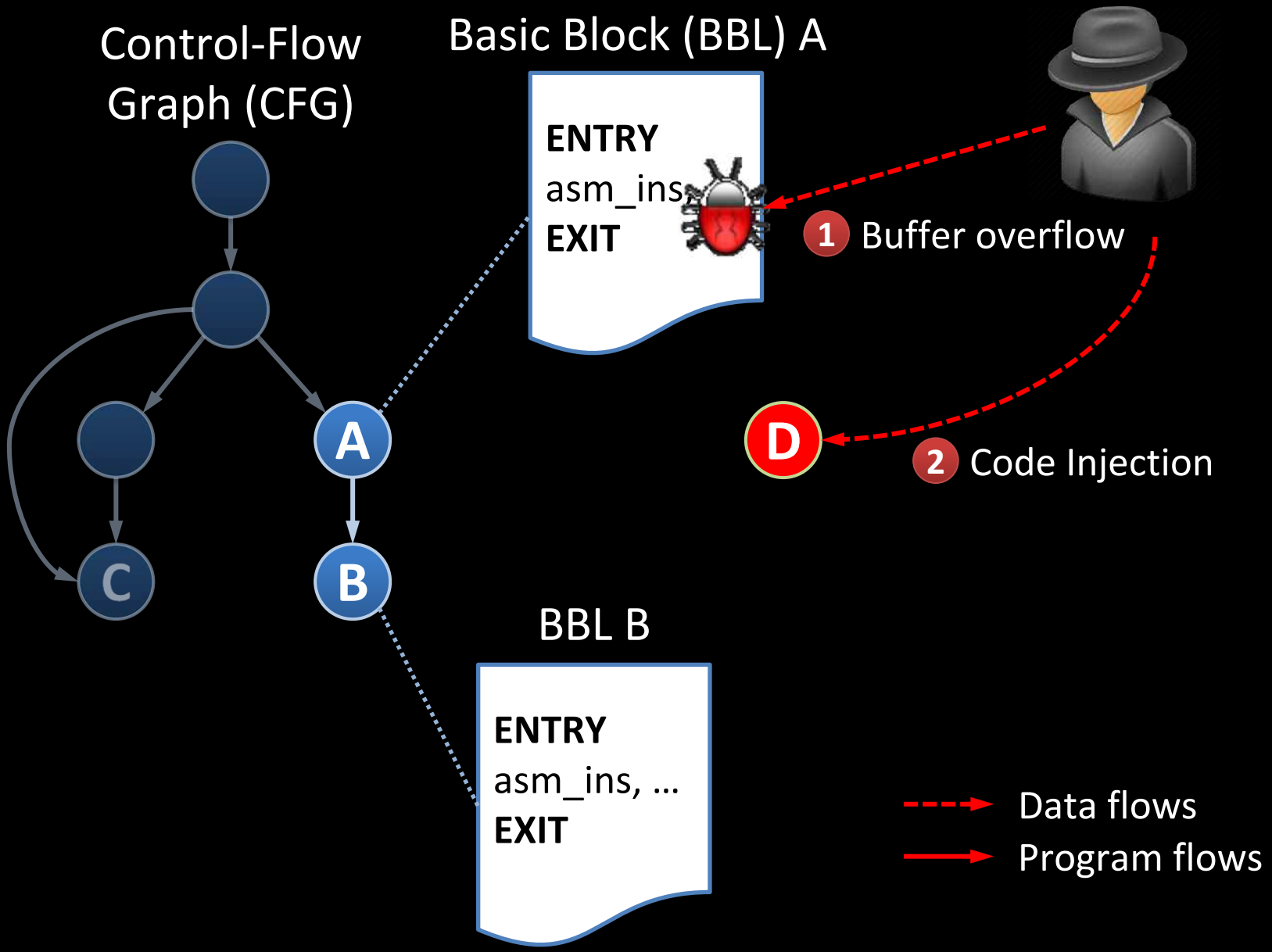
Data flows

Program flows

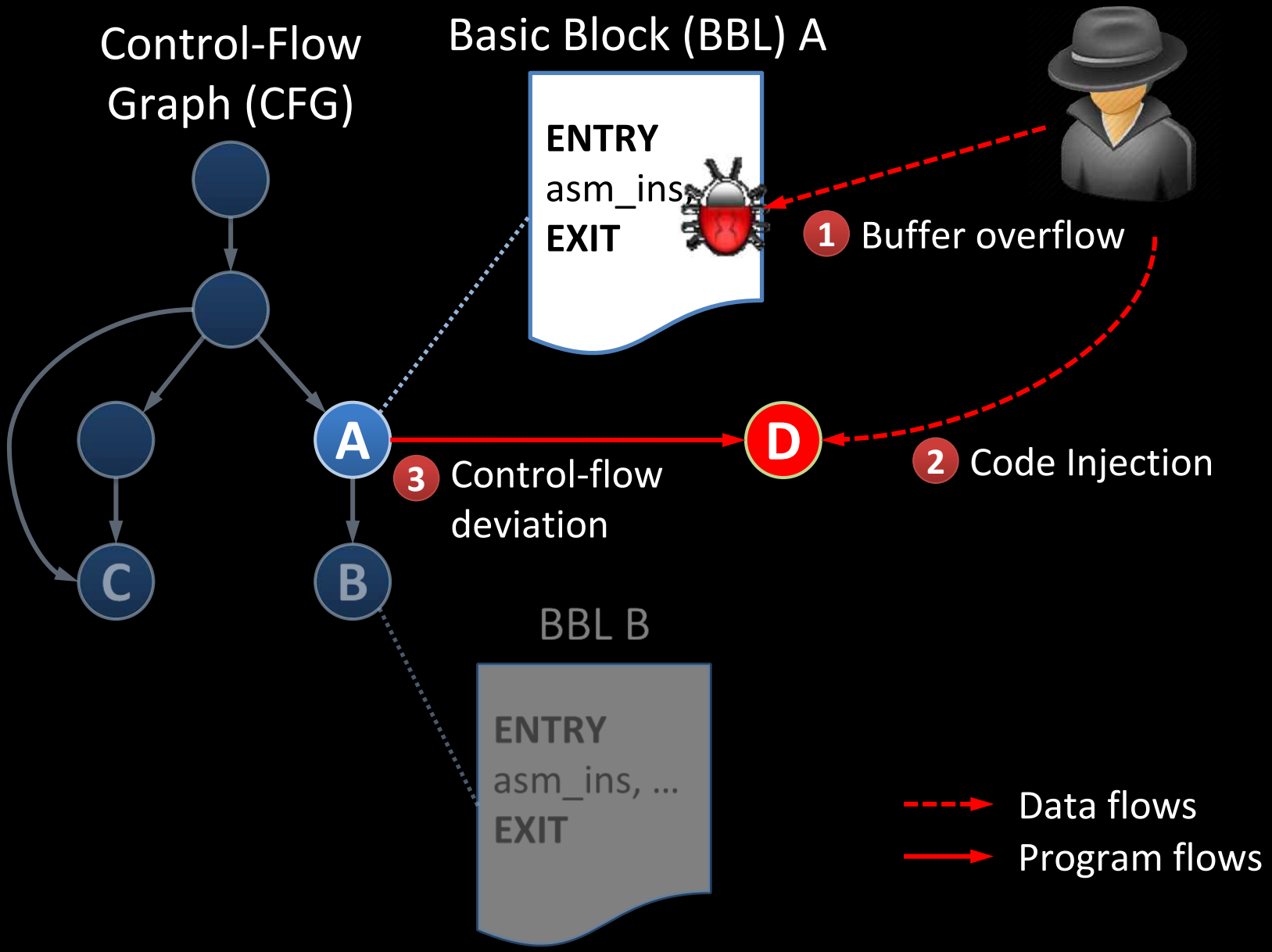
General Principle of Code Injection Attacks



General Principle of Code Injection Attacks



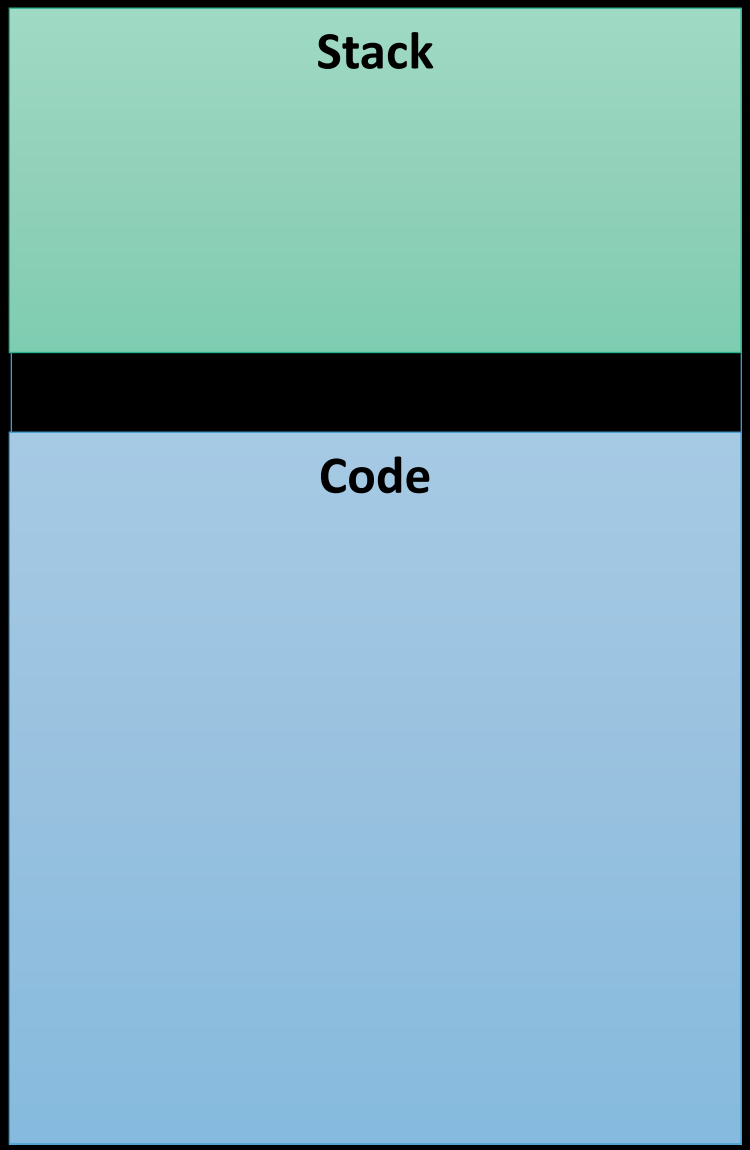
General Principle of Code Injection Attacks



Example: Code Injection Attack

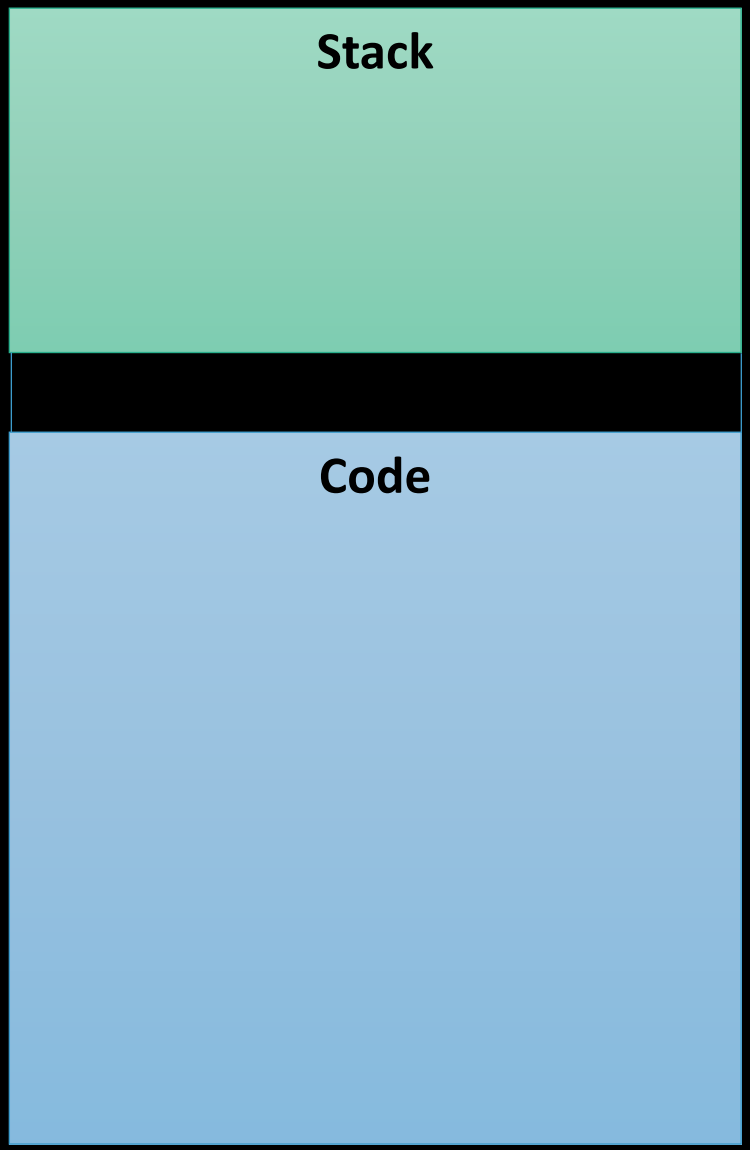
Example: Code Injection Attack

Program Memory



Example: Code Injection Attack

Program Memory



Example: Code Injection Attack

Program Memory

Stack

Code

<main>:

```
x86_ins…
CALL func()
x86_ins…
```

Example: Code Injection Attack

Program Memory

Stack

Code

<main>:

```
x86_ins…
CALL func()
x86_ins…
```

Example: Code Injection Attack

Program Memory

Stack

Code

<main>:

```
x86_ins…
CALL func()
x86_ins…
```

Example: Code Injection Attack

Program Memory

Stack

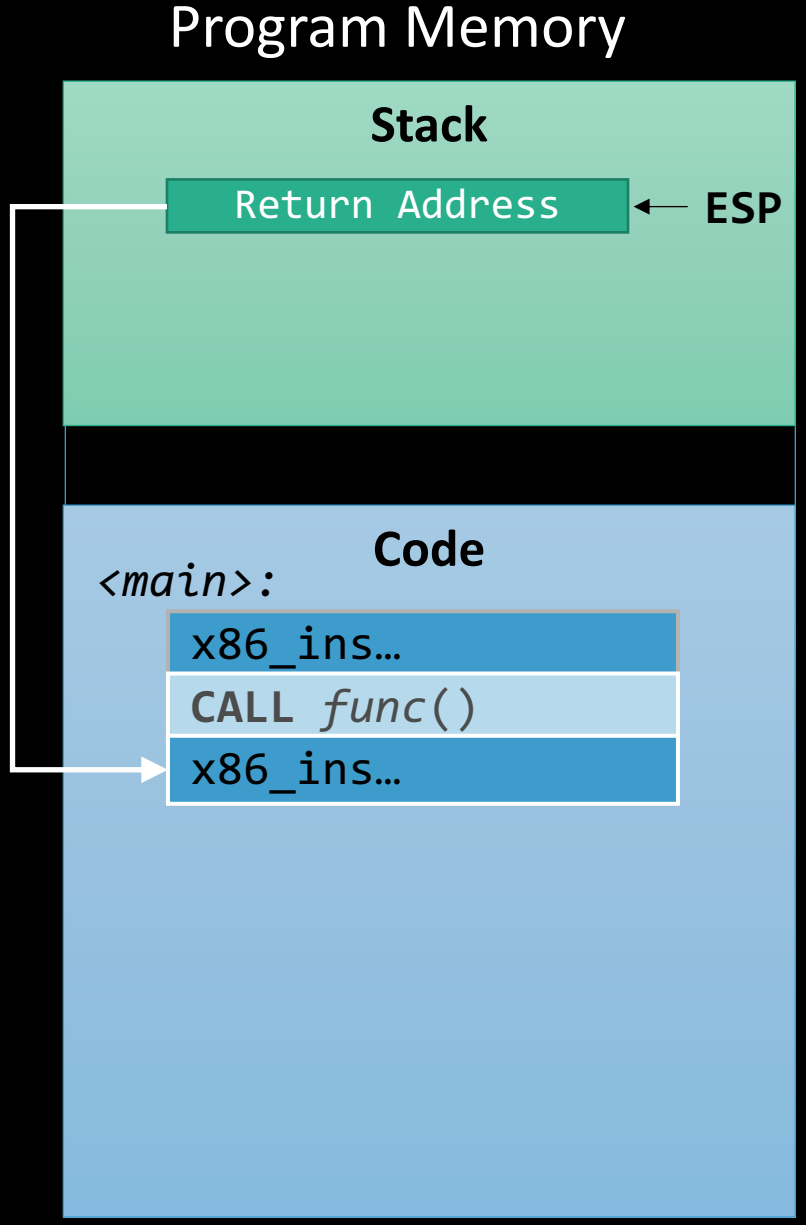
Return Address ← **ESP**

Code

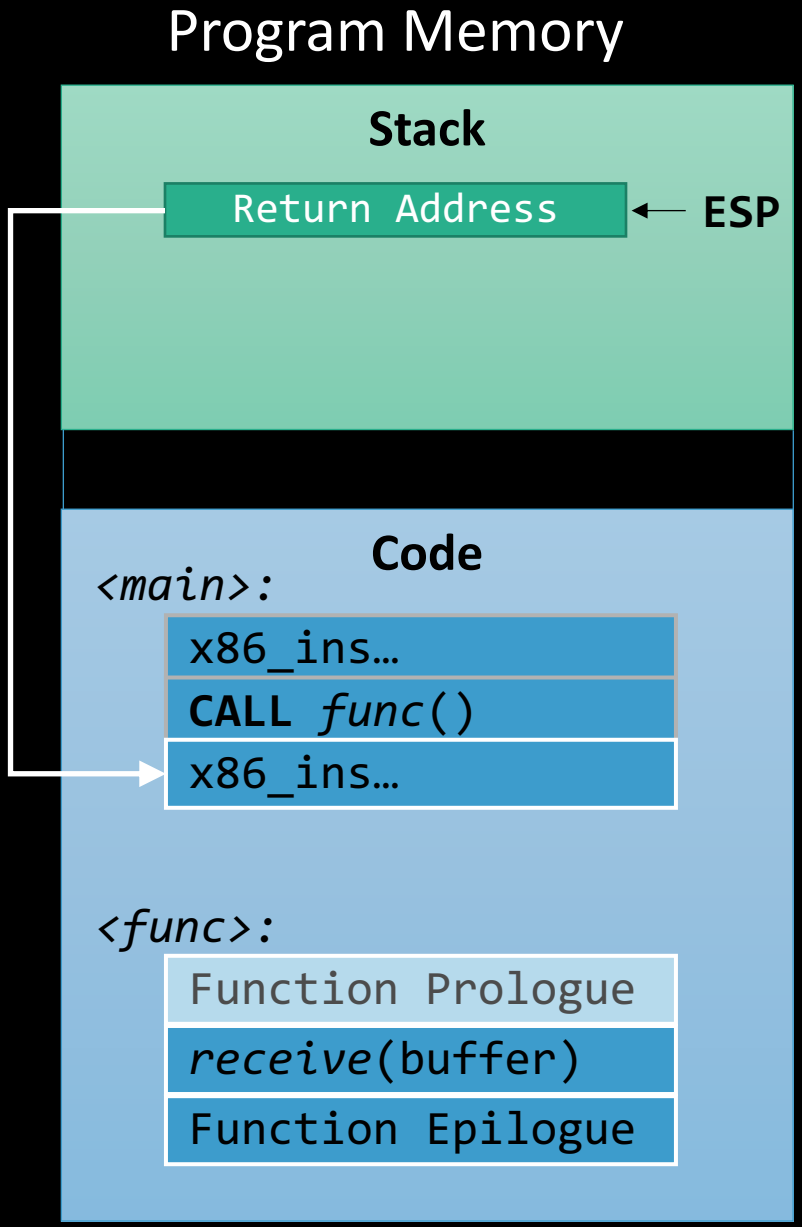
<main>:

```
x86_ins…
CALL func()
x86_ins…
```

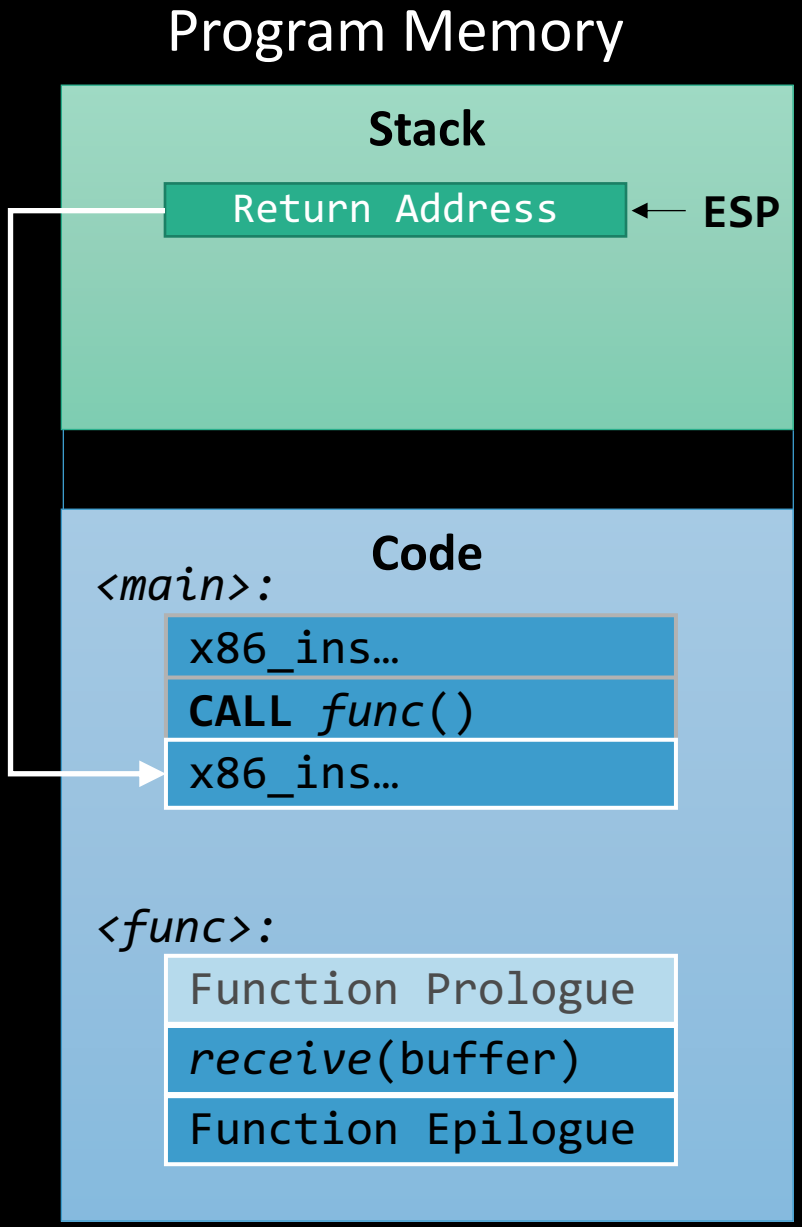
Example: Code Injection Attack



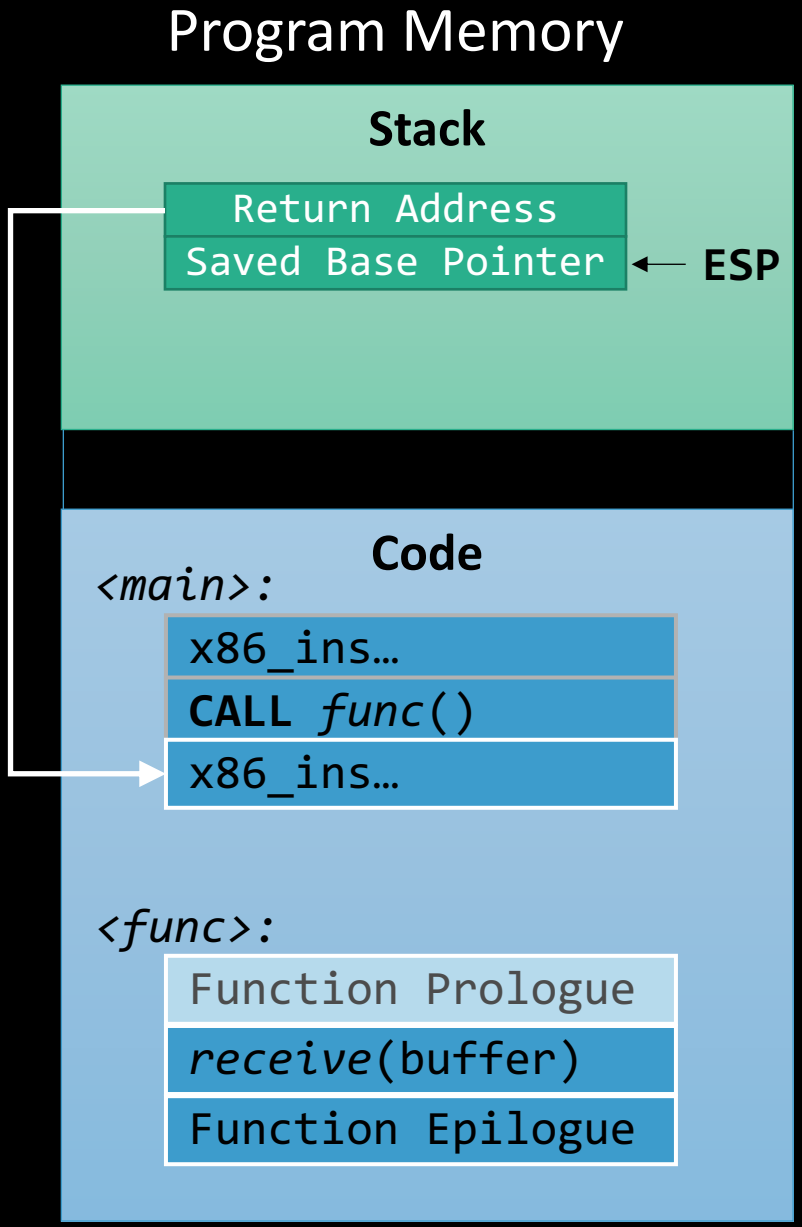
Example: Code Injection Attack



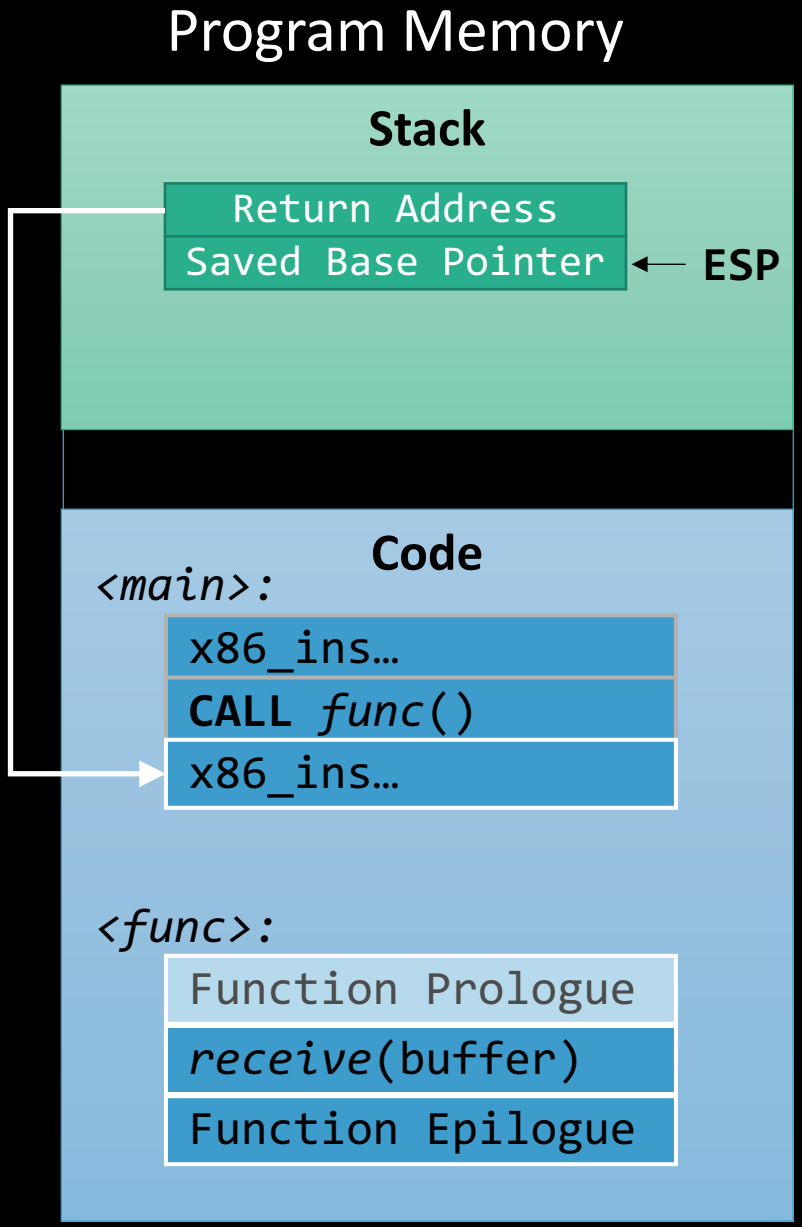
Example: Code Injection Attack



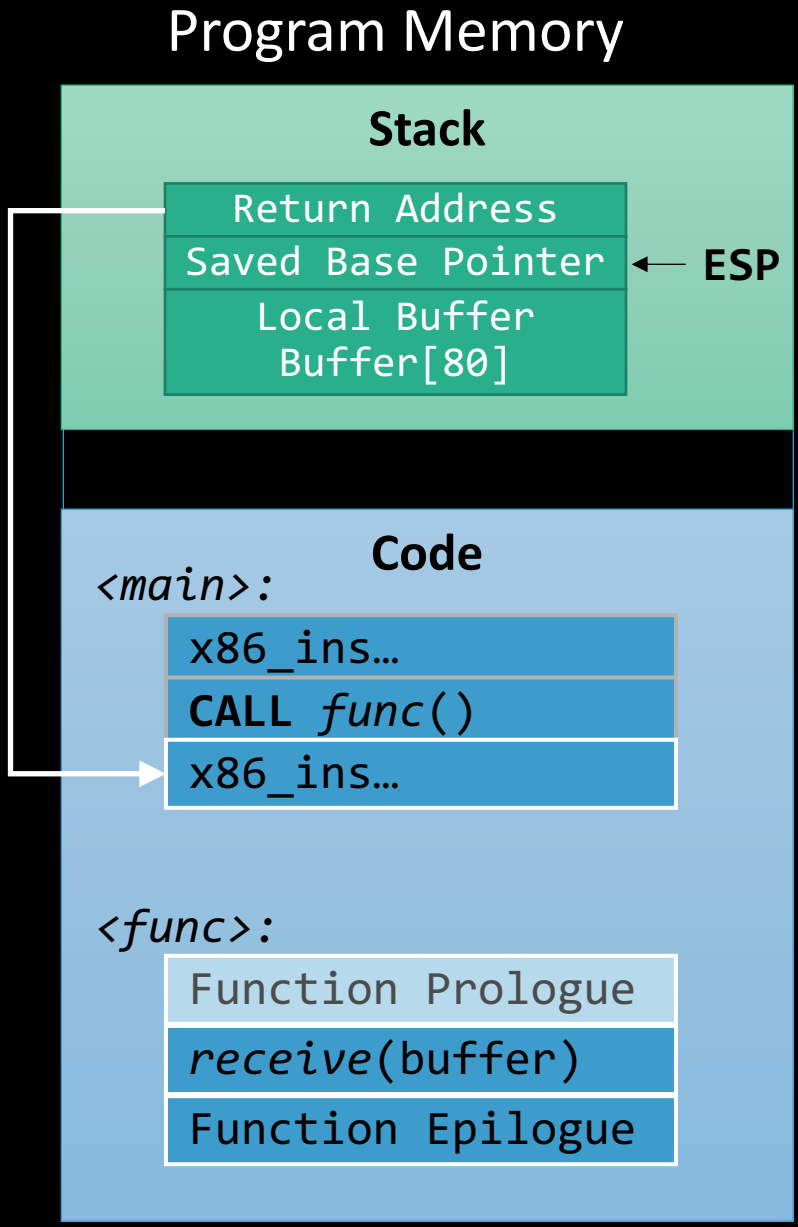
Example: Code Injection Attack



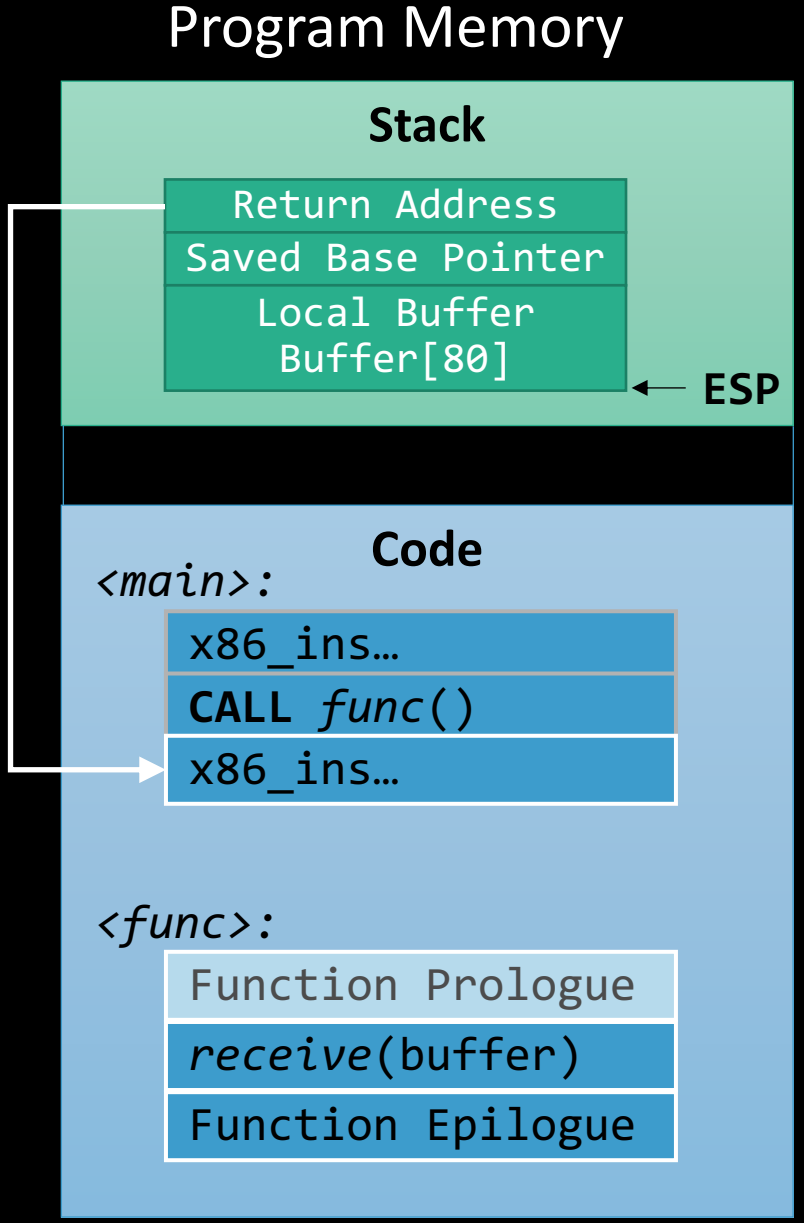
Example: Code Injection Attack



Example: Code Injection Attack



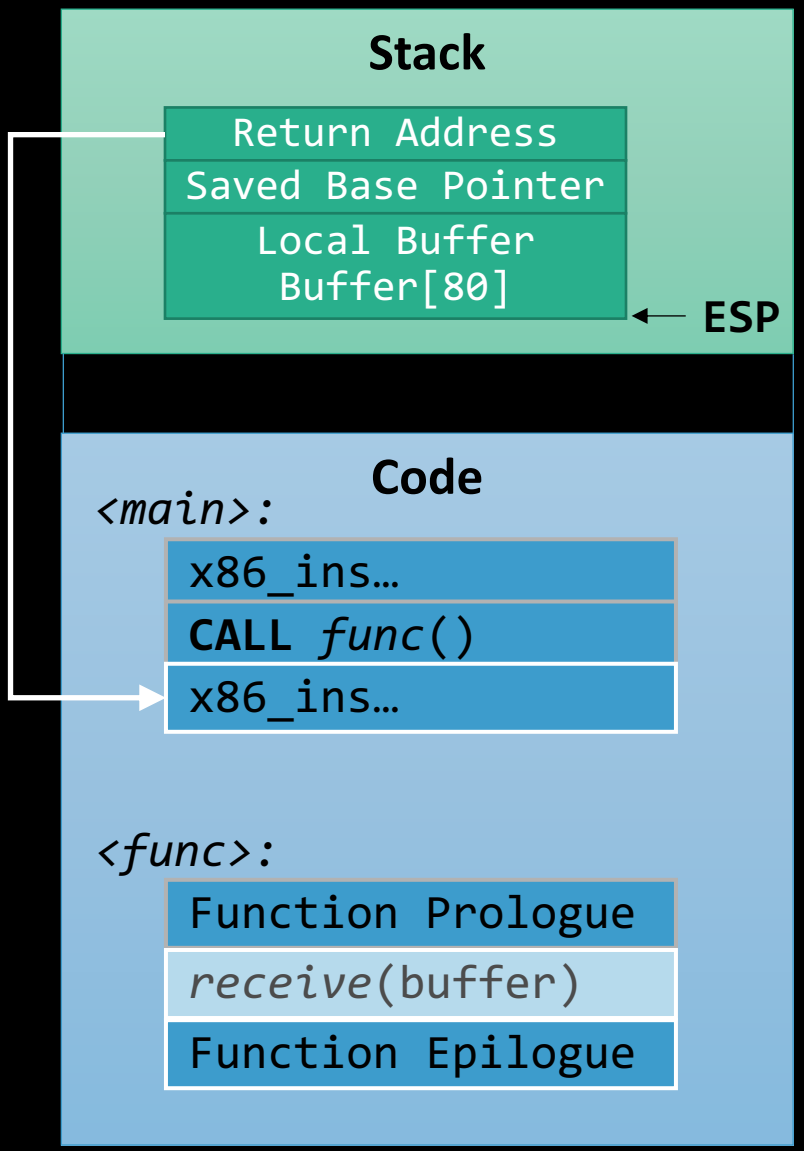
Example: Code Injection Attack



Example: Code Injection Attack

Corrupt Control Structures

Program Memory



Example: Code Injection Attack

Program Memory

Stack

Return Address
Saved Base Pointer
Local Buffer
Buffer[80]

← **ESP**

Corrupt Control Structures

Code

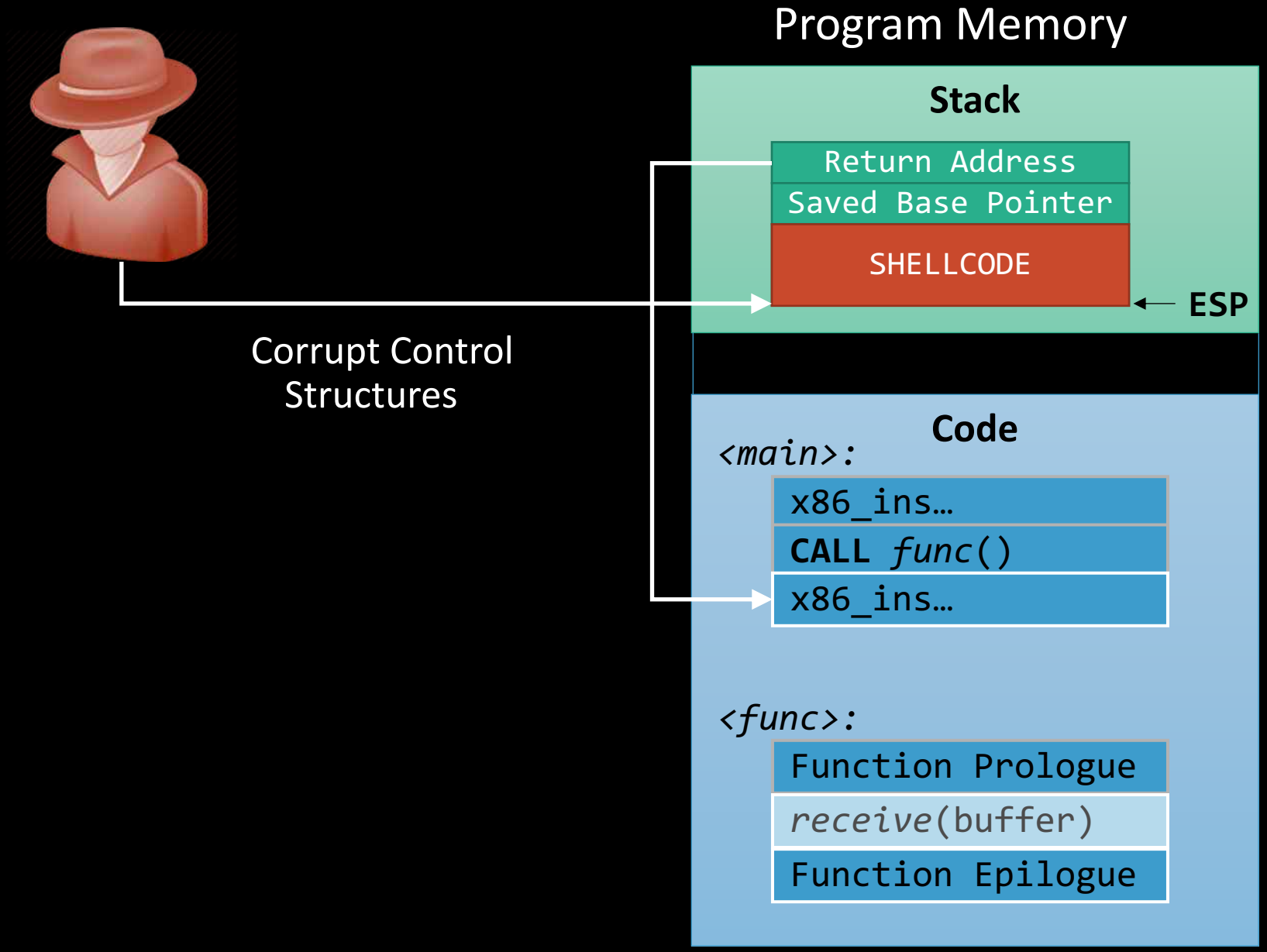
<main>:

x86_ins…
CALL *func*()
x86_ins…

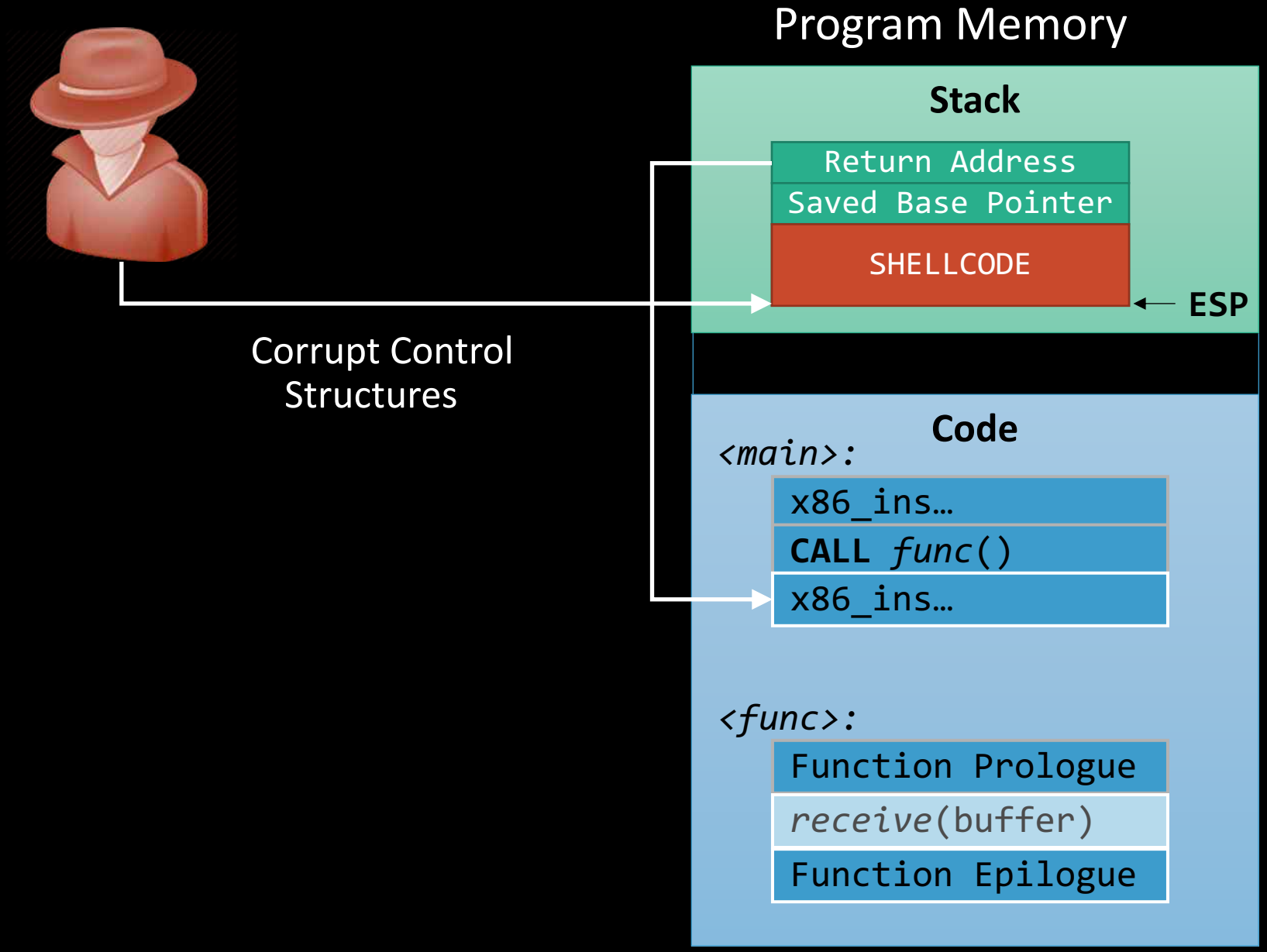
<func>:

Function Prologue
receive(buffer)
Function Epilogue

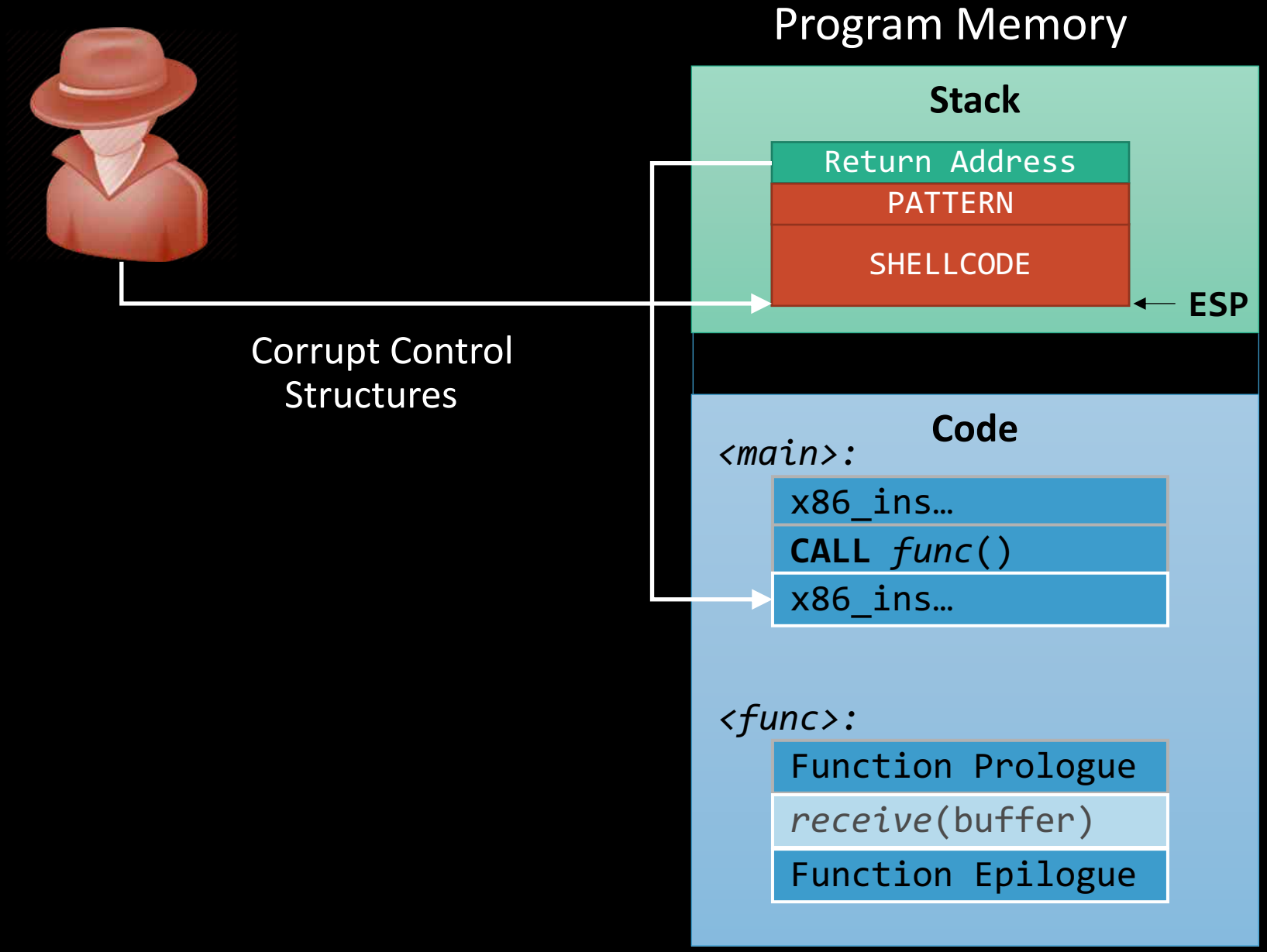
Example: Code Injection Attack



Example: Code Injection Attack



Example: Code Injection Attack



Example: Code Injection Attack

Corrupt Control Structures

Program Memory

Stack

NEW RETURN ADDR

PATTERN

SHELLCODE

ESP

Code

<main>:

```
x86_ins…
CALL func()
x86_ins…
```

<func>:

```
Function Prologue
receive(buffer)
Function Epilogue
```

Example: Code Injection Attack

Corrupt Control Structures

Program Memory

Stack

NEW RETURN ADDR

PATTERN

SHELLCODE

ESP

Code

<main>:

```
x86_ins…
CALL func()
x86_ins…
```

<func>:

```
Function Prologue
receive(buffer)
Function Epilogue
```

Example: Code Injection Attack

Corrupt Control Structures

Program Memory

Stack

NEW RETURN ADDR

PATTERN

SHELLCODE

ESP

Code

<main>:

```
x86_ins…
CALL func()
x86_ins…
```

<func>:

```
Function Prologue
receive(buffer)
Function Epilogue
```

func() now returns!

Example: Code Injection Attack

Corrupt Control Structures

Program Memory

Stack

NEW RETURN ADDR

PATTERN

SHELLCODE

ESP

Code

<main>:

x86_ins…

CALL *func*()

x86_ins…

<func>:

Function Prologue

receive(buffer)

Function Epilogue

func() now returns!

Example: Code Injection Attack

Corrupt Control Structures

Program Memory

Stack

NEW RETURN ADDR ← **ESP**

PATTERN

SHELLCODE

Code

<main>:

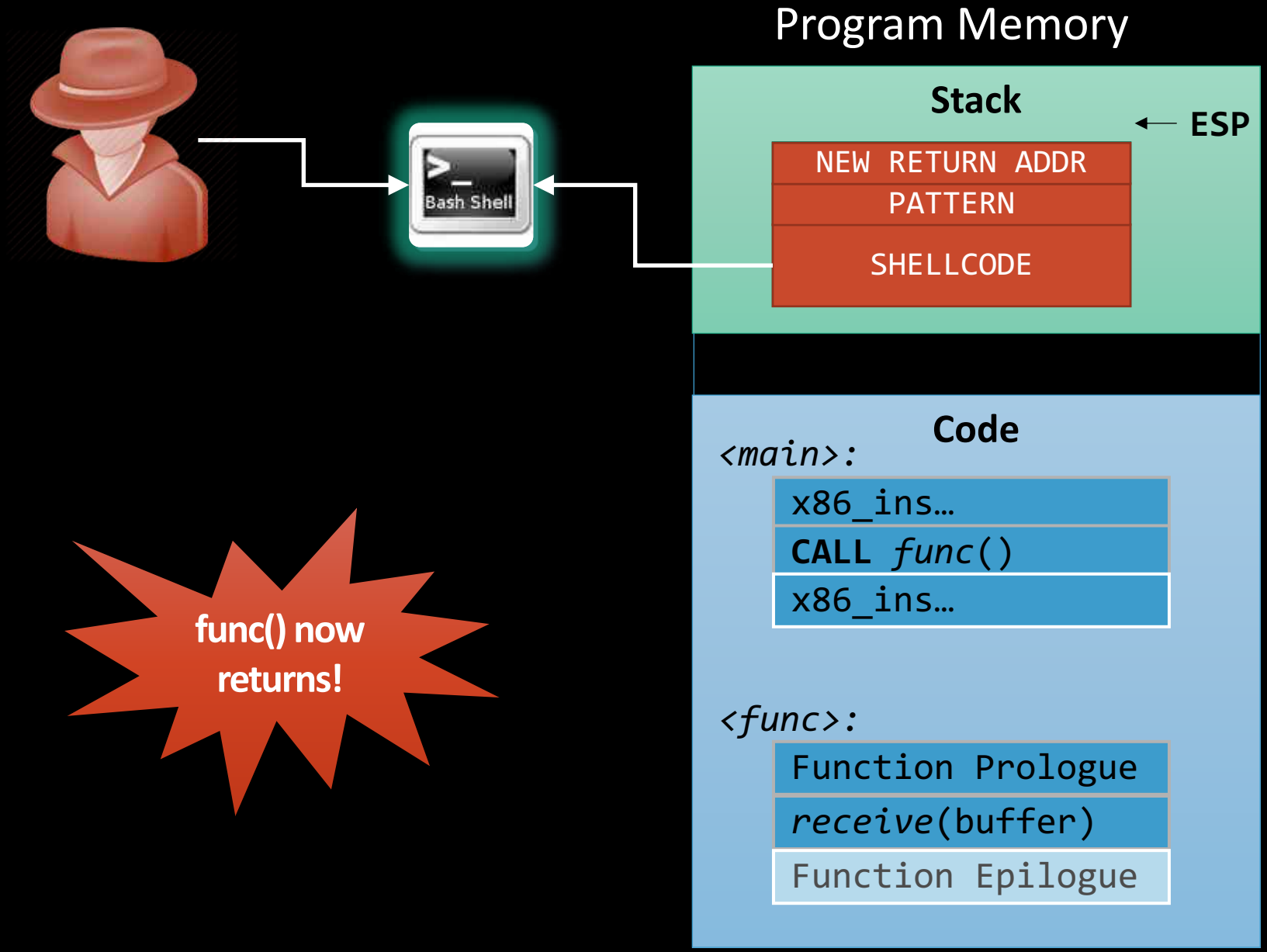
```
x86_ins…
CALL func()
x86_ins…
```

<func>:

```
Function Prologue
receive(buffer)
Function Epilogue
```

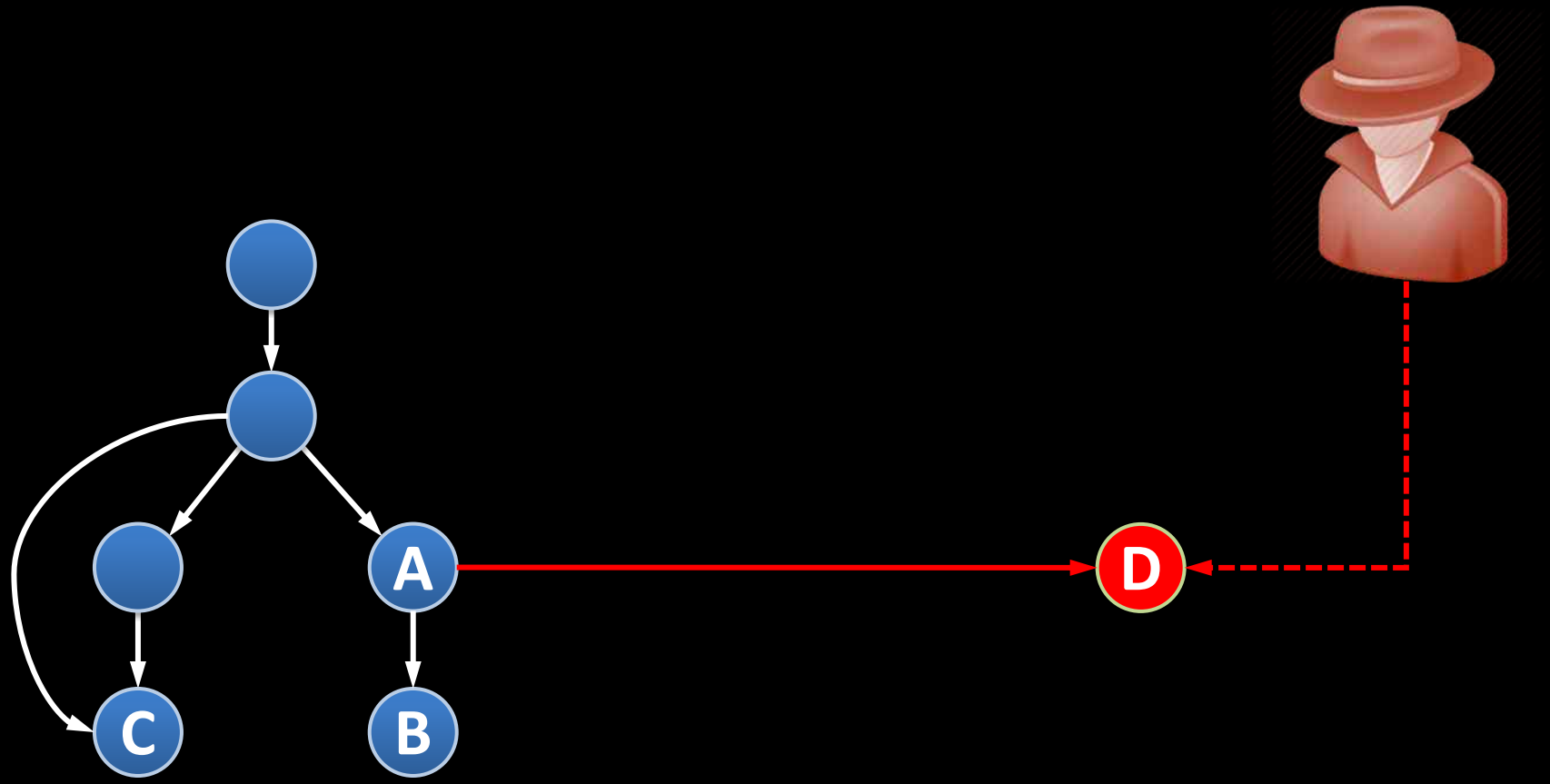
func() now returns!

Example: Code Injection Attack



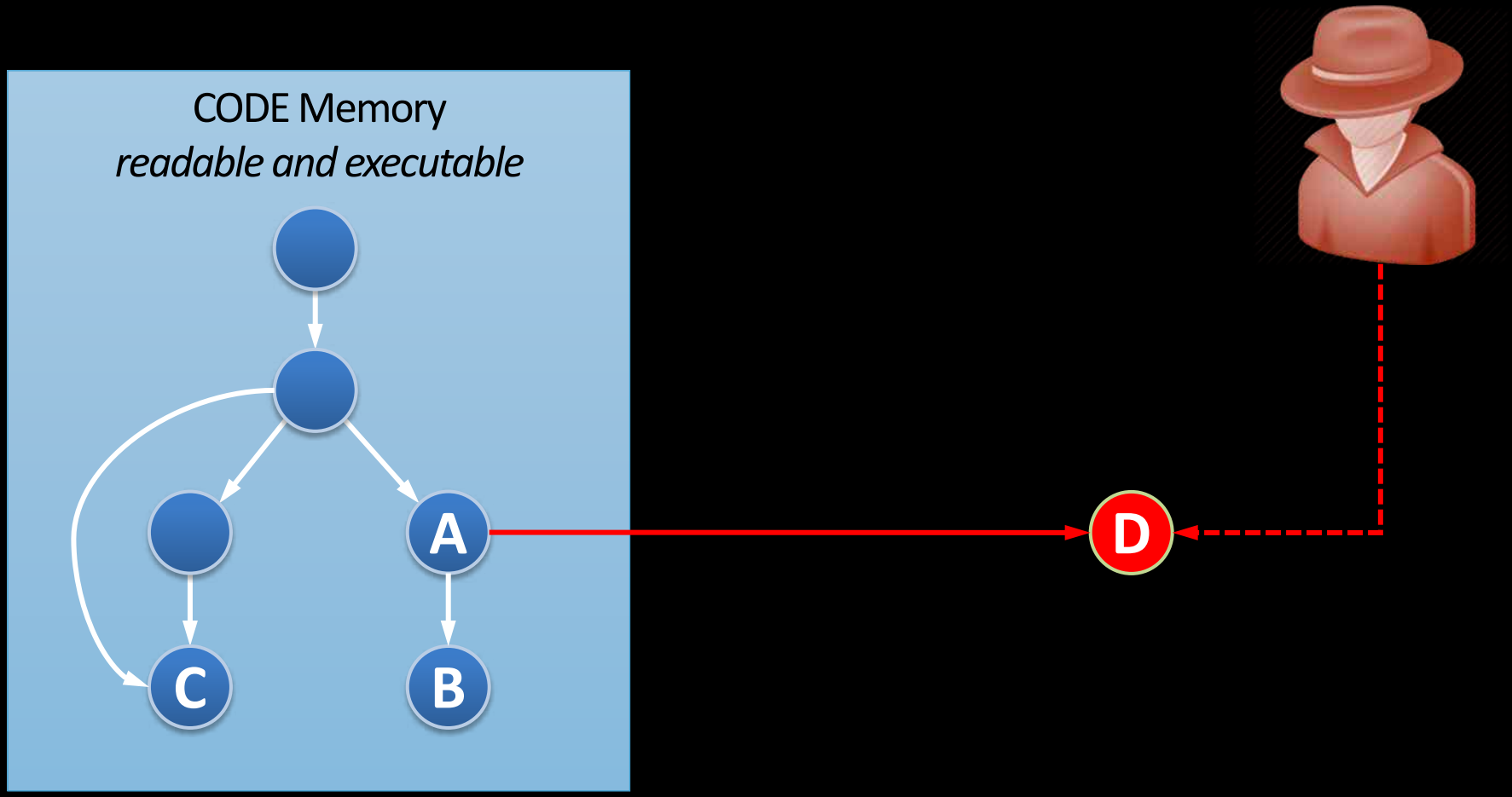
Data Execution Prevention (DEP)

- Prevent execution from a writeable memory (data) area



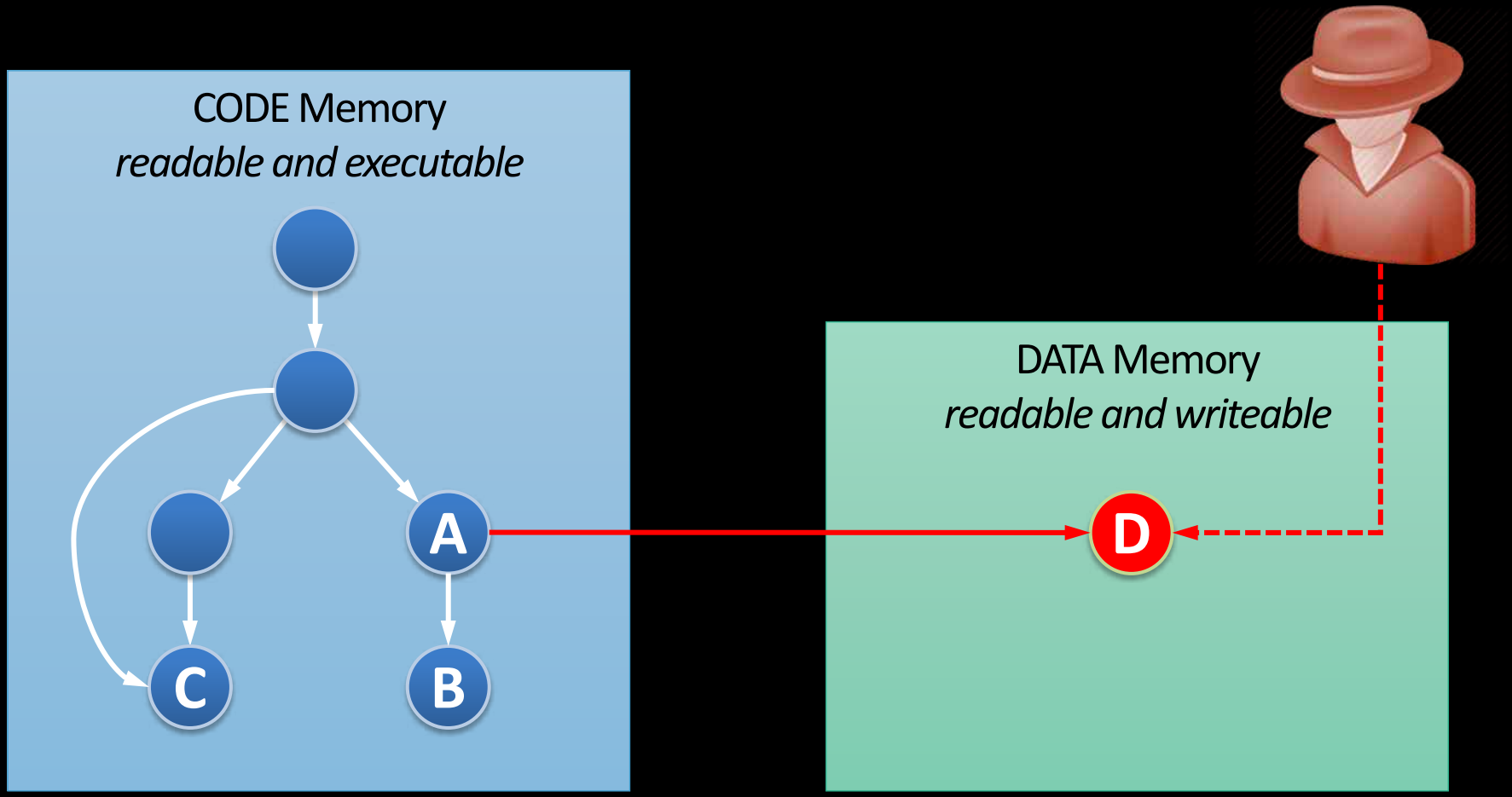
Data Execution Prevention (DEP)

- Prevent execution from a writeable memory (data) area



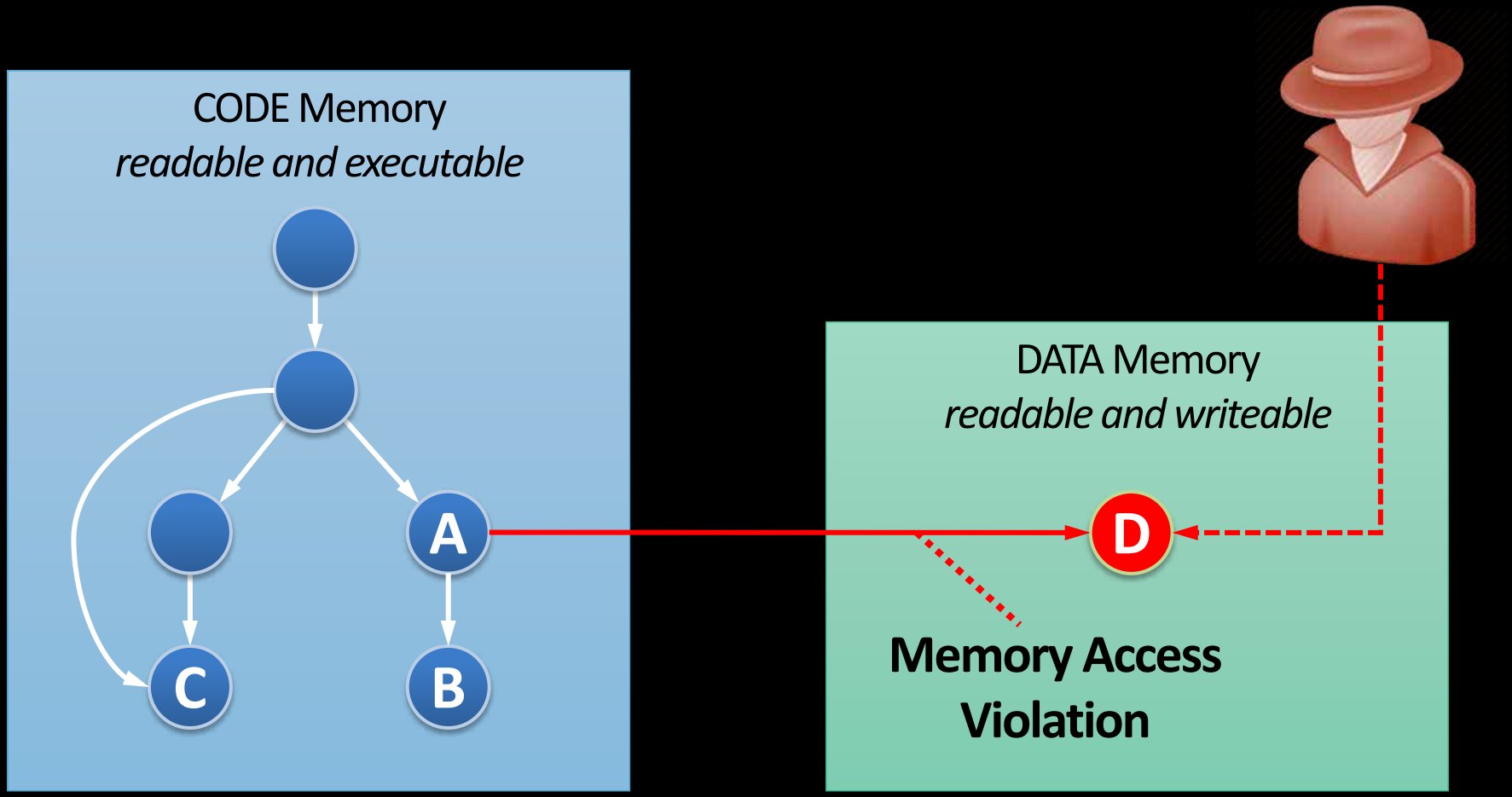
Data Execution Prevention (DEP)

- Prevent execution from a writeable memory (data) area



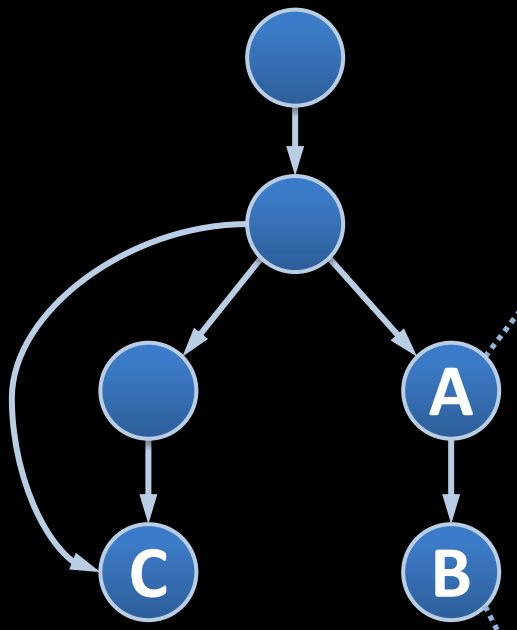
Data Execution Prevention (DEP)

- Prevent execution from a writeable memory (data) area



General Principle of Code-Reuse Attacks

Control-Flow Graph (CFG)



Basic Block (BBL) A

ENTRY
asm_ins,
EXIT

BBL B

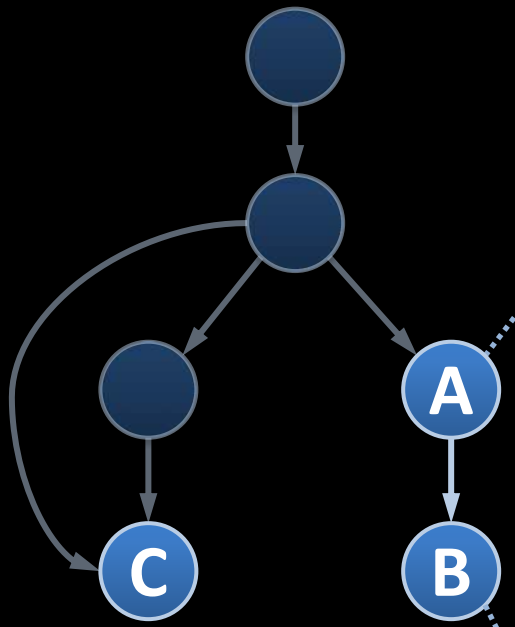
ENTRY
asm_ins, …
EXIT

Data flows

Program flows

General Principle of Code-Reuse Attacks

Control-Flow Graph (CFG)



Basic Block (BBL) A

ENTRY
asm_ins,
EXIT

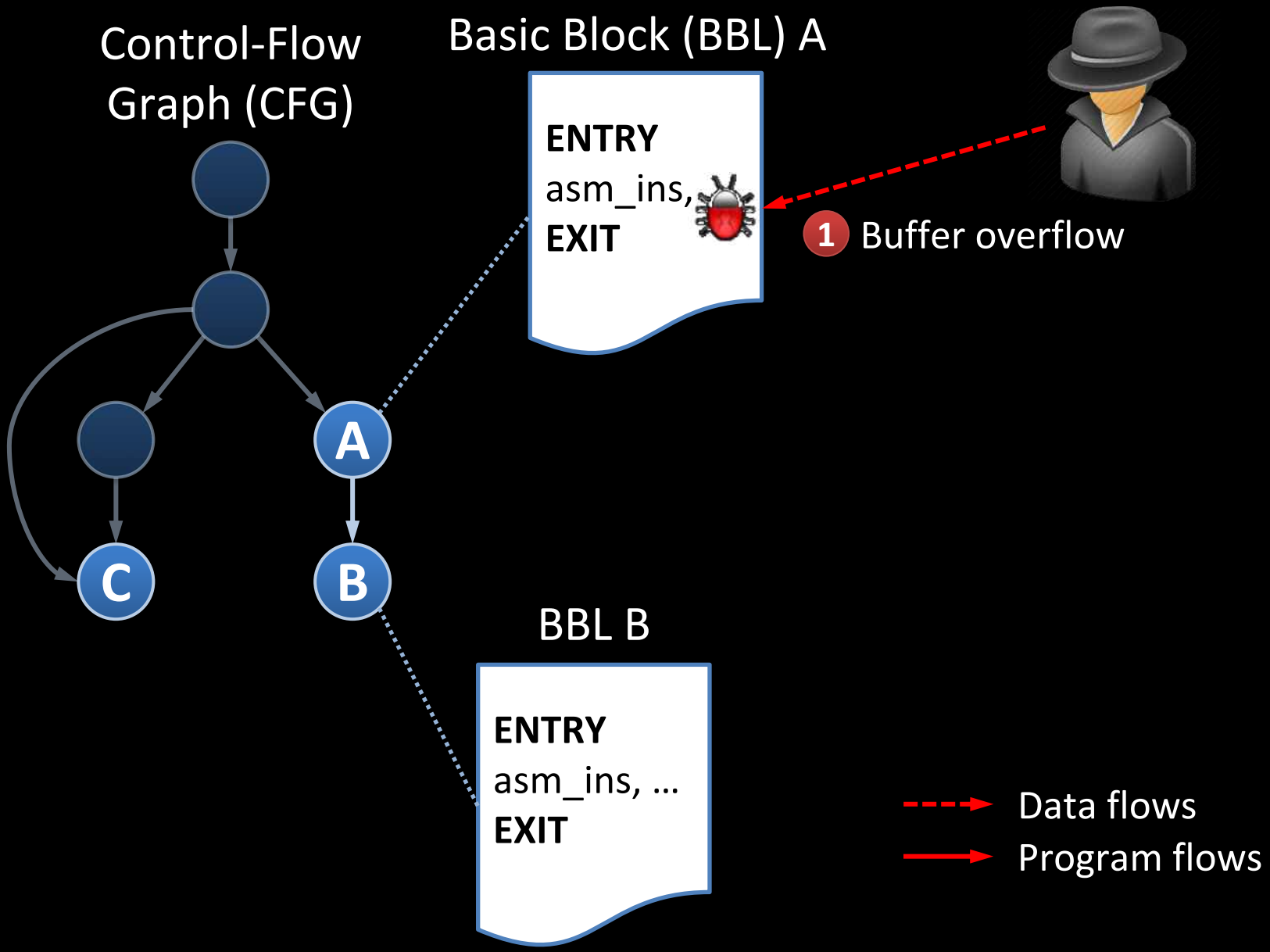
BBL B

ENTRY
asm_ins, …
EXIT

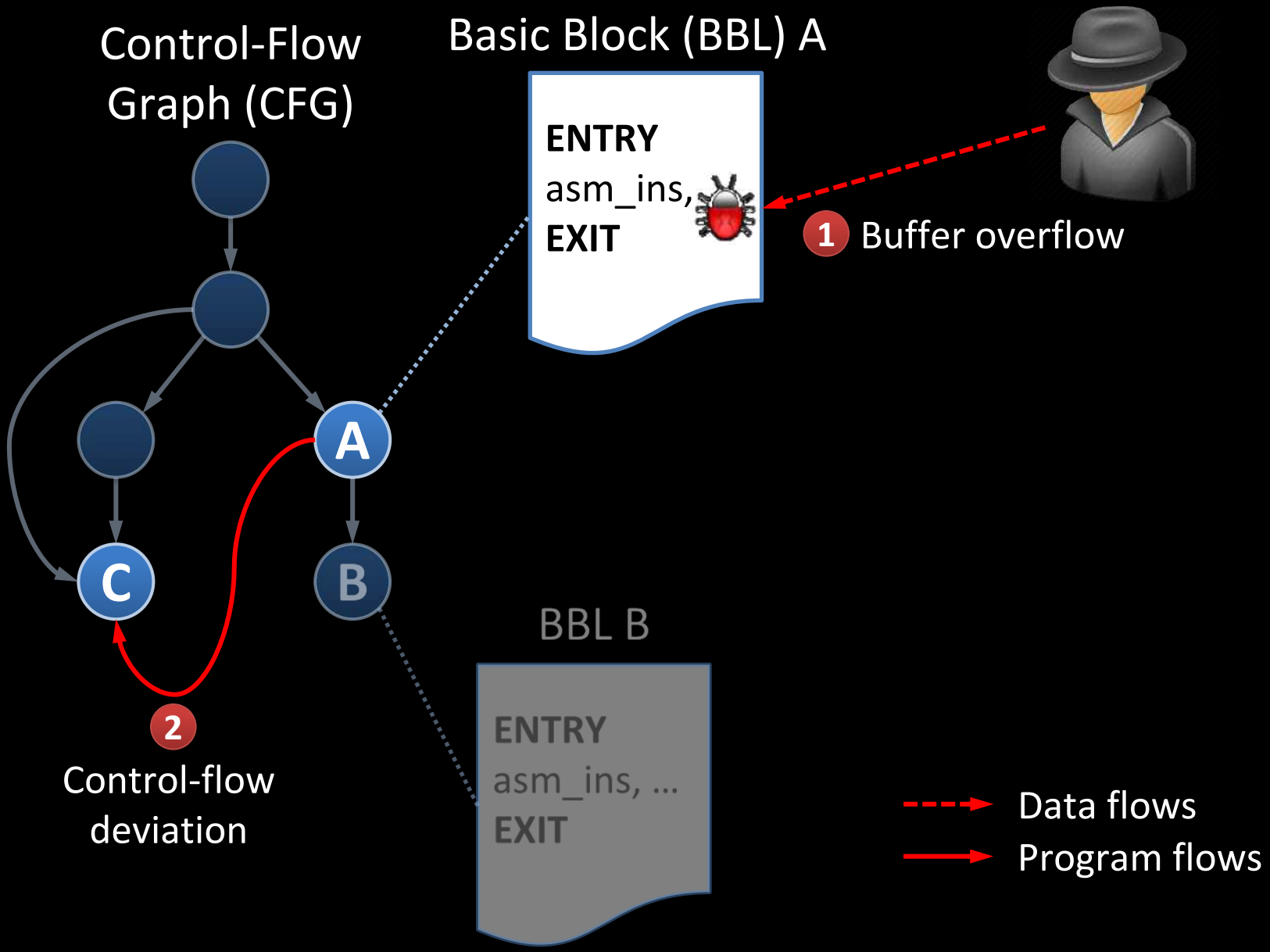
Data flows

Program flows

General Principle of Code-Reuse Attacks



General Principle of Code-Reuse Attacks



Generalization of return-into-libc attacks: return-oriented programming [Shacham, ACM CCS 2007]

Big Picture

The New Y

Saturday, January 6, 2007

Daily Blog Tips awarded th

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

Big Picture

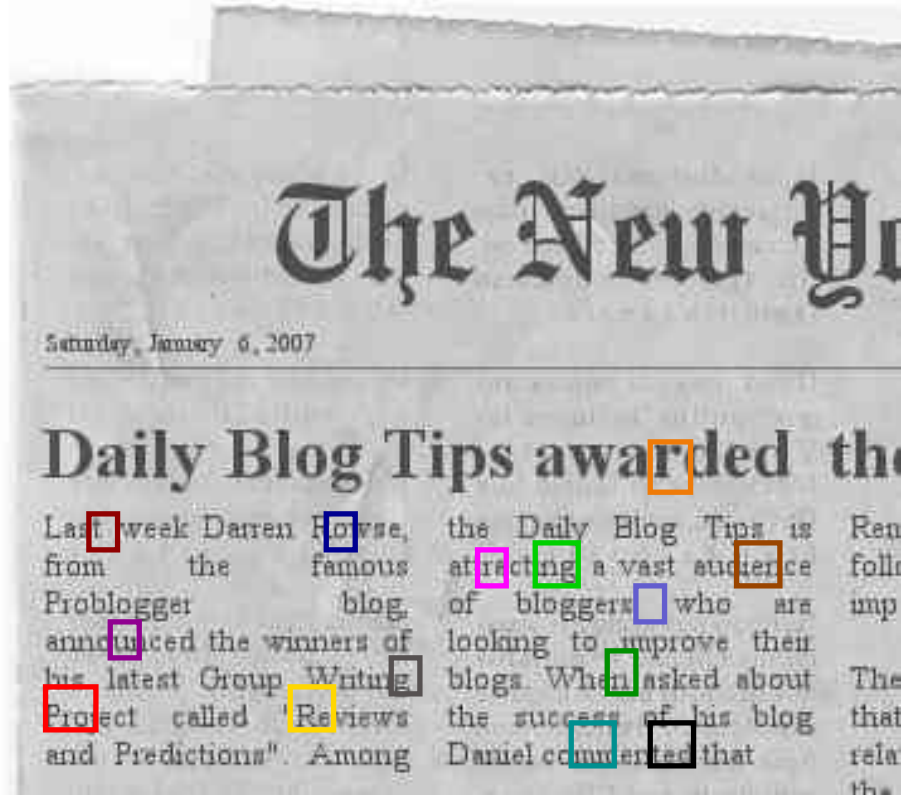
The New Y

Saturday, January 6, 2007

Daily Blog Tips awarded th

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

Big Picture



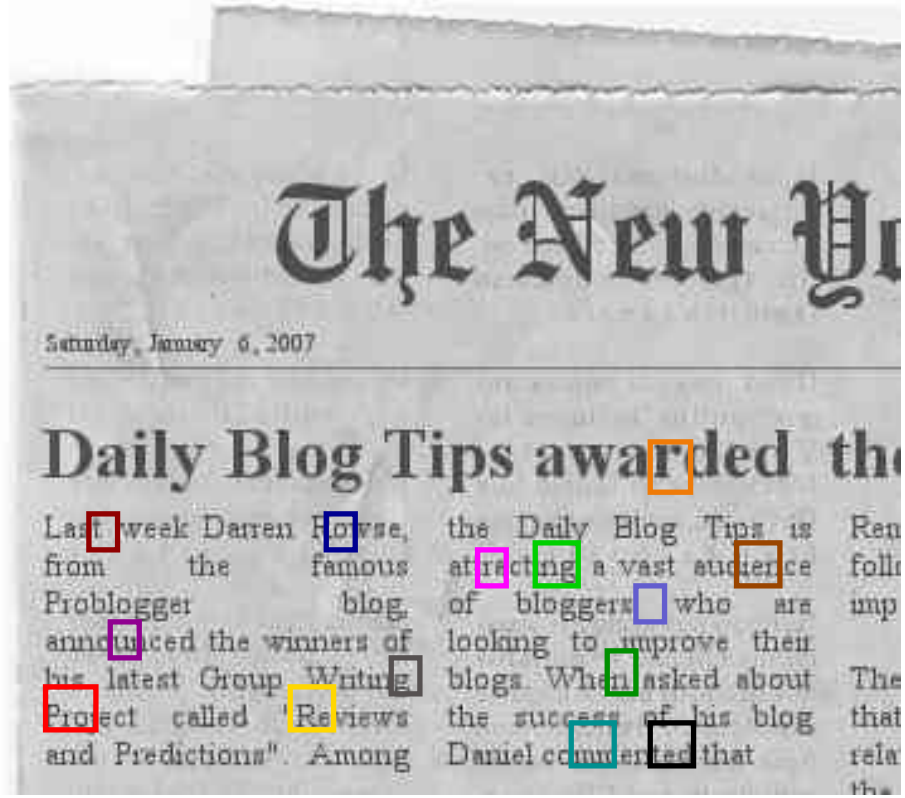
The New Y

Saturday, January 6, 2007

Daily Blog Tips awarded th

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

Big Picture



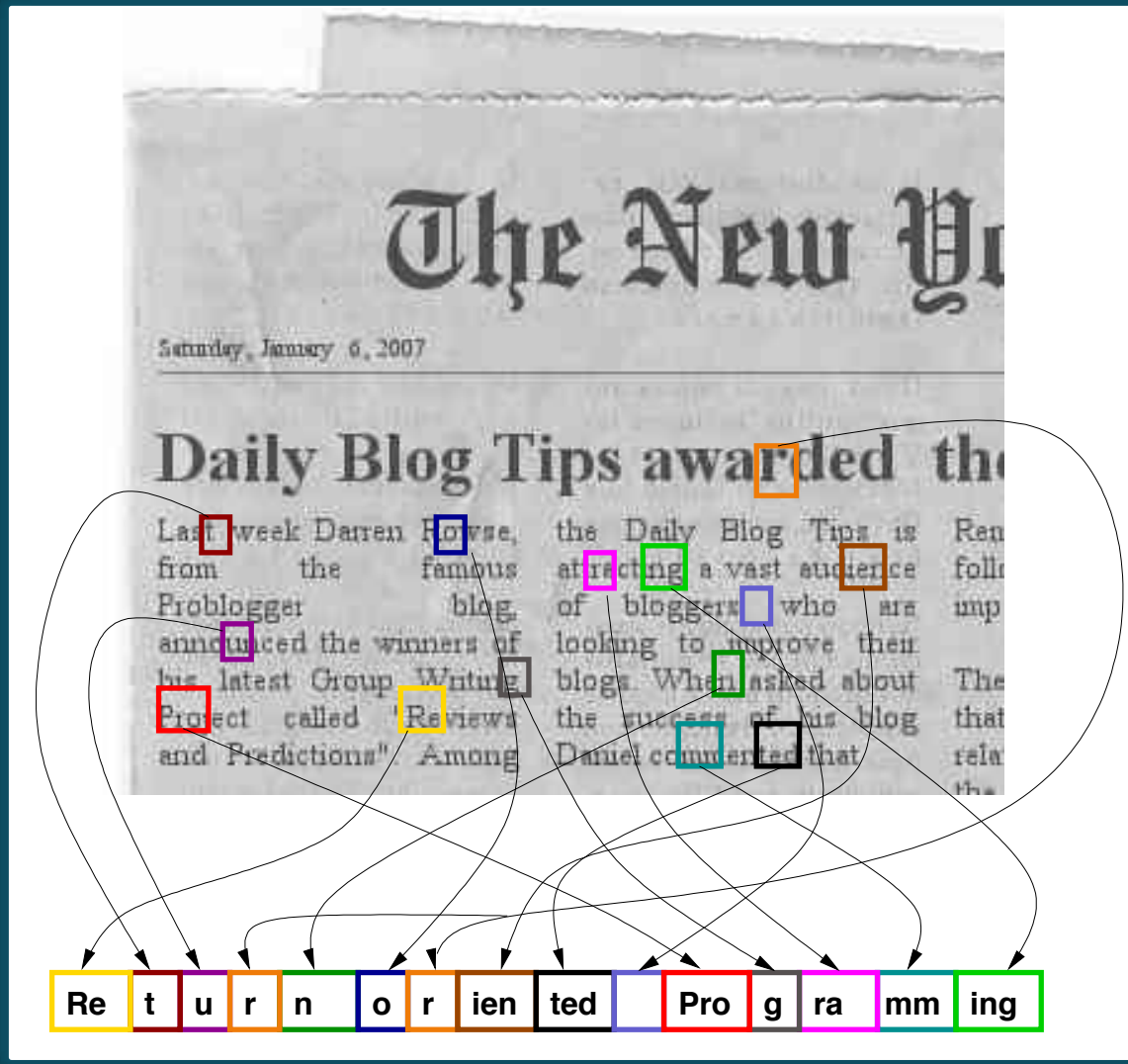
The New Y

Saturday, January 6, 2007

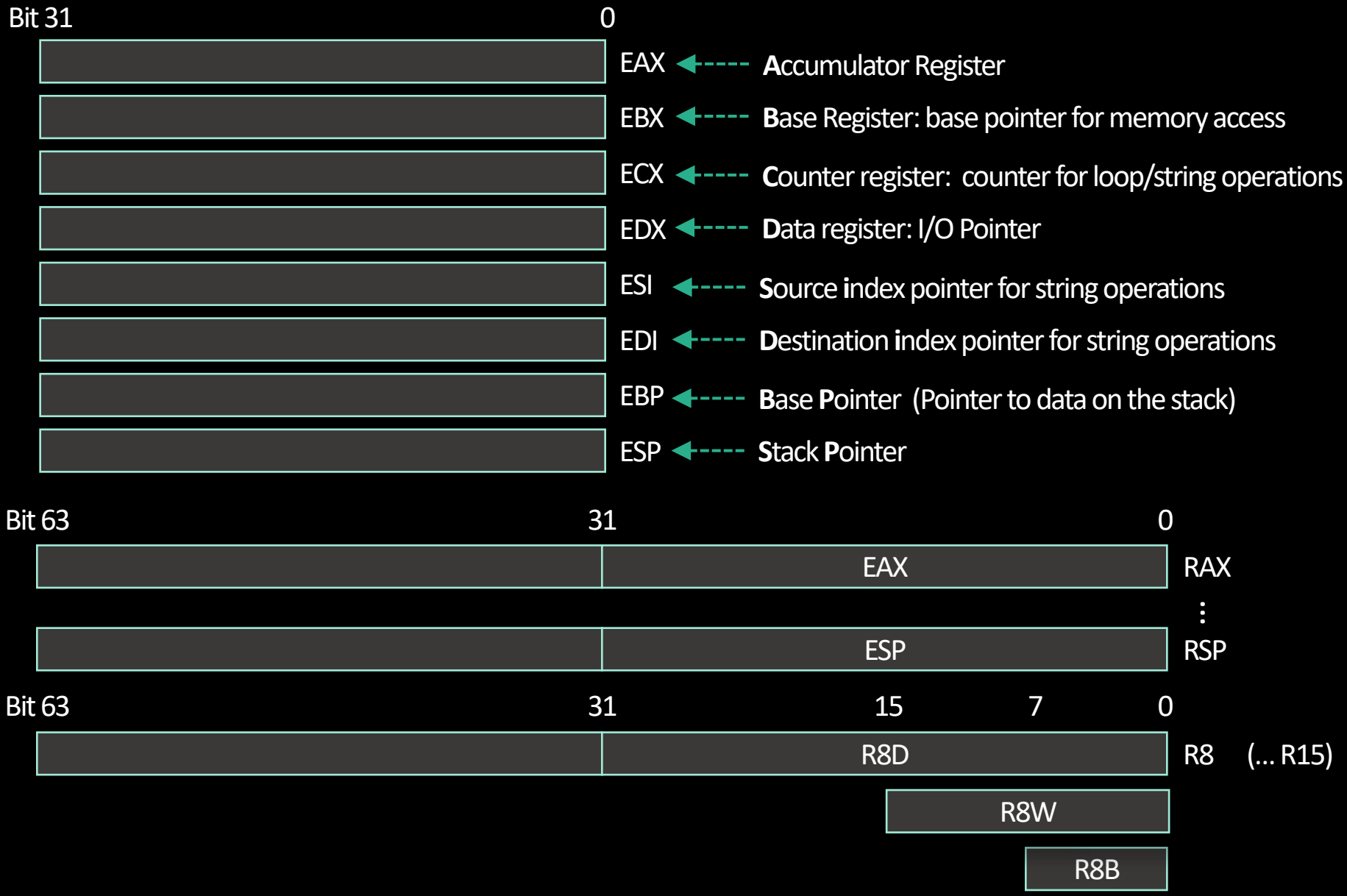
Daily Blog Tips awarded th

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

Big Picture


The New Yo
Saturday, January 6, 2007
Daily Blog Tips awarded the
Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among
the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that
Re
t
u
r
n
o
r
ien
ted
Pro
g
ra
mm
ing

x86 and x64 CPU registers



Attack Technique: Overview on x86

Attack Technique: Overview on x86

Program Stack

Attack Technique: Overview on x86

Program Code

Sequence 1

Sequence 2

Sequence 3

Program Stack

Attack Technique: Overview on x86

Program Code

Sequence 1

Sequence 2

Sequence 3

Program Stack

Attack Technique: Overview on x86

Program Stack

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Attack Technique: Overview on x86

Program Stack

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

EAX:

EBX:

Attack Technique: Overview on x86

Program Stack

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

EAX:

EBX:

Attack Technique: Overview on x86

Program Stack

Return Address 3

0xAABBCCDD

0X80102030

Return Address 2

Return Address 1

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

EAX:

EBX:

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

```
Return Address 3
0xAABBCCDD
0X80102030
Return Address 2
Return Address 1
```

ESP →

EAX:

EBX:

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

```
Return Address 3
0xAABBCCDD
0X80102030
Return Address 2
Return Address 1
```

ESP →

EAX:

EBX:

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

```
Return Address 3
0xAABBCCDD
0X80102030
Return Address 2
Return Address 1
```

ESP →

EAX:

EBX:

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

```
Return Address 3
0xAABBCCDD
0X80102030
Return Address 2
Return Address 1
```

ESP

EAX:

EBX:

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

```
Return Address 3
0xAABBCCDD
0X80102030
Return Address 2
Return Address 1
```

ESP

EAX:

EBX:

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

ESP → Return Address 3

0xAABBCCDD

0X80102030

Return Address 2

Return Address 1

EAX:	0x80102030
EBX:	0xAABBCCDD

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

ESP → Return Address 3

0xAABBCCDD

0X80102030

Return Address 2

Return Address 1

EAX:	0x80102030
EBX:	0xAABBCCDD

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

Program Stack

ESP

```
Return Address 3
0xAABBCCDD
0X80102030
Return Address 2
Return Address 1
```

EAX:	0x80102030
EBX:	0xAABBCCDD

Attack Technique: Overview on x86

Program Code

Sequence 1

```
x86_ins
ret
```

Sequence 2

```
pop eax
pop ebx
ret
```

Sequence 3

```
x86_ins
ret
```

...

ESP

Program Stack

```
Return Address 3
0xAABBCCDD
0X80102030
Return Address 2
Return Address 1
```

EAX:	0x80102030
EBX:	0xAABBCCDD

Memory Load Gadget on x86

Memory Load Gadget on x86

Program Stack

Memory Load Gadget on x86

Program Stack

Data Area

0x80102030: 0xDEADBEEF

Memory Load Gadget on x86

Program Stack

Data Area

0x80102030: 0xDEADBEEF

Memory Load Gadget on x86

Program Stack

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Data Area

0x80102030: 0xDEADBEEF

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

Data Area

0x80102030: 0xDEADBEEF

EAX:

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

Data Area

0x80102030: 0xDEADBEEF

EAX:

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

…

Return Address 2

0x80102010

ESP → Return Address 1

Data Area

0x80102030: 0xDEADBEEF

EAX:

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

```
...
Return Address 2
0x80102010
Return Address 1
```

ESP

Data Area

0x80102030: `0xDEADBEEF`

EAX:

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

…

Return Address 2

ESP → 0x80102010

Return Address 1

Data Area

0x80102030: 0xDEADBEEF

EAX:

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

```
...
Return Address 2
0x80102010
Return Address 1
```

ESP

Data Area

0x80102030: 0xDEADBEEF

EAX: 0x80102010

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

ESP →

…
Return Address 2
0x80102010
Return Address 1

Data Area

0x80102030:

0xDEADBEEF

EAX: 0x80102010

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

ESP →

…
Return Address 2
0x80102010
Return Address 1

Data Area

0x80102030:

0xDEADBEEF

EAX: 0x80102010

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

ESP →

```
...
Return Address 2
0x80102010
Return Address 1
```

Data Area

0x80102030:

```
0xDEADBEEF
```

EAX: 0x80102010

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

ESP →

```
...
Return Address 2
0x80102010
Return Address 1
```

Data Area

0x80102030:

```
0xDEADBEEF
```

EAX: `0xDEADBEEF`

Memory Load Gadget on x86

Program Code

Sequence 2

```
mov eax,[eax+0x20]
ret
```

Sequence 1

```
pop eax
ret
```

Program Stack

ESP →

```
...
Return Address 2
0x80102010
Return Address 1
```

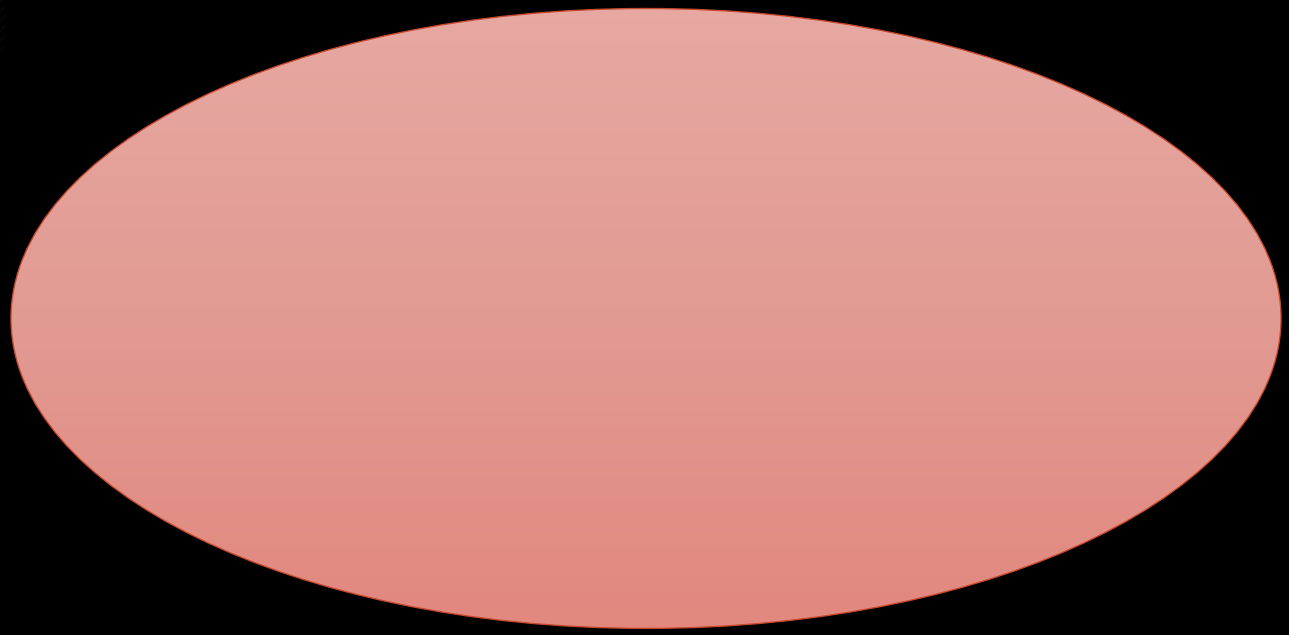
Data Area

0x80102030:

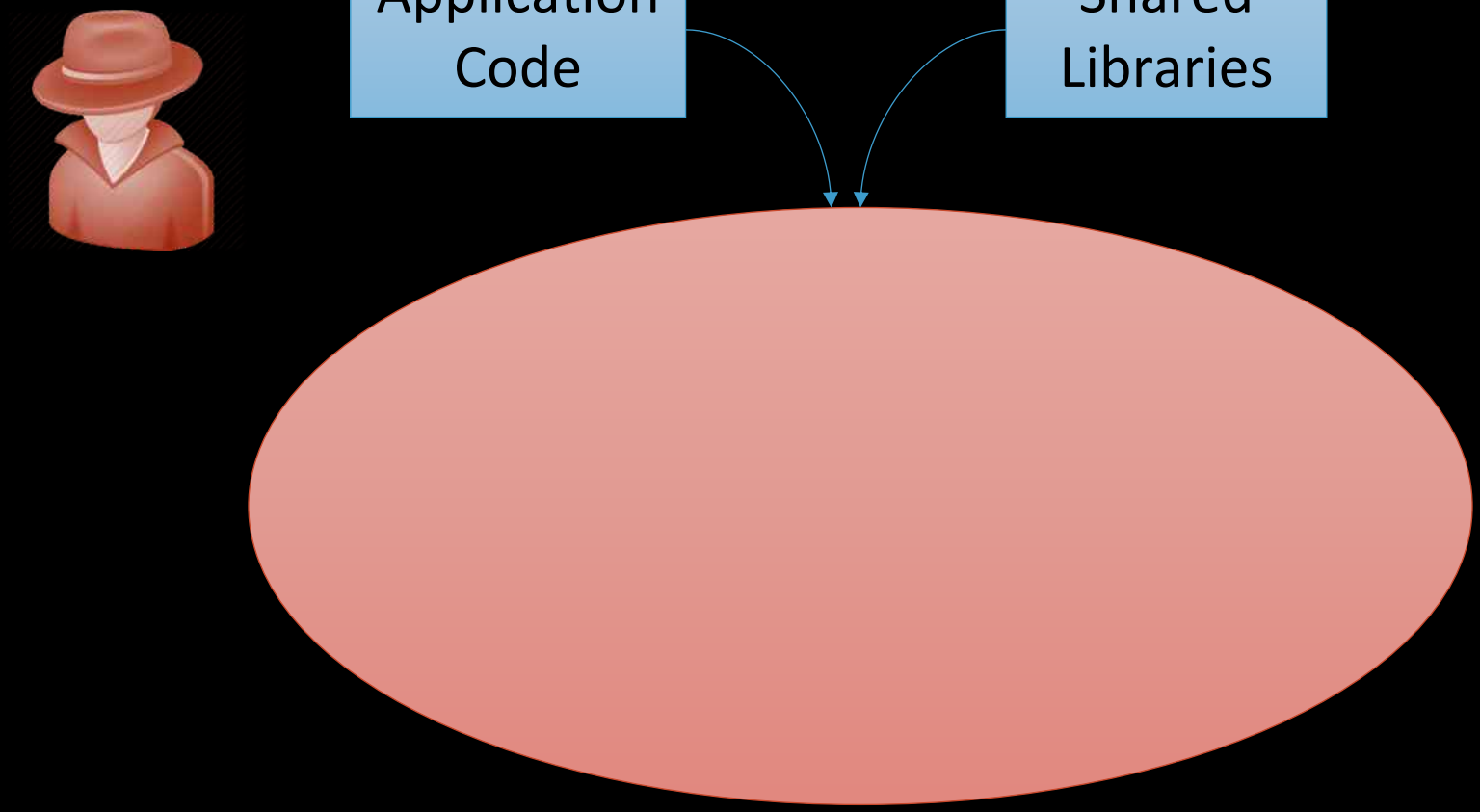
```
0xDEADBEEF
```

EAX: `0xDEADBEEF`

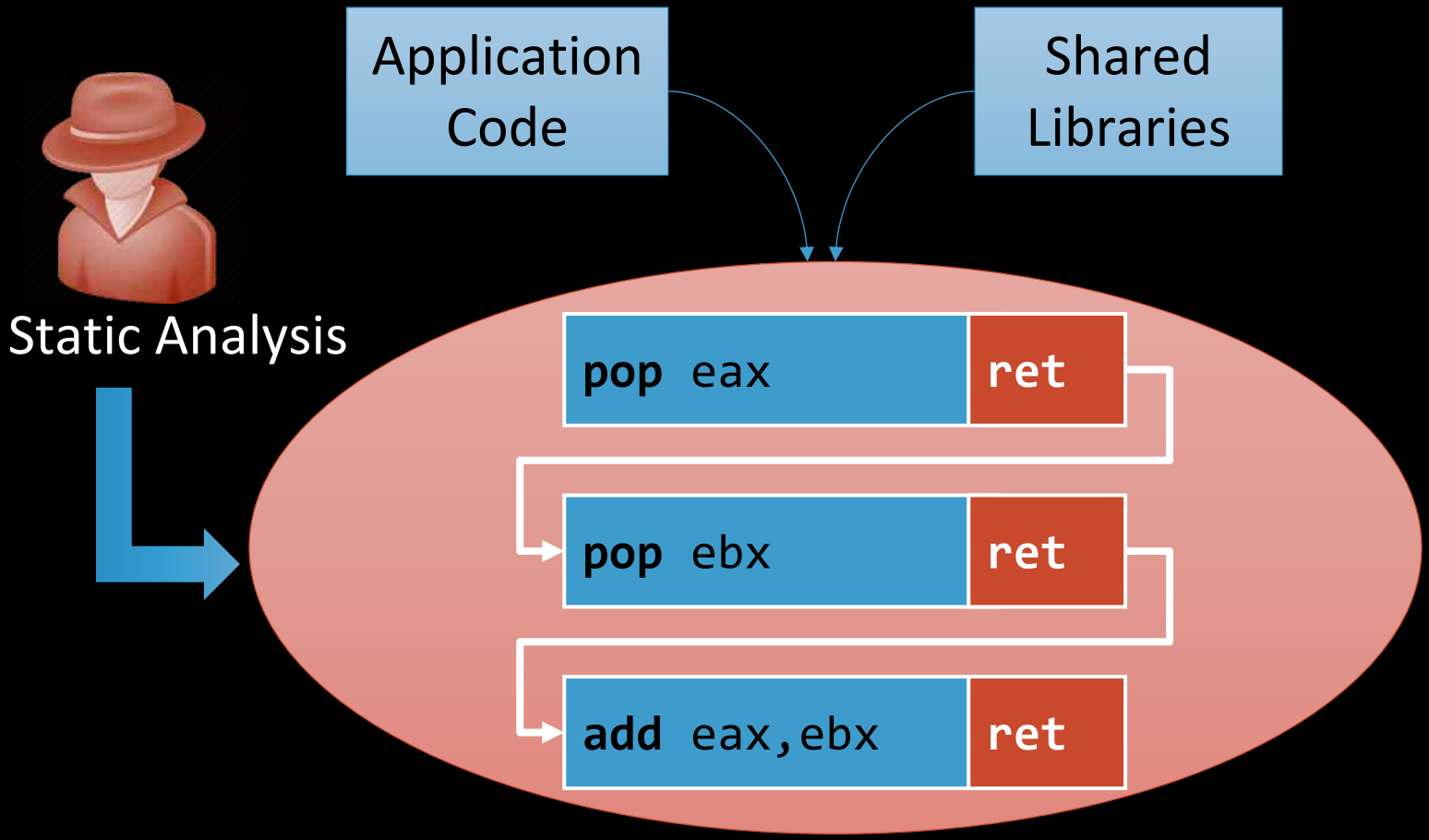
Code Base and Turing-Completeness



Code Base and Turing-Completeness



Code Base and Turing-Completeness



Code Base and Turing-Completeness

Static Analysis

Application Code

Shared Libraries

Cond. JMP

STORE

LOAD

GADGET SPACE

Turing-complete language

Mandatory

Code Base and Turing-Completeness

Static Analysis

Application Code

Shared Libraries

MOV

CALL

Uncond. JMP

Logic.

Cond. JMP

STORE

LOAD

Arith.

GADGET SPACE

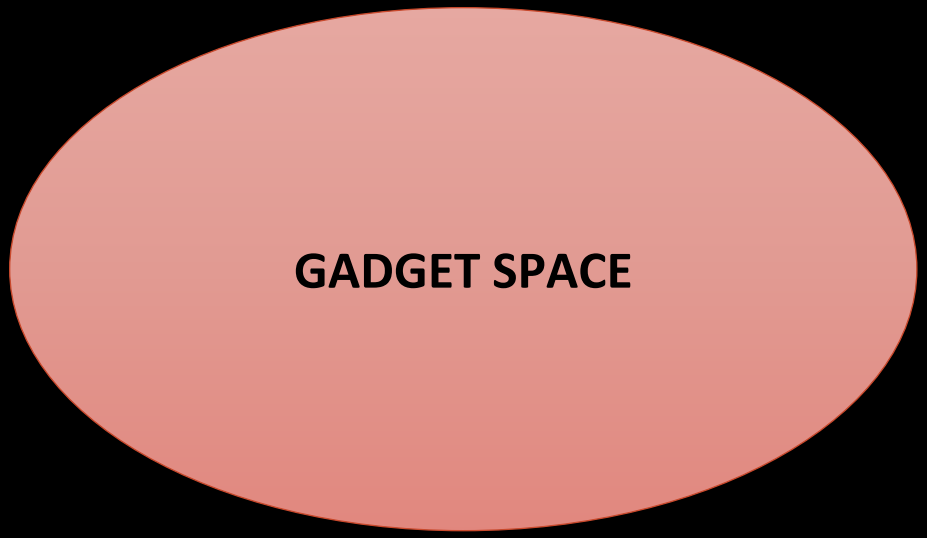
Optional

Mandatory

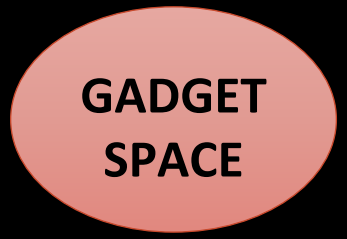
Turing-complete language

Gadget Space on Different Architectures

Architectures with no memory alignment, e.g., Intel x86

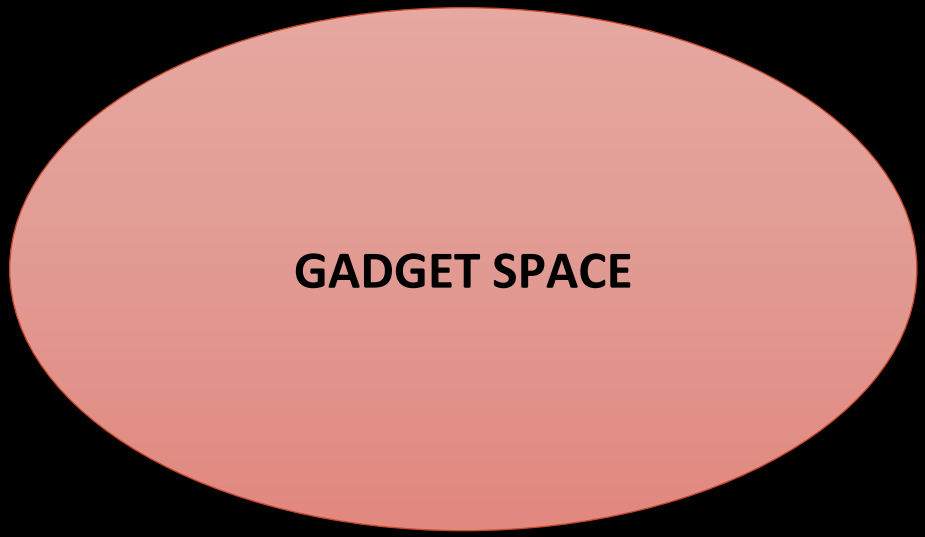


Architectures with memory alignment, e.g., SPARC, ARM

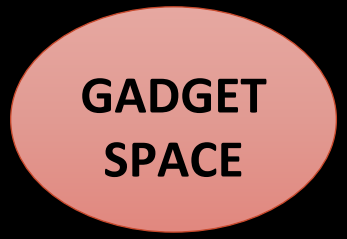


Gadget Space on Different Architectures

Architectures with no memory alignment, e.g., Intel x86

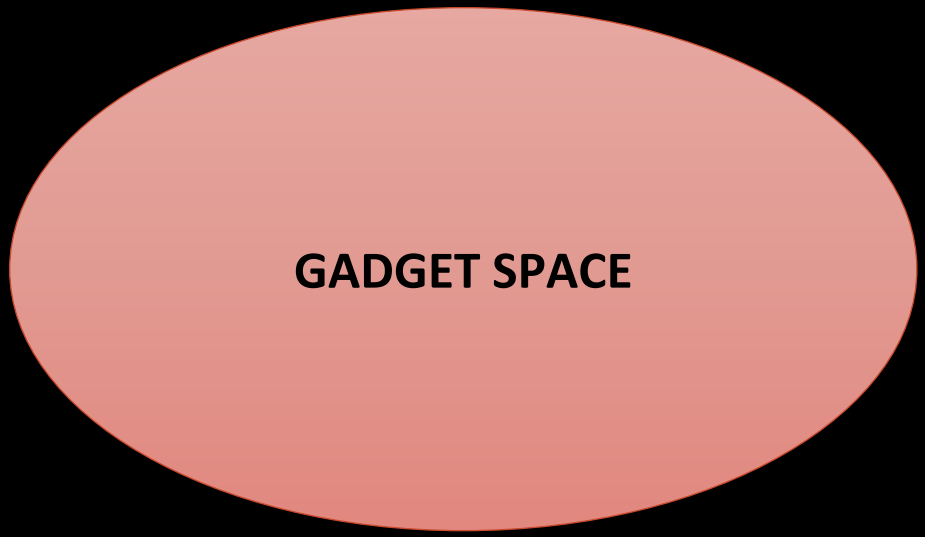


Architectures with memory alignment, e.g., SPARC, ARM

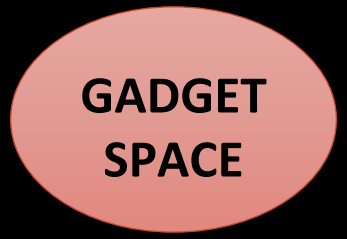


Gadget Space on Different Architectures

Architectures with no memory alignment, e.g., Intel x86



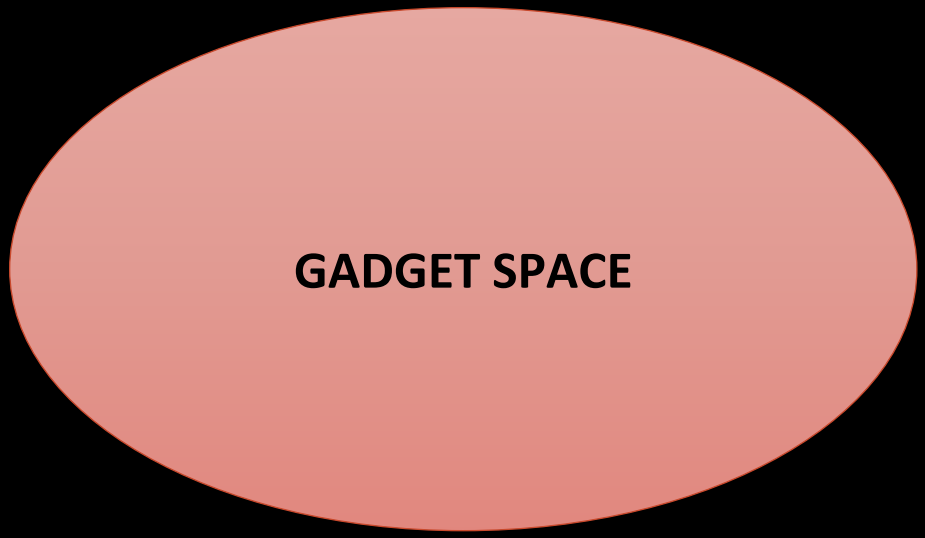
Architectures with memory alignment, e.g., SPARC, ARM

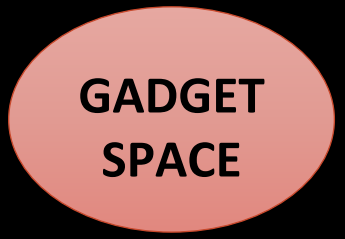


Gadget Space on Different Architectures

Architectures with no memory alignment, e.g., Intel x86

Architectures with memory alignment, e.g., SPARC, ARM





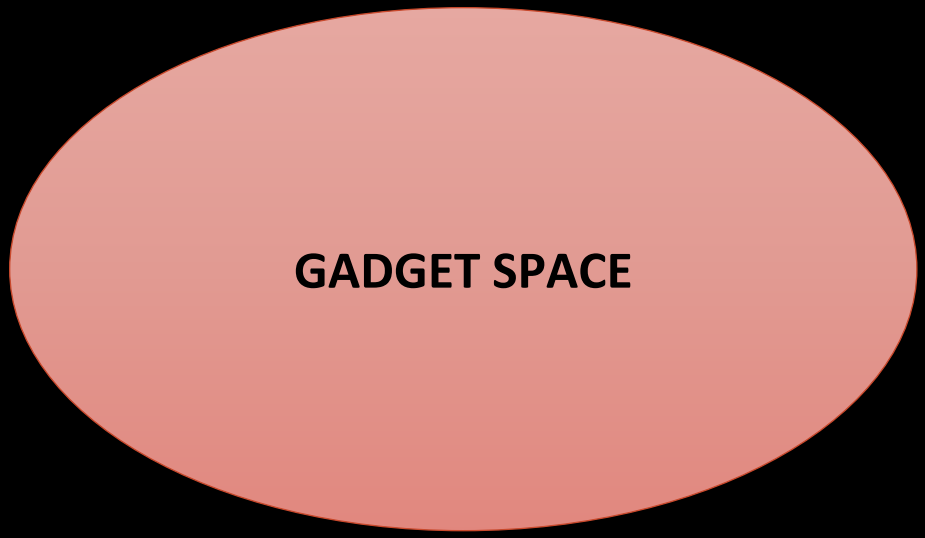
Intended Code

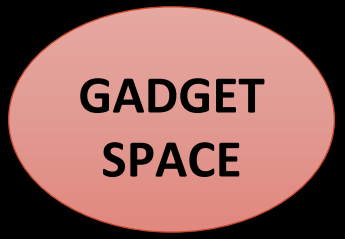
```
mov eax,0x13
jmp 3aae9
```

Gadget Space on Different Architectures

Architectures with no memory alignment, e.g., Intel x86

Architectures with memory alignment, e.g., SPARC, ARM





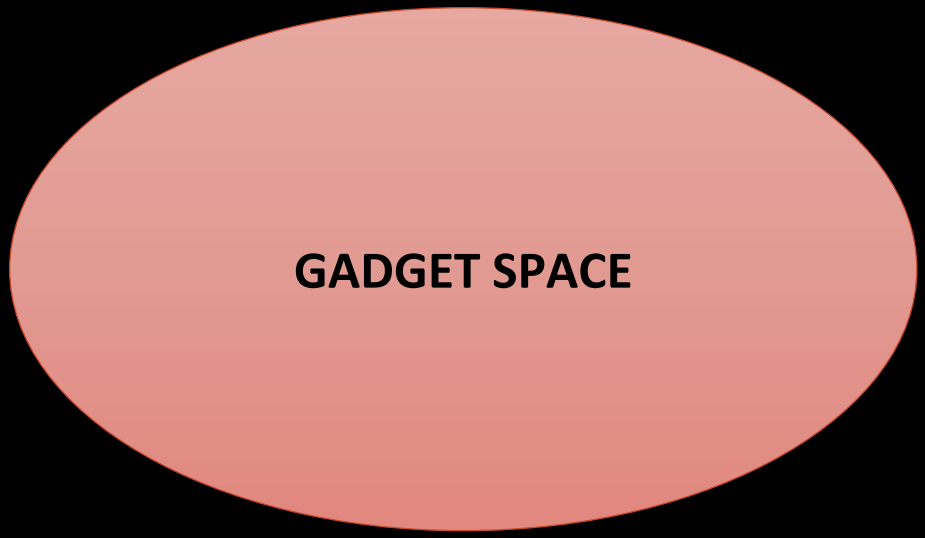
Intended Code

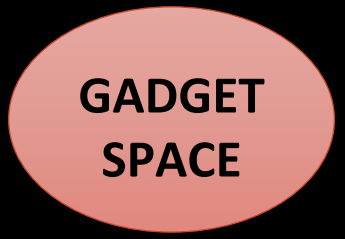
```
mov eax,0x13
jmp 3aae9
```

Gadget Space on Different Architectures

Architectures with no memory alignment, e.g., Intel x86

Architectures with memory alignment, e.g., SPARC, ARM





Intended Code

```
mov eax,0x13
jmp 3aae9
```

Unintended Code

```
add eax,[al]
add cl,ch
ret
```

Research on Code-Reuse Attacks

SELECTED

1997 — ret2libc *Solar Designer*

2001 — Advanced ret2libc *Nergal*

2005 — Borrowed Code Chunk Exploitation *Krahmer*

2007 — ROP on x86 *Shacham (CCS)*

2008 — ROP on SPARC *Buchanan et al (CCS)*; ROP on Atmel AVR *Francillon et al (CCS)*

2009 — ROP Rootkits *Hund et al (USENIX)*; ROP on PowerPC *FX Lindner (BlackHat)*; ROP on ARM/iOS *Miller et al (BlackHat)*

2010 — ROP without Returns *Checkoway et al (CCS)*; Practical ROP *Zovi (RSA Conference)*; Pwn2Own (iOS/IE) *Iozzo et al / Nils*

2011/2012 — Real-World Exploits





2013 — JIT-ROP *Snow et al (IEEE S&P)*

2014 — Blind ROP *Bittau et al (IEEE S&P)*; Out-Of-Control *Göktas et al (IEEE S&P)*; Stitching Gadgets *Davi et al (USENIX)*; Flushing Attacks *Schuster et al (RAID)*; ROP is Dangerous *Carlini et al (USENIX)*

2015 — COOP *Schuster et al (IEEE S&P)*; Losing Control *Liebchen et al (CCS)*; Control-Flow Bending *Carlini et al (USENIX)*

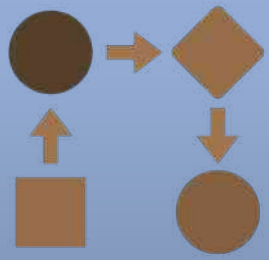
Overview

Background on Run-Time Attacks

Building Code Randomization Defenses

Code-Reuse Attacks

Building Control-Flow Integrity Defenses



Data-Oriented Exploits

Probabilistic vs Enforcement-Based Defense Approach

(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]

Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]

Probabilistic vs Enforcement-Based Defense Approach

(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]

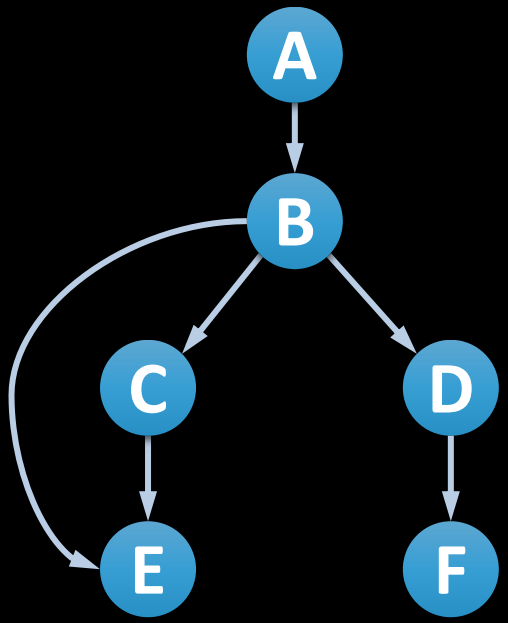
Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]

Probabilistic vs Enforcement-Based Defense Approach

(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



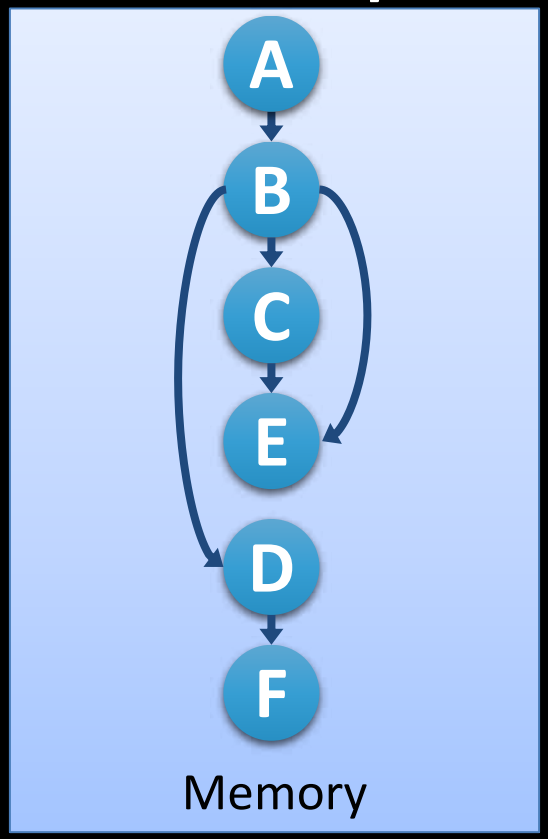
Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]

Probabilistic vs Enforcement-Based Defense Approach

(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



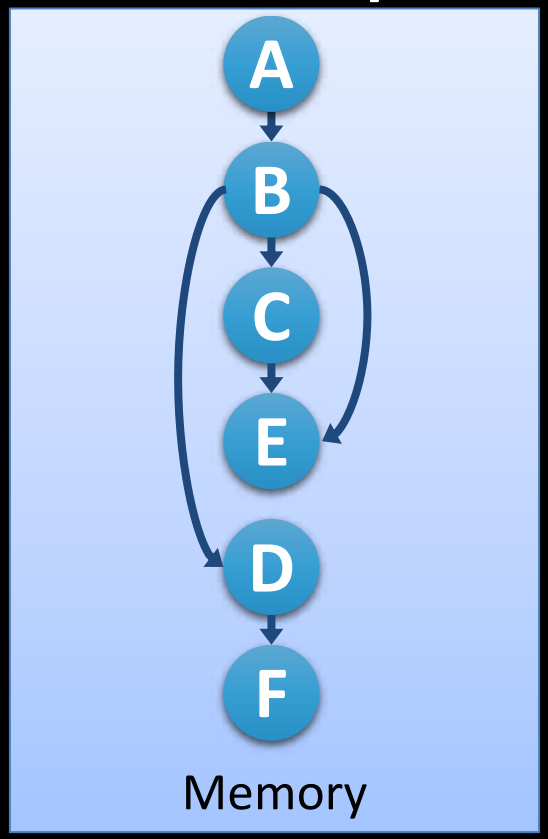
Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]

Probabilistic vs Enforcement-Based Defense Approach

(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



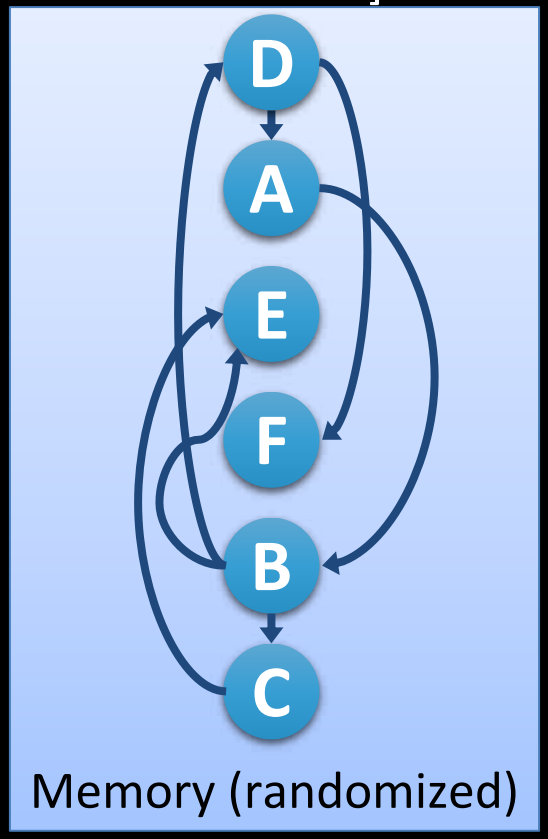
Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]

Probabilistic vs Enforcement-Based Defense Approach

(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



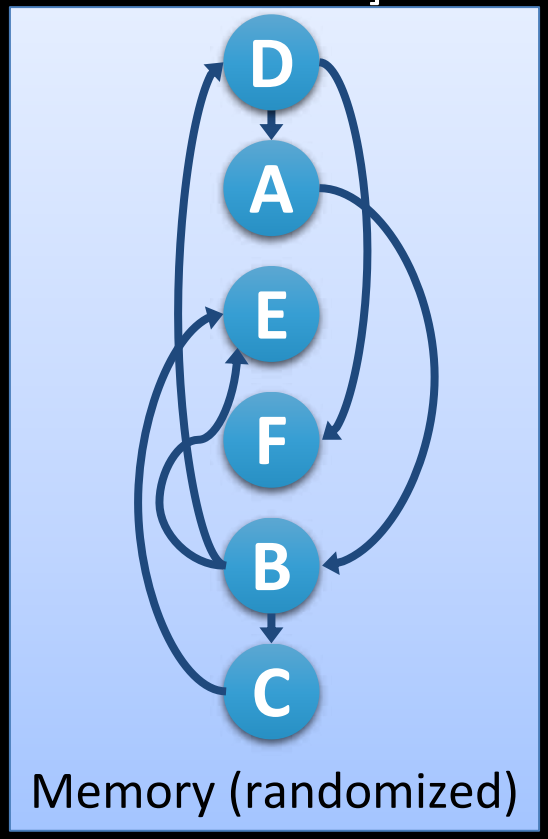
Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]

Probabilistic vs Enforcement-Based Defense Approach

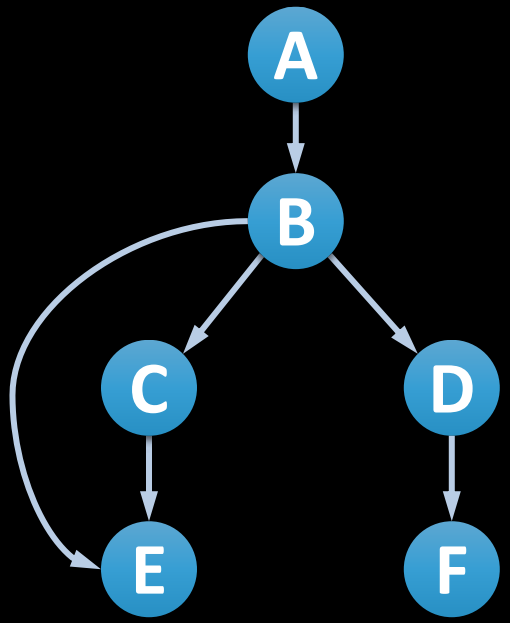
(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



Control-Flow Integrity (CFI)

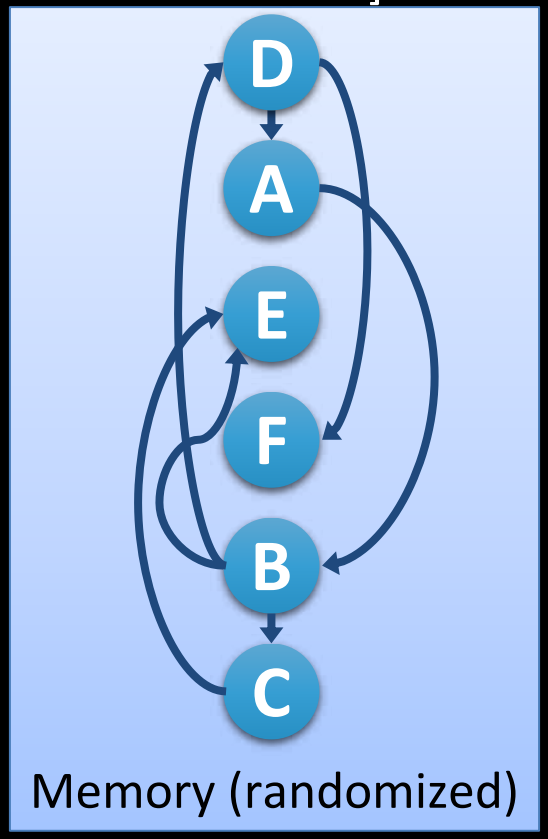
[Abadi et al., CCS 2005 & TISSEC 2009]



Probabilistic vs Enforcement-Based Defense Approach

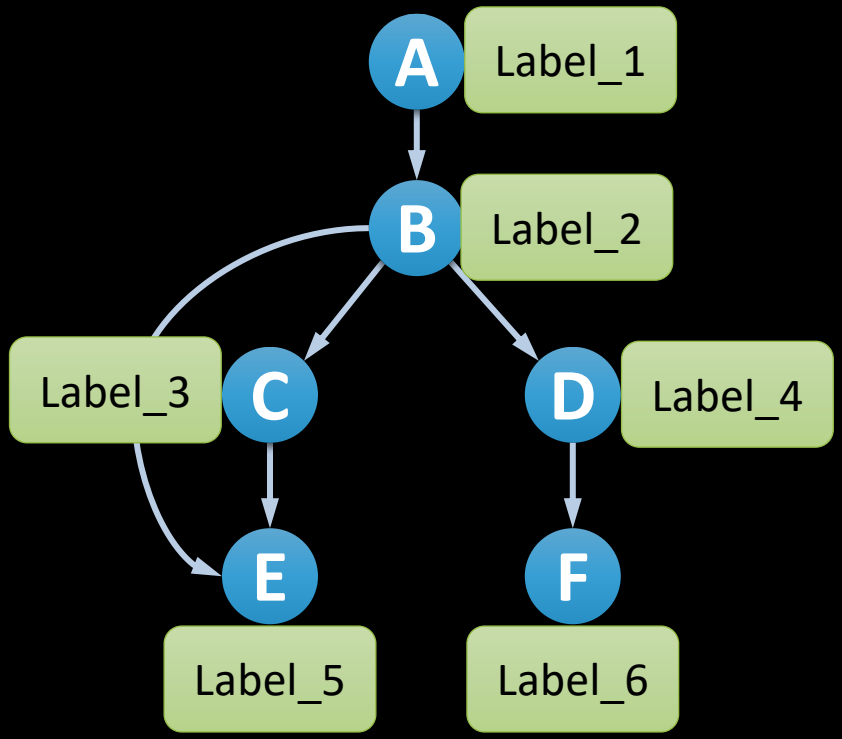
(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



Control-Flow Integrity (CFI)

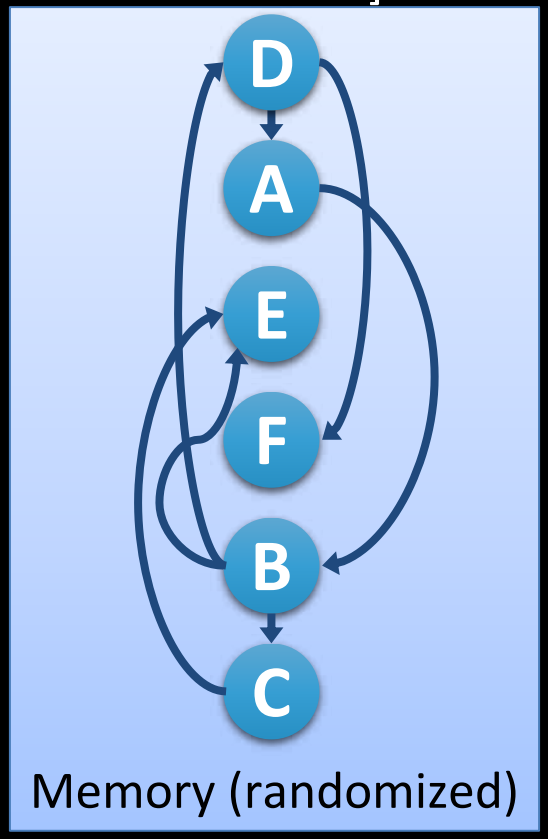
[Abadi et al., CCS 2005 & TISSEC 2009]



Probabilistic vs Enforcement-Based Defense Approach

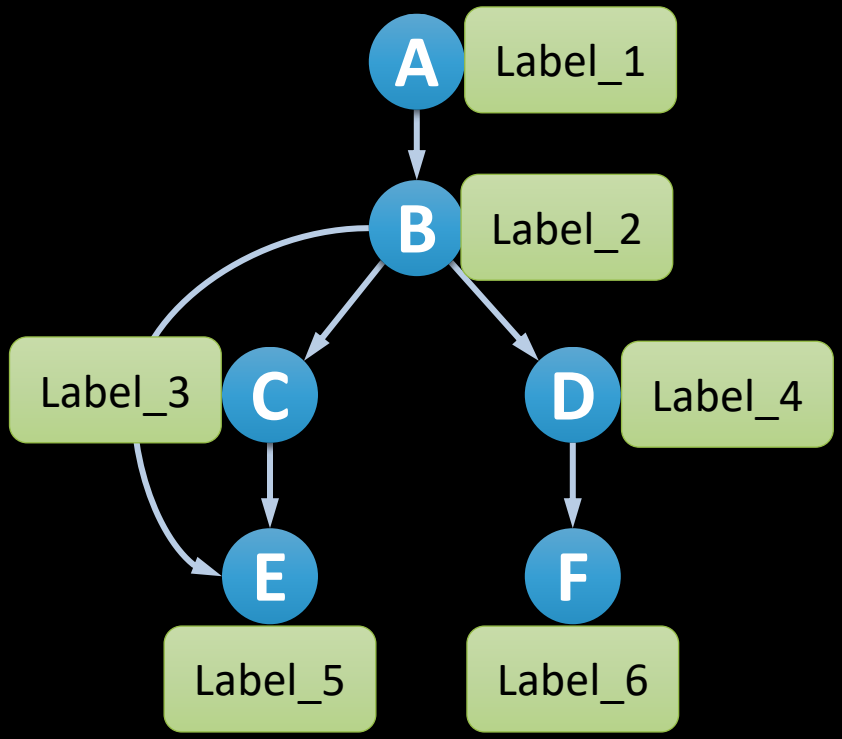
(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



Control-Flow Integrity (CFI)

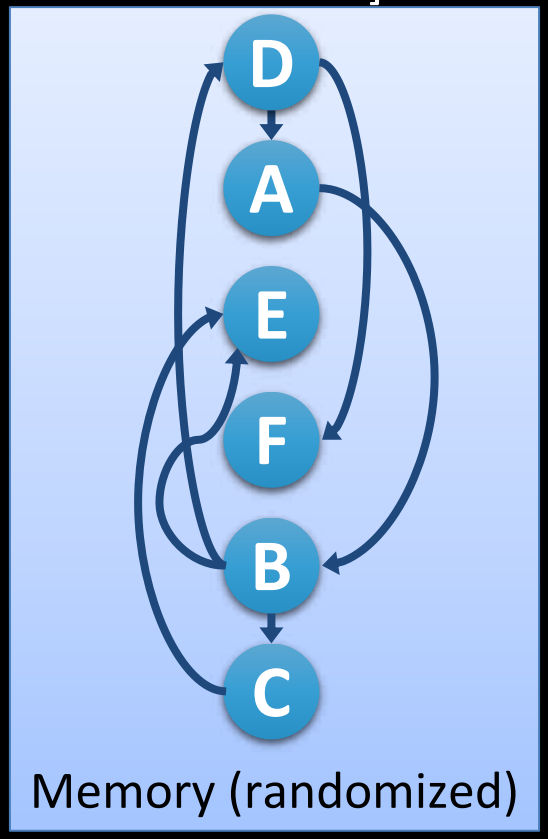
[Abadi et al., CCS 2005 & TISSEC 2009]



Probabilistic vs Enforcement-Based Defense Approach

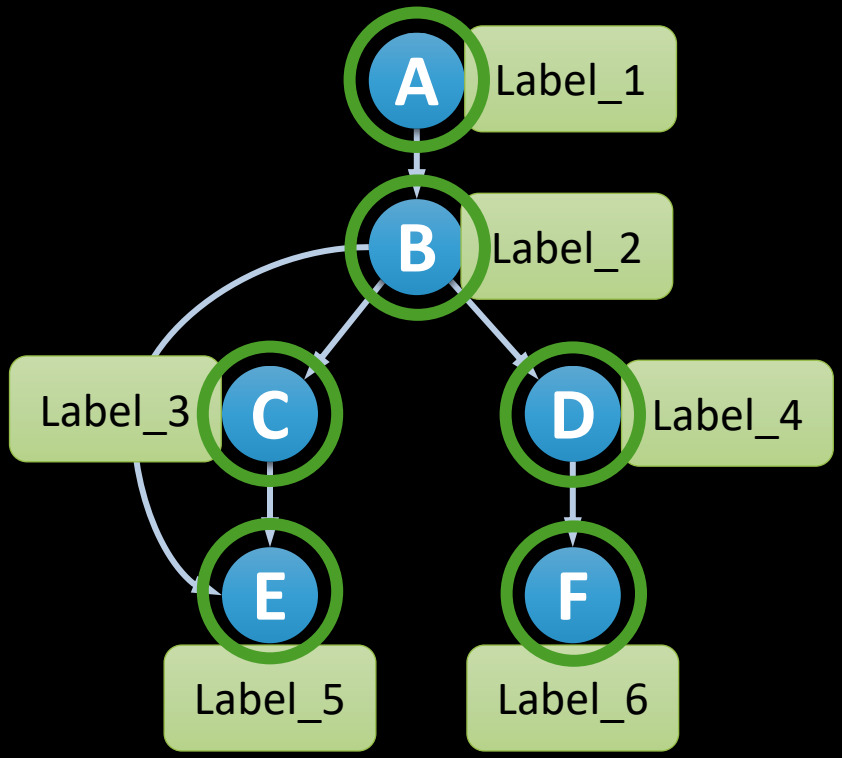
(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



Control-Flow Integrity (CFI)

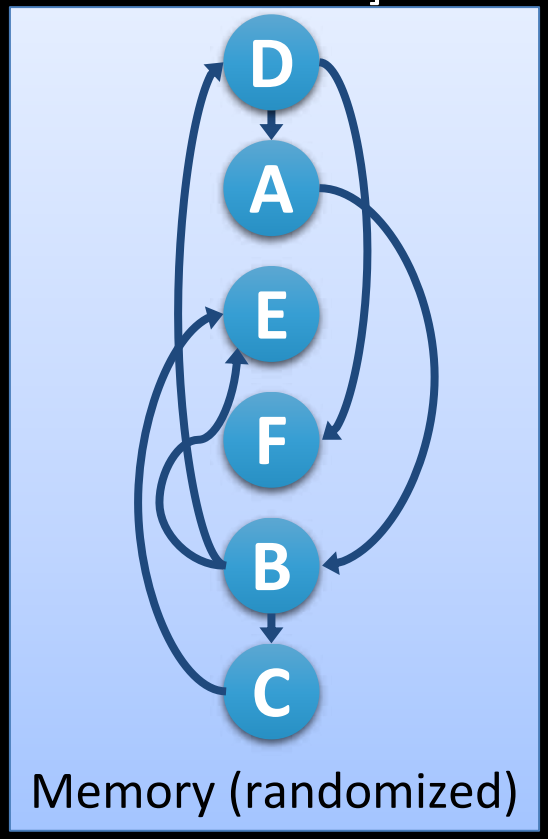
[Abadi et al., CCS 2005 & TISSEC 2009]



Probabilistic vs Enforcement-Based Defense Approach

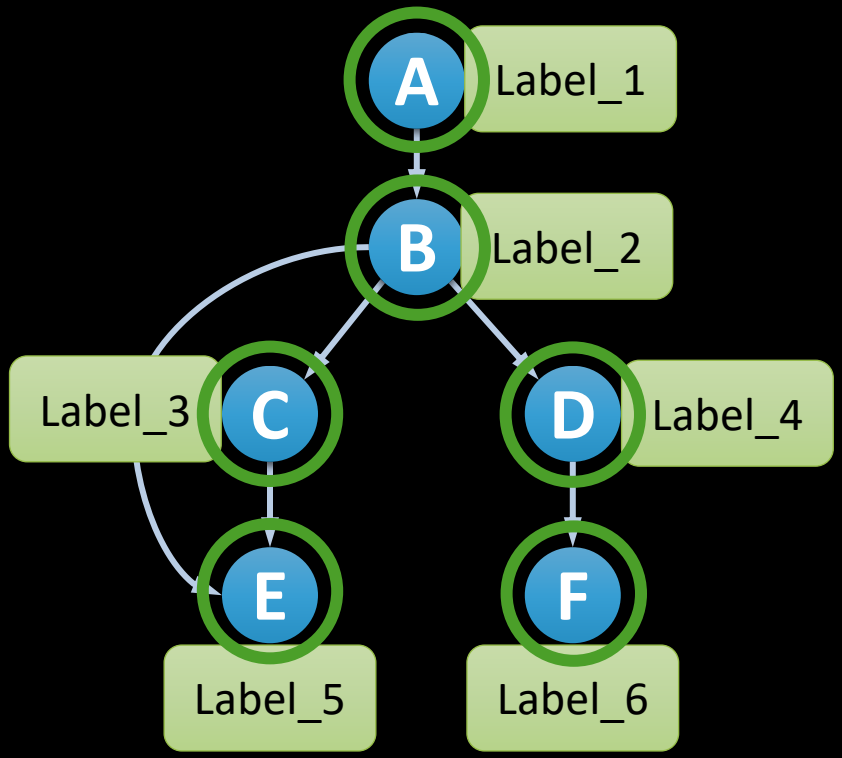
(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



Control-Flow Integrity (CFI)

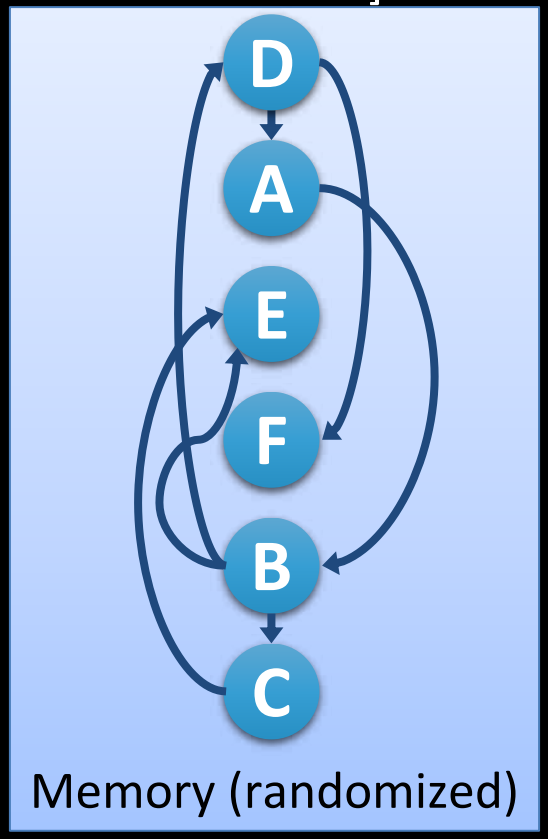
[Abadi et al., CCS 2005 & TISSEC 2009]



Probabilistic vs Enforcement-Based Defense Approach

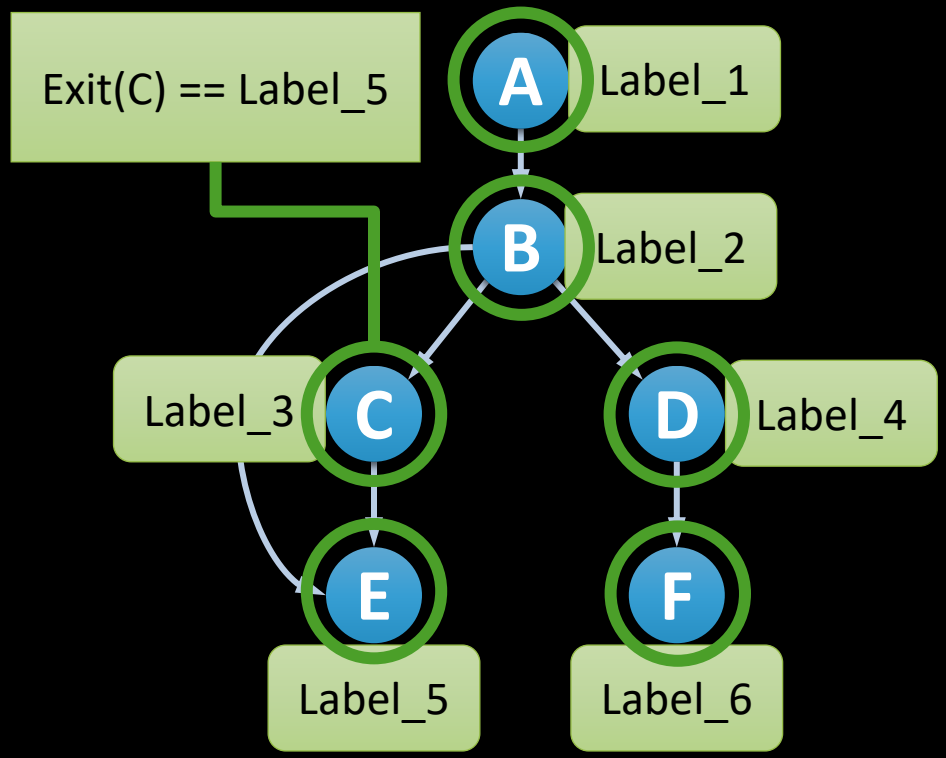
(Fine-grained) Code Randomization

[Cohen 1993 & Larsen et al., SoK IEEE S&P 2014]



Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]



Basics of Code Randomization

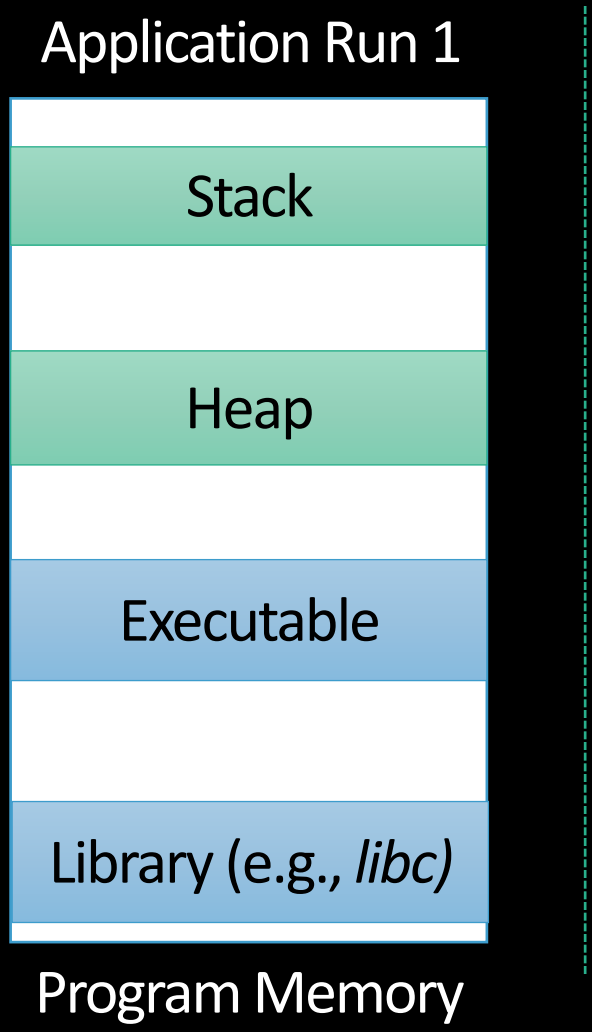
- ASLR randomizes the base address of code/data segments

Basics of Code Randomization

- ASLR randomizes the base address of code/data segments

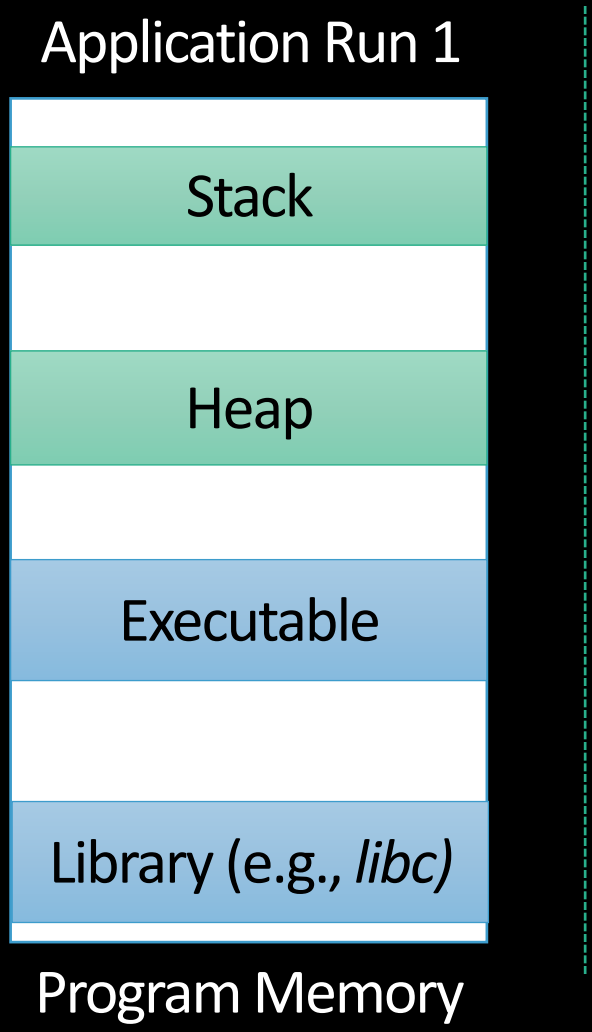
Basics of Code Randomization

- ASLR randomizes the base address of code/data segments



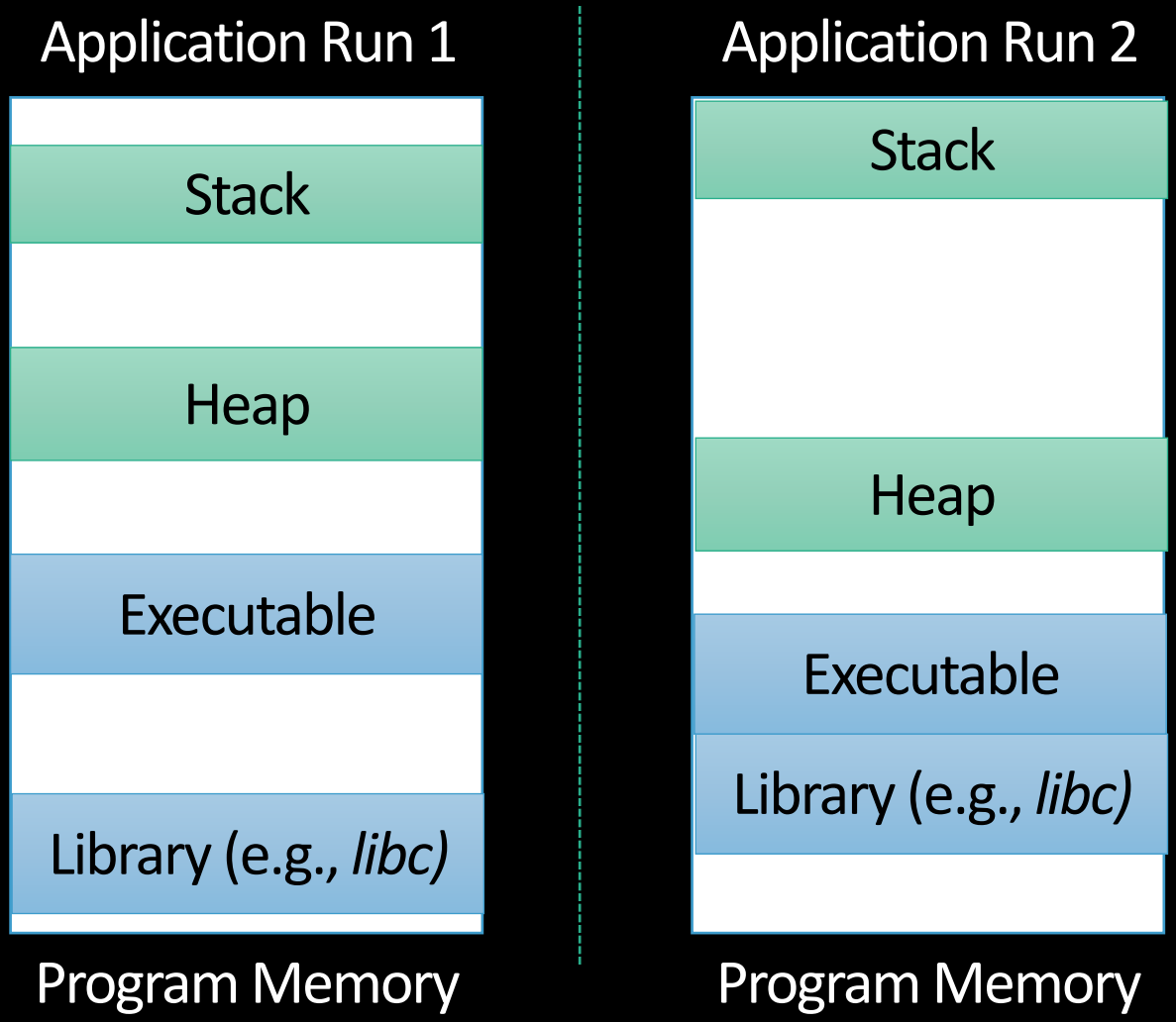
Basics of Code Randomization

- ASLR randomizes the base address of code/data segments



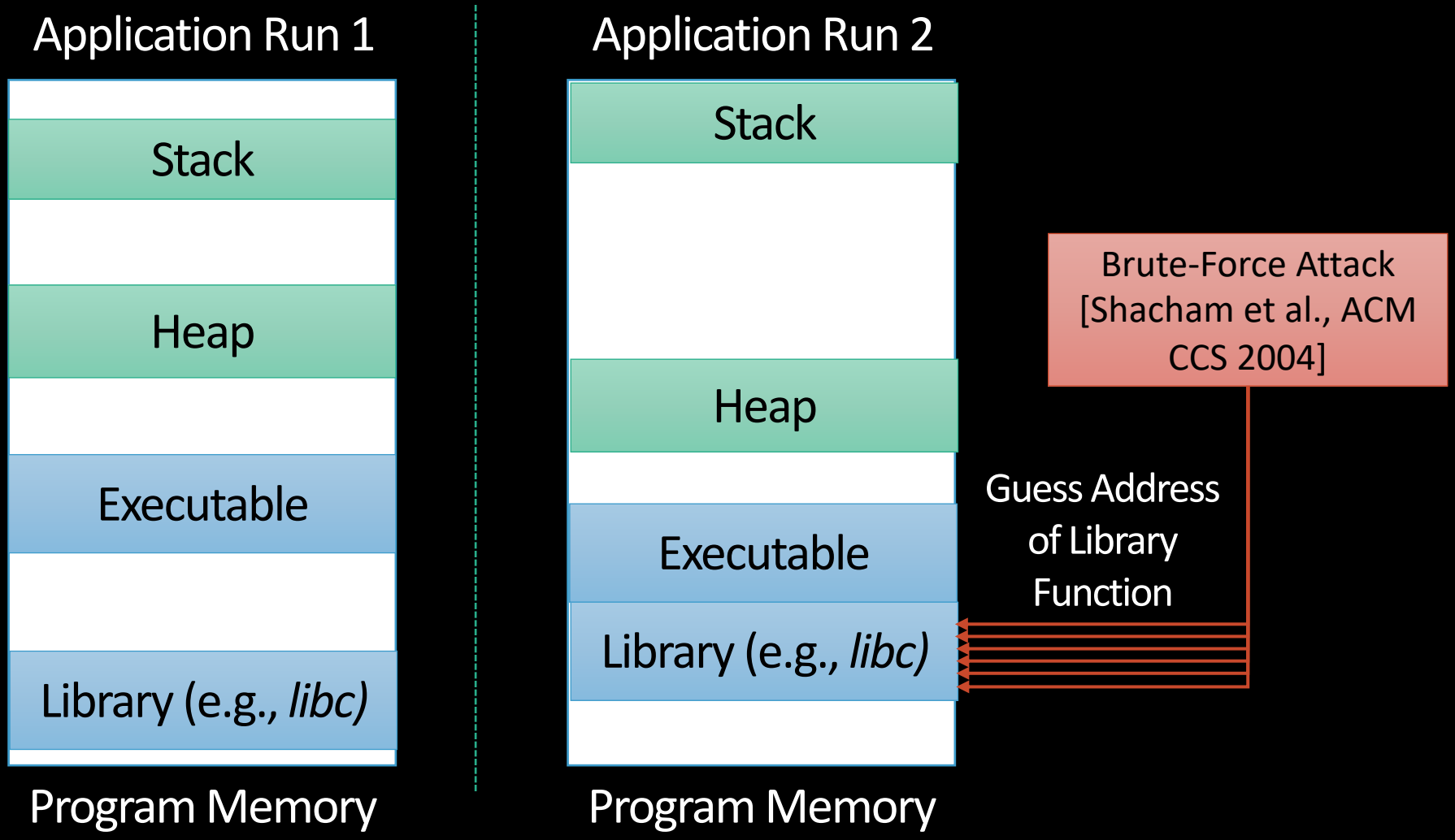
Basics of Code Randomization

- ASLR randomizes the base address of code/data segments

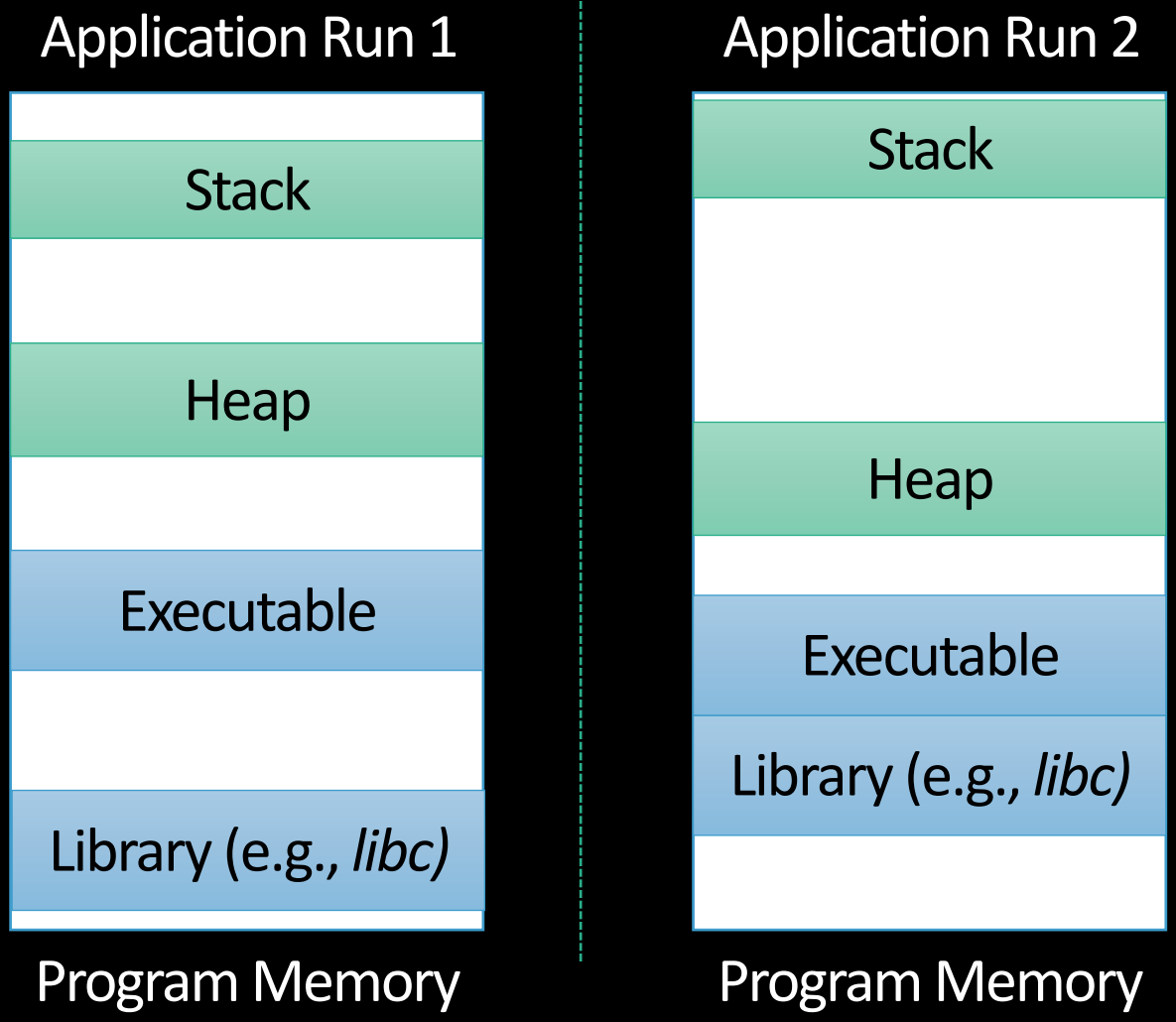


Basics of Code Randomization

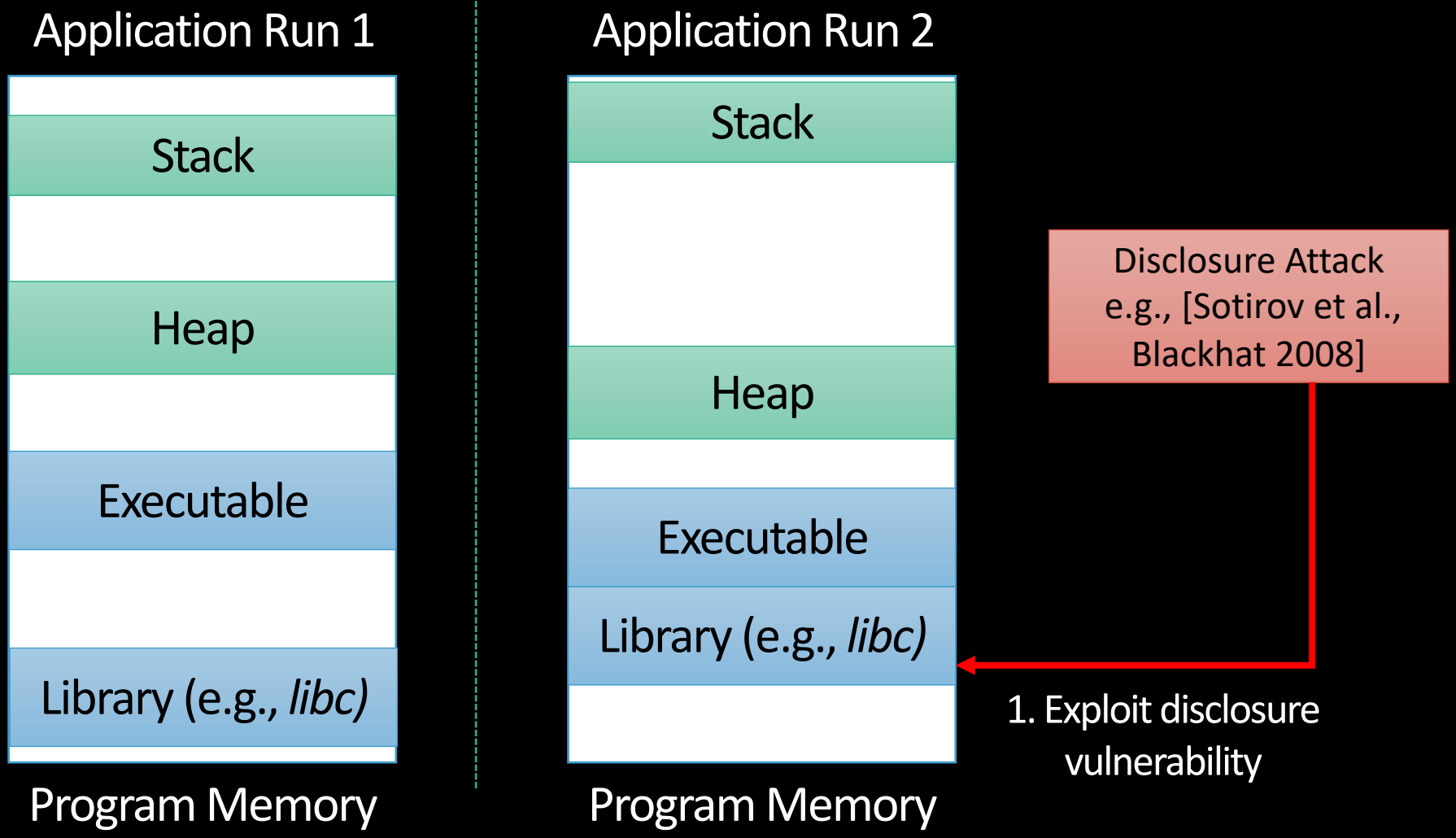
- ASLR randomizes the base address of code/data segments



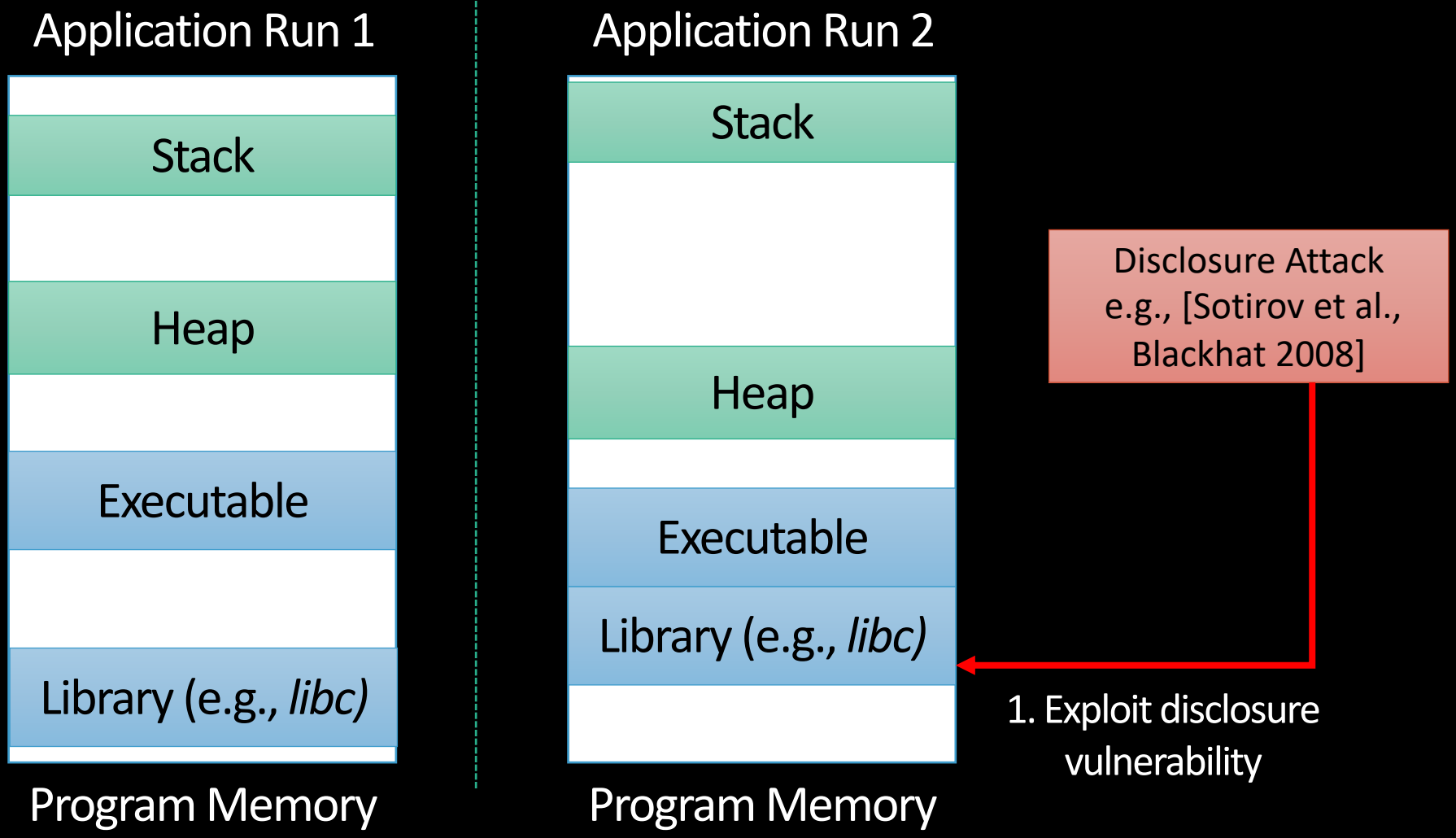
Big Picture


Application Run 1
Stack
Heap
Executable
Library (e.g., *libc)*
Program Memory
Application Run 2
Stack
Heap
Executable
Library (e.g., *libc)*
Program Memory

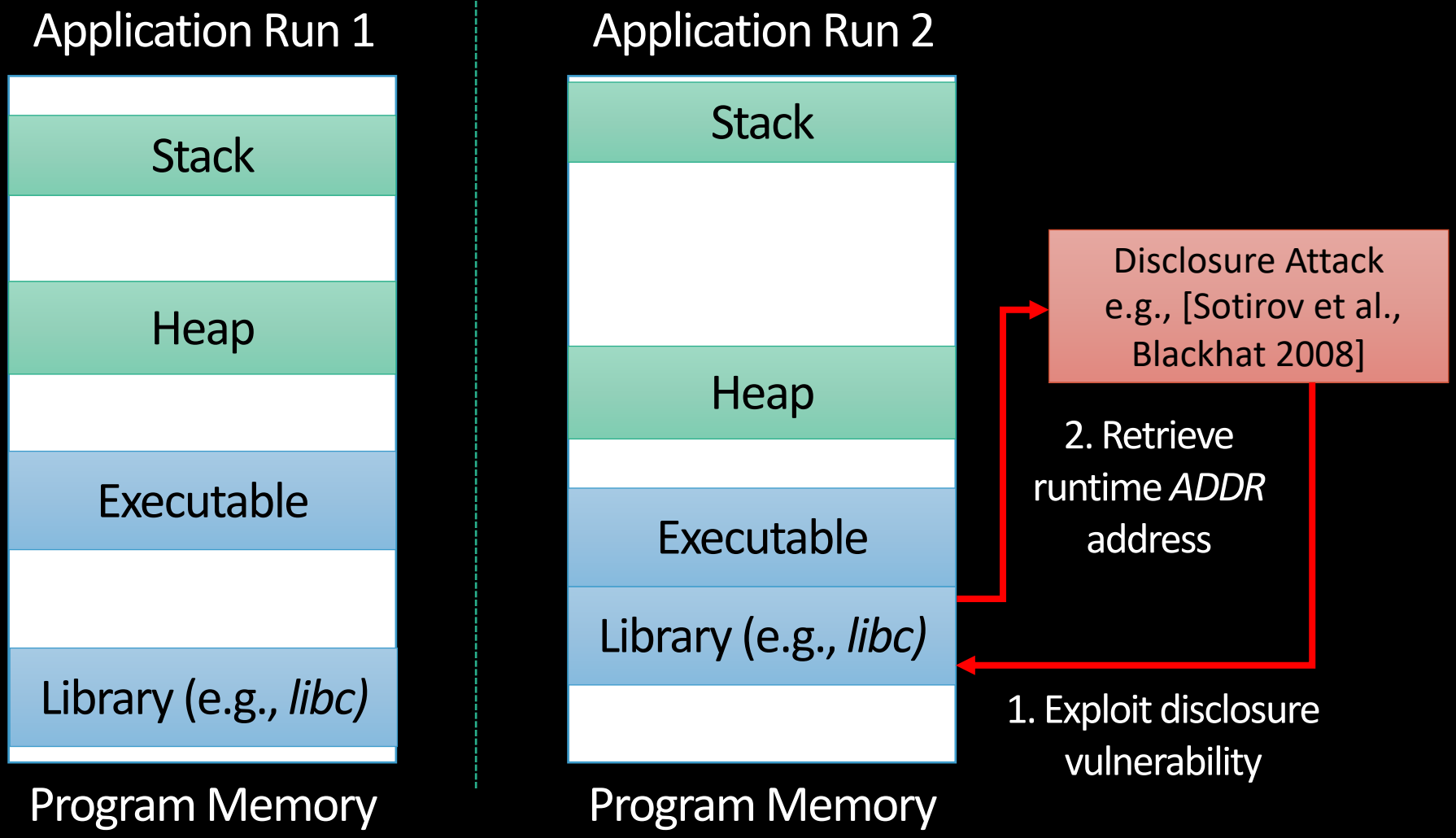
Big Picture


Application Run 1
Stack
Heap
Executable
Library (e.g., *libc)*
Program Memory
Application Run 2
Stack
Heap
Executable
Library (e.g., *libc)*
Program Memory
Disclosure Attack
e.g., [Sotirov et al., Blackhat 2008]
1. Exploit disclosure vulnerability

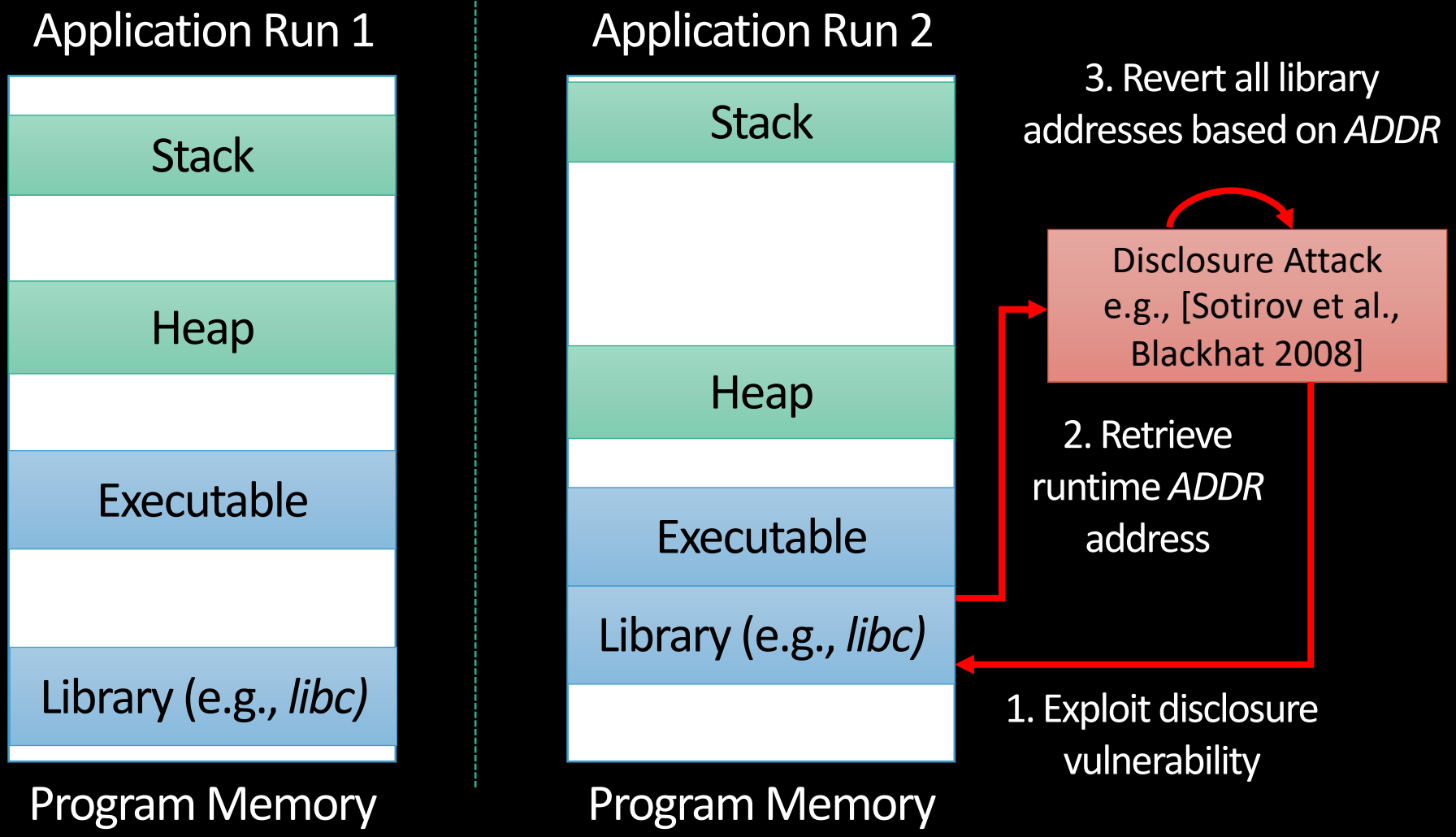
Big Picture


Application Run 1
Stack
Heap
Executable
Library (e.g., *libc)*
Program Memory
Application Run 2
Stack
Heap
Executable
Library (e.g., *libc)*
Program Memory
Disclosure Attack
e.g., [Sotirov et al., Blackhat 2008]
1. Exploit disclosure vulnerability

Big Picture


Application Run 1
Stack
Heap
Executable
Library (e.g., libc)
Program Memory
Application Run 2
Stack
Heap
Executable
Library (e.g., libc)
Program Memory
Disclosure Attack
e.g., [Sotirov et al., Blackhat 2008]
2. Retrieve runtime ADDR address
1. Exploit disclosure vulnerability

Big Picture


Application Run 1
Stack
Heap
Executable
Library (e.g., libc)
Program Memory
Application Run 2
Stack
Heap
Executable
Library (e.g., libc)
Program Memory
3. Revert all library addresses based on ADDR
Disclosure Attack e.g., [Sotirov et al., Blackhat 2008]
2. Retrieve runtime ADDR address
1. Exploit disclosure vulnerability

Making exploitation harder with fine-grained code randomization

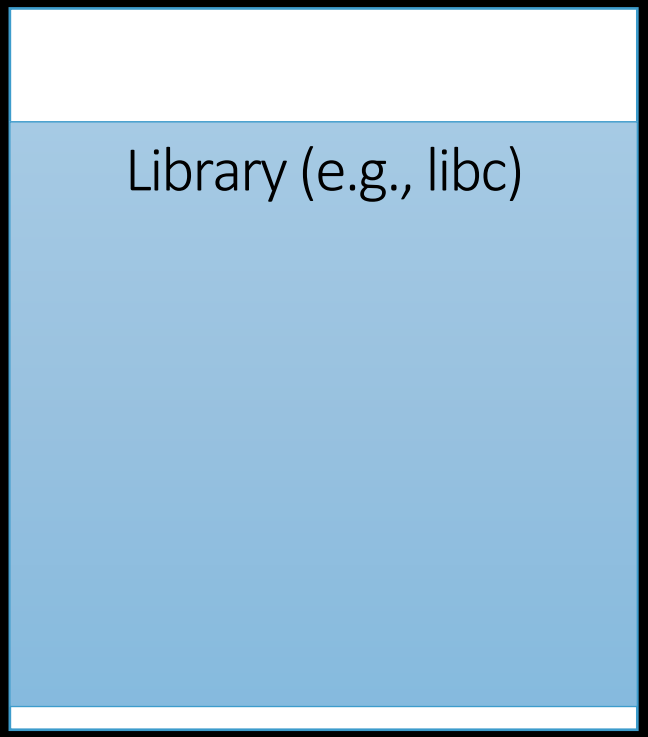
Fine-Grained ASLR

Fine-Grained ASLR

Application Run 1

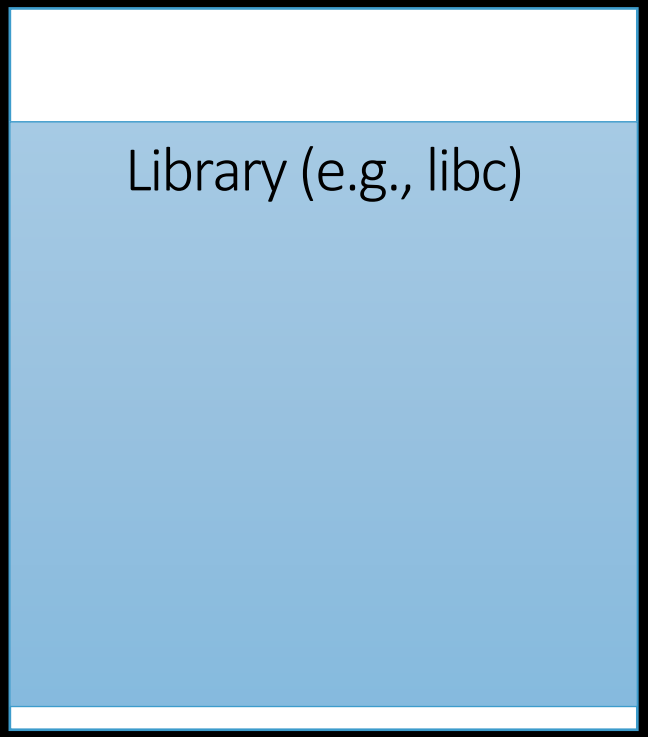
Fine-Grained ASLR

Application Run 1

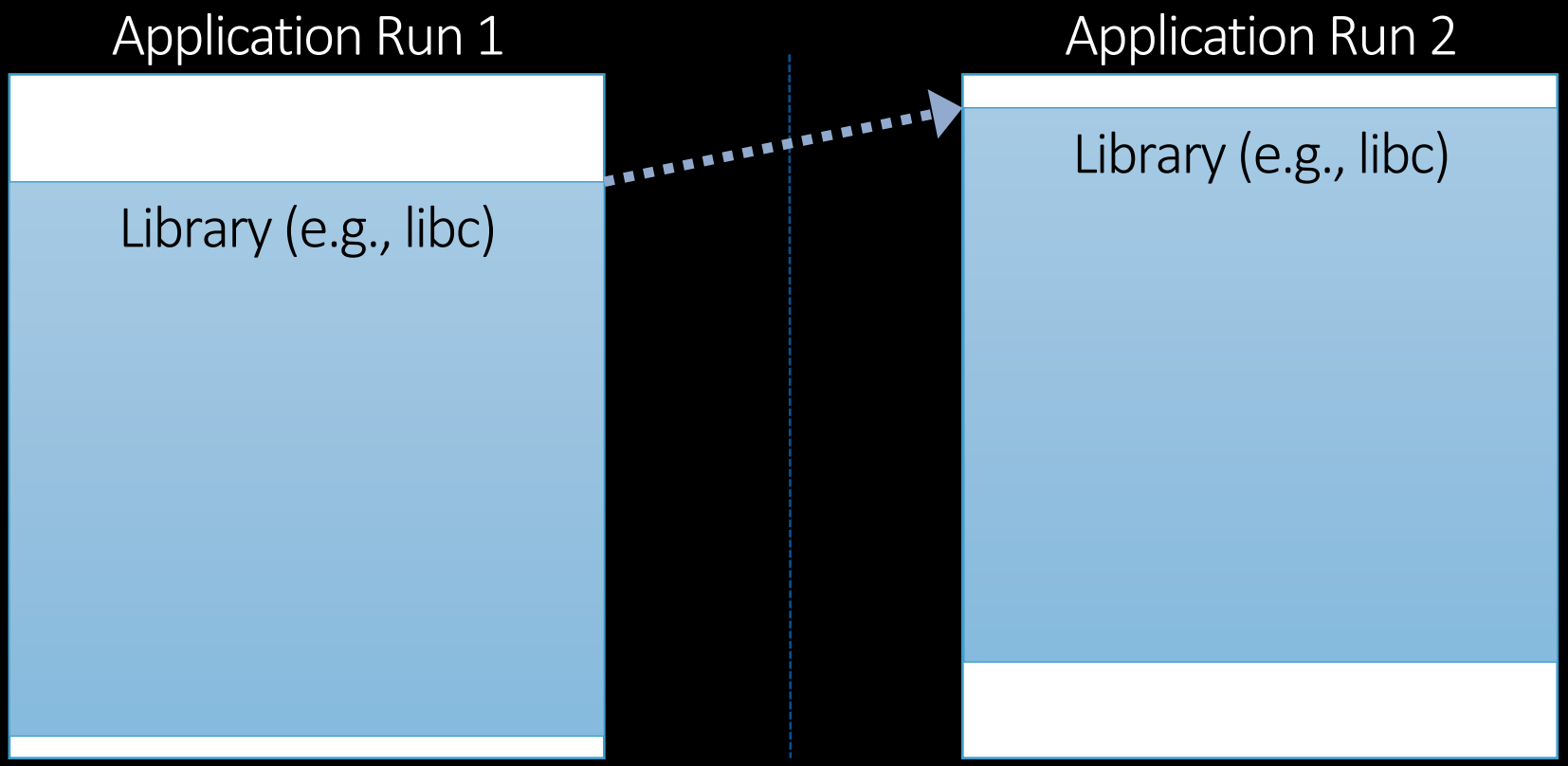


Fine-Grained ASLR

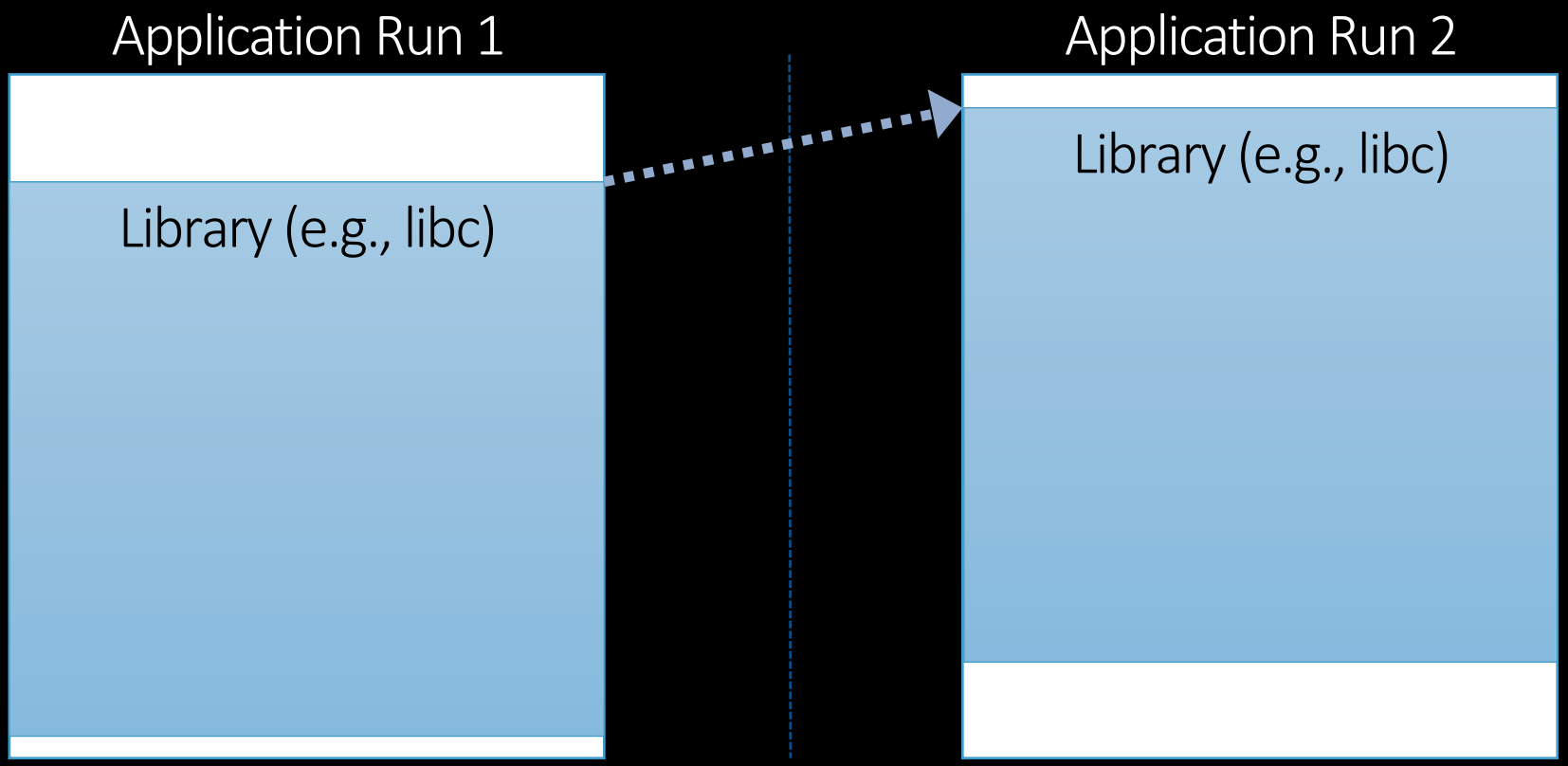
Application Run 1



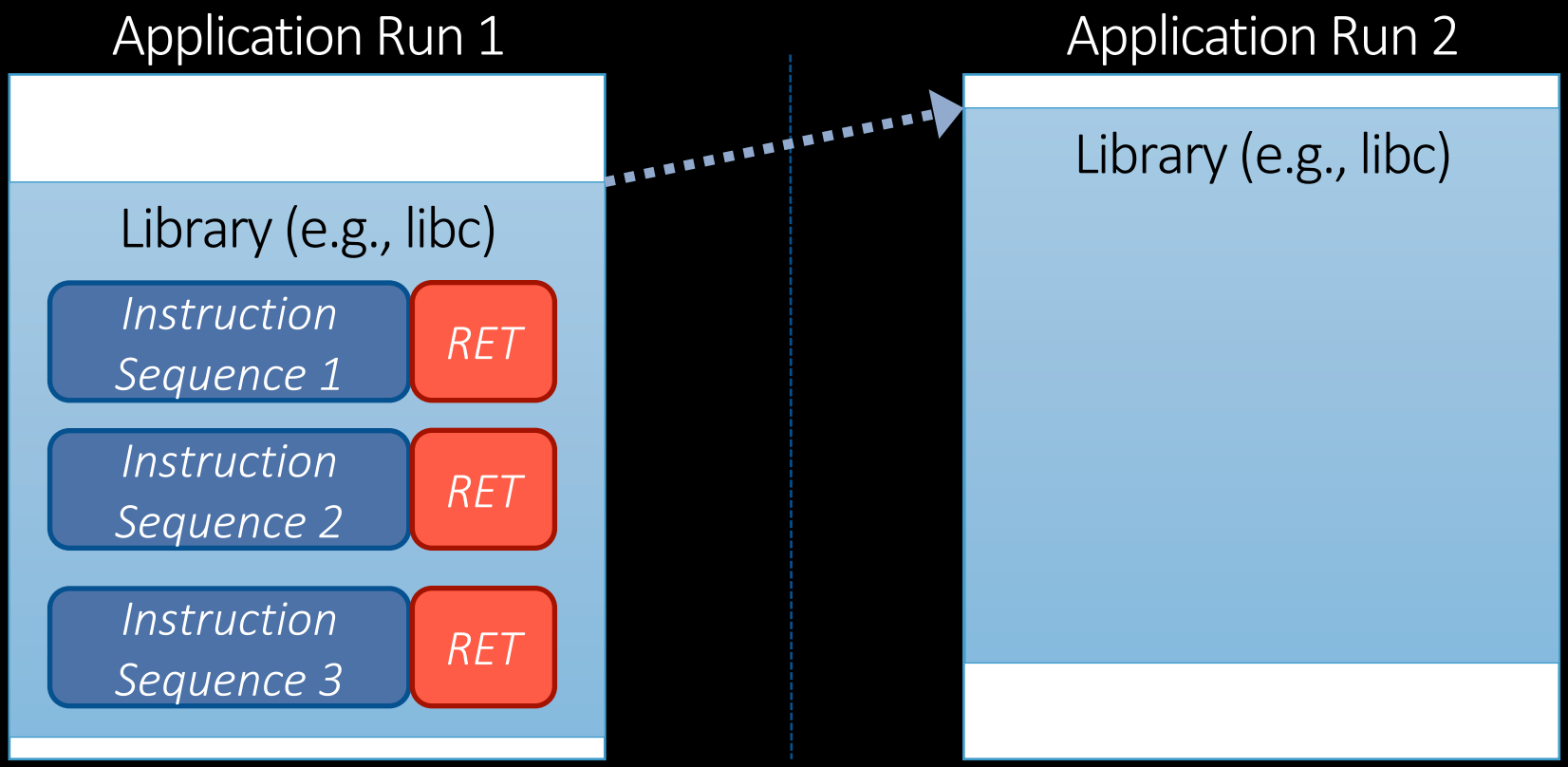
Fine-Grained ASLR



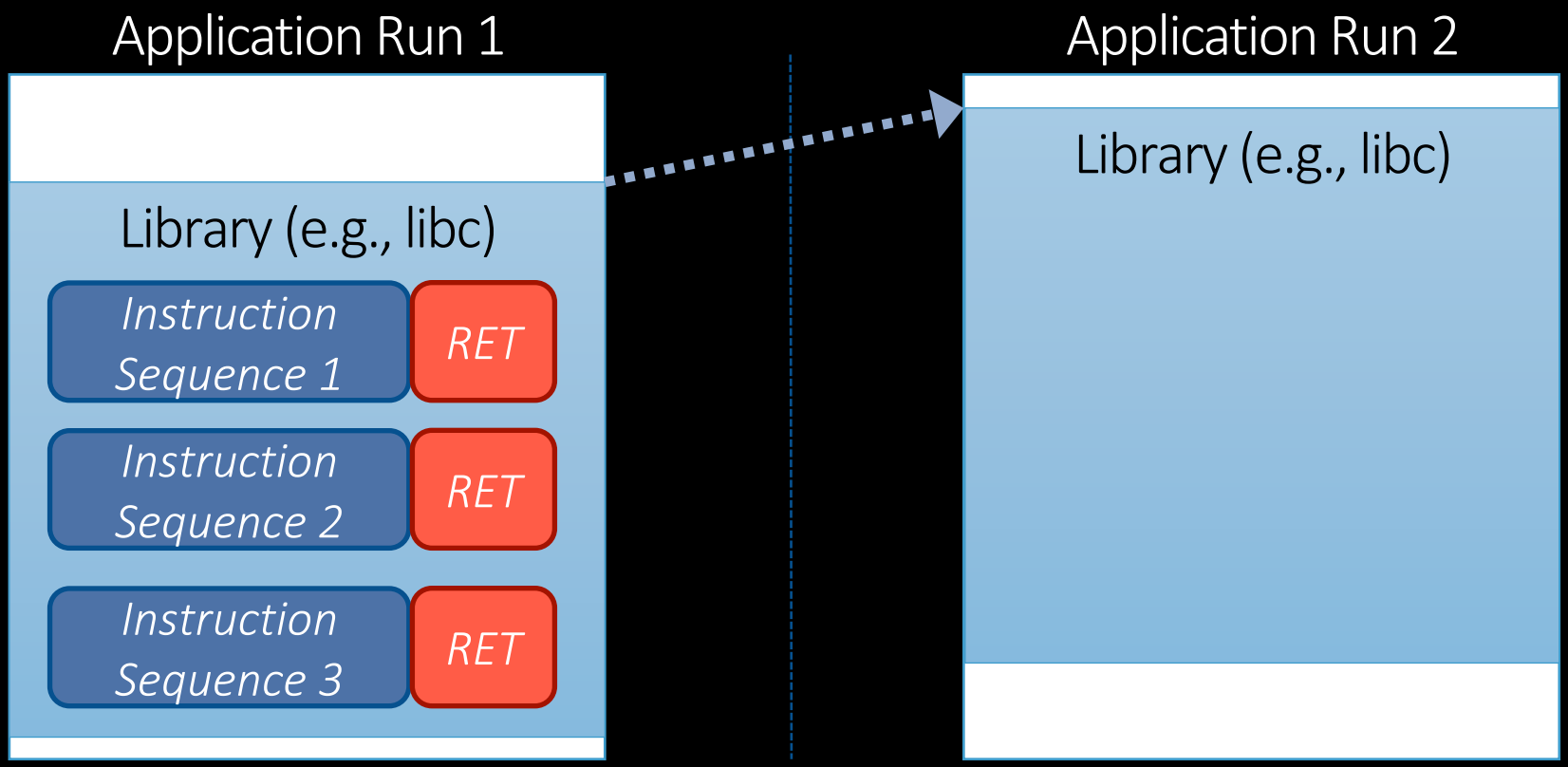
Fine-Grained ASLR



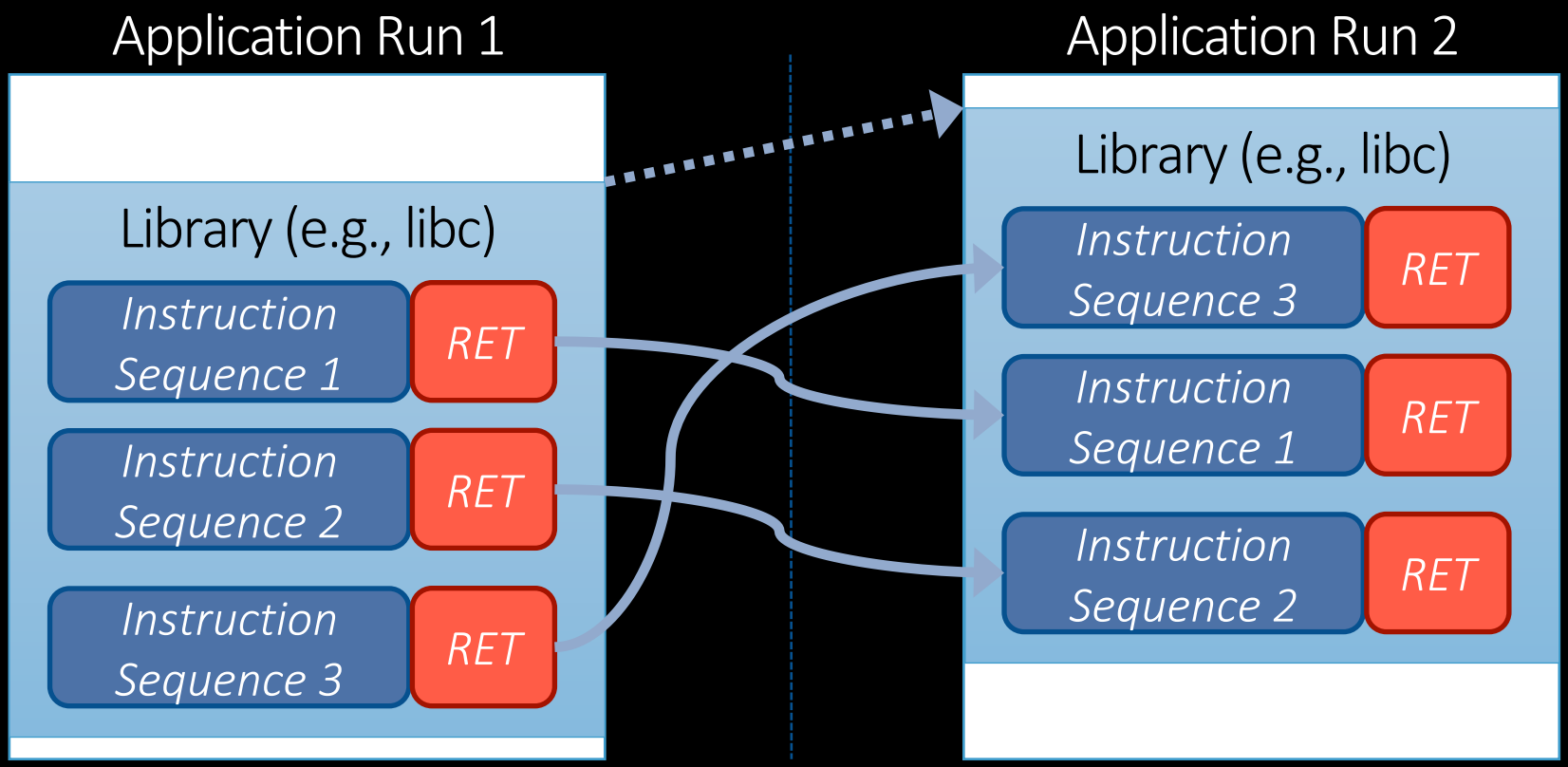
Fine-Grained ASLR



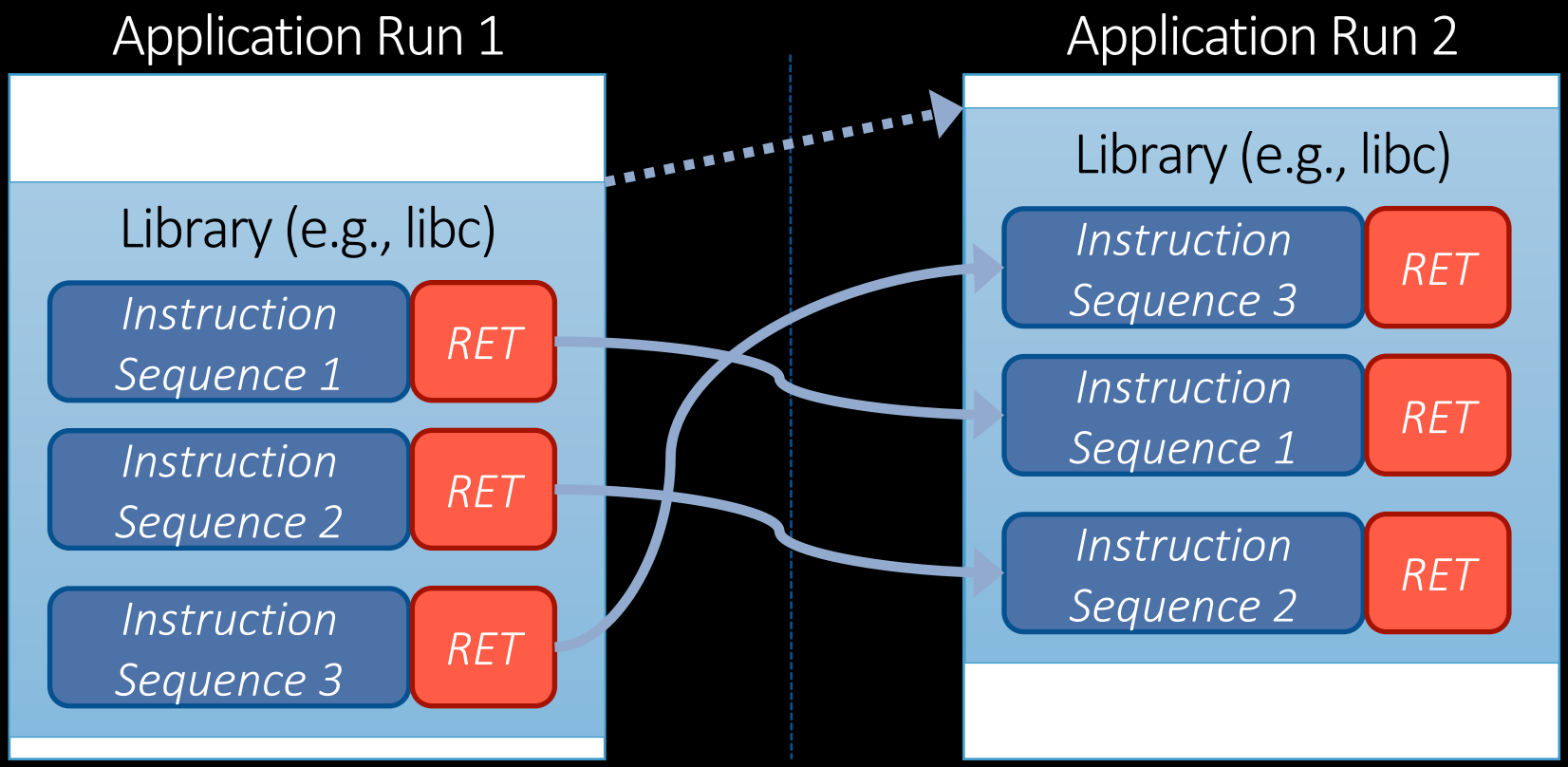
Fine-Grained ASLR



Fine-Grained ASLR



Fine-Grained ASLR



- **ORP** [Pappas et al., IEEE S&P 2012]: Instruction reordering/substitution within a BBL

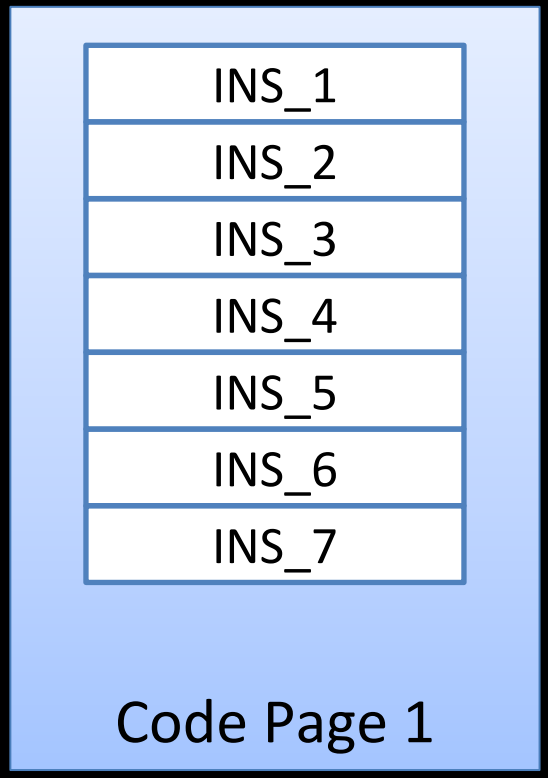
  ILR [Hiser et al., IEEE S&P 2012]: Randomizing each instruction‘s location
- **STIR** [Wartell et al., ACM CCS 2012] & **XIFER** [with Davi et al., AsiaCCS 2013]: Permutation of BBLs

Does Fine-Grained ASLR Provide a Viable Defense in the Long Run?

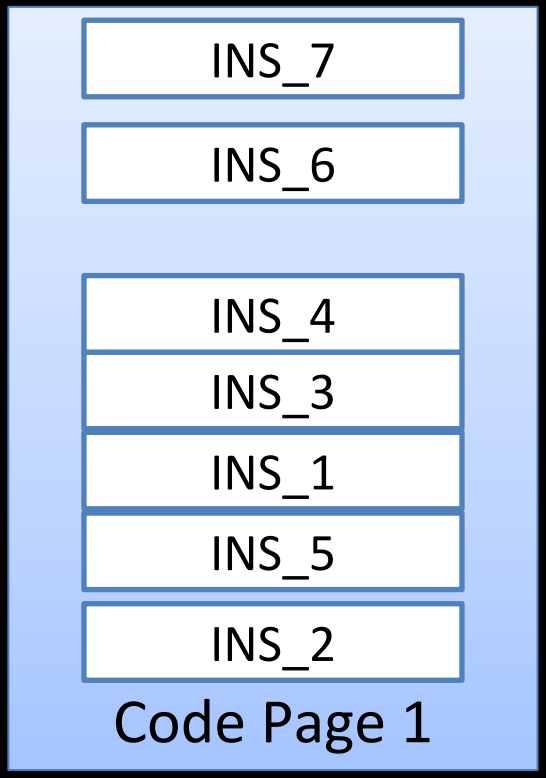
Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization

[Kevin Snow, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, Fabian Monrose, Ahmad-Reza Sadeghi, Best Student Paper at IEEE S&P 2013]

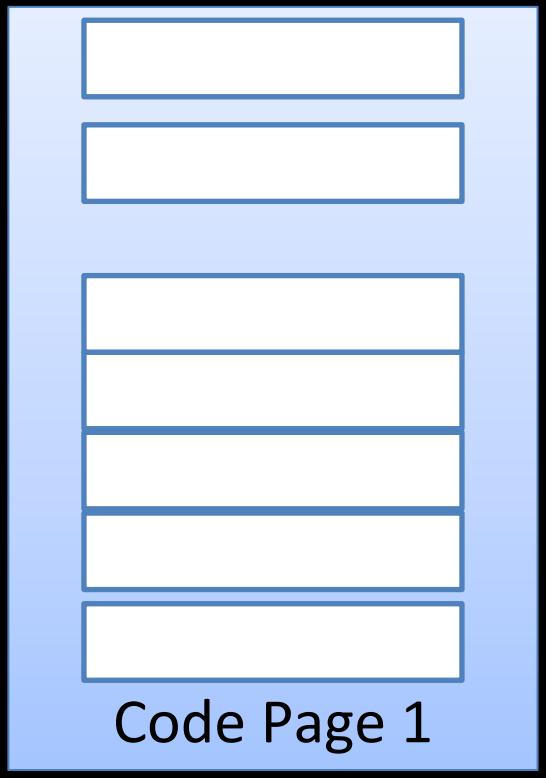
High-Level Idea



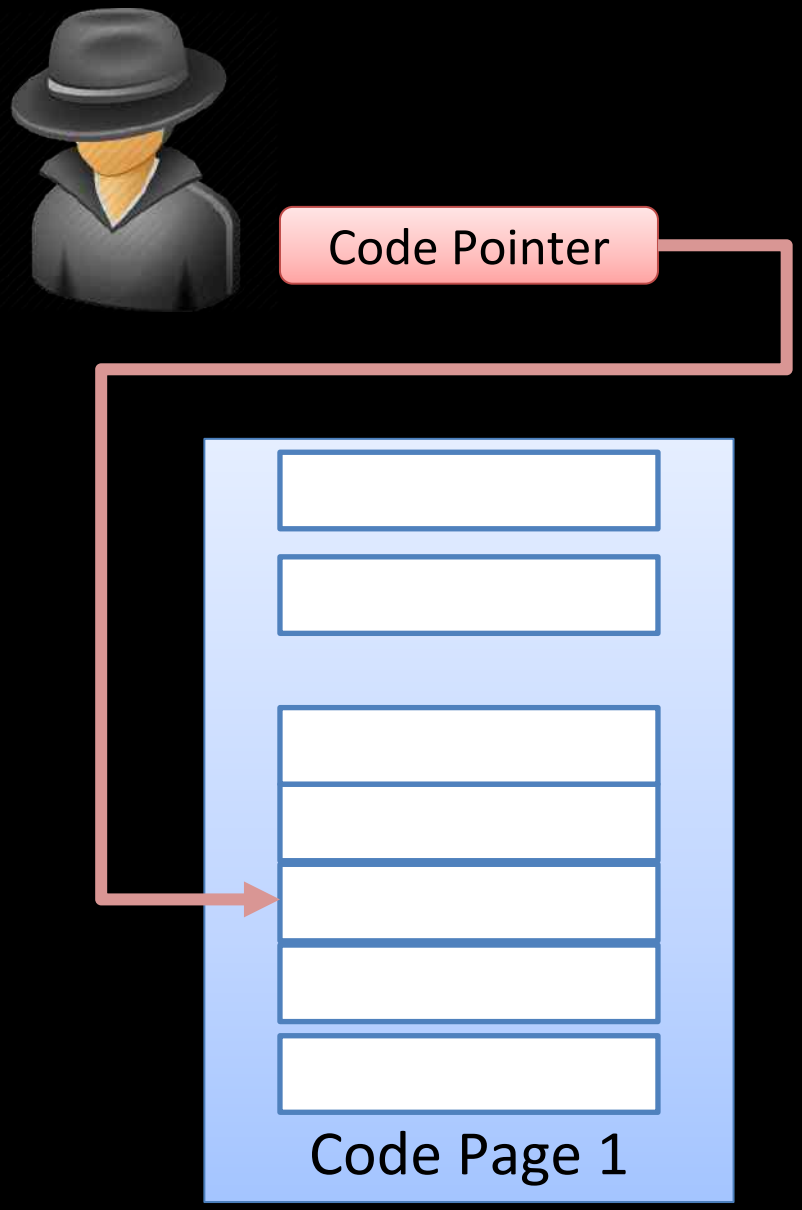
High-Level Idea



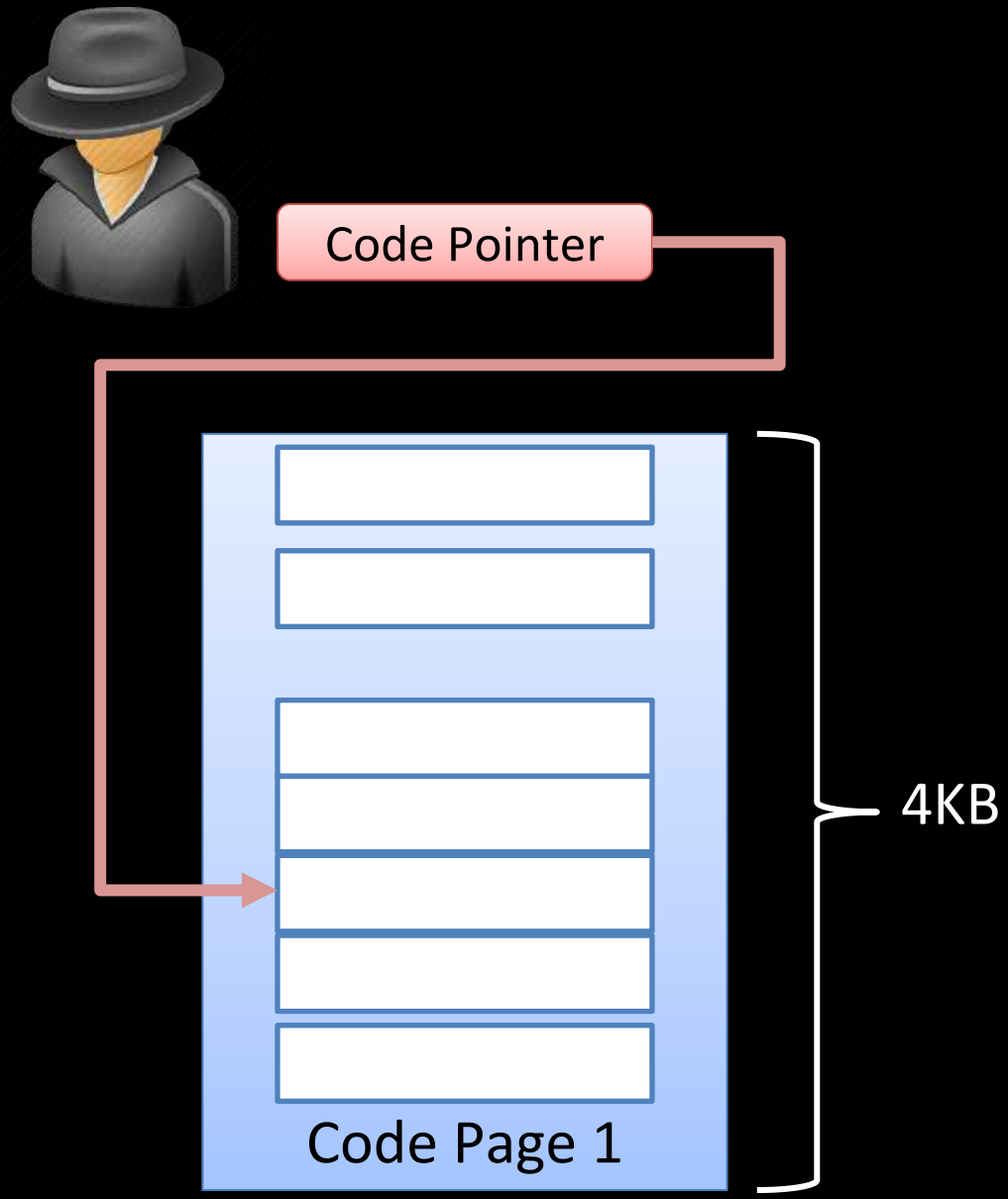
High-Level Idea



High-Level Idea

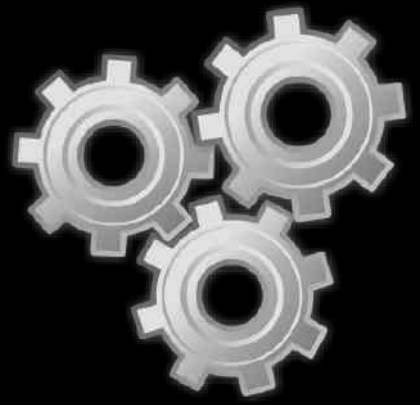


High-Level Idea

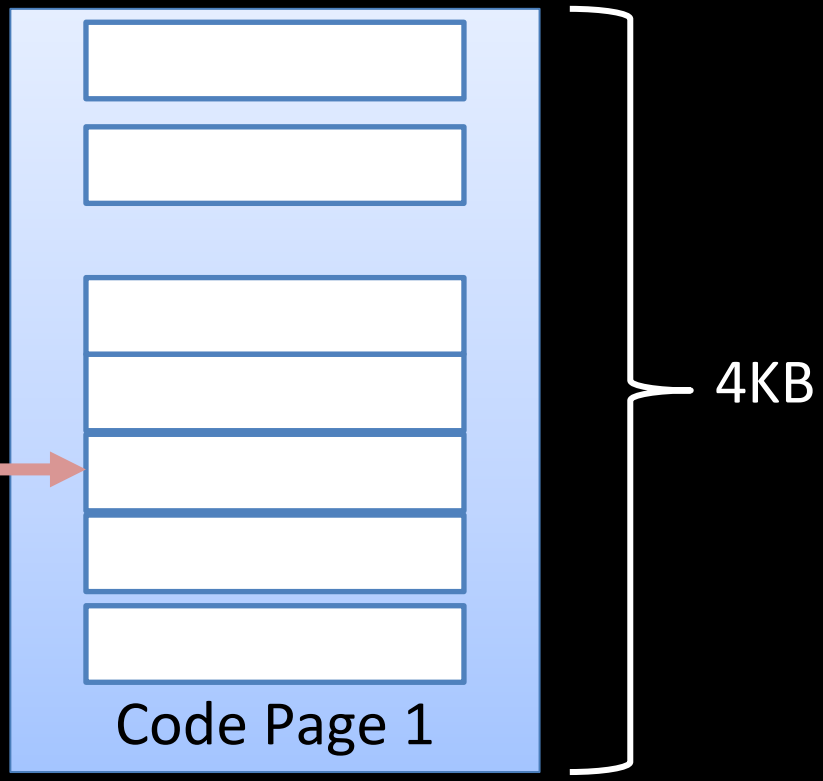


High-Level Idea

Scripting Engine

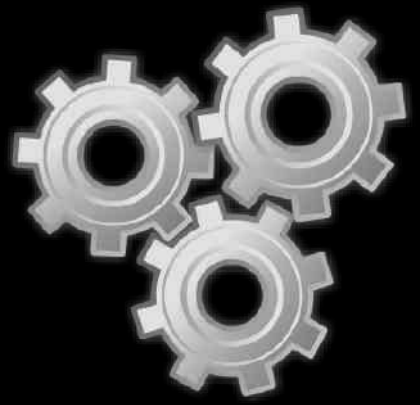


Code Pointer



High-Level Idea

Scripting Engine



Code Pointer

Page Start

Disassemble

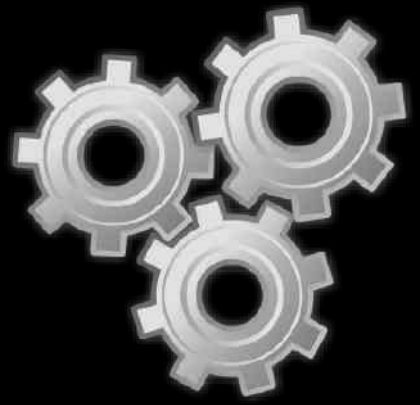
4KB

Code Page 1

Page End

High-Level Idea

Scripting Engine



Code Pointer

Page Start

Disassemble

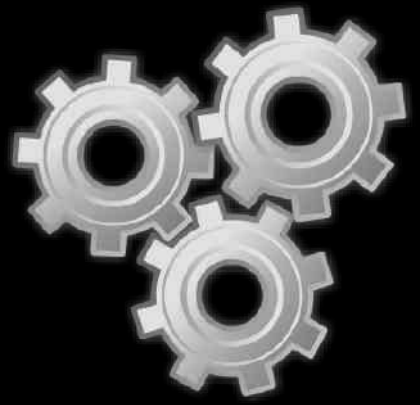
4KB

Code Page 1

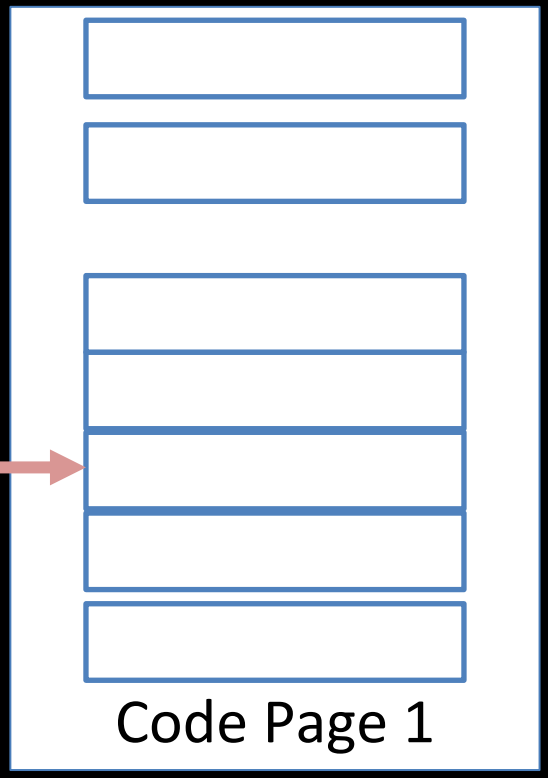
Page End

High-Level Idea

Scripting Engine

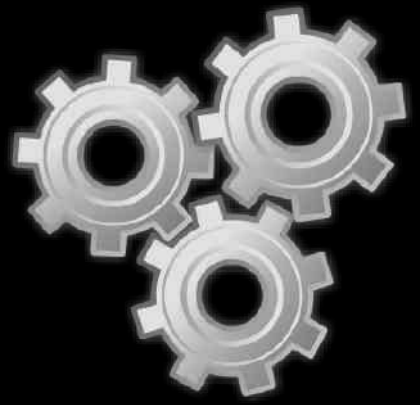


Code Pointer

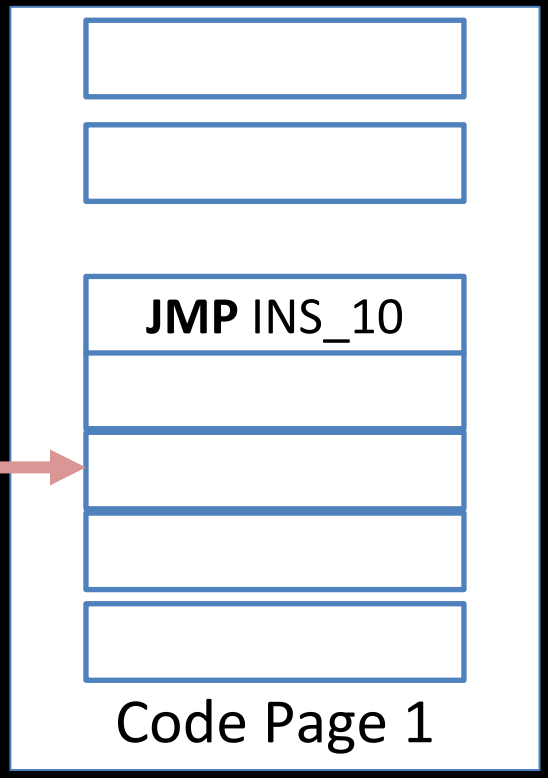


High-Level Idea

Scripting Engine

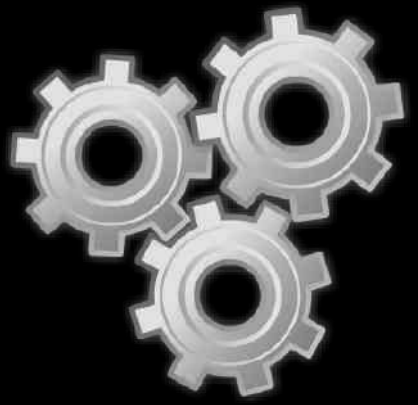


Code Pointer

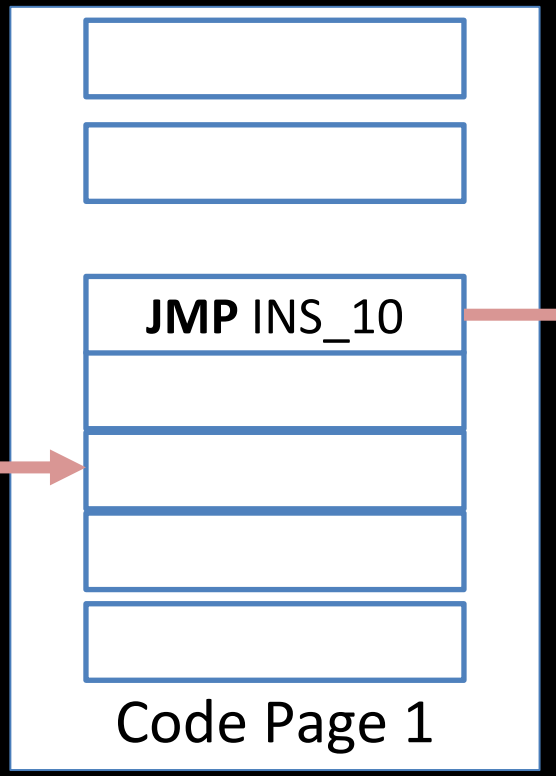


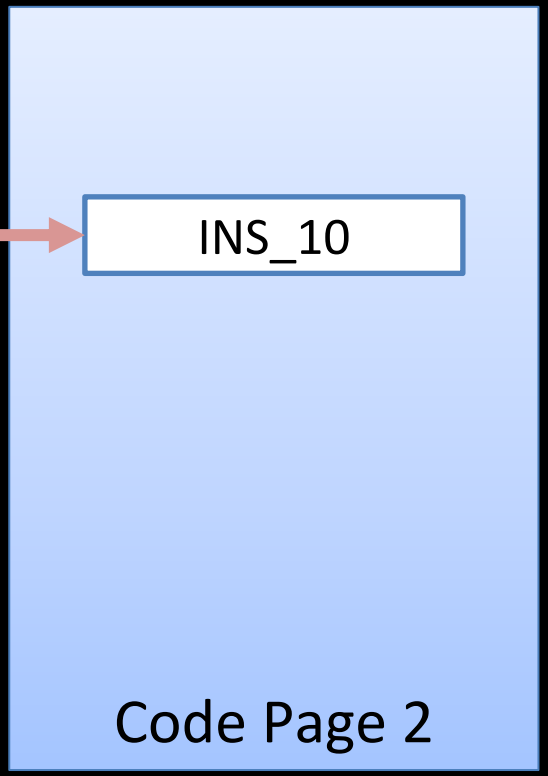
High-Level Idea

Scripting Engine



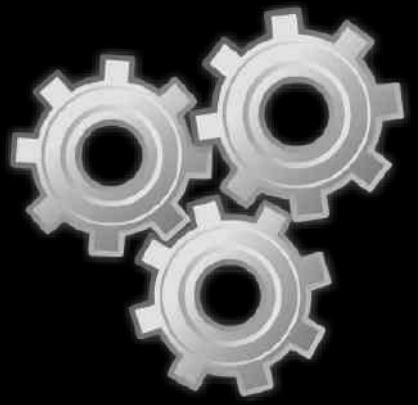
Code Pointer



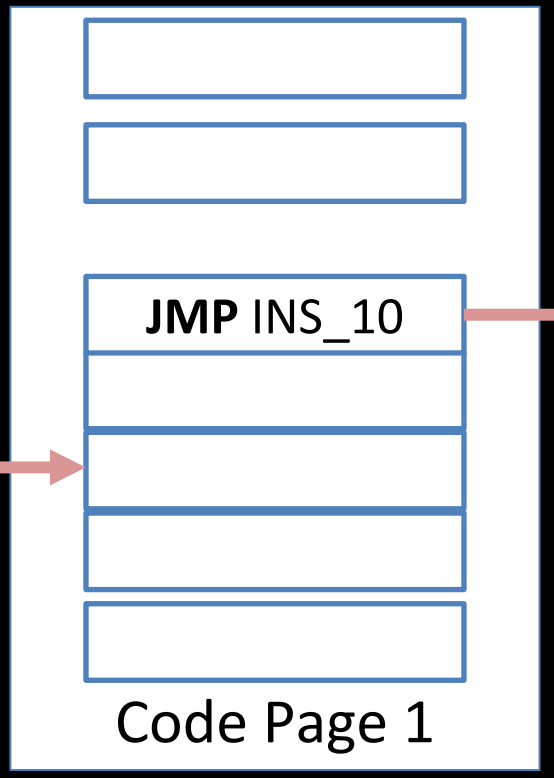


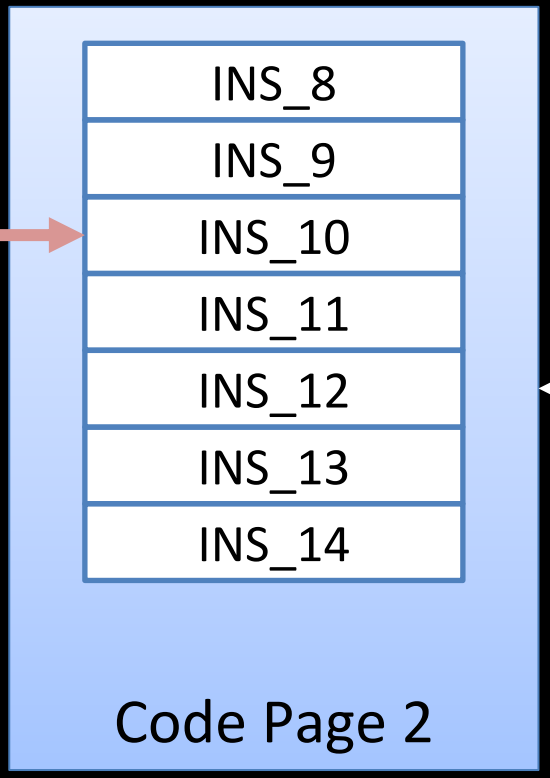
High-Level Idea

Scripting Engine



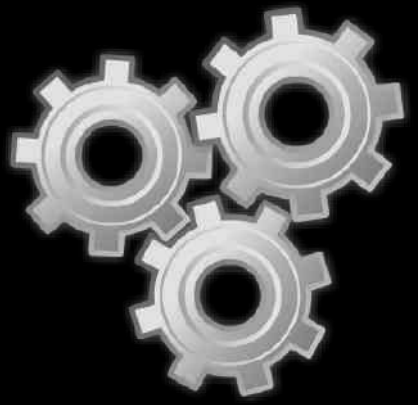
Code Pointer



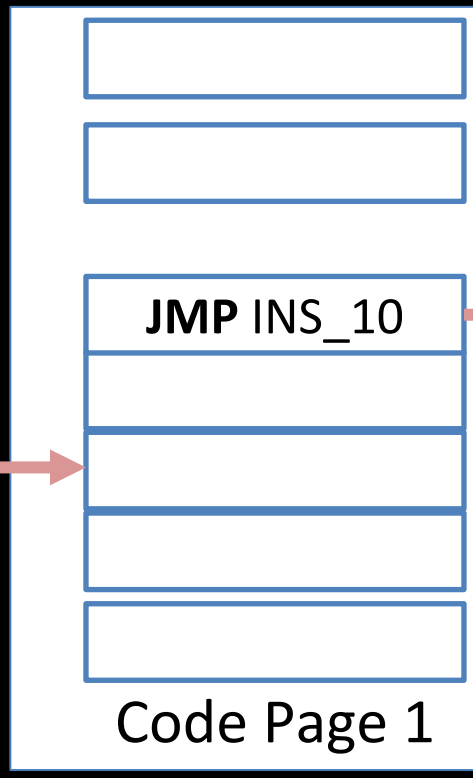


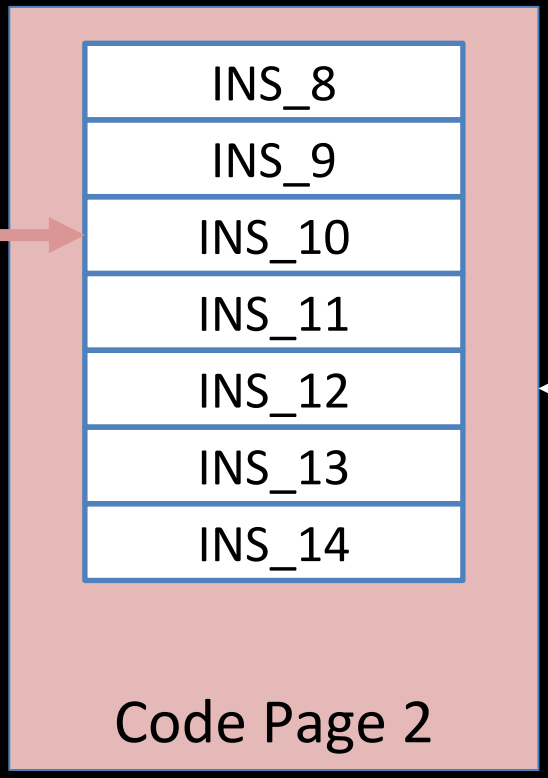
High-Level Idea

Scripting Engine



Code Pointer





Execute-Only Memory

Remove the read permission from code pages

see e.g., Readactor [IEEE S&P 2015]

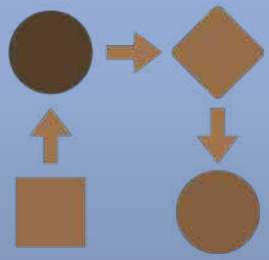
Overview

Background on Run-Time Attacks

Building Code Randomization Defenses

Code-Reuse Attacks

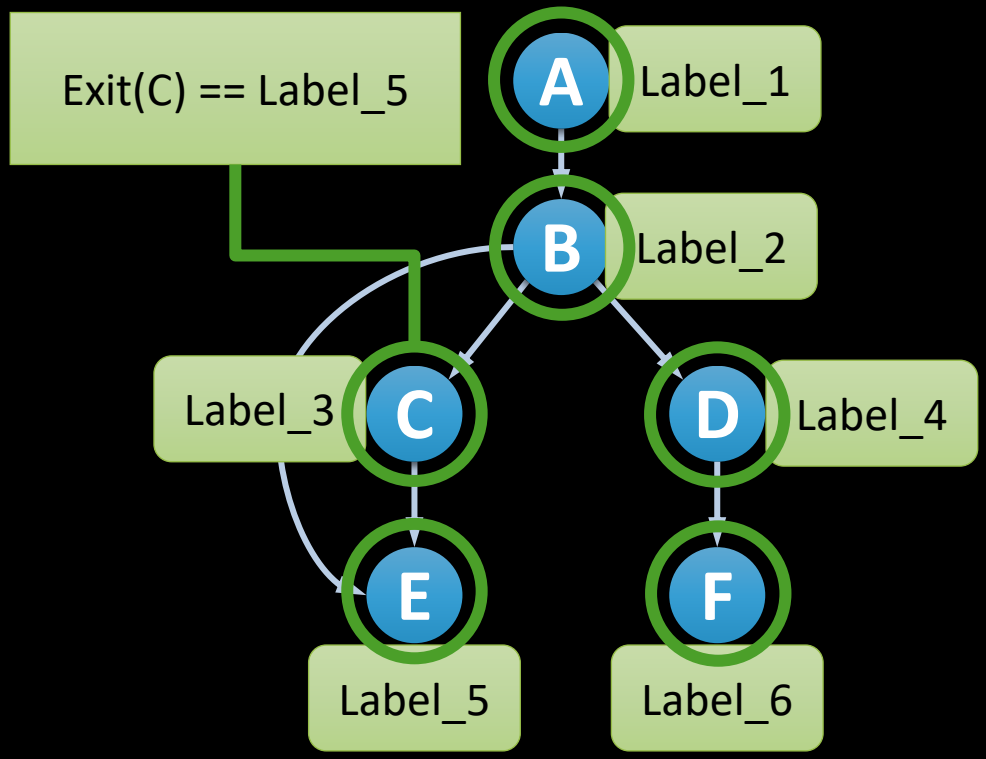
Building Control-Flow Integrity Defenses



Data-Oriented Exploits

Control-Flow Integrity (CFI)

[Abadi et al., CCS 2005 & TISSEC 2009]

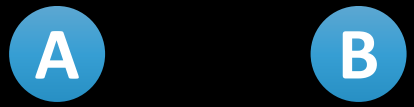


Label Problem for Returns

- Static CFI label checking leads to coarse-grained protection for returns

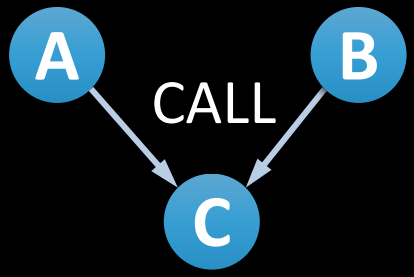
Label Problem for Returns

- Static CFI label checking leads to coarse-grained protection for returns



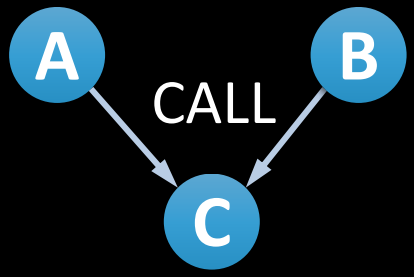
Label Problem for Returns

- Static CFI label checking leads to coarse-grained protection for returns



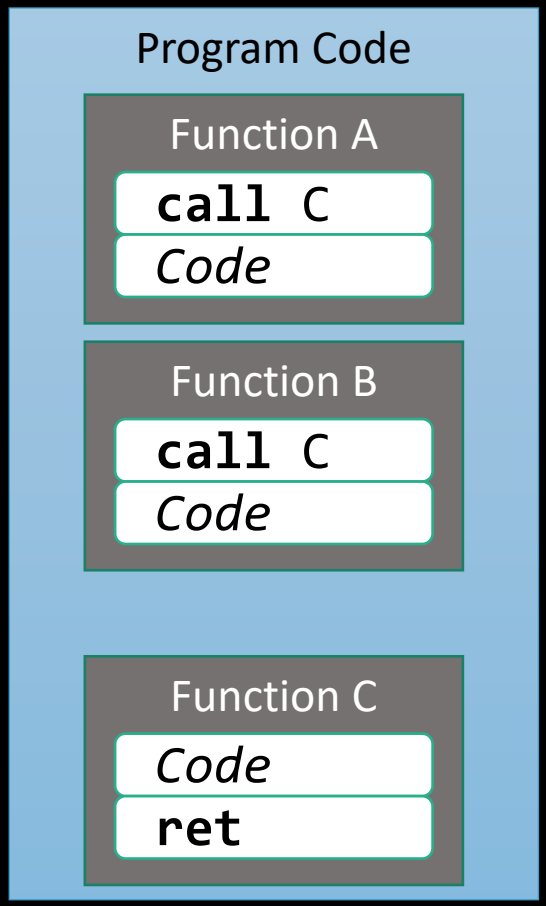
Label Problem for Returns

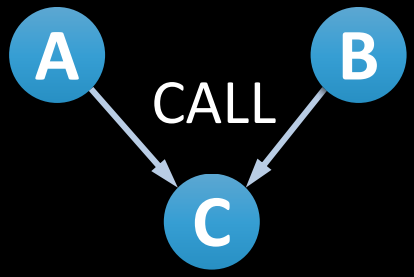
- Static CFI label checking leads to coarse-grained protection for returns



Label Problem for Returns

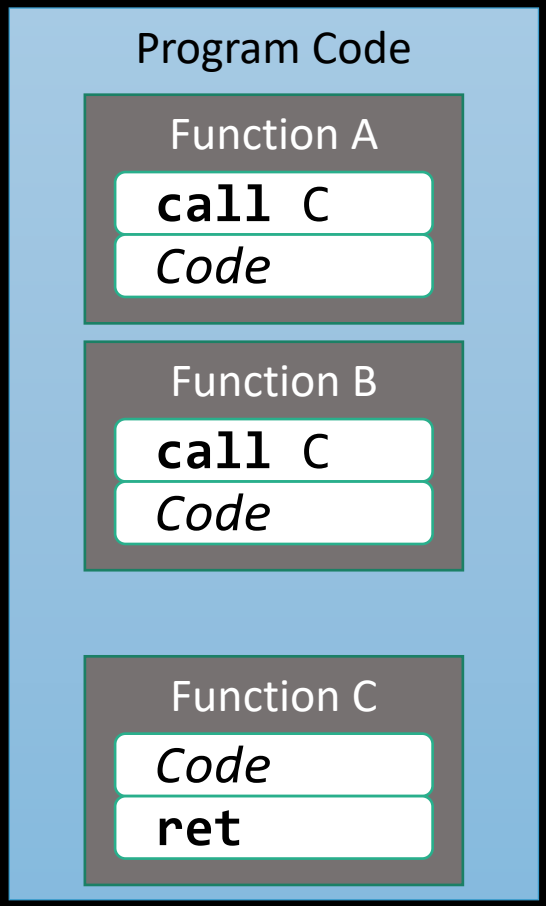
- Static CFI label checking leads to coarse-grained protection for returns

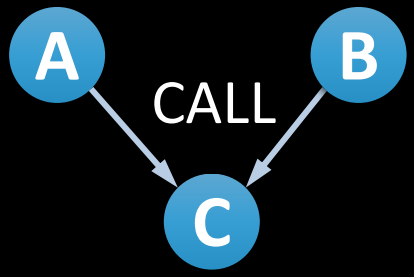




Label Problem for Returns

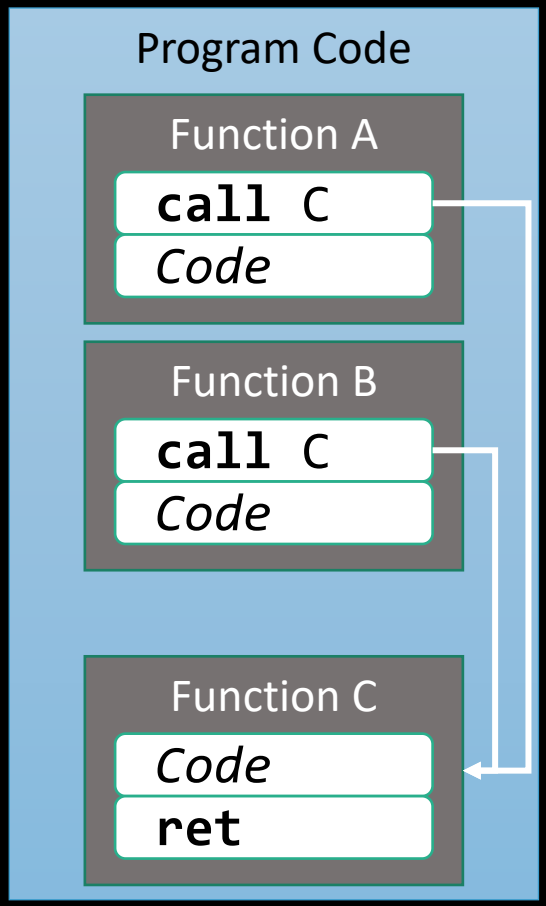
- Static CFI label checking leads to coarse-grained protection for returns

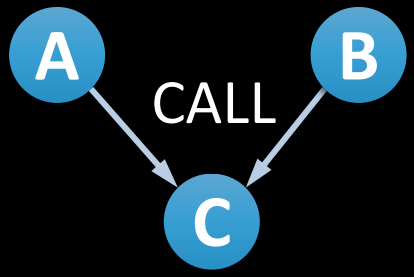




Label Problem for Returns

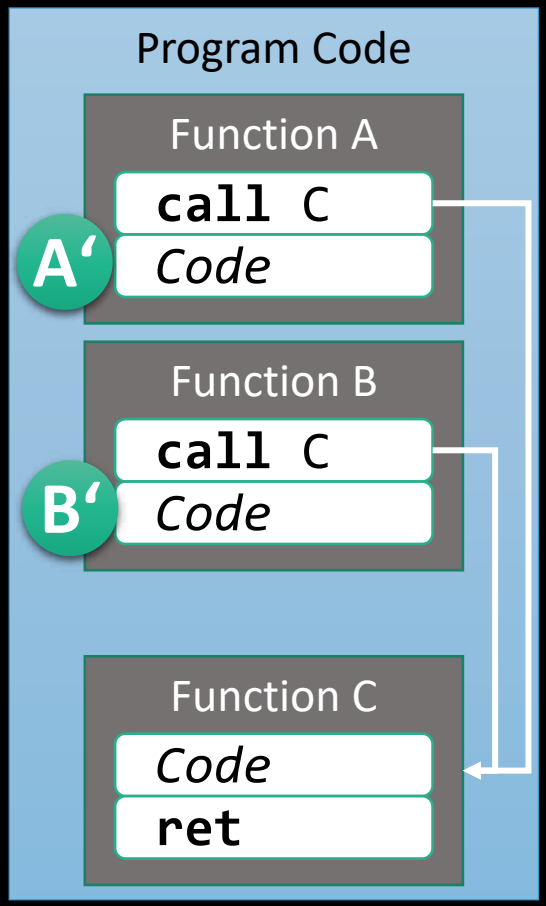
- Static CFI label checking leads to coarse-grained protection for returns

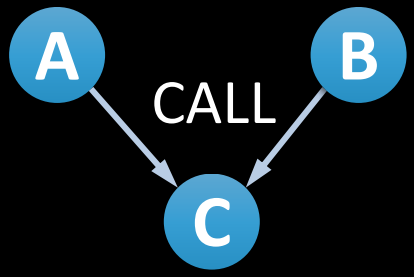




Label Problem for Returns

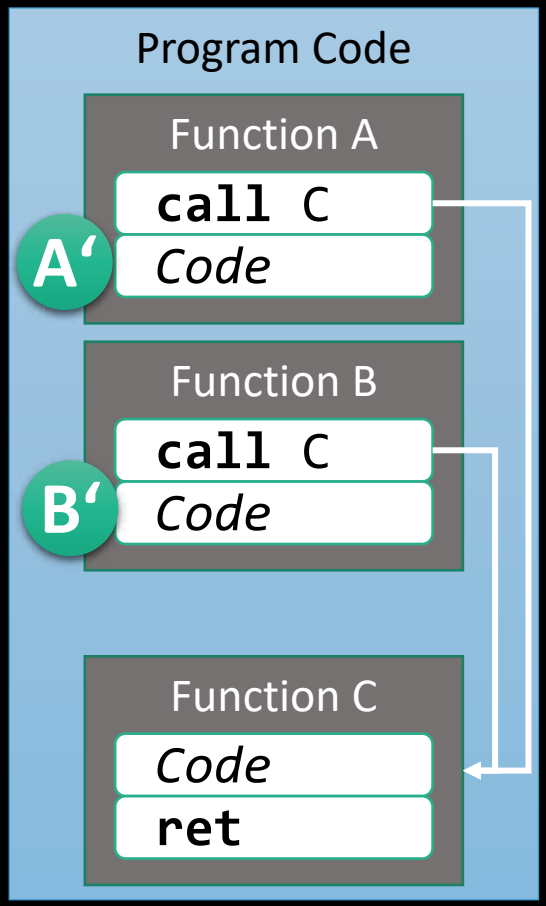
- Static CFI label checking leads to coarse-grained protection for returns

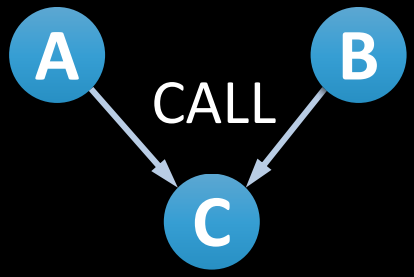




Label Problem for Returns

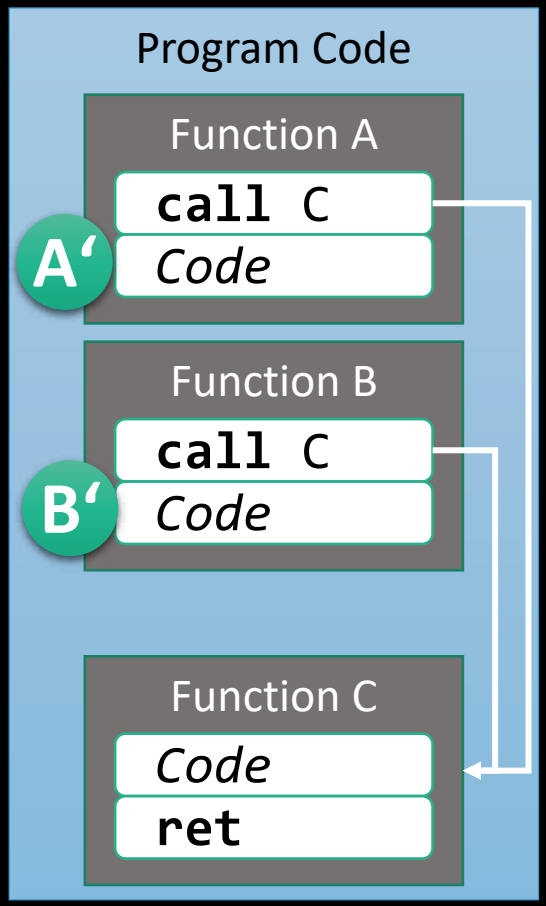
- Static CFI label checking leads to coarse-grained protection for returns

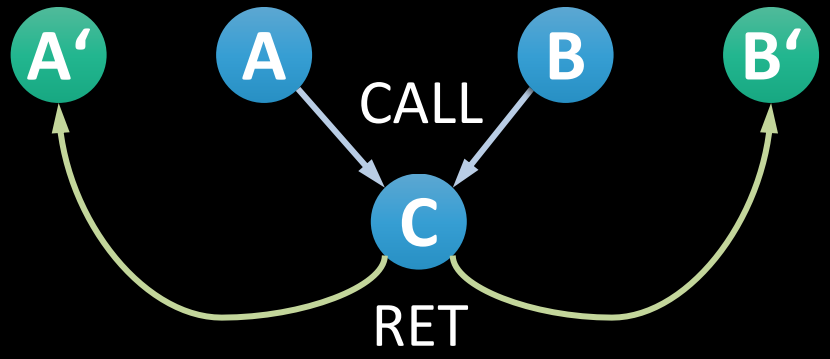




Label Problem for Returns

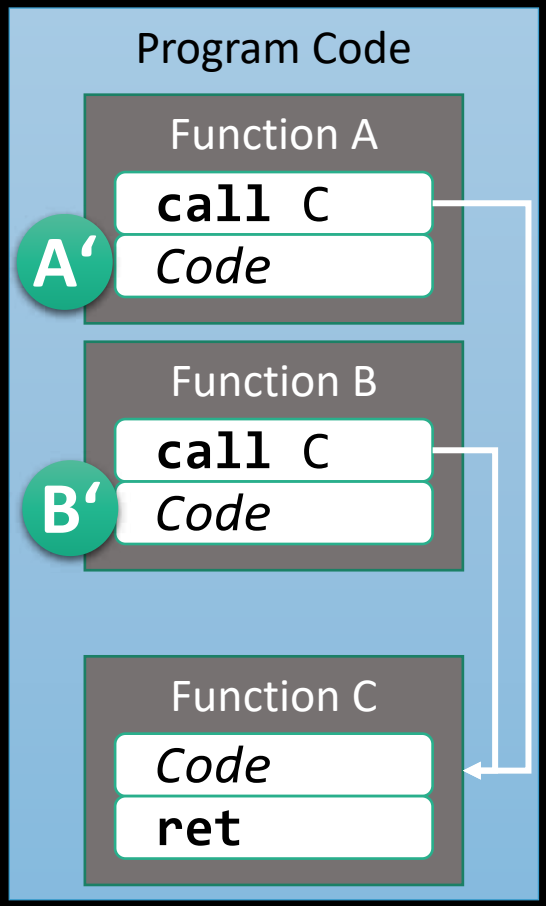
- Static CFI label checking leads to coarse-grained protection for returns

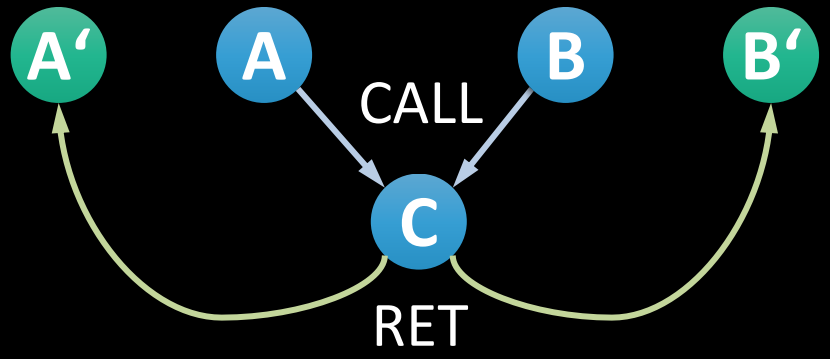




Label Problem for Returns

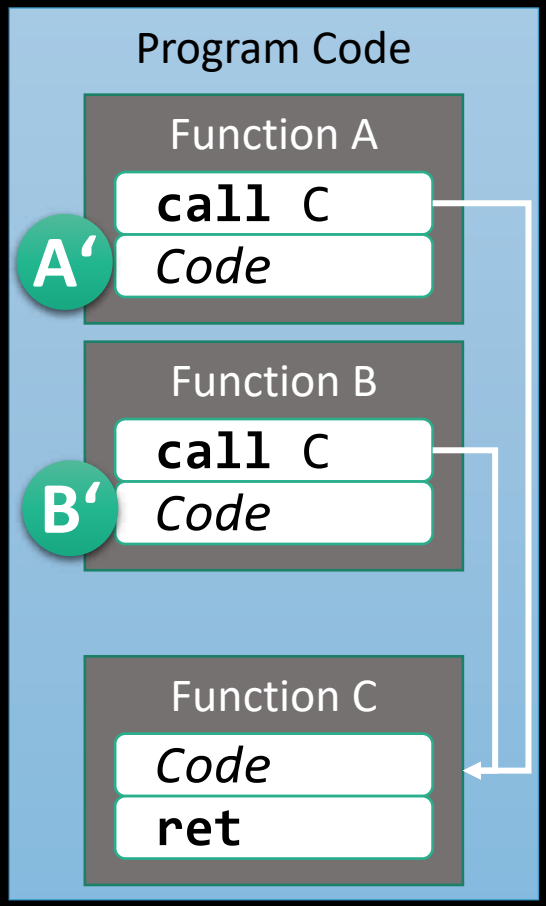
- Static CFI label checking leads to coarse-grained protection for returns

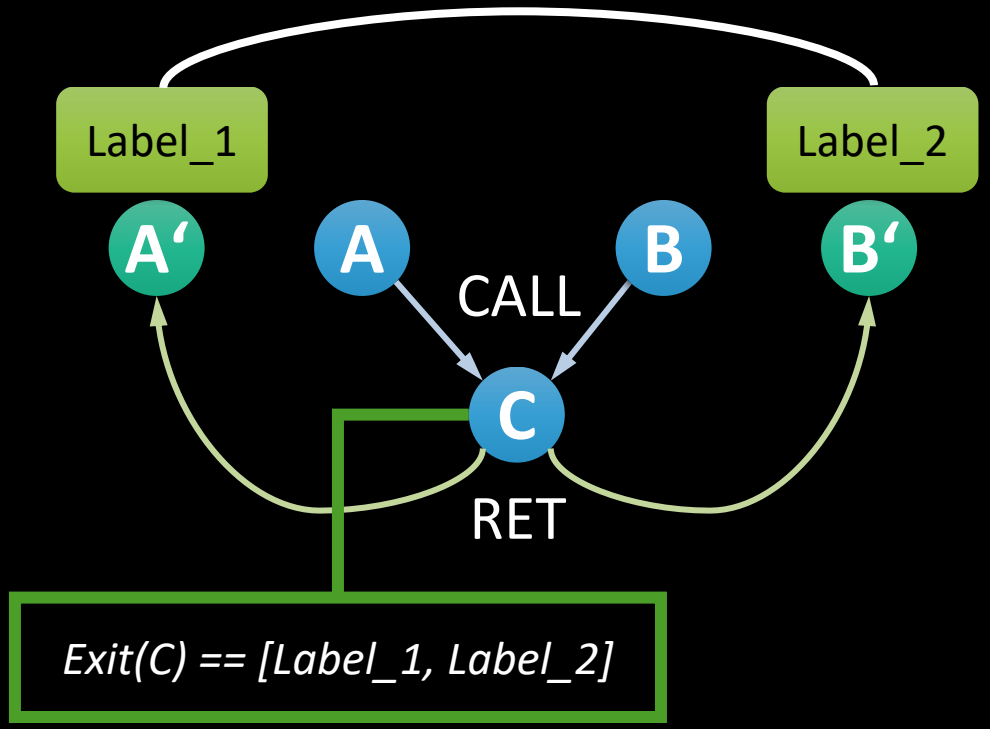




Label Problem for Returns

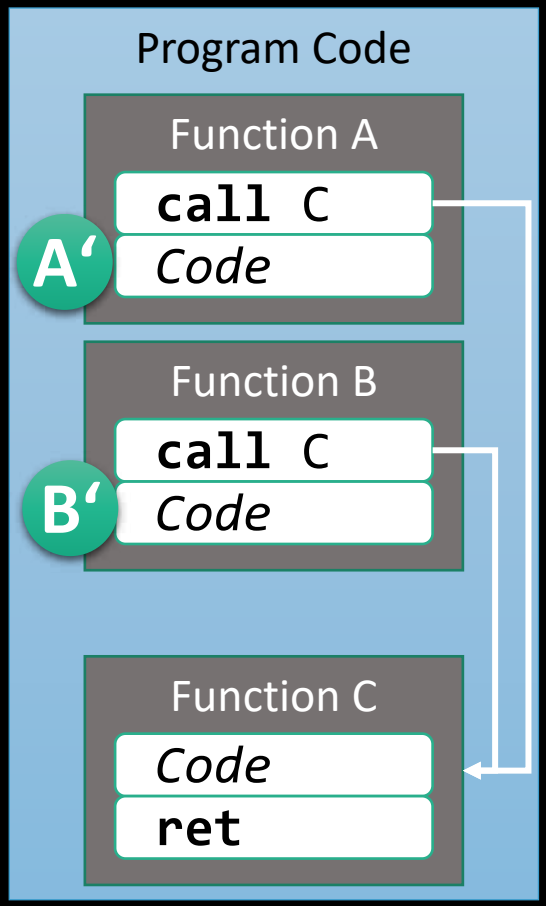
- Static CFI label checking leads to coarse-grained protection for returns

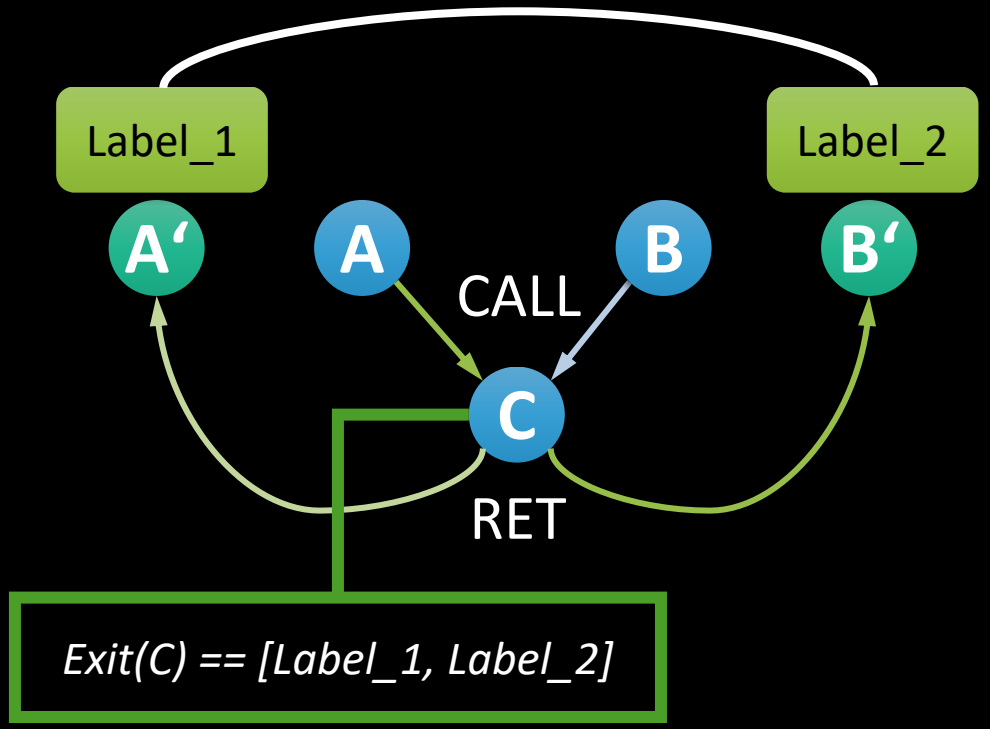




Label Problem for Returns

- Static CFI label checking leads to coarse-grained protection for returns





Shadow Stack / Return Address Stack

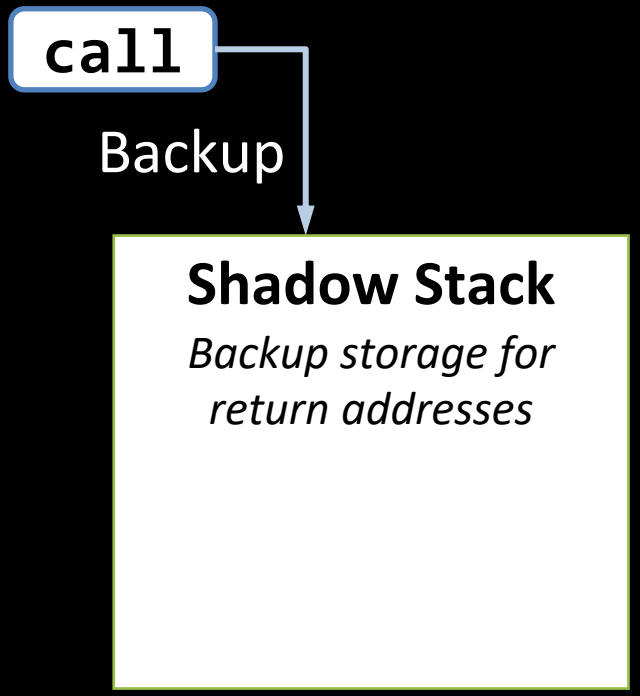
- Shadow stack allows for fine-grained return address protection but incurs higher overhead

Shadow Stack

Backup storage for return addresses

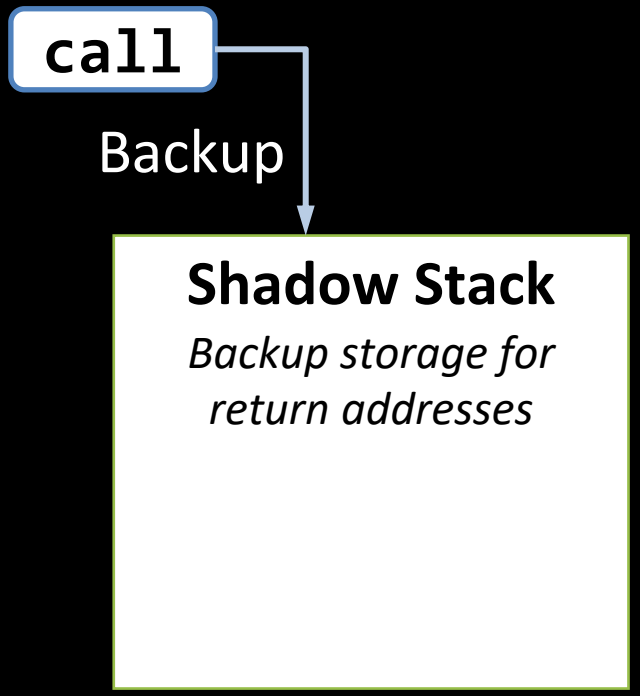
Shadow Stack / Return Address Stack

- Shadow stack allows for fine-grained return address protection but incurs higher overhead



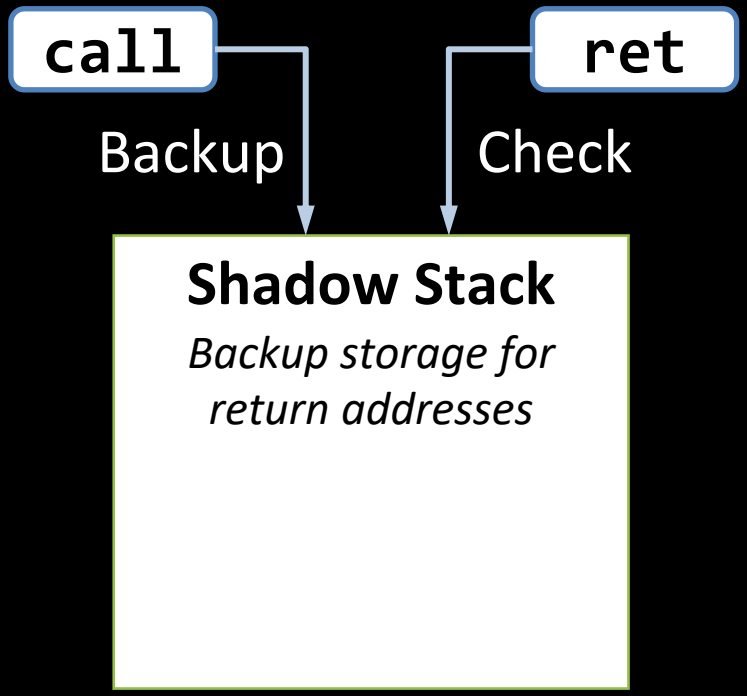
Shadow Stack / Return Address Stack

- Shadow stack allows for fine-grained return address protection but incurs higher overhead



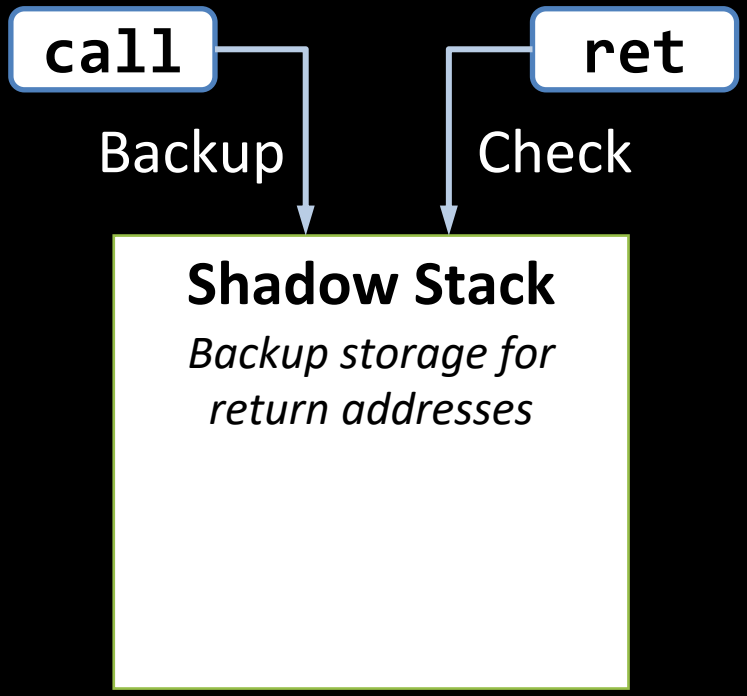
Shadow Stack / Return Address Stack

- Shadow stack allows for fine-grained return address protection but incurs higher overhead



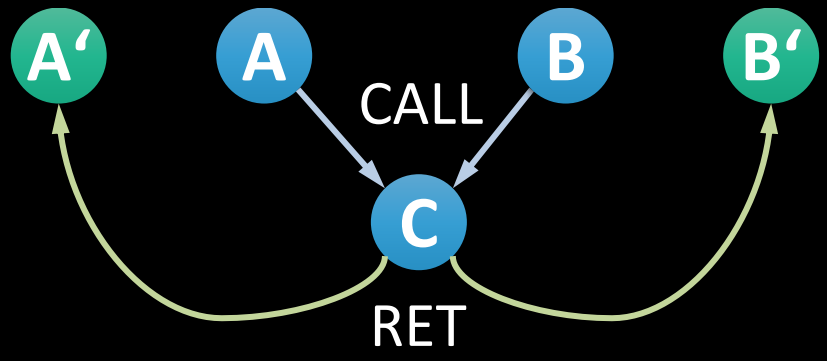
Shadow Stack / Return Address Stack

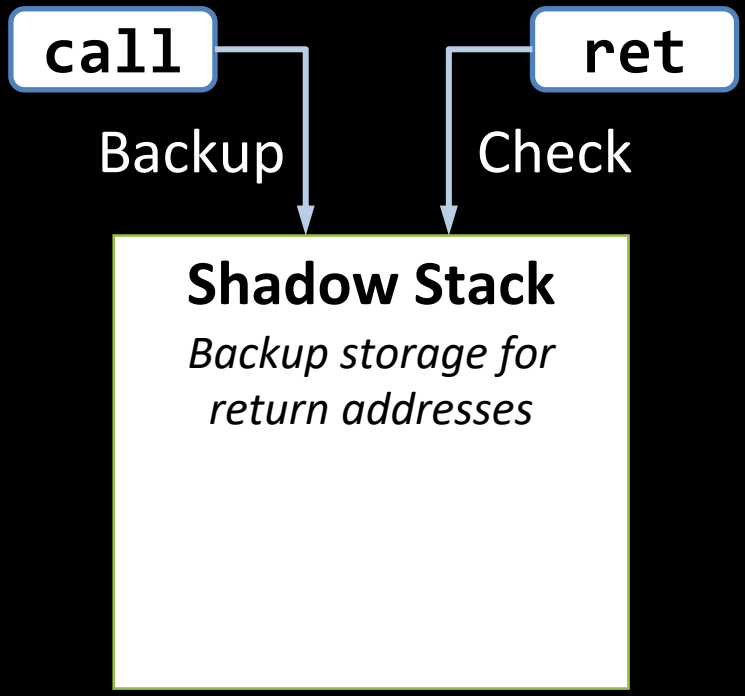
- Shadow stack allows for fine-grained return address protection but incurs higher overhead



Shadow Stack / Return Address Stack

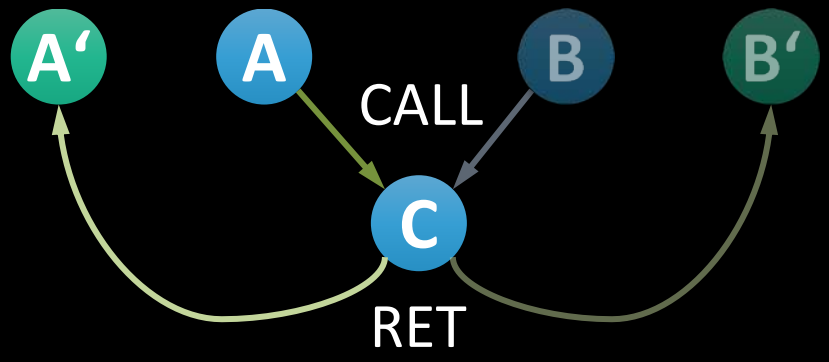
- Shadow stack allows for fine-grained return address protection but incurs higher overhead

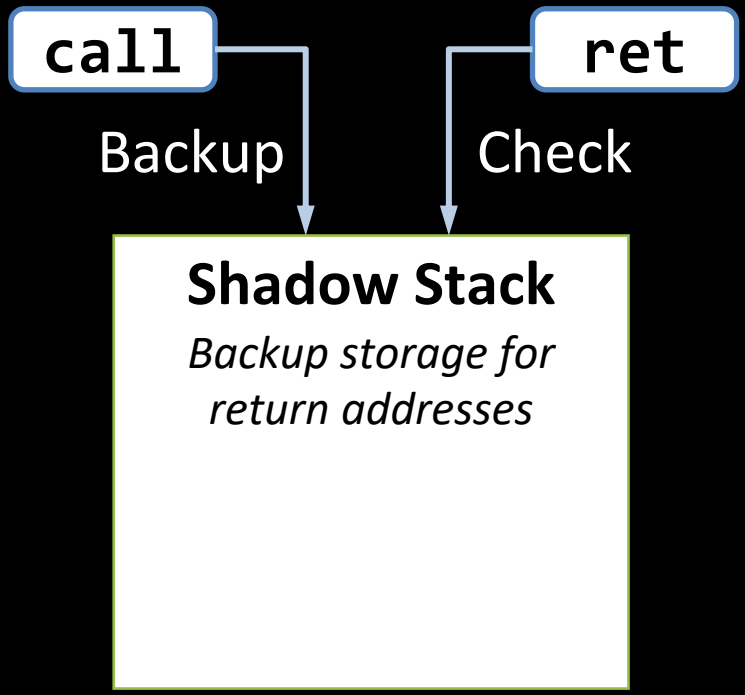




Shadow Stack / Return Address Stack

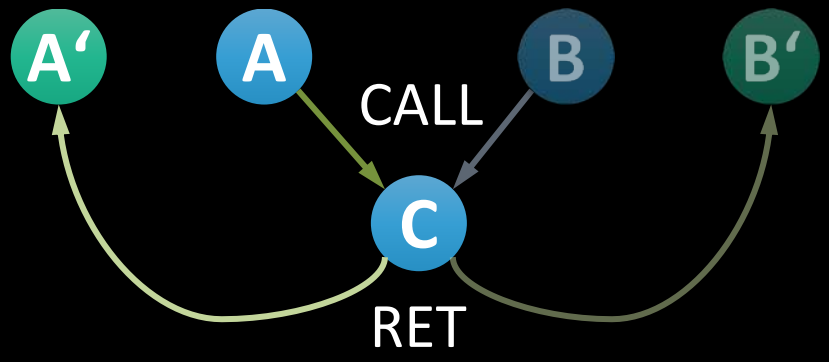
- Shadow stack allows for fine-grained return address protection but incurs higher overhead

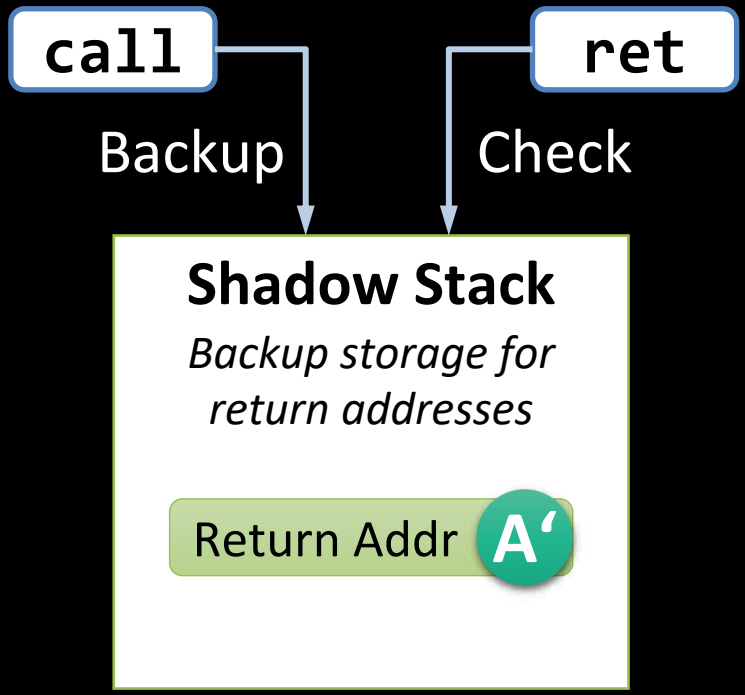




Shadow Stack / Return Address Stack

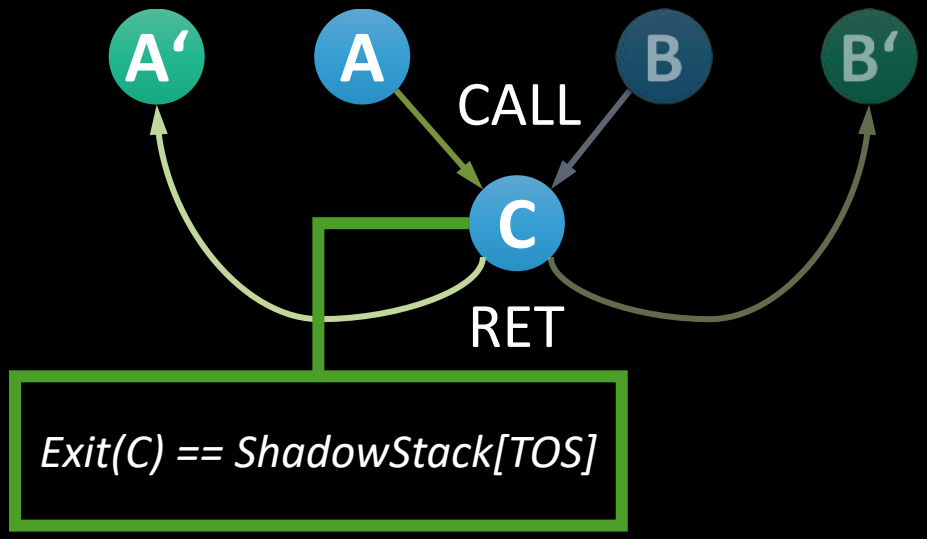
- Shadow stack allows for fine-grained return address protection but incurs higher overhead

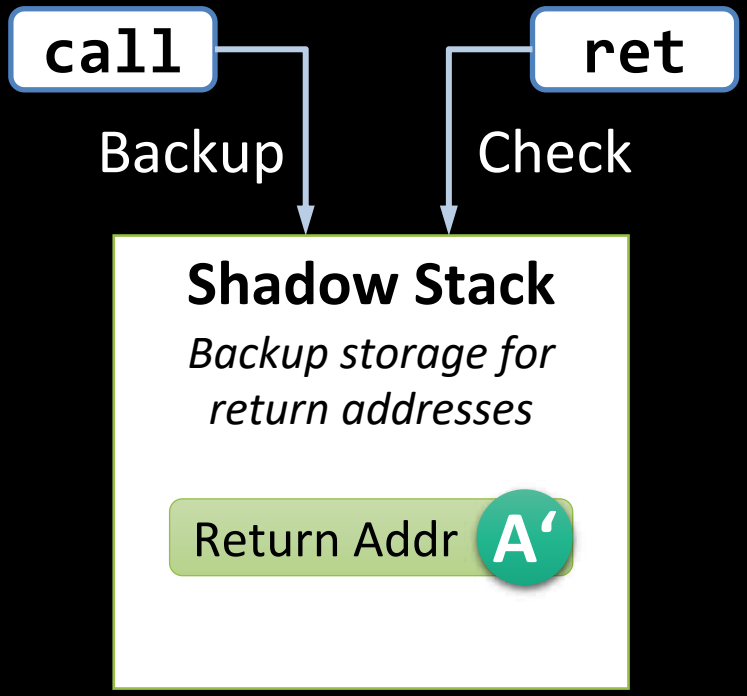




Shadow Stack / Return Address Stack

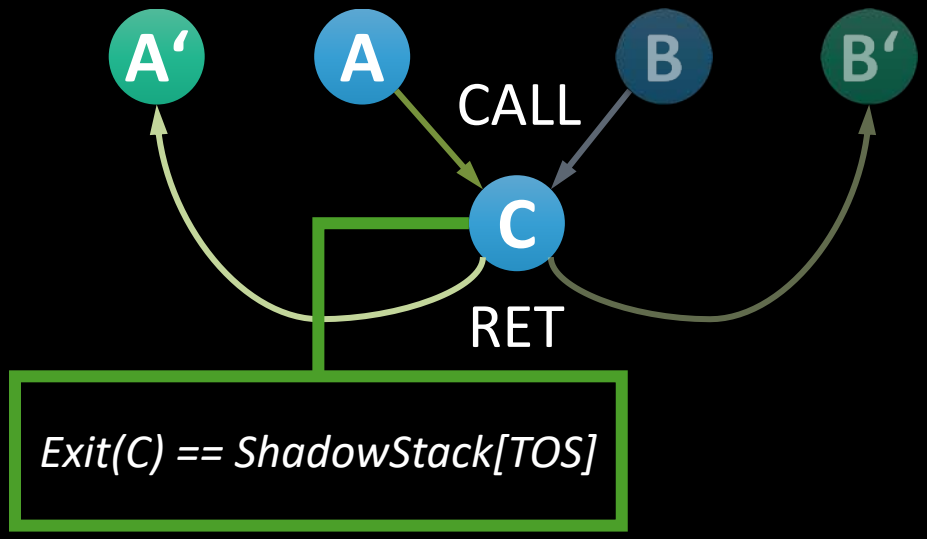
- Shadow stack allows for fine-grained return address protection but incurs higher overhead

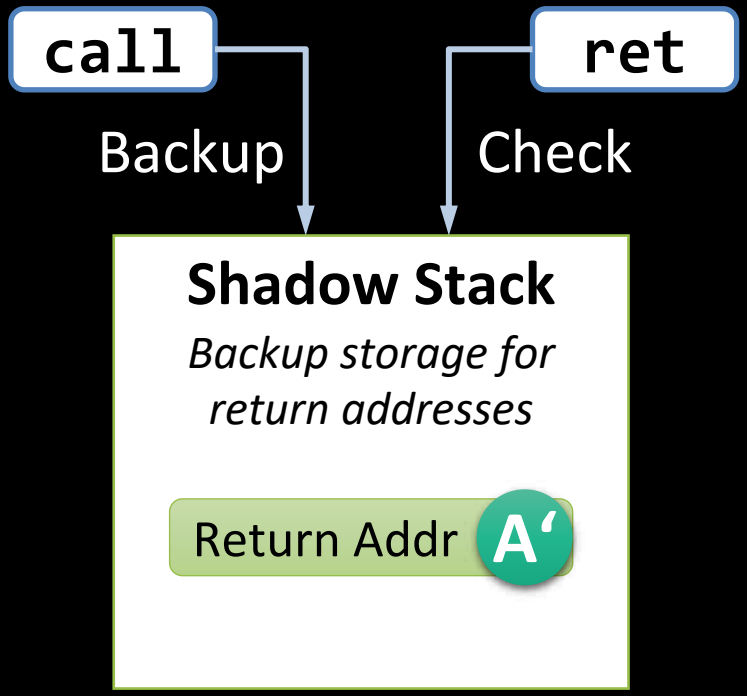




Shadow Stack / Return Address Stack

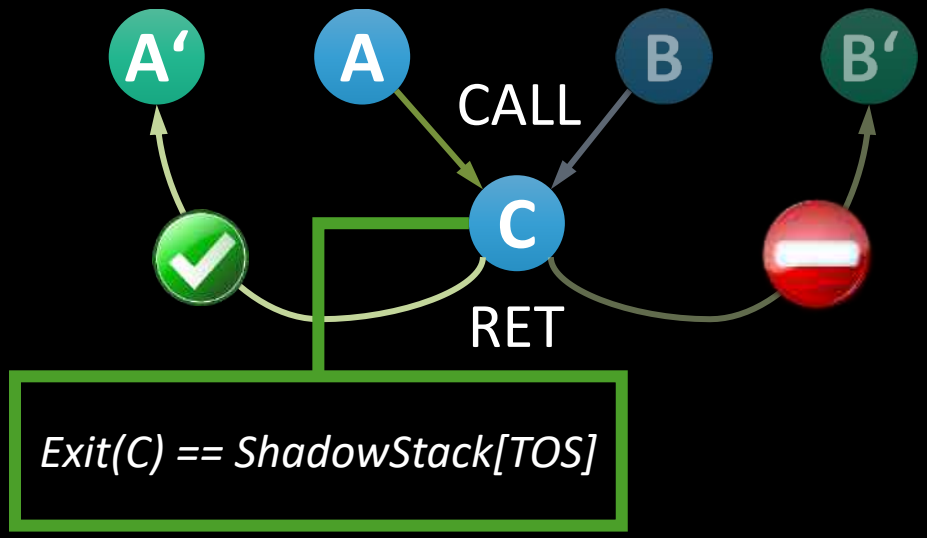
- Shadow stack allows for fine-grained return address protection but incurs higher overhead

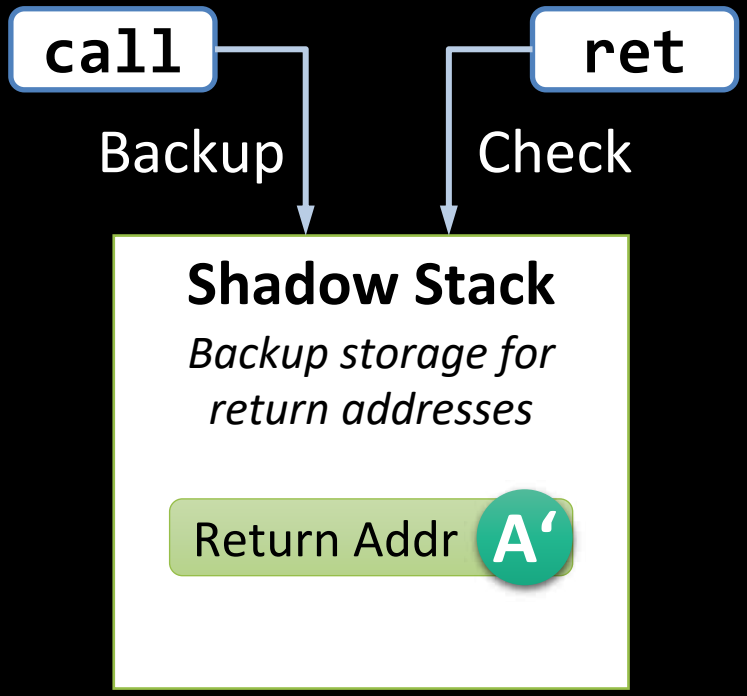




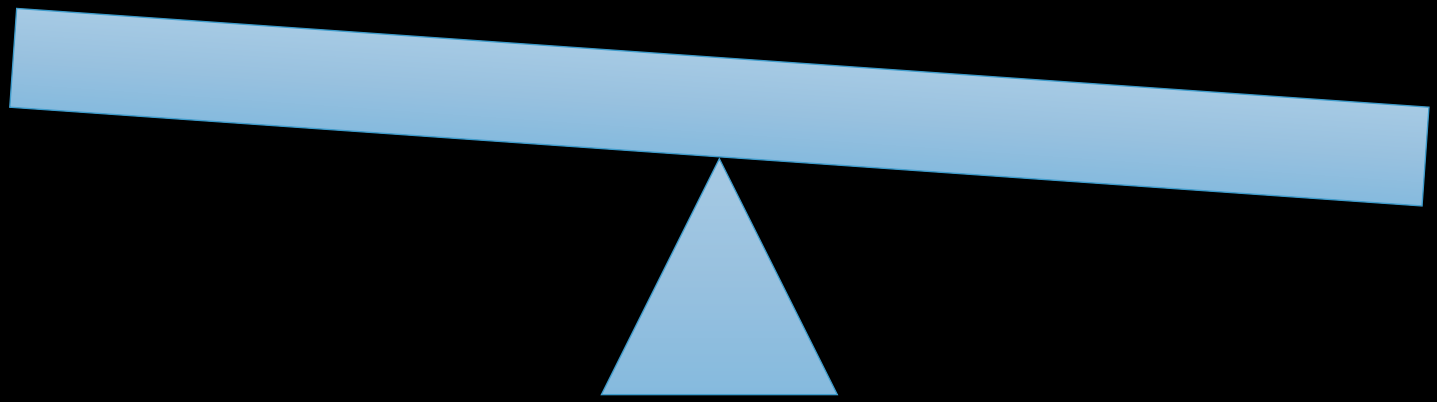
Shadow Stack / Return Address Stack

- Shadow stack allows for fine-grained return address protection but incurs higher overhead

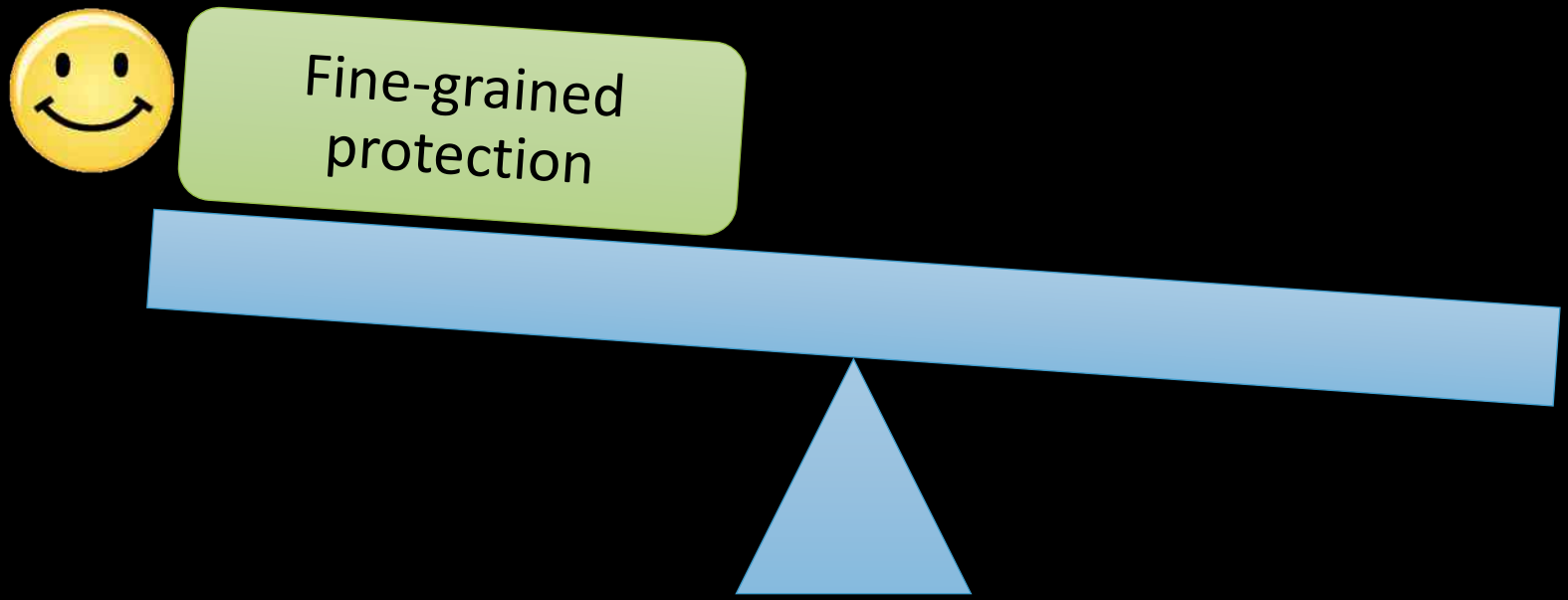




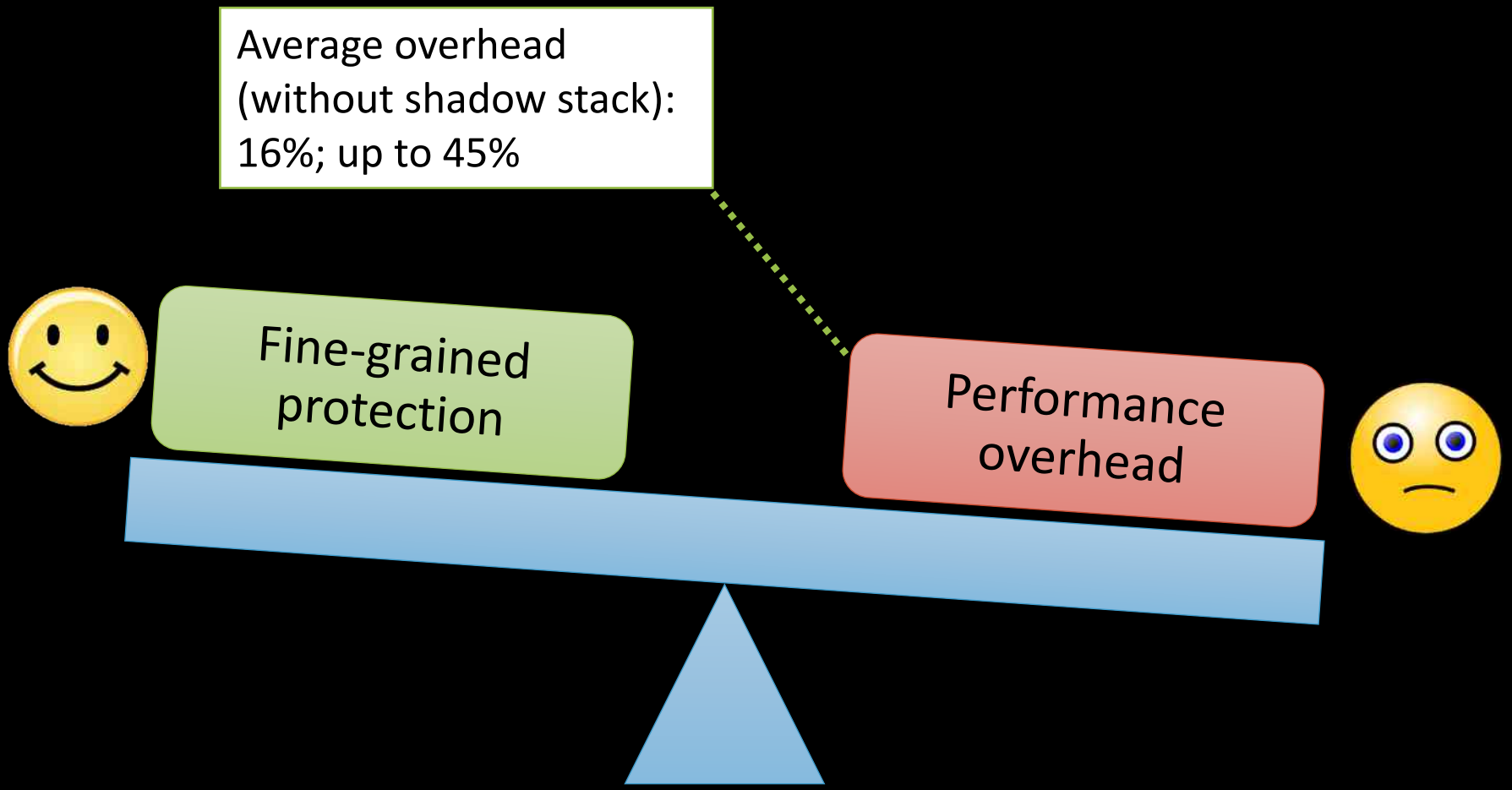
CFI: Benefits and Limitations



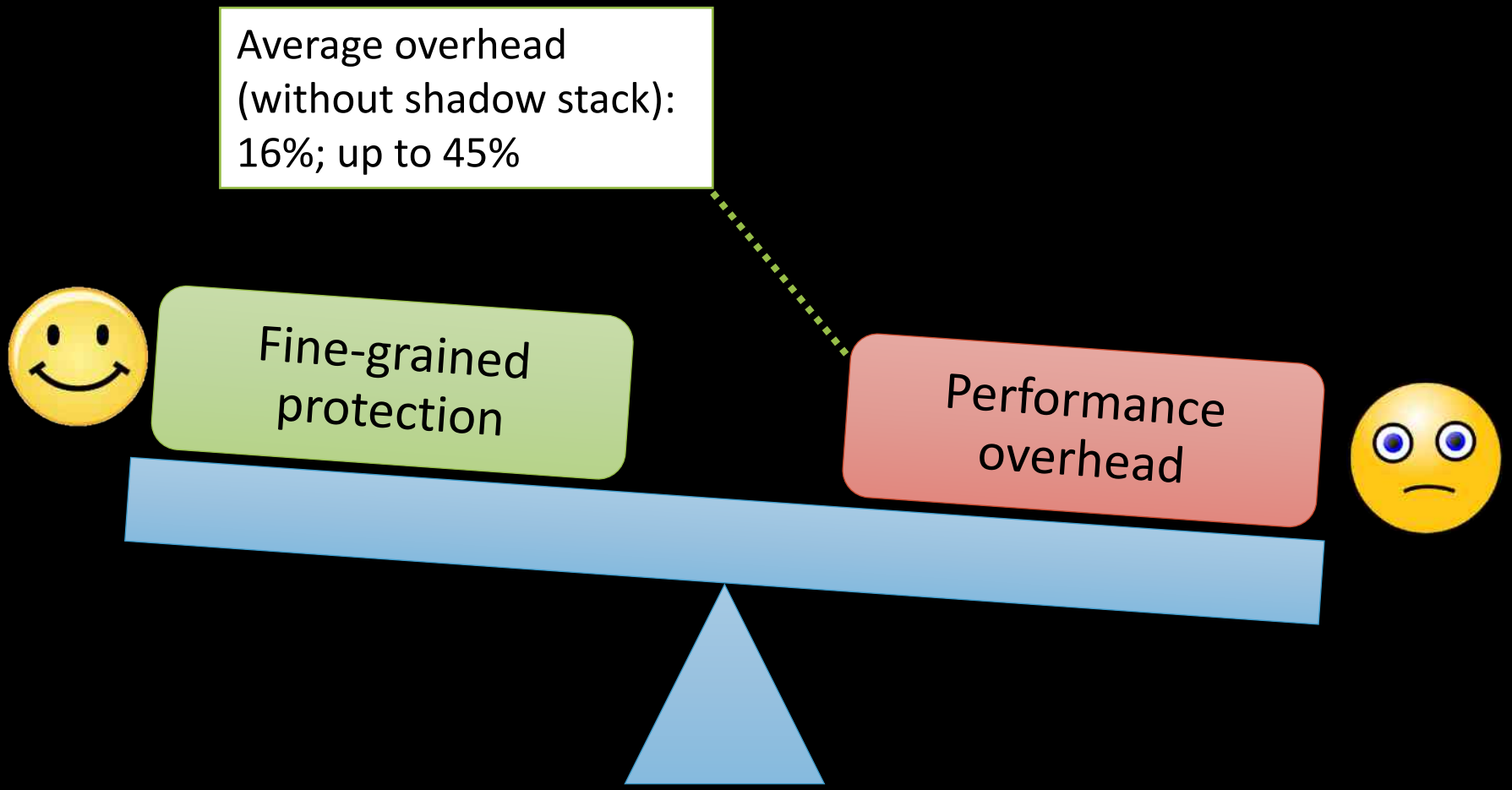
CFI: Benefits and Limitations



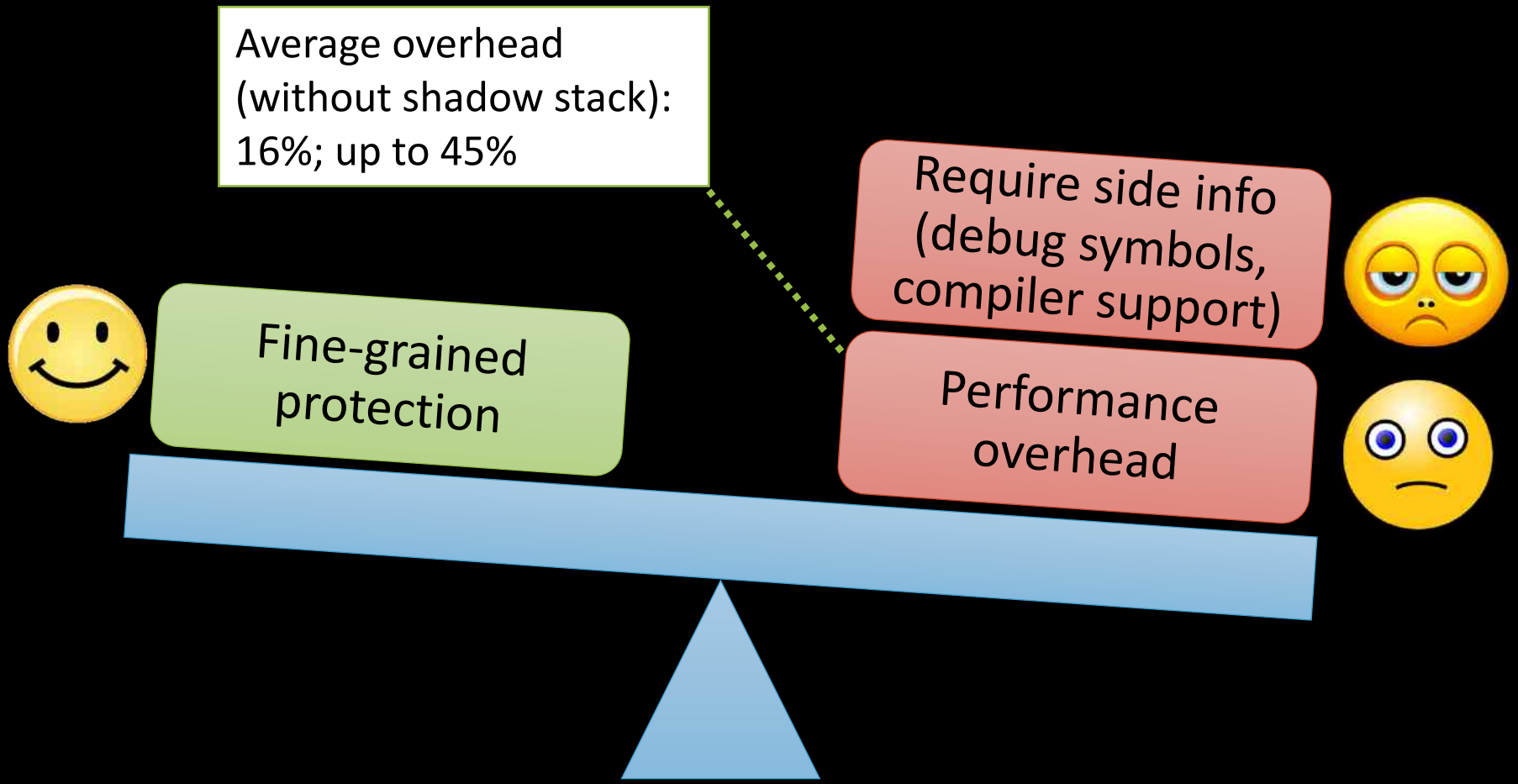
CFI: Benefits and Limitations



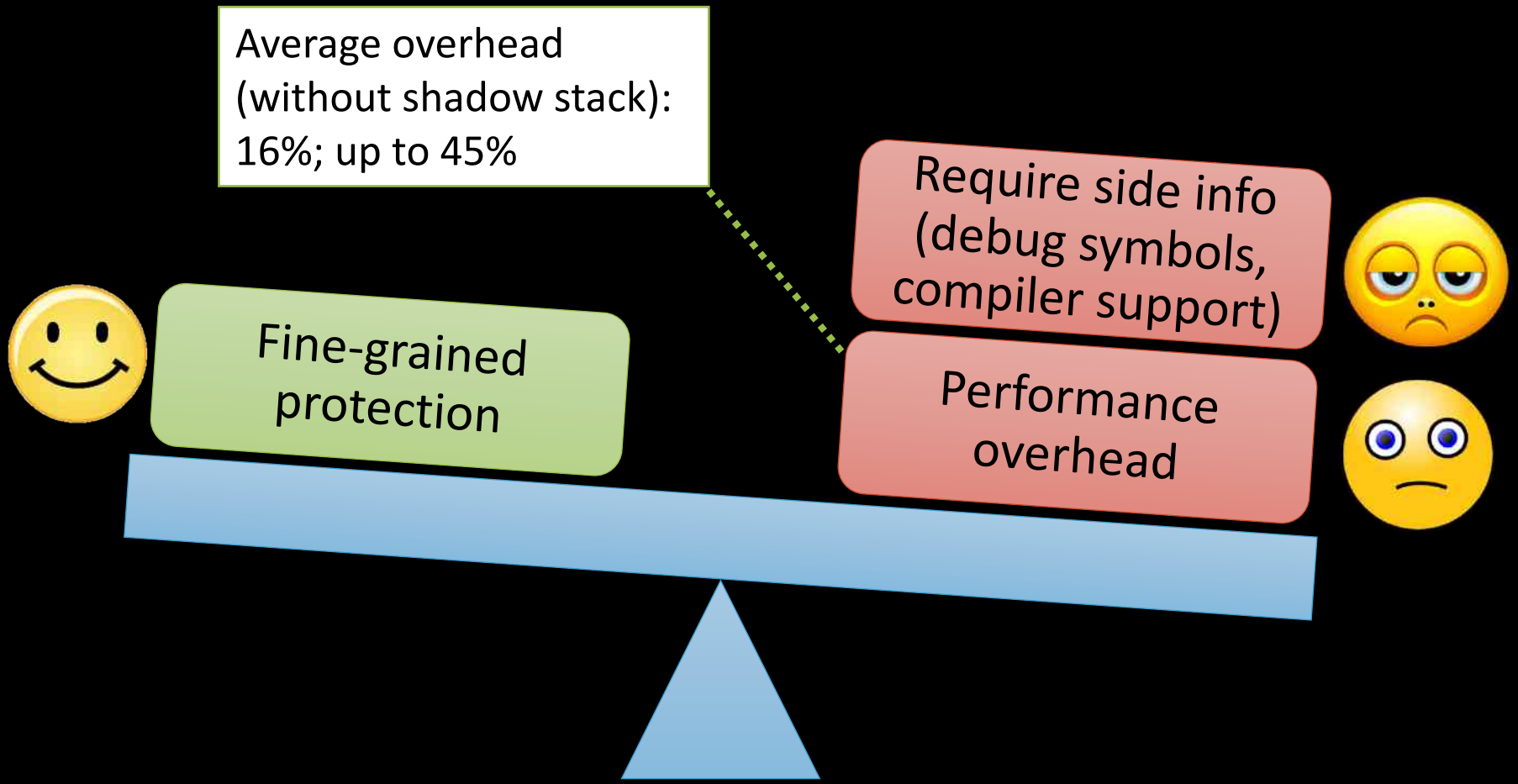
CFI: Benefits and Limitations



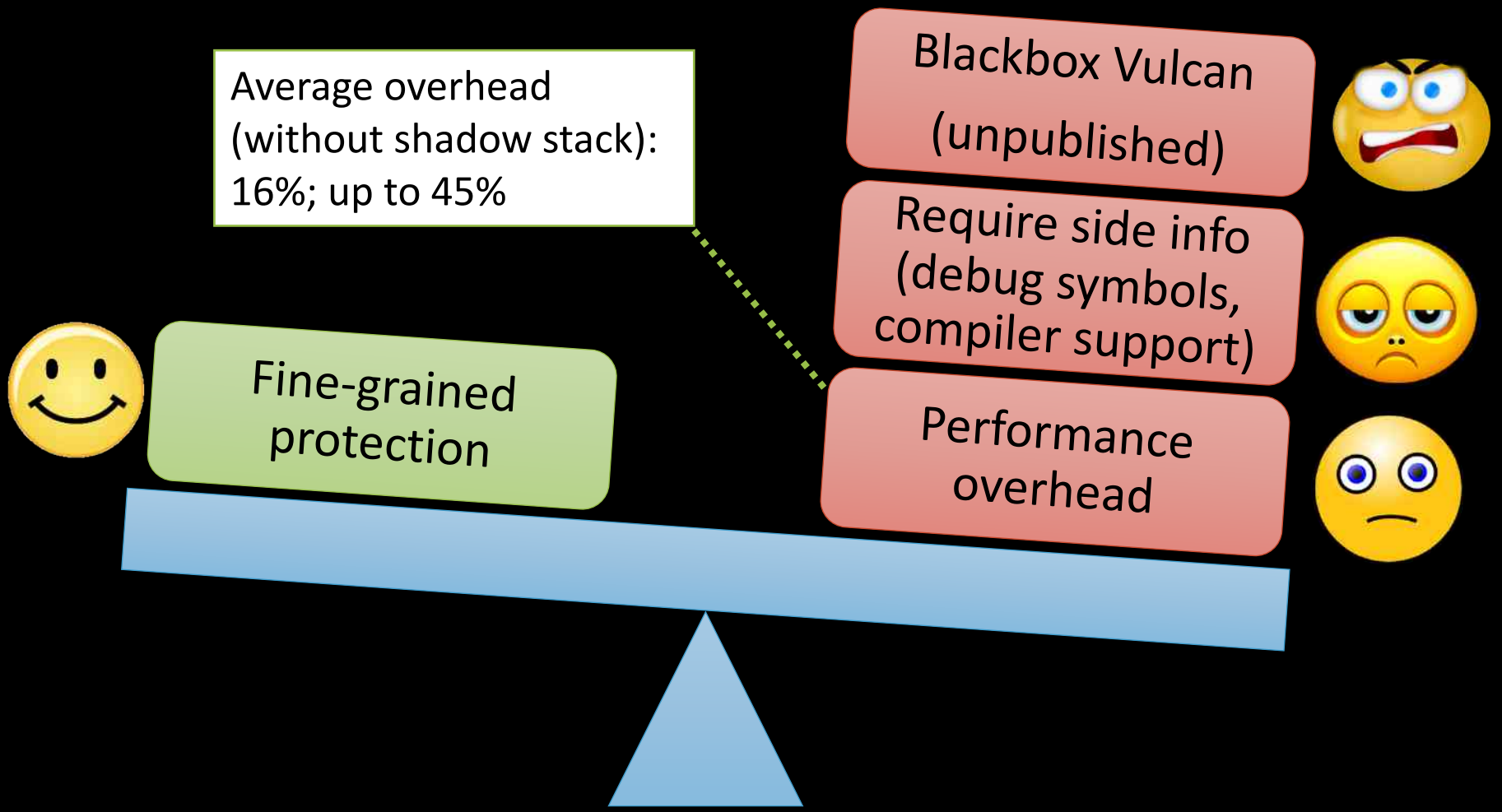
CFI: Benefits and Limitations



CFI: Benefits and Limitations



CFI: Benefits and Limitations



Hot Research Topic:

"Practical" (coarse-grained)

Control Flow Integrity (CFI)

Hot Research Topic:

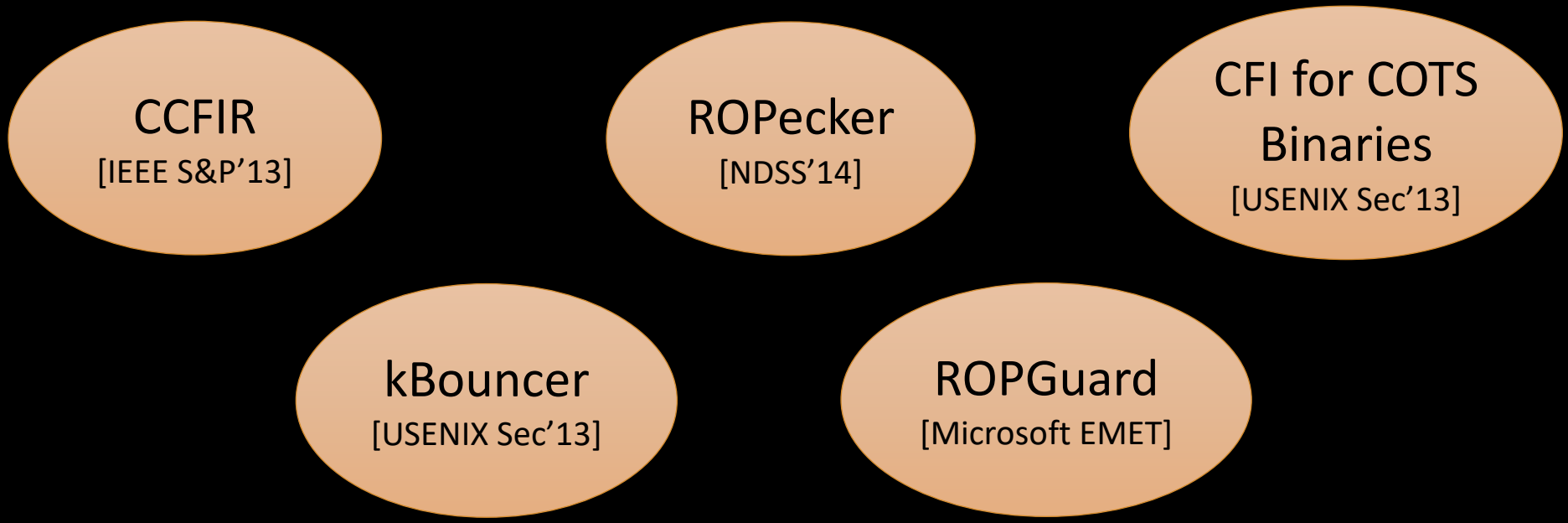
"Practical" (coarse-grained)

Control Flow Integrity (CFI)

Hot Research Topic:

"Practical" (coarse-grained) Control Flow Integrity (CFI)

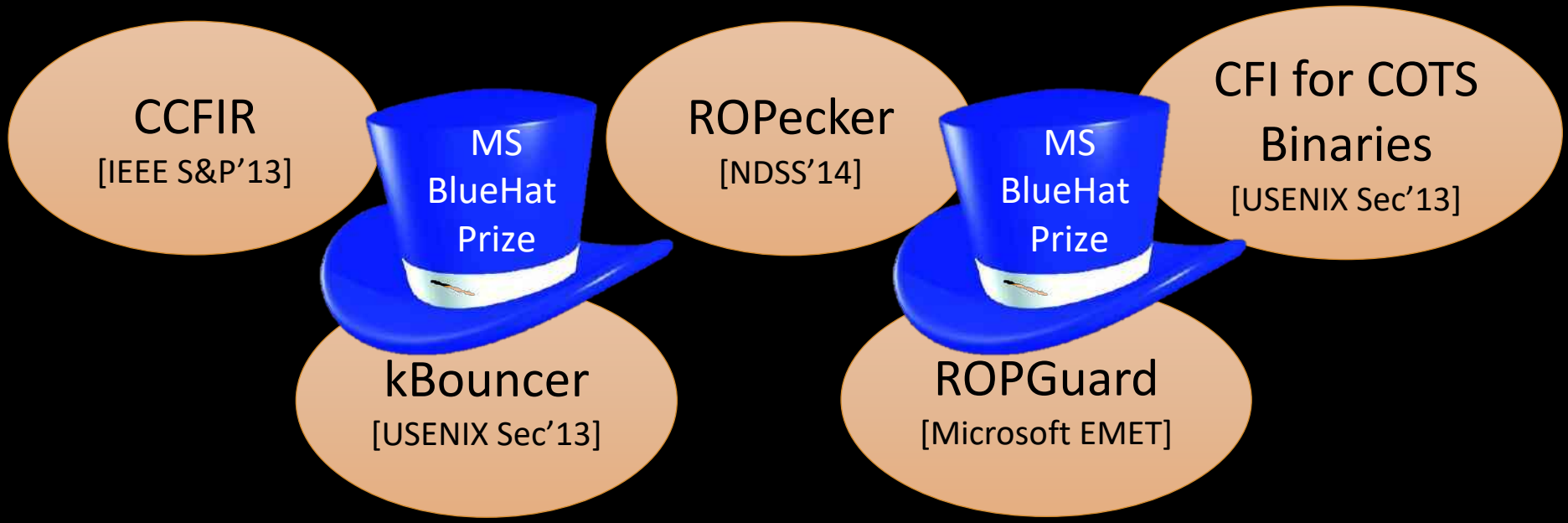
Recently, many solutions proposed



Hot Research Topic:

"Practical" (coarse-grained) Control Flow Integrity (CFI)

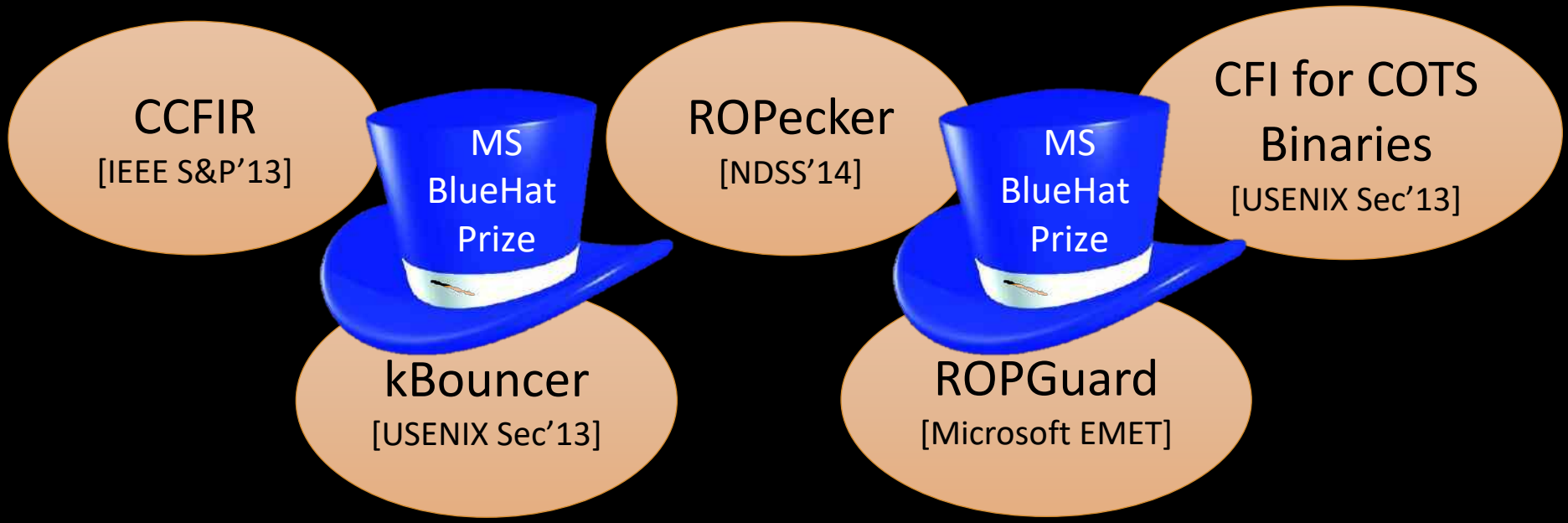
Recently, many solutions proposed



Hot Research Topic:

"Practical" (coarse-grained) Control Flow Integrity (CFI)

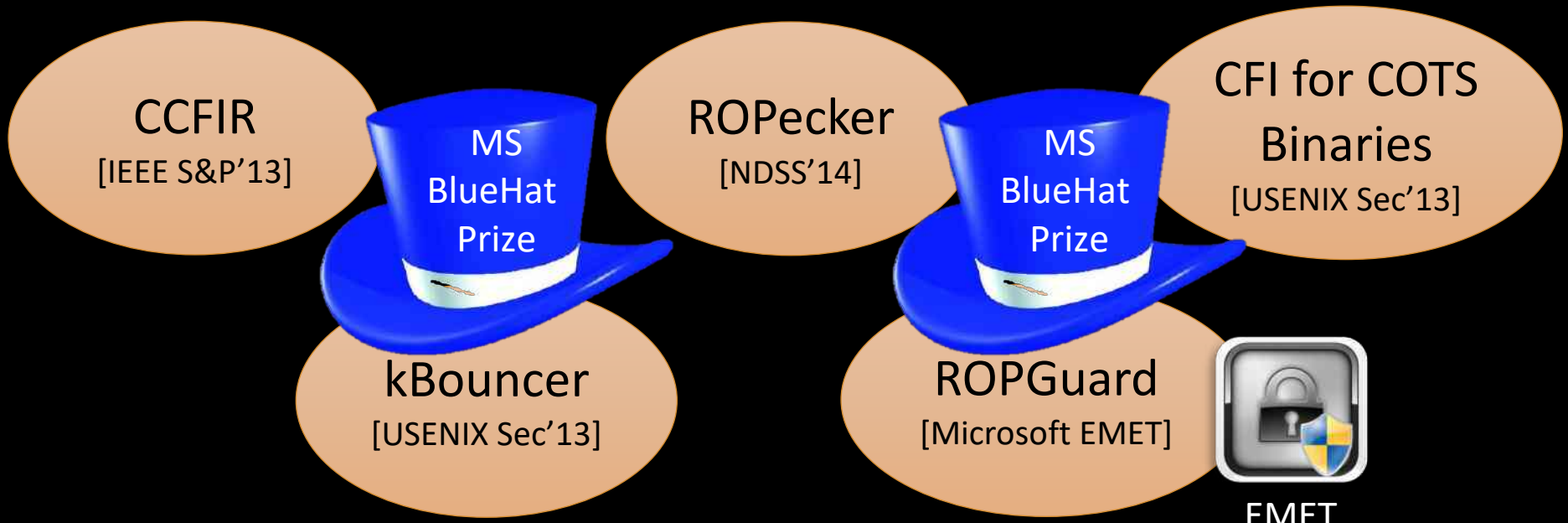
Recently, many solutions proposed



Hot Research Topic:

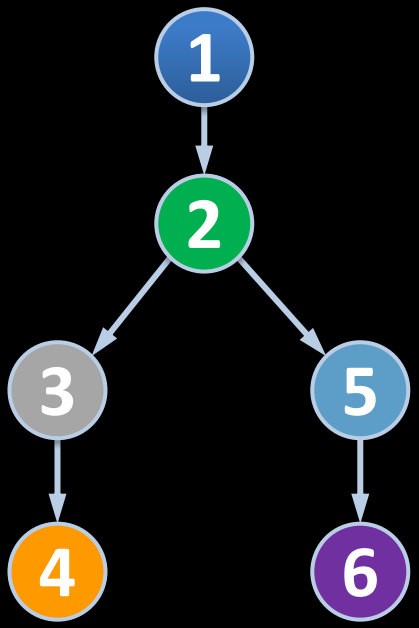
"Practical" (coarse-grained) Control Flow Integrity (CFI)

Recently, many solutions proposed

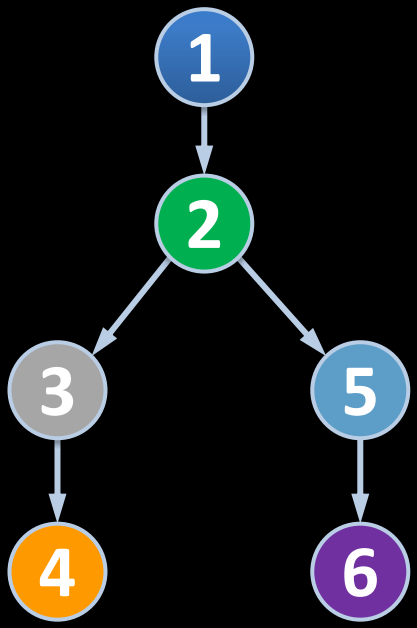


http://technet.microsoft.com/en-us/security/jj653751

Coarse-grained CFI leads to CFG imprecision

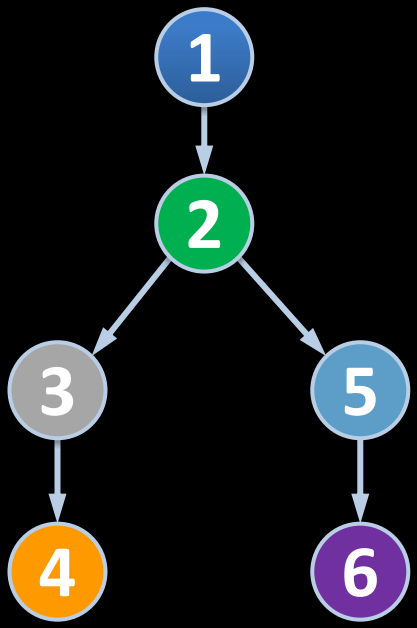


Coarse-grained CFI leads to CFG imprecision



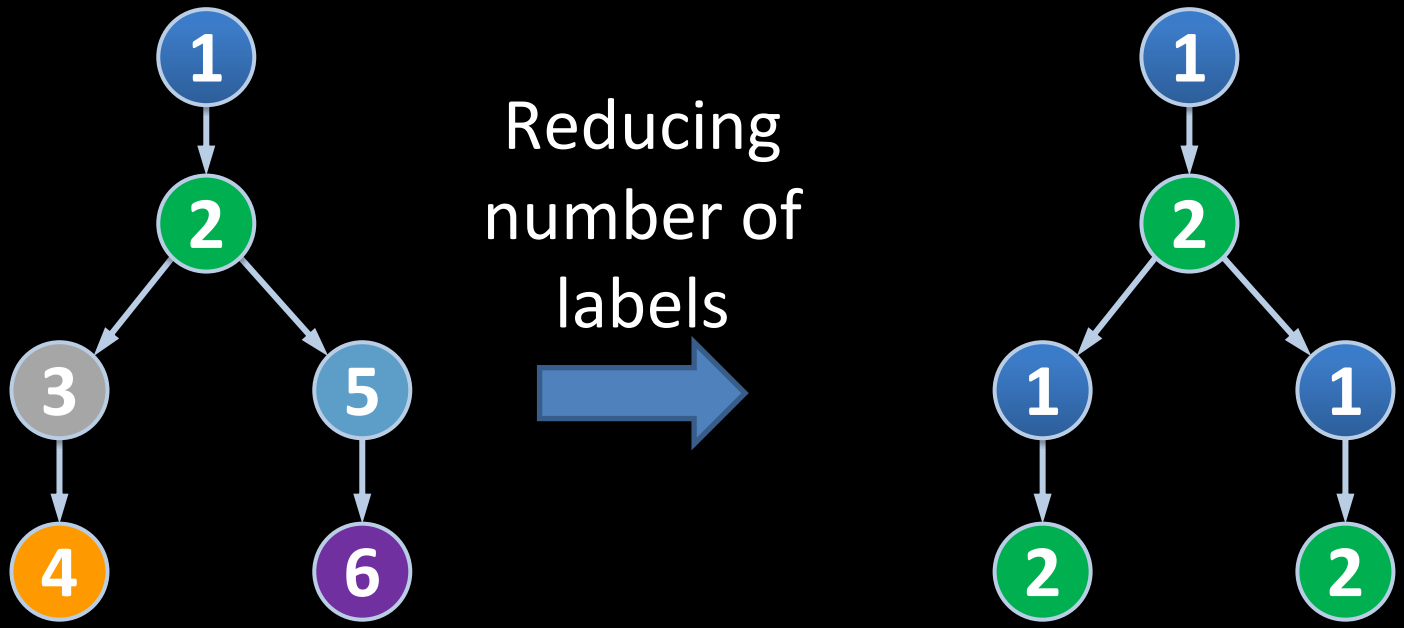
Reducing number of labels

Coarse-grained CFI leads to CFG imprecision



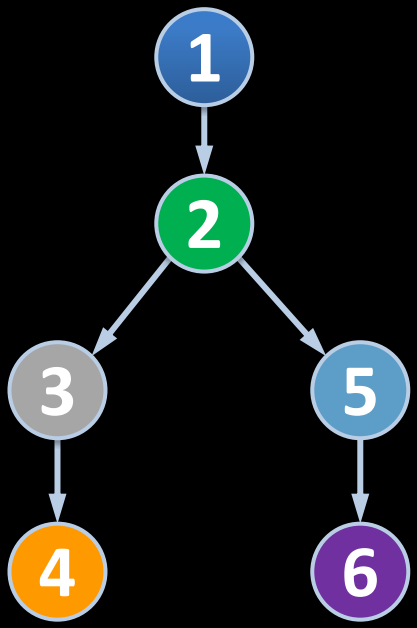
Reducing number of labels

Coarse-grained CFI leads to CFG imprecision

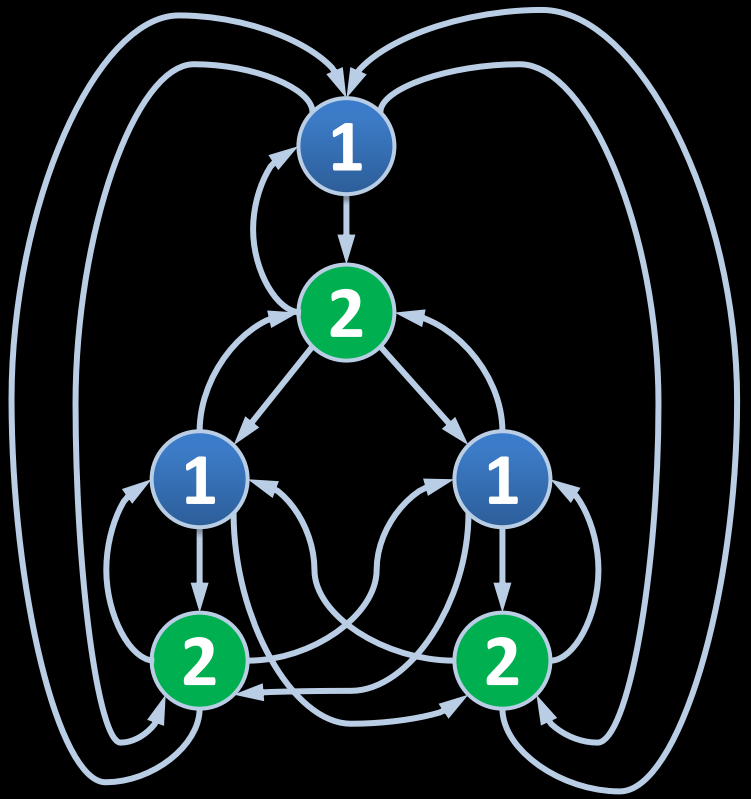


Coarse-grained CFI leads to CFG imprecision

Allowed paths: 1→2 and 2→1

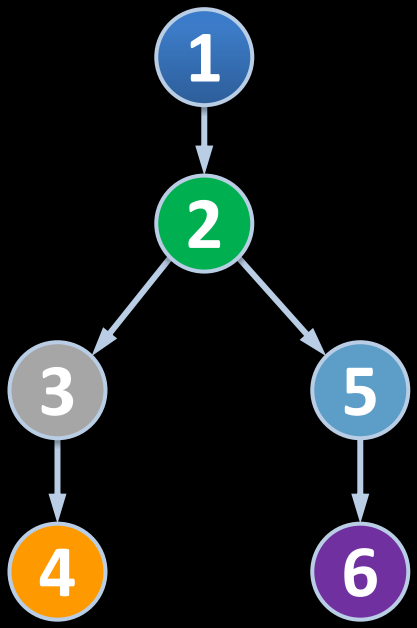


Reducing number of labels

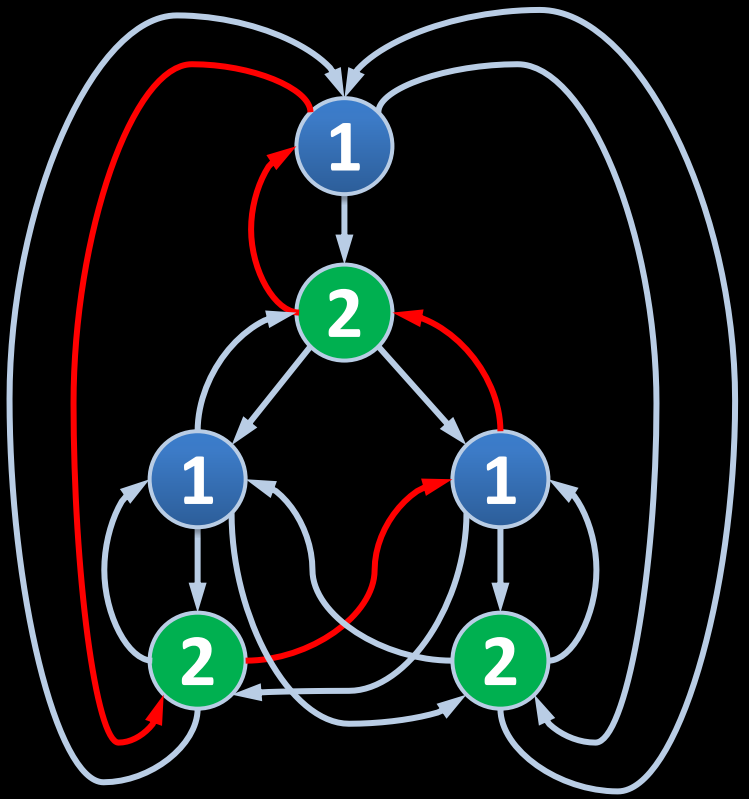


Coarse-grained CFI leads to CFG imprecision

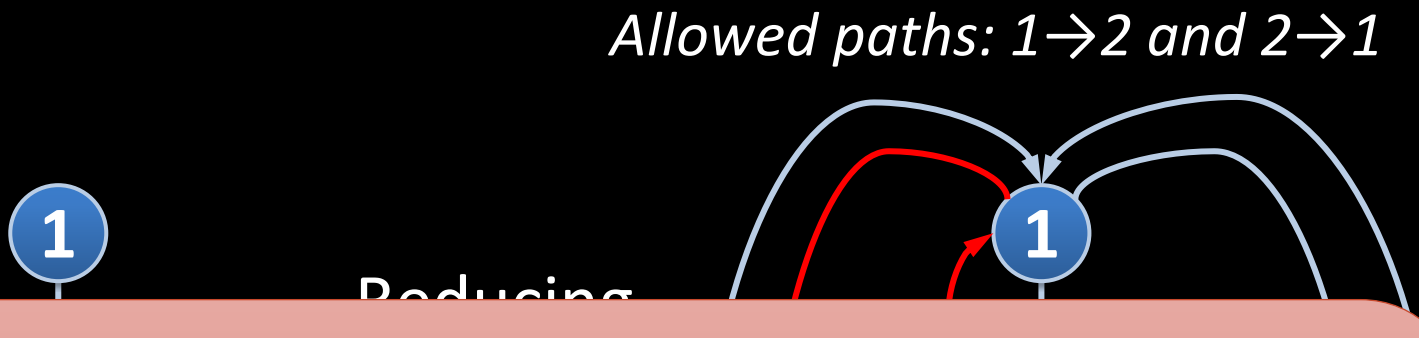
Allowed paths: 1→2 and 2→1



Reducing number of labels



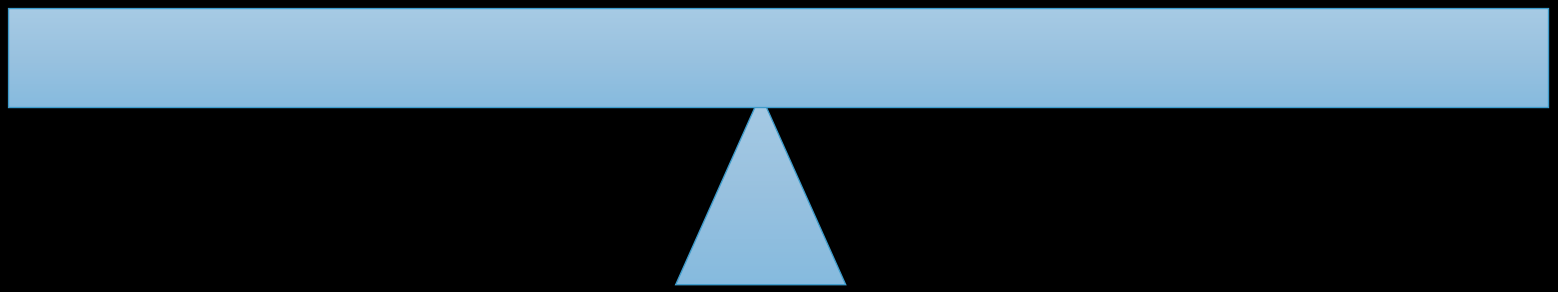
Coarse-grained CFI leads to CFG imprecision



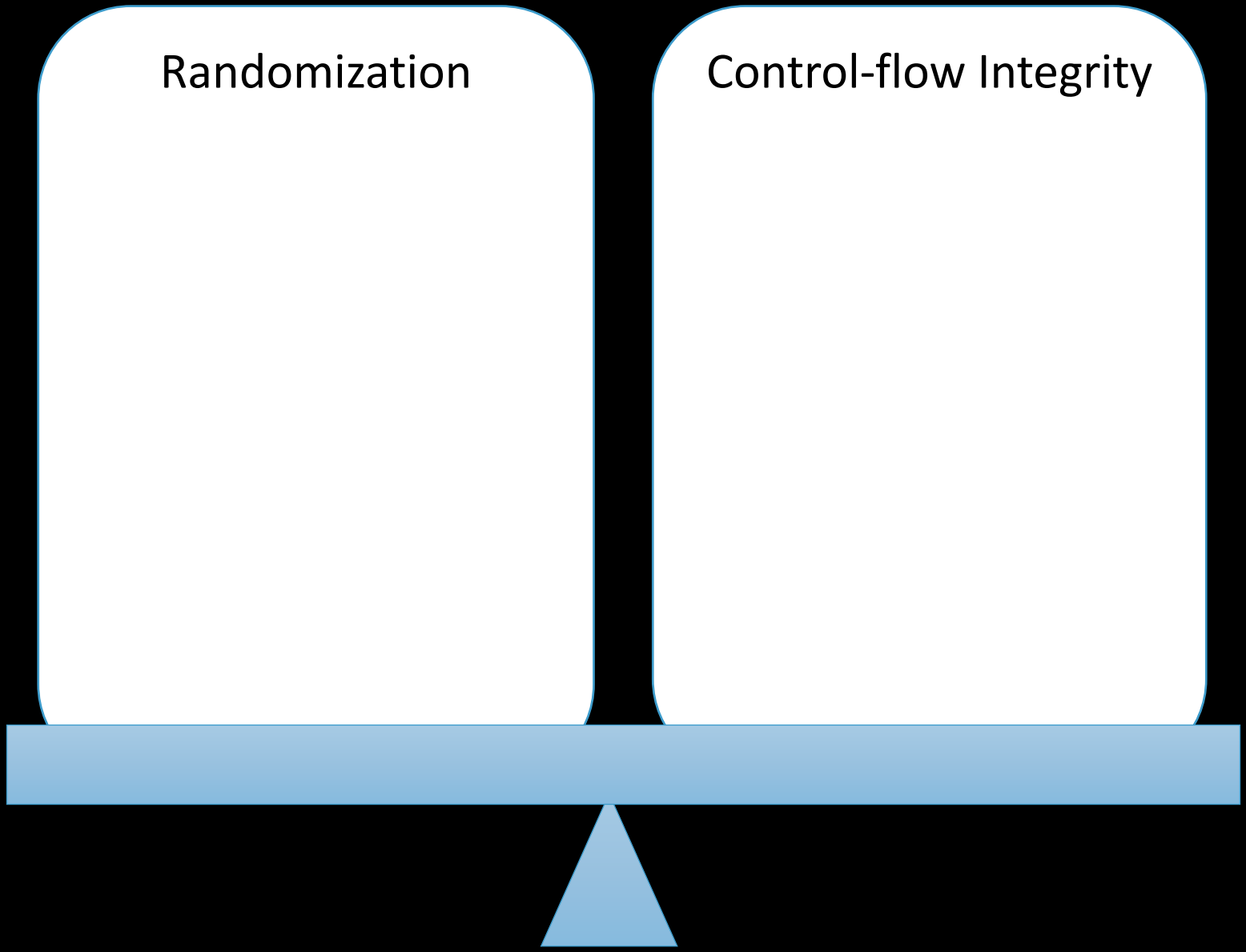
Many attacks have been proposed that exploit the coarse-grained checks of these CFI schemes

Out-of-Control [Goktas et al., IEEE S&P 2014]
Stitching the Gadgets [Davi et al., USENIX Sec. 2014]
ROP is still dangerous [Carlini and Wagner, USENIX Sec. 2014]

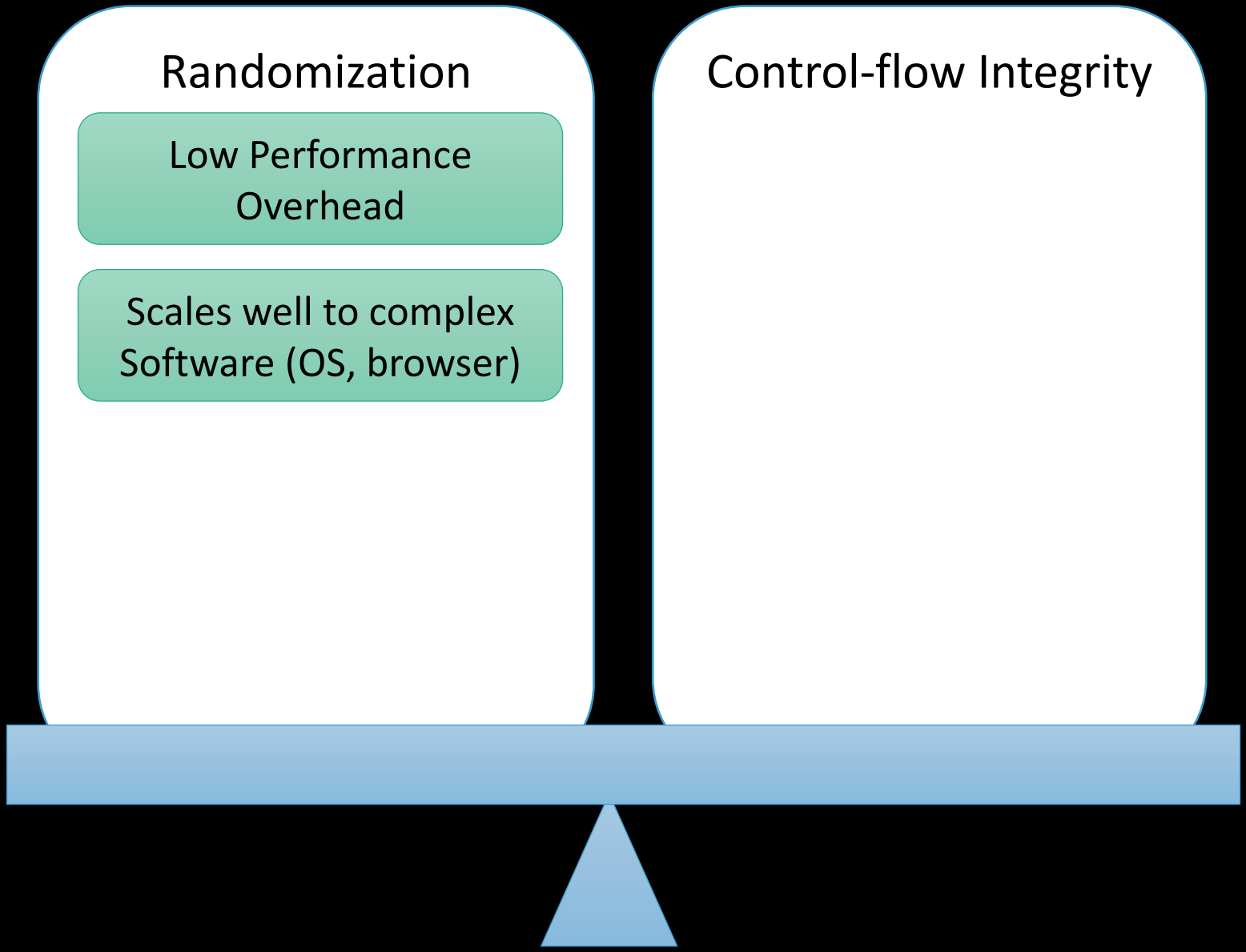
Randomization vs. CFI



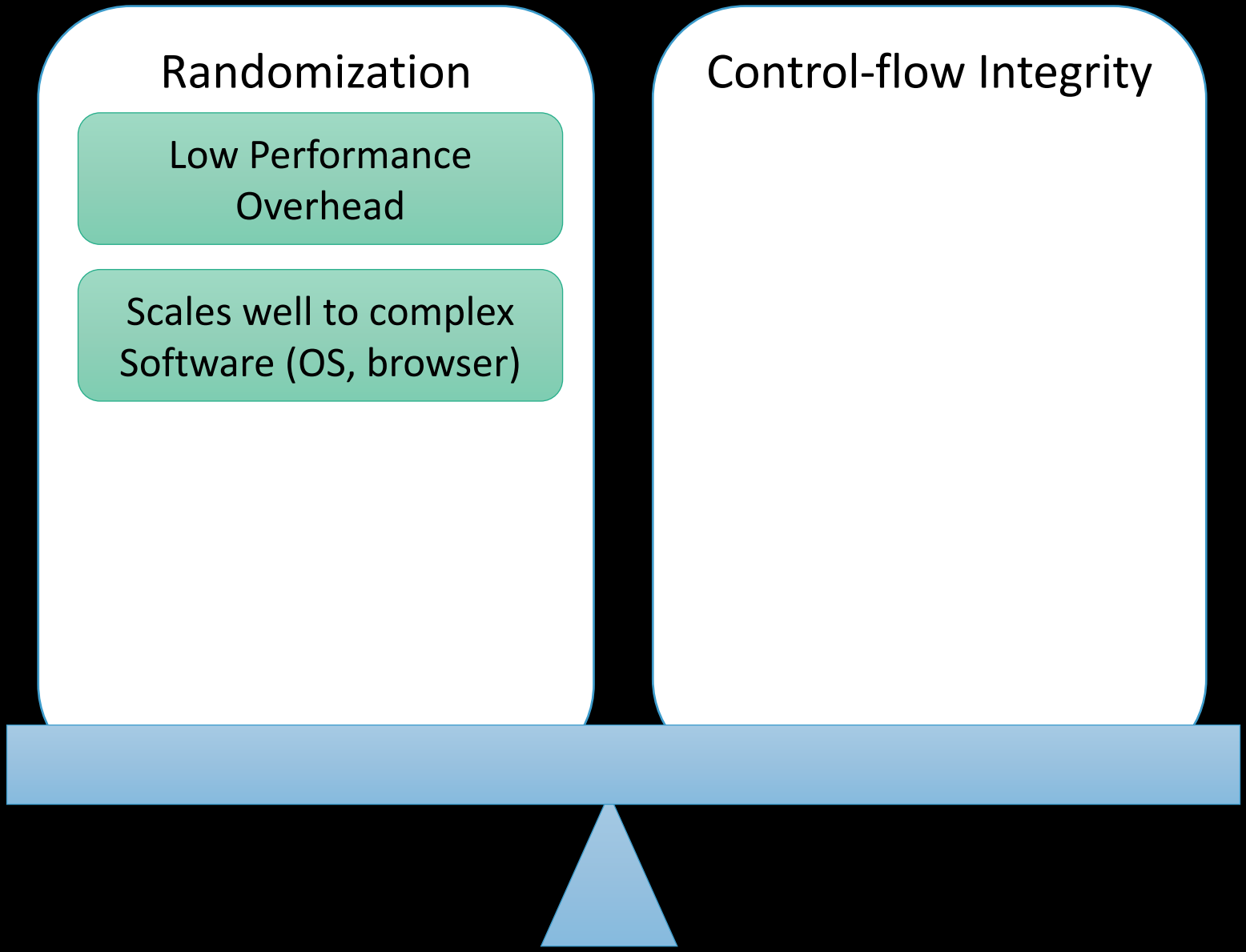
Randomization vs. CFI



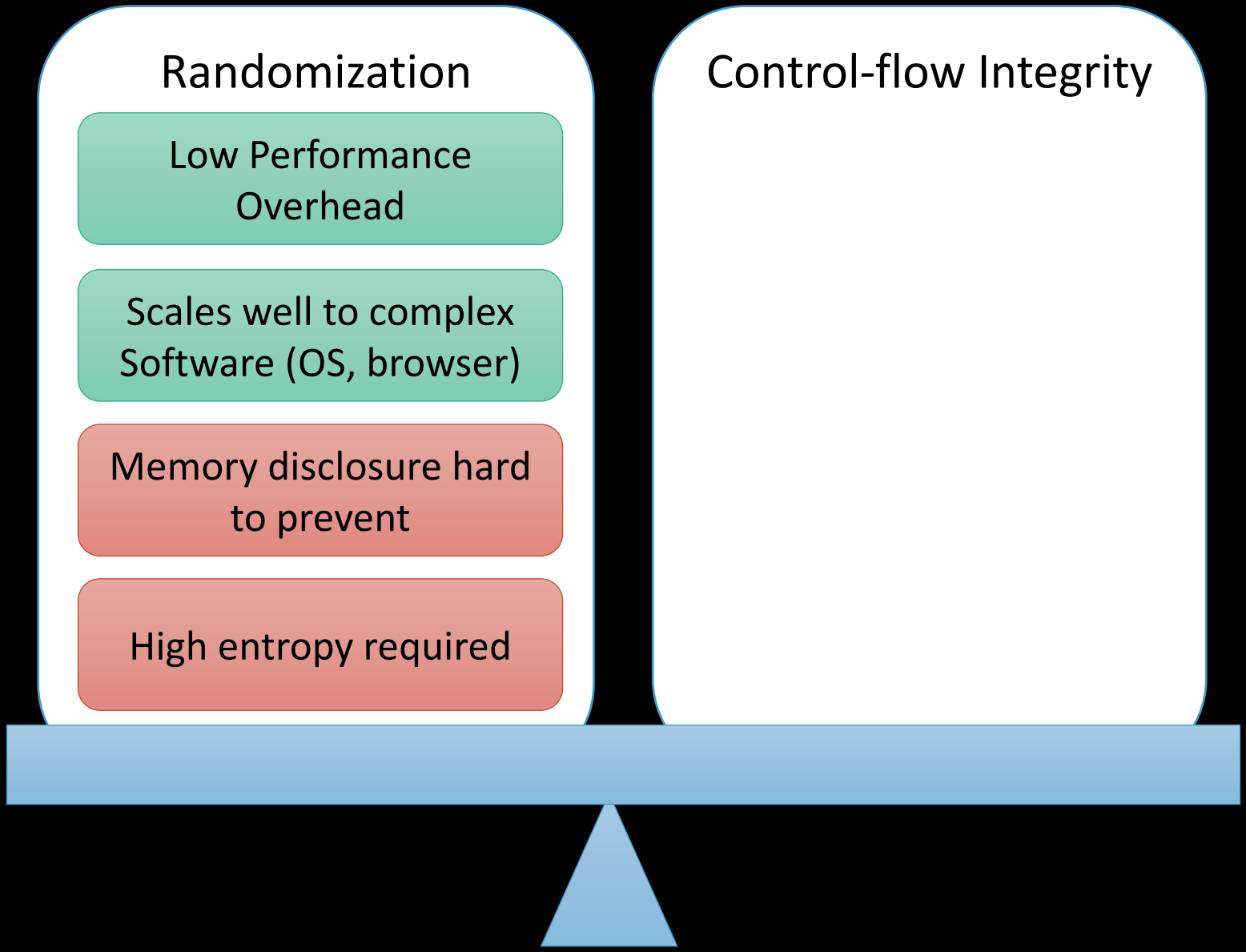
Randomization vs. CFI



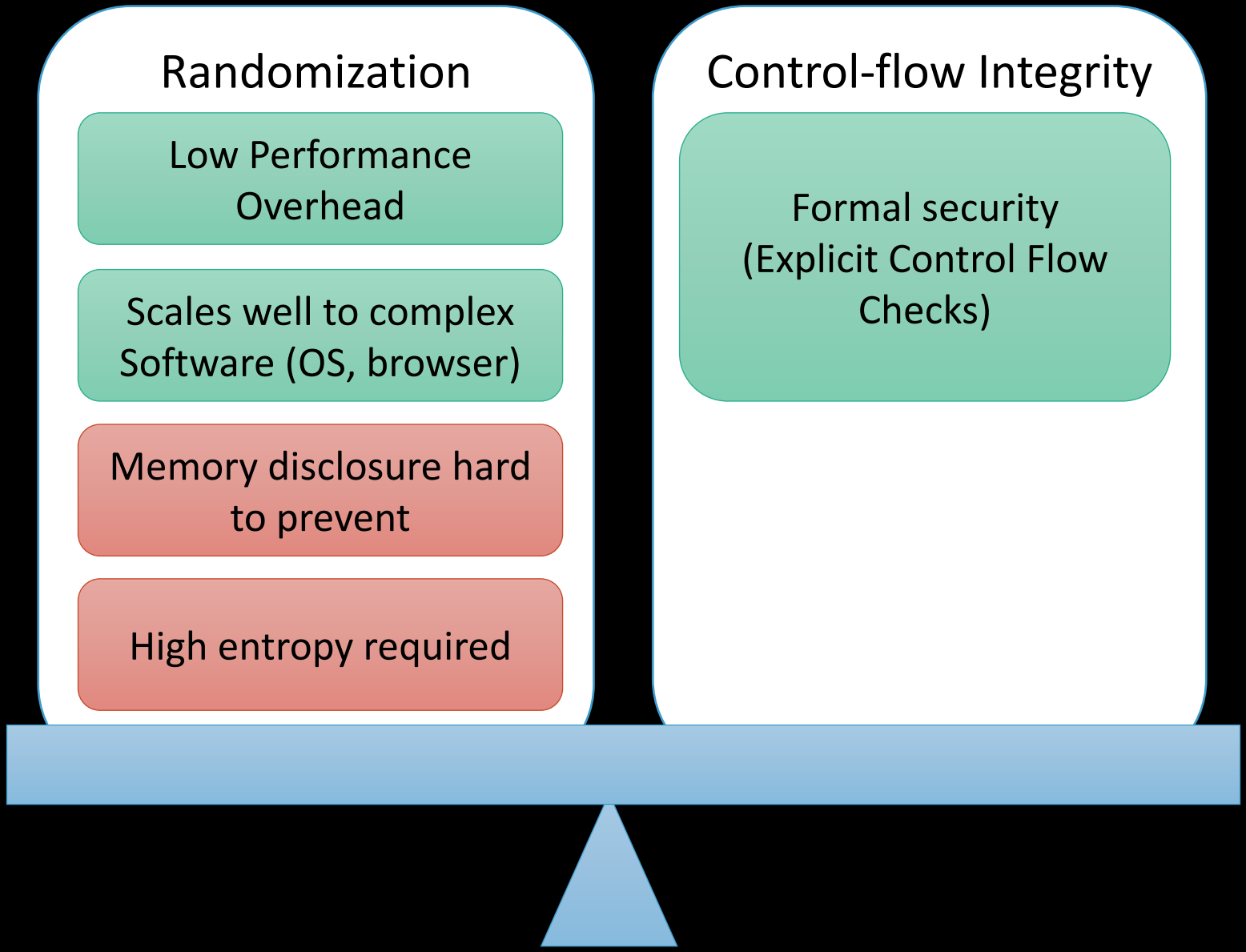
Randomization vs. CFI



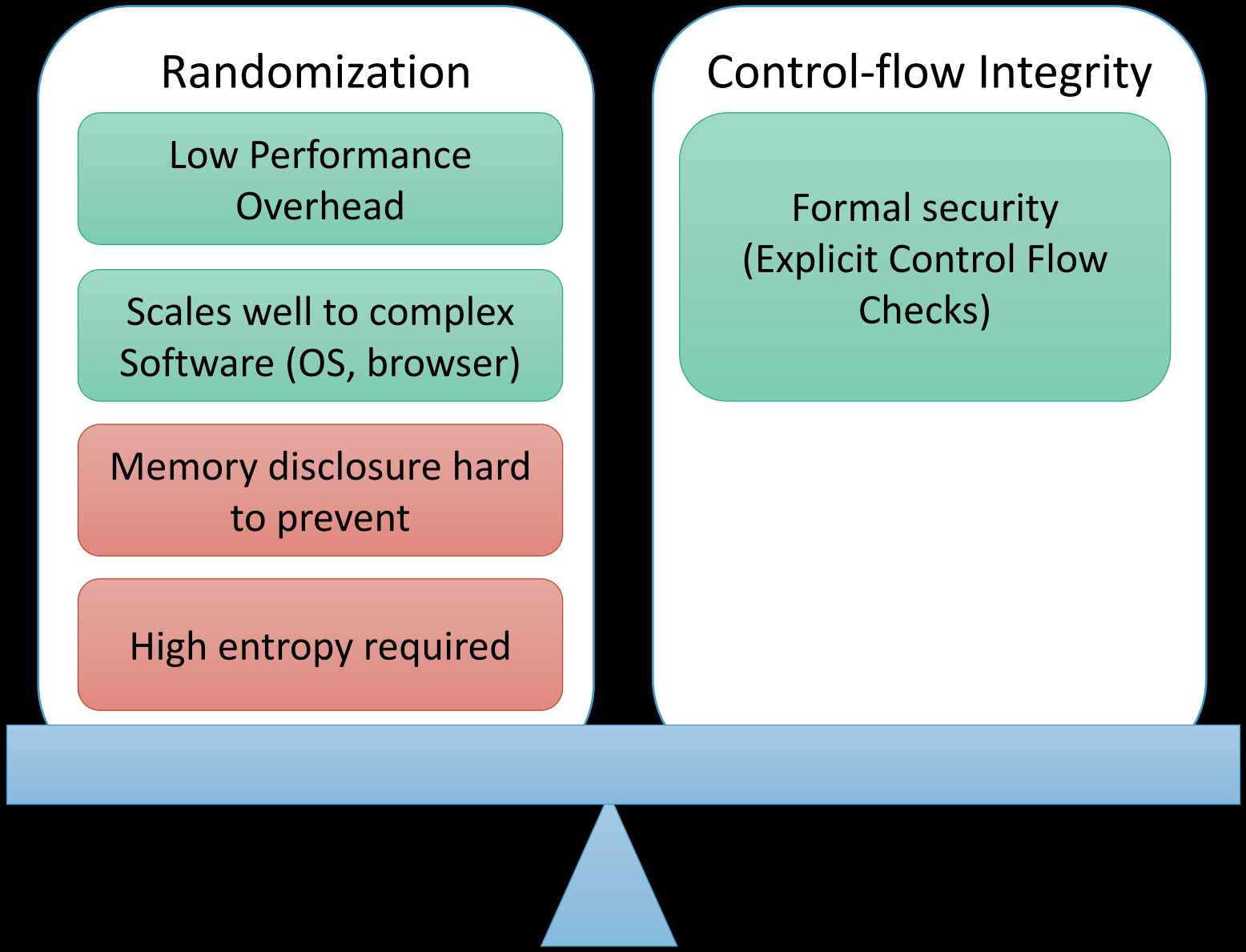
Randomization vs. CFI



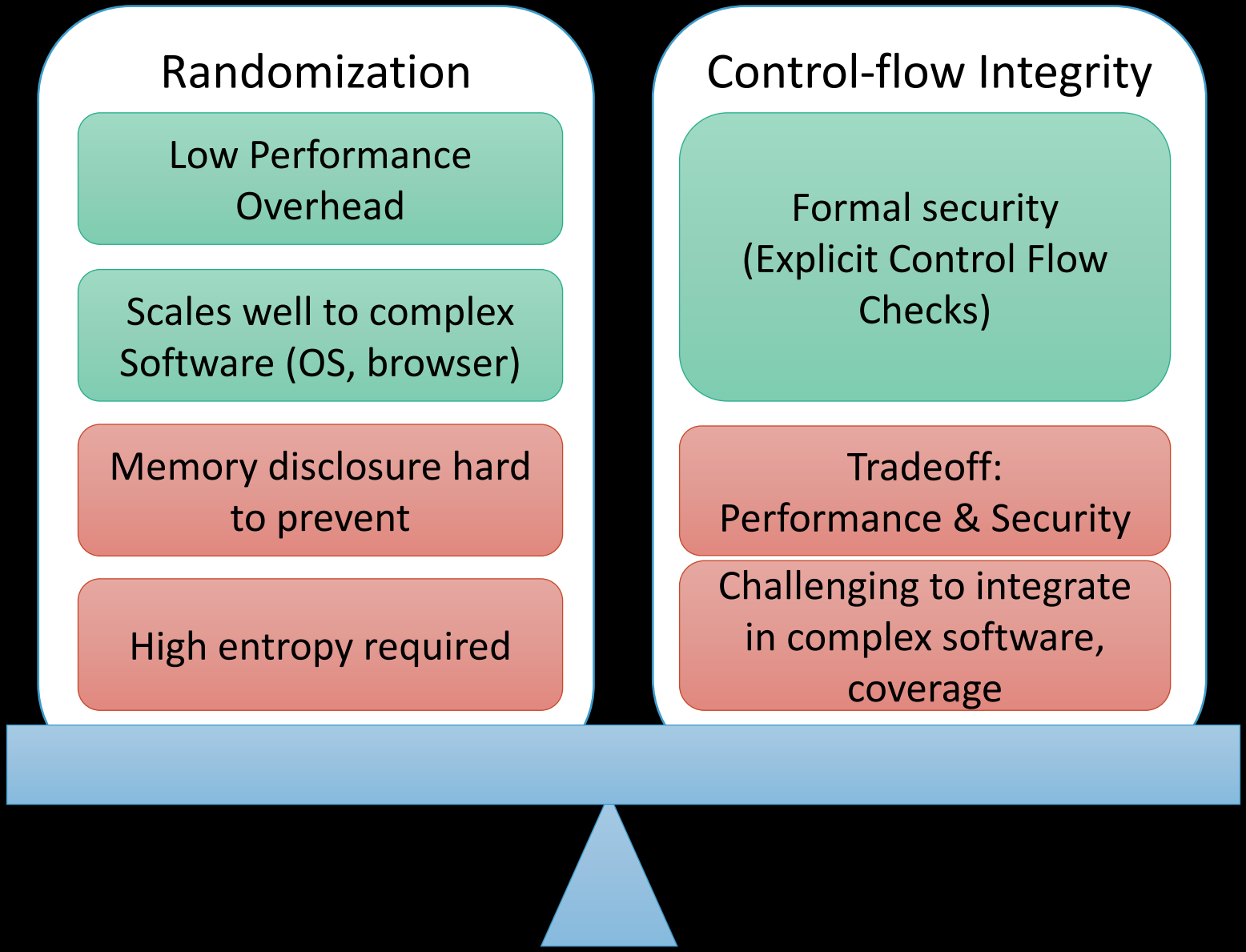
Randomization vs. CFI



Randomization vs. CFI



Randomization vs. CFI



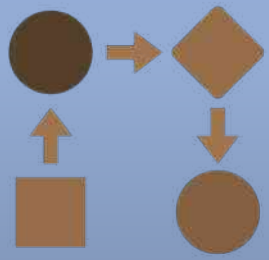
Overview

Background on Run-Time Attacks

Building Code Randomization Defenses

Code-Reuse Attacks

Building Control-Flow Integrity Defenses



Data-Oriented Exploits

Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]

Adversary

Memory write
Program flow

Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]

Adversary

Memory write
Program flow

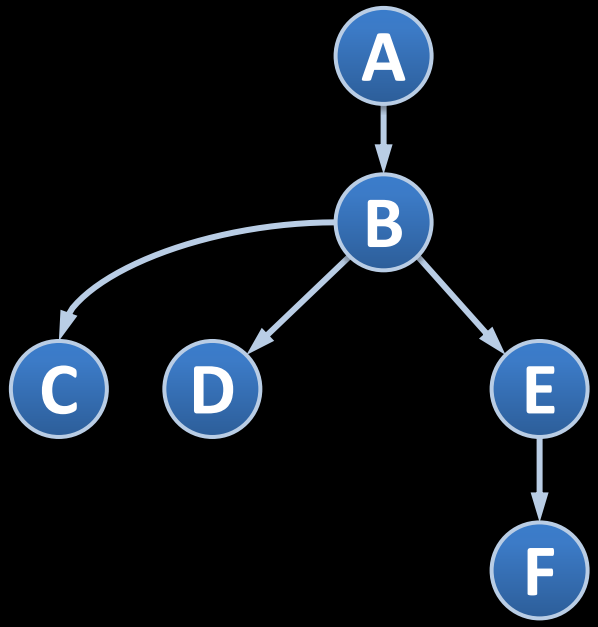
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Adversary

Memory write
Program flow

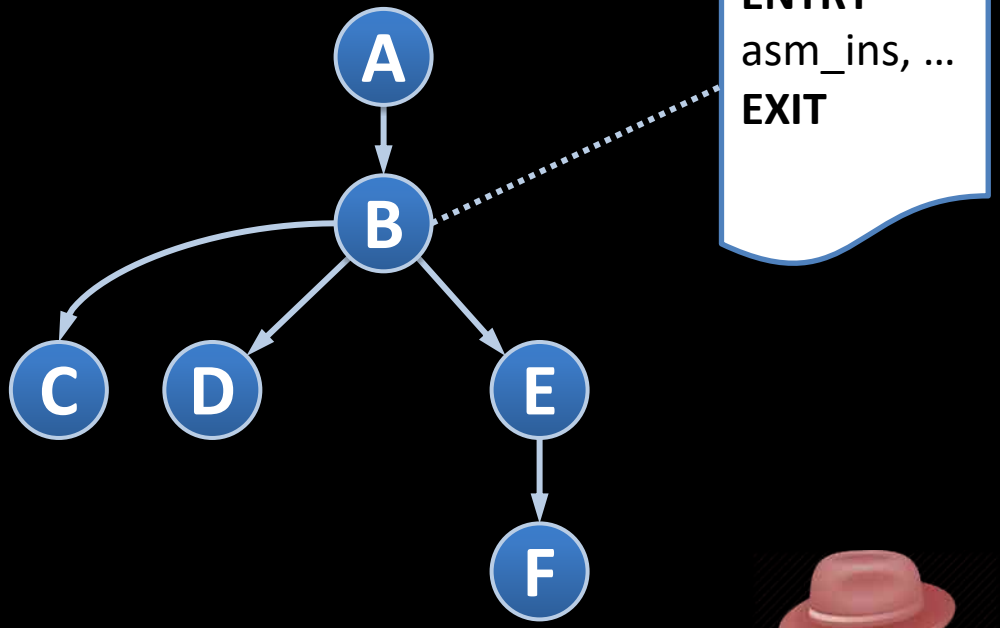
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Basic Block

ENTRY
asm_ins, …
EXIT

Adversary

Memory write
Program flow

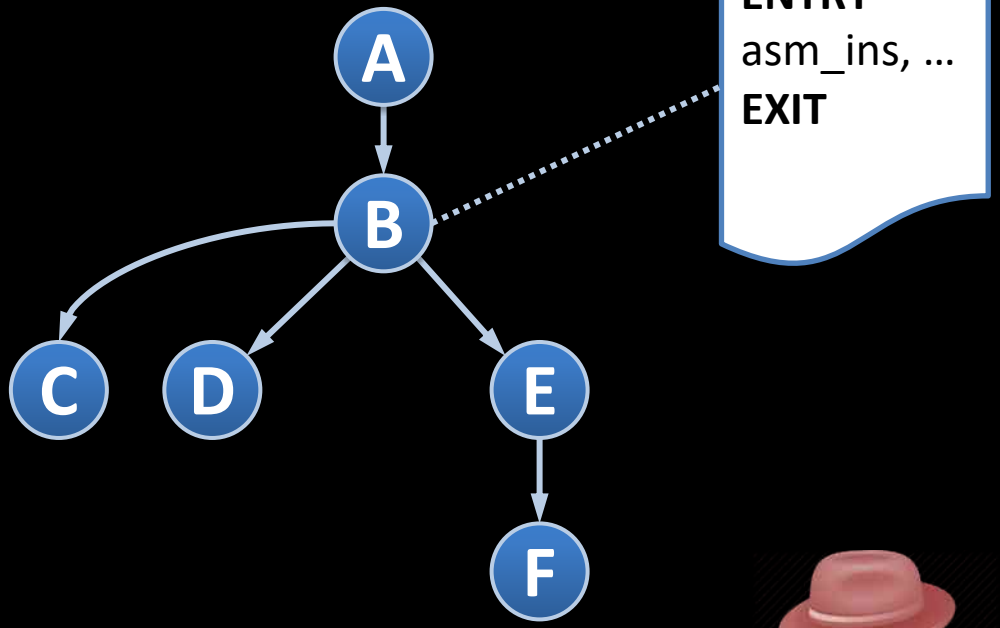
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Basic Block

ENTRY
asm_ins, …
EXIT

Adversary

Memory write
Program flow

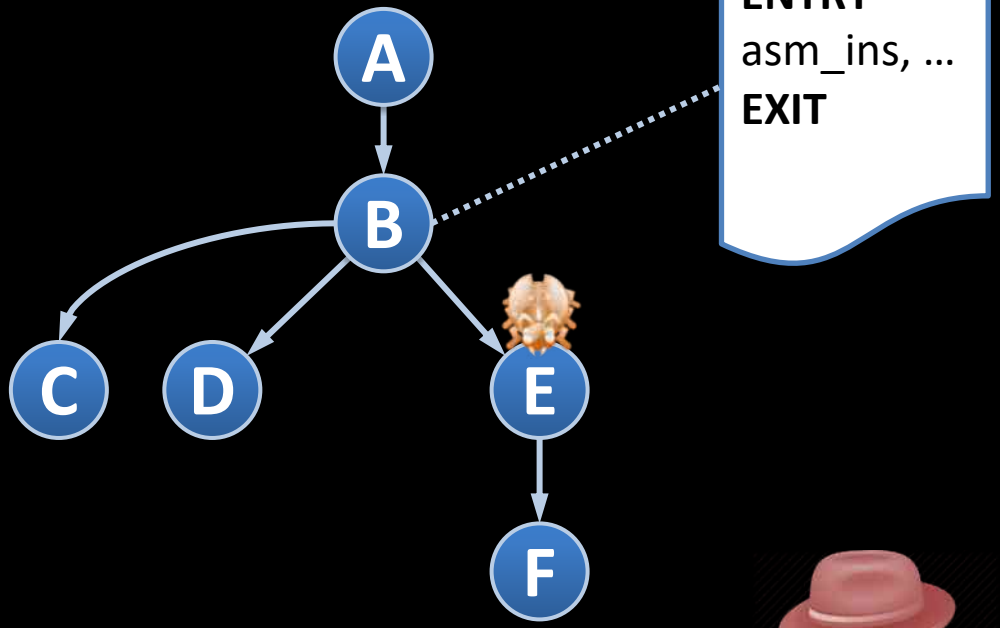
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Basic Block

ENTRY
asm_ins, …
EXIT

Adversary

Memory write
Program flow

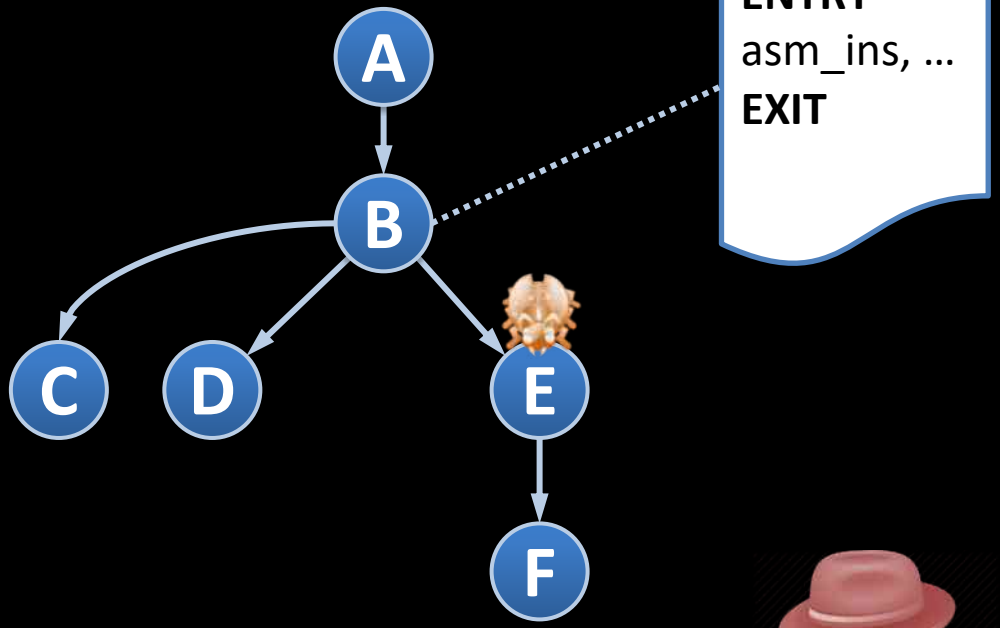
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Basic Block

ENTRY
asm_ins, …
EXIT

Adversary

Memory write
Program flow

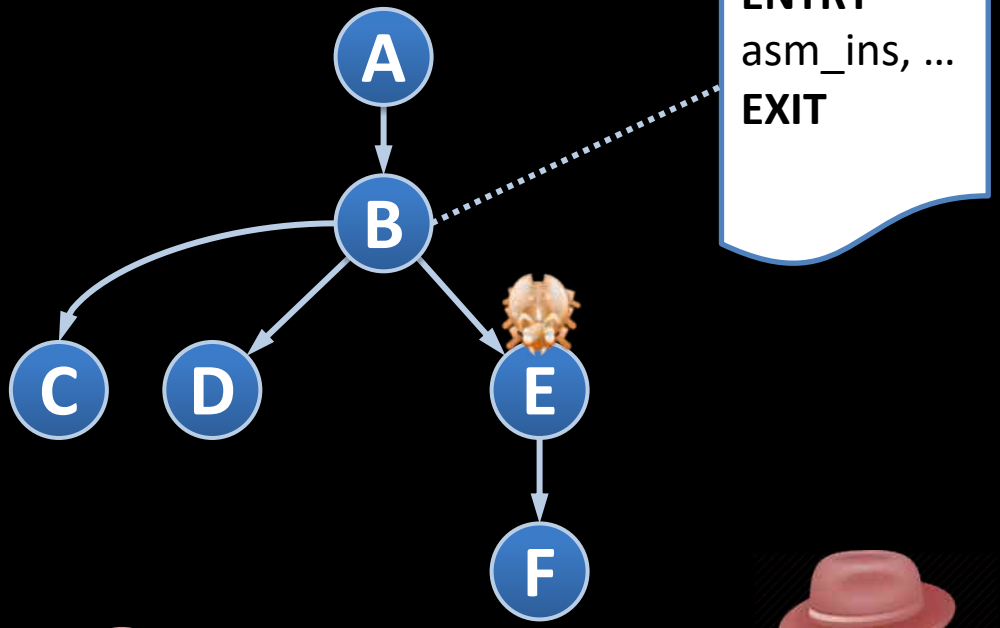
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



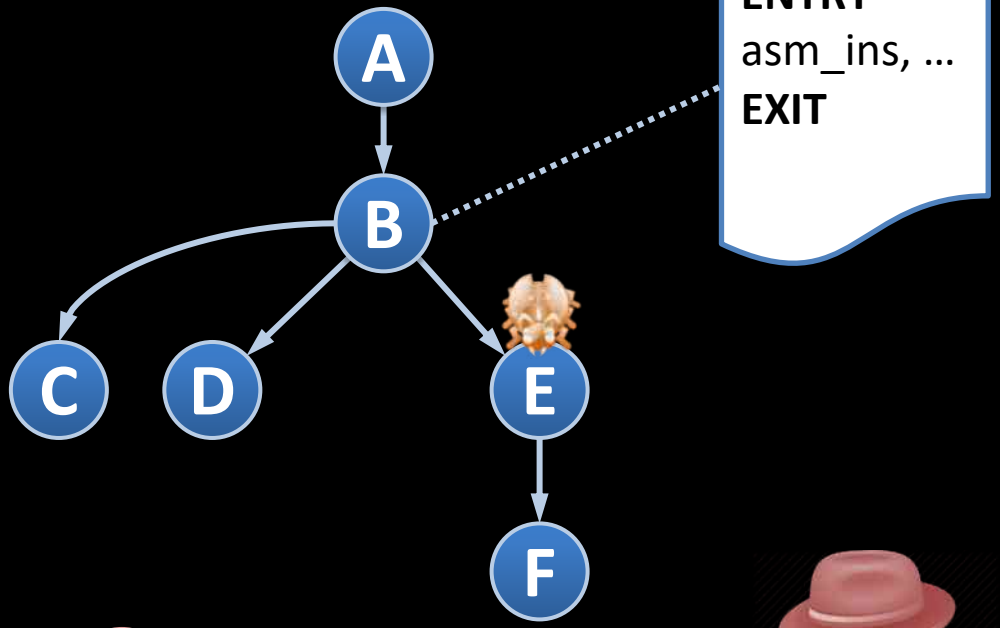
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



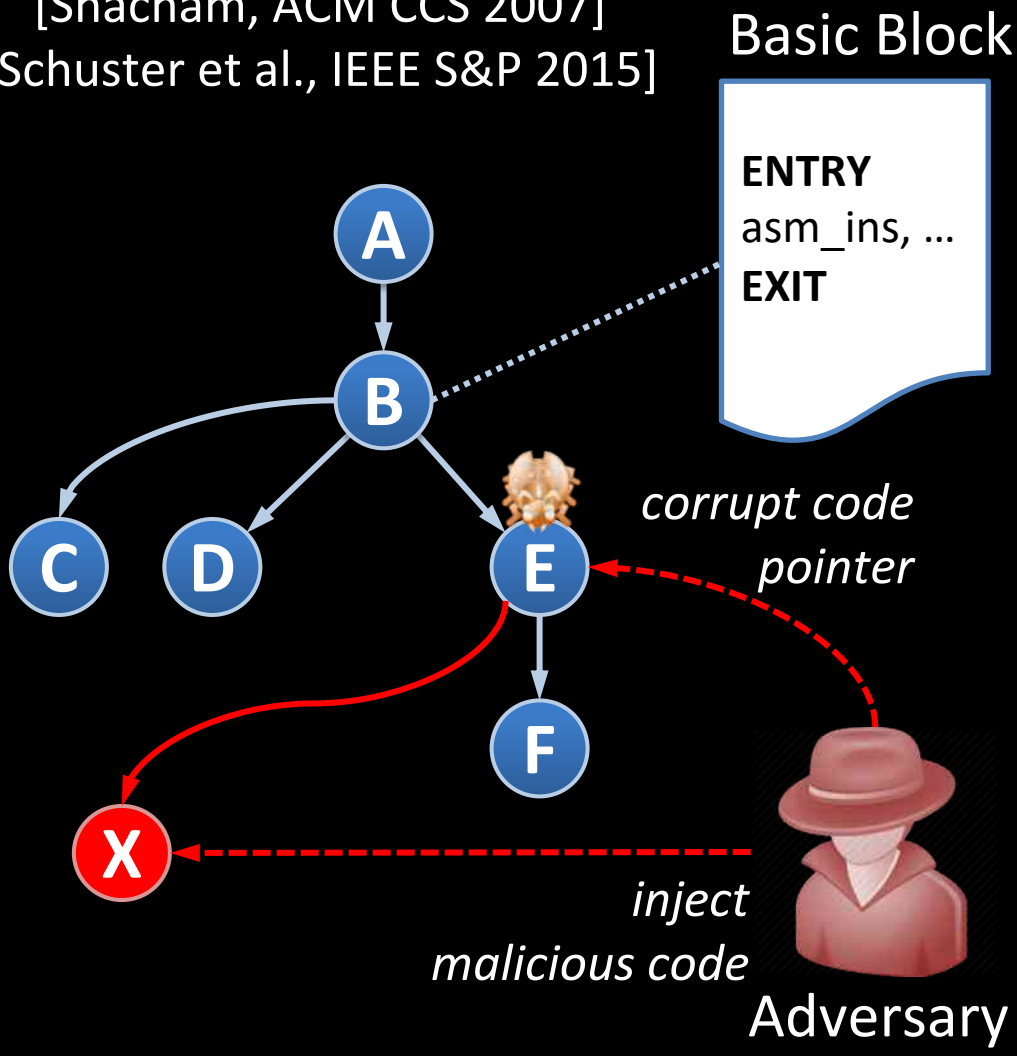
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



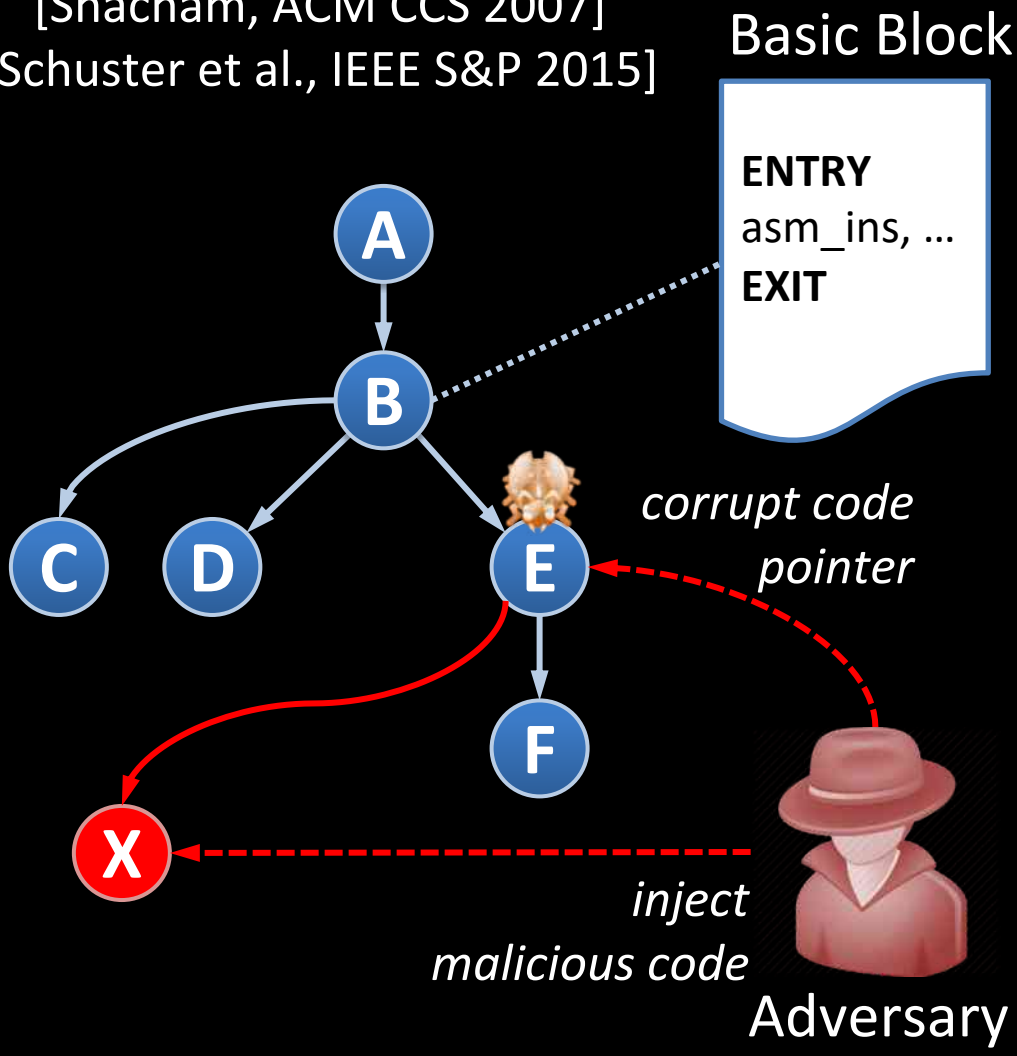
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



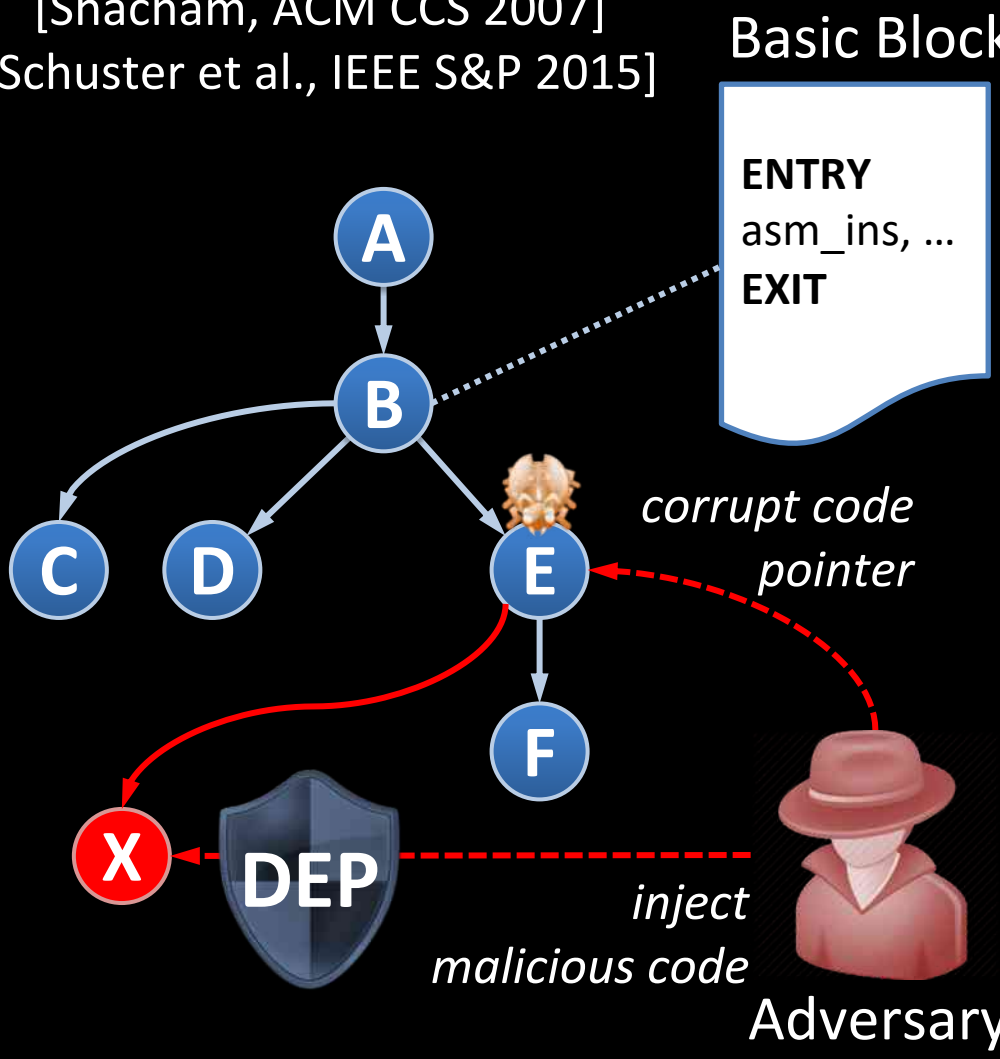
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



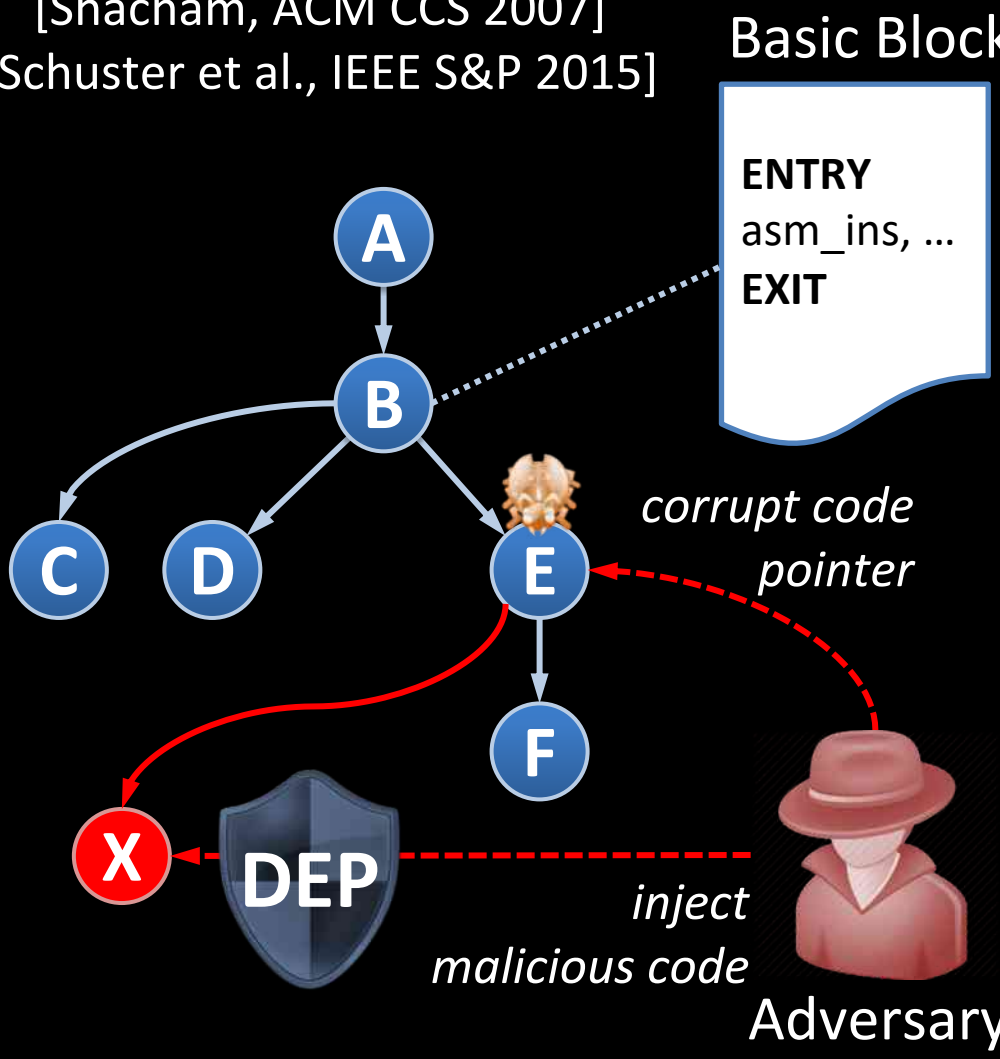
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



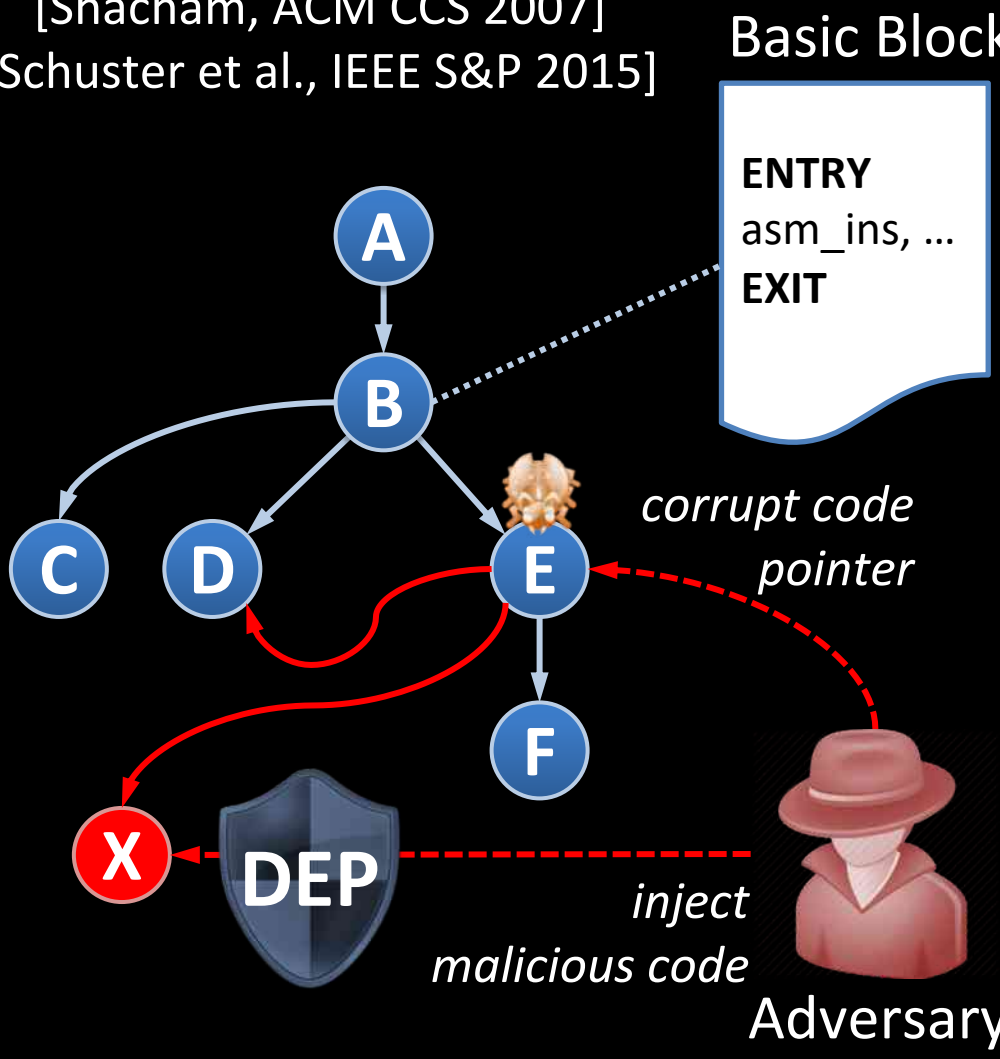
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



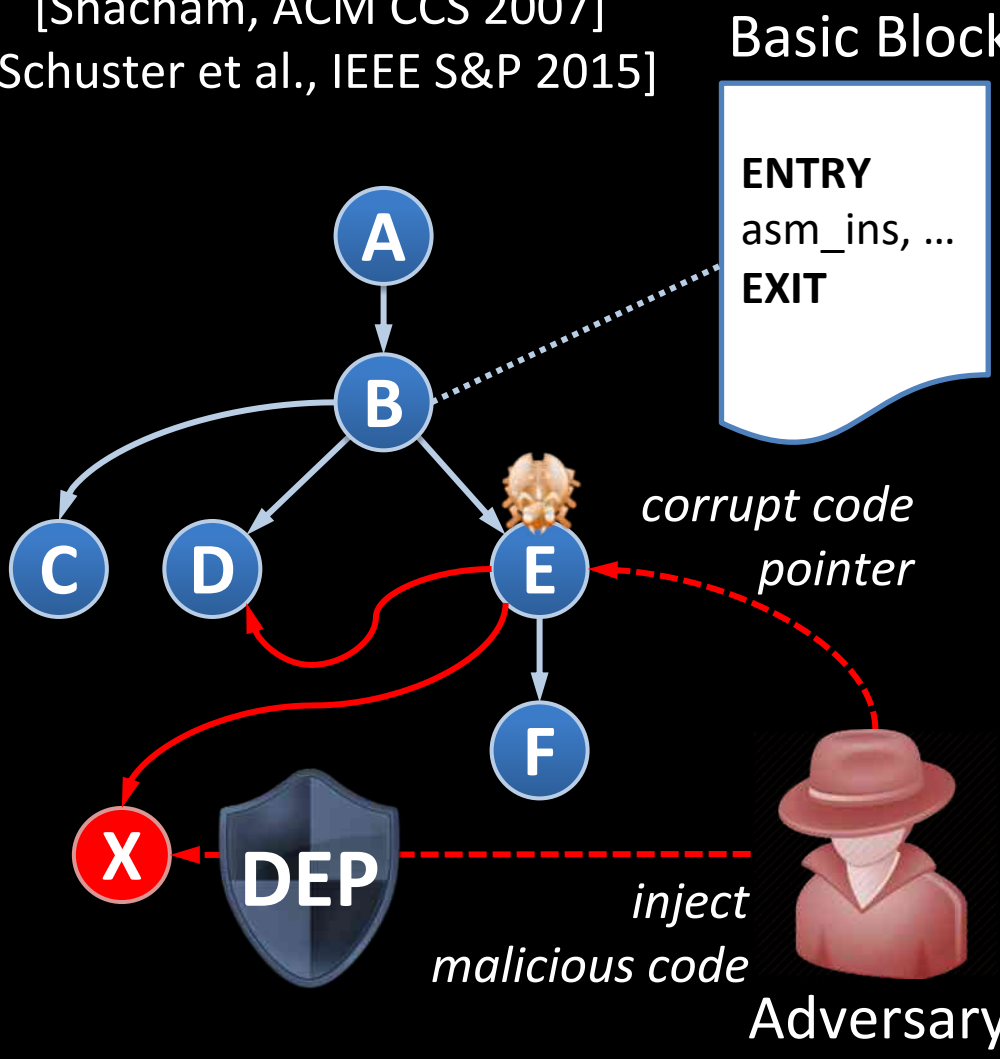
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



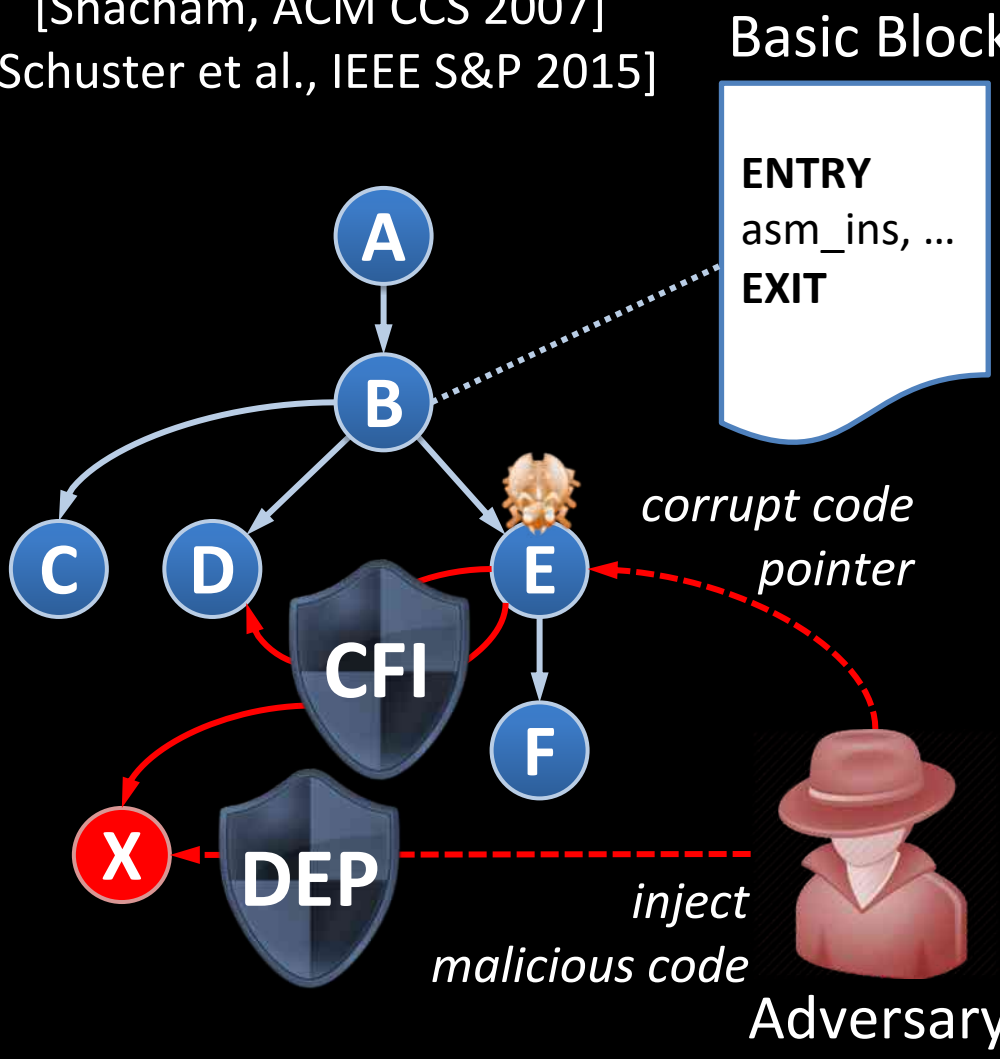
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Memory write
Program flow

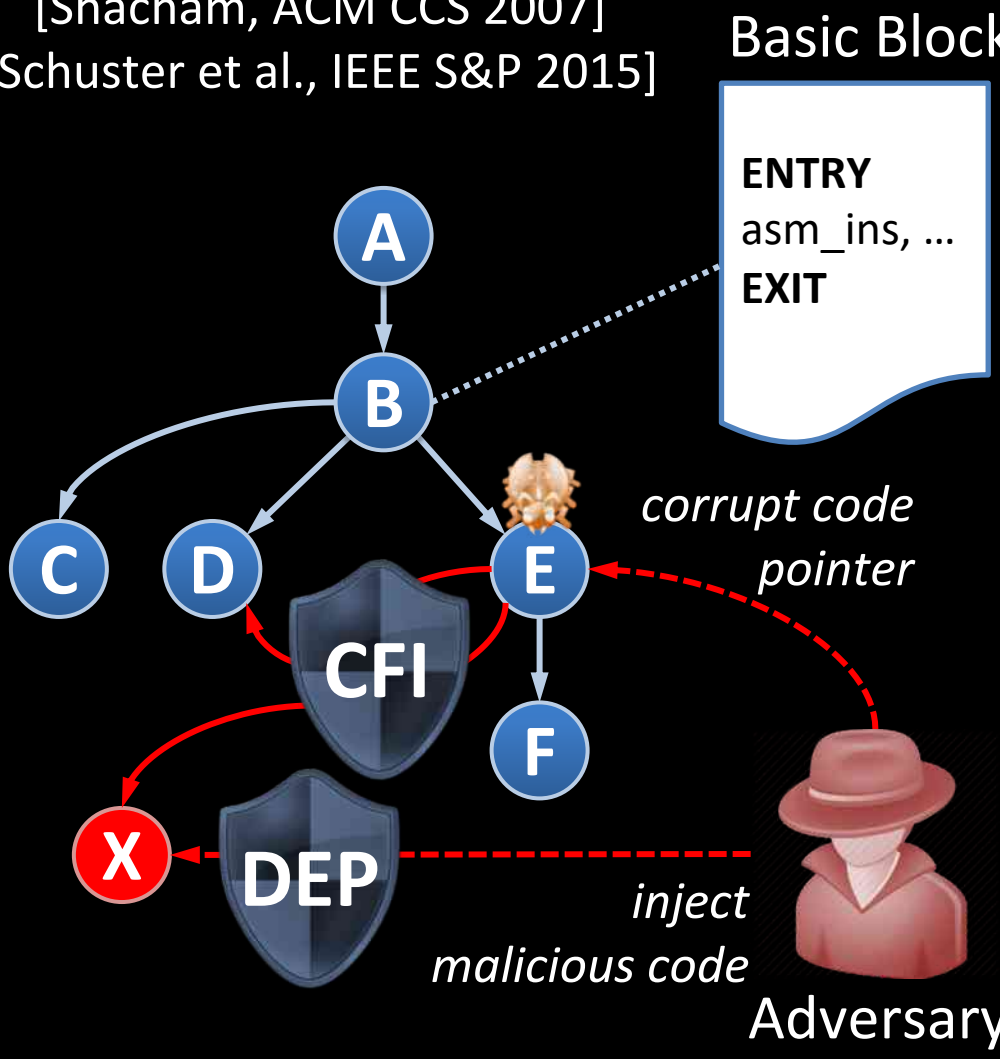
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]

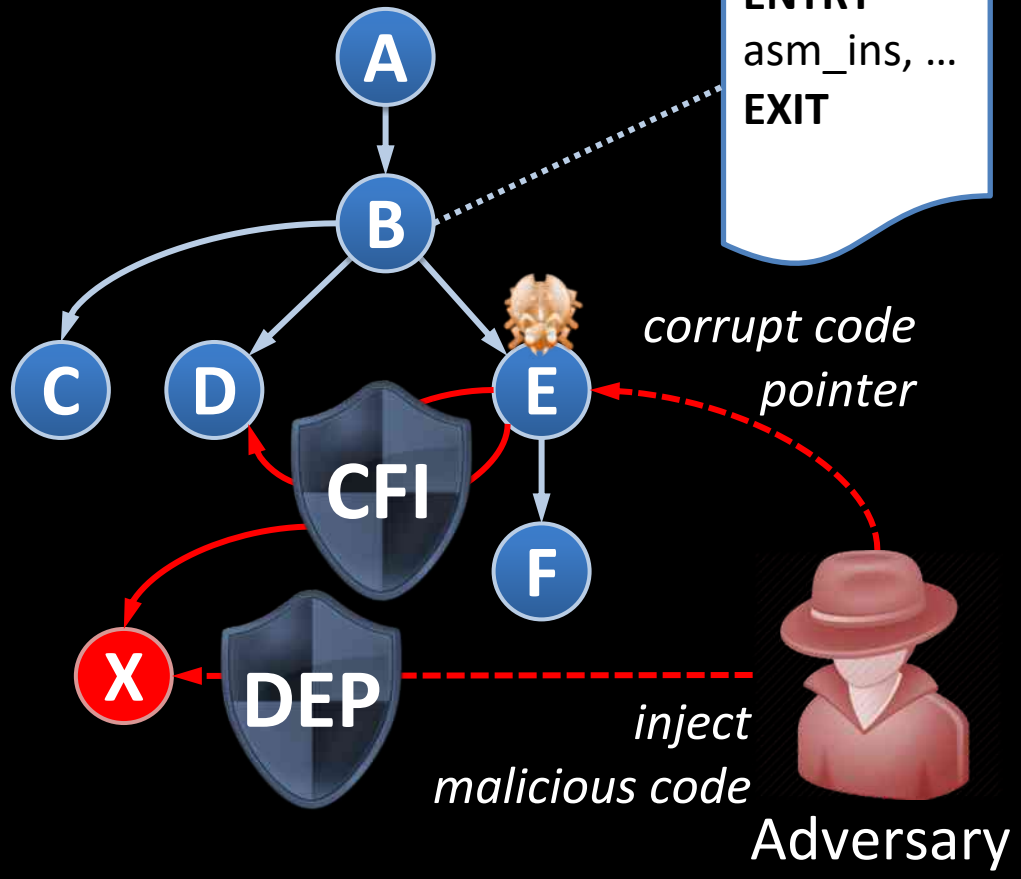


Memory write
Program flow

Problem Space of Zero-Day Exploits

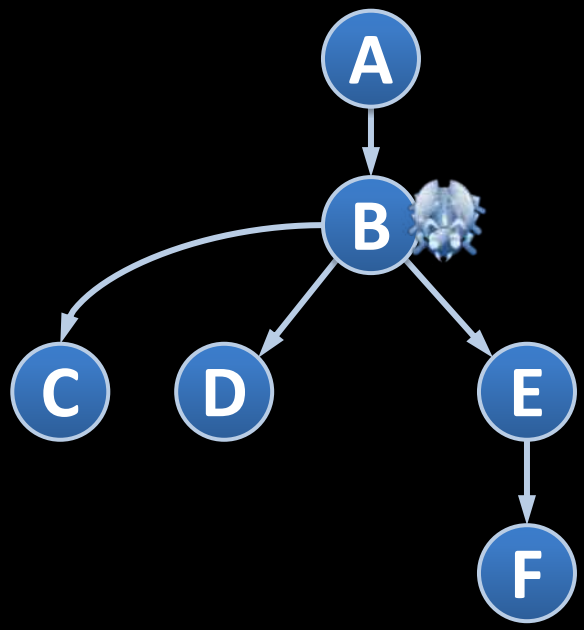
Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]



Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]

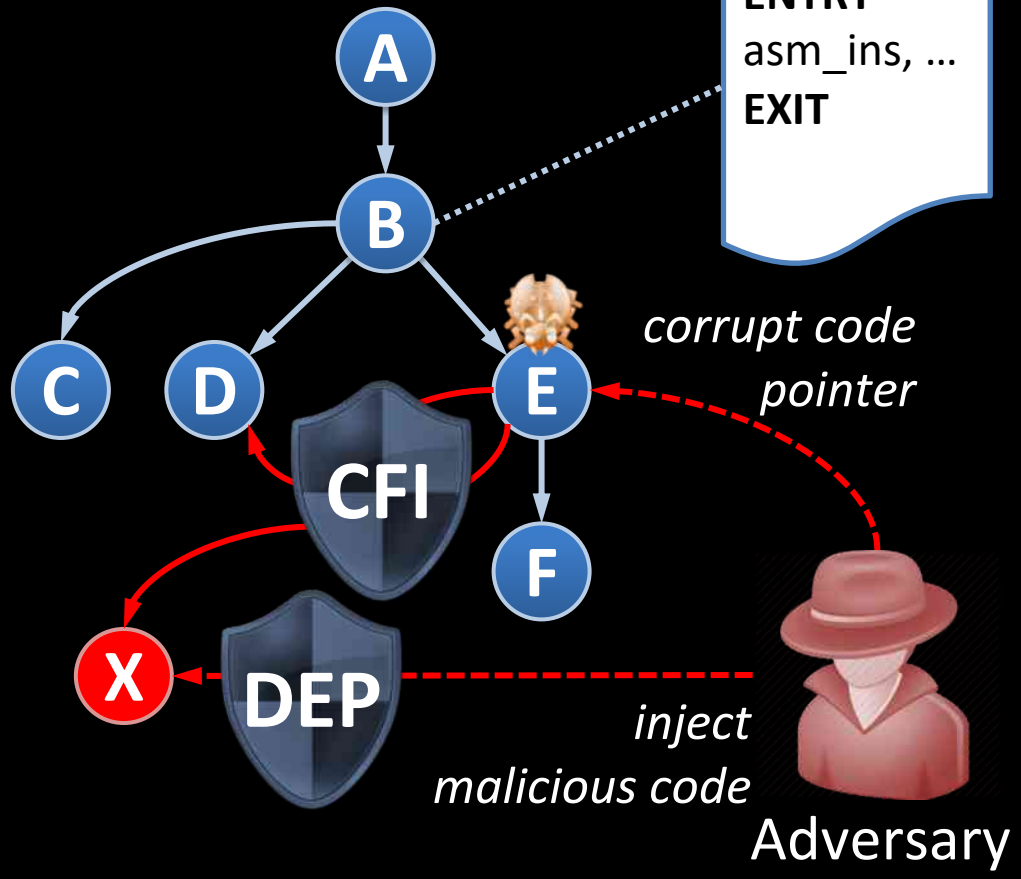


Memory write
Program flow

Problem Space of Zero-Day Exploits

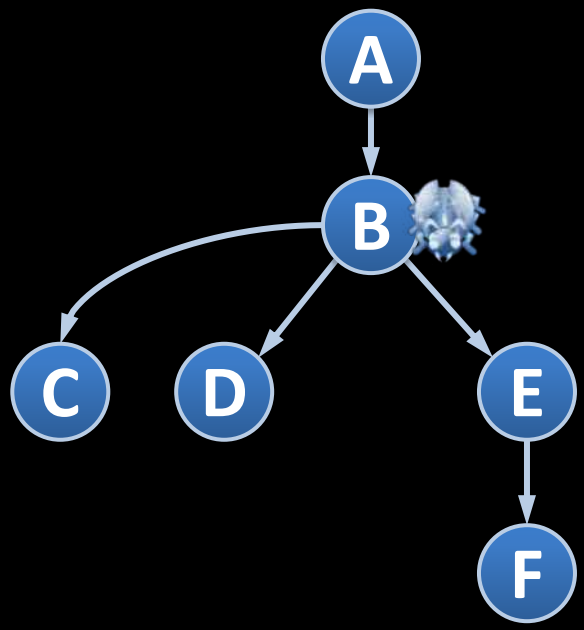
Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]



Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Memory write
Program flow

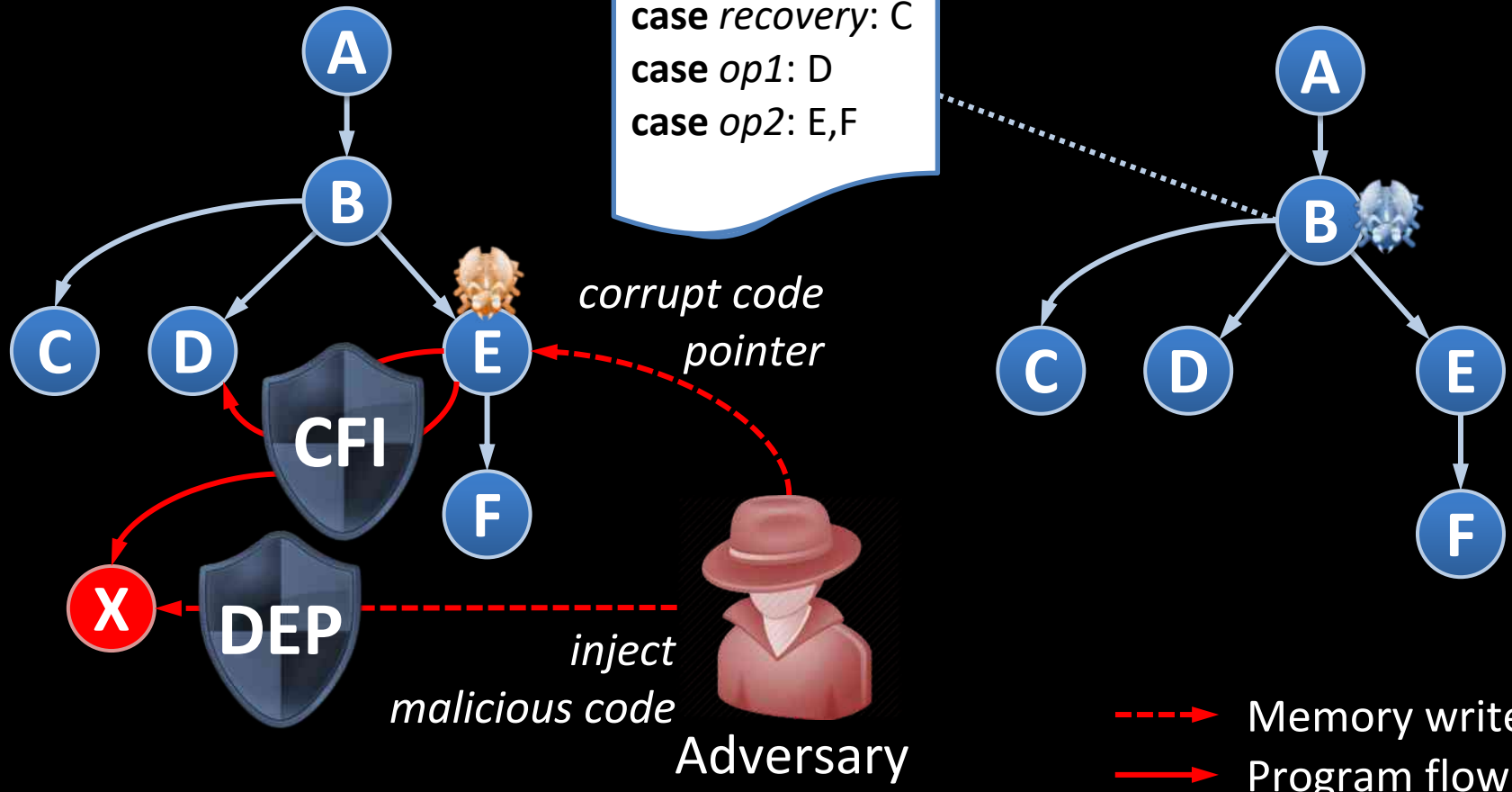
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



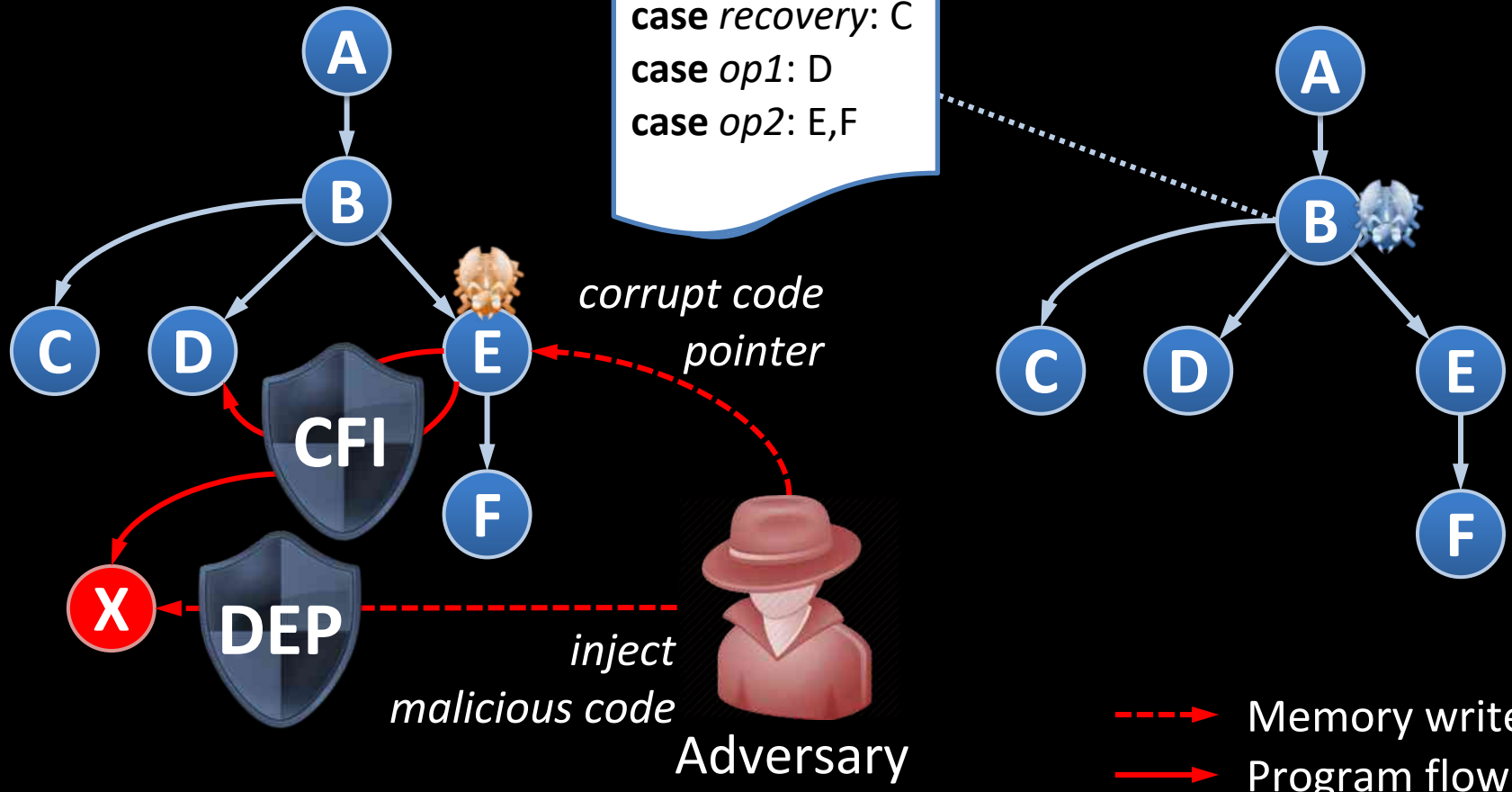
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



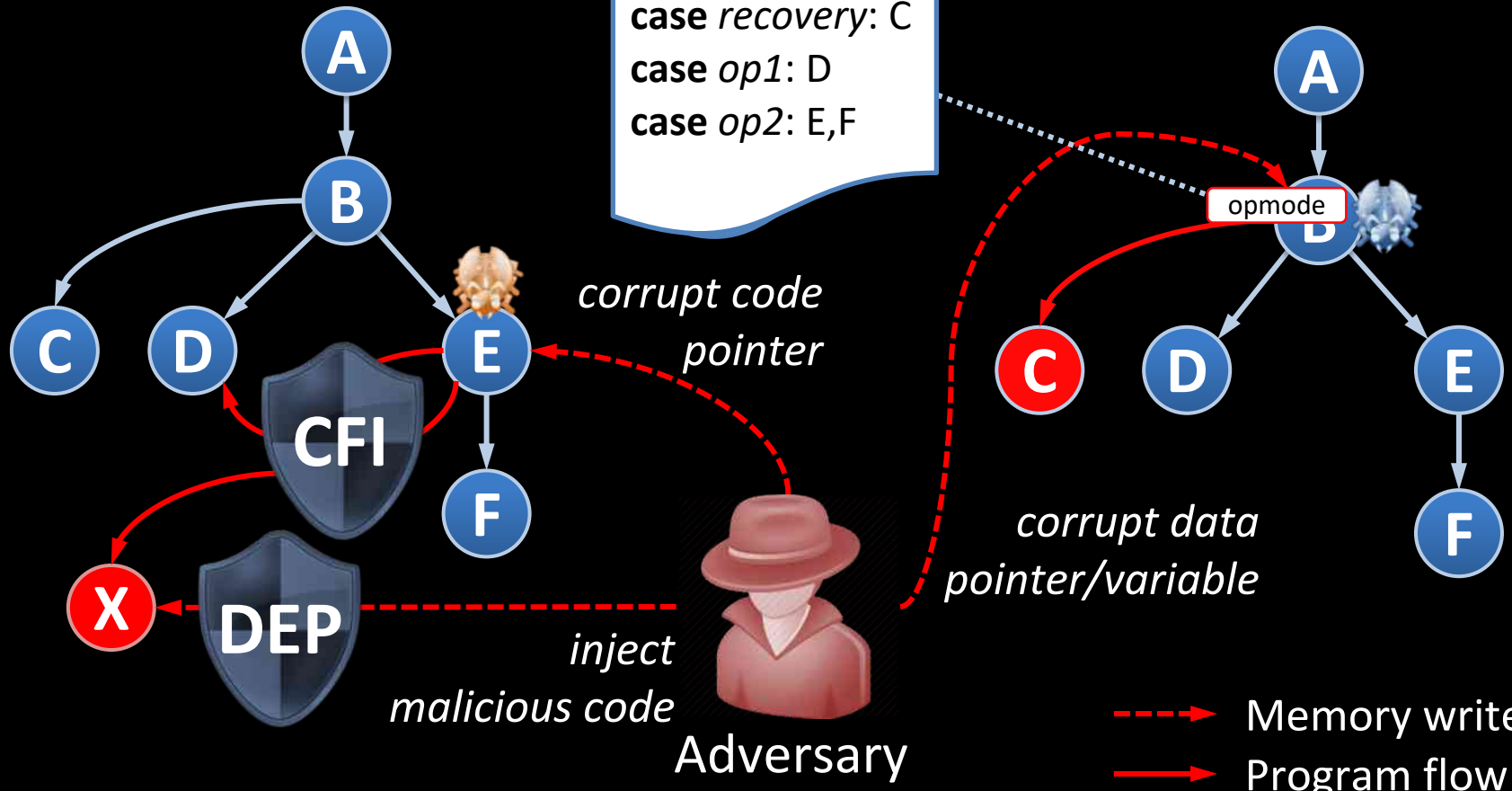
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



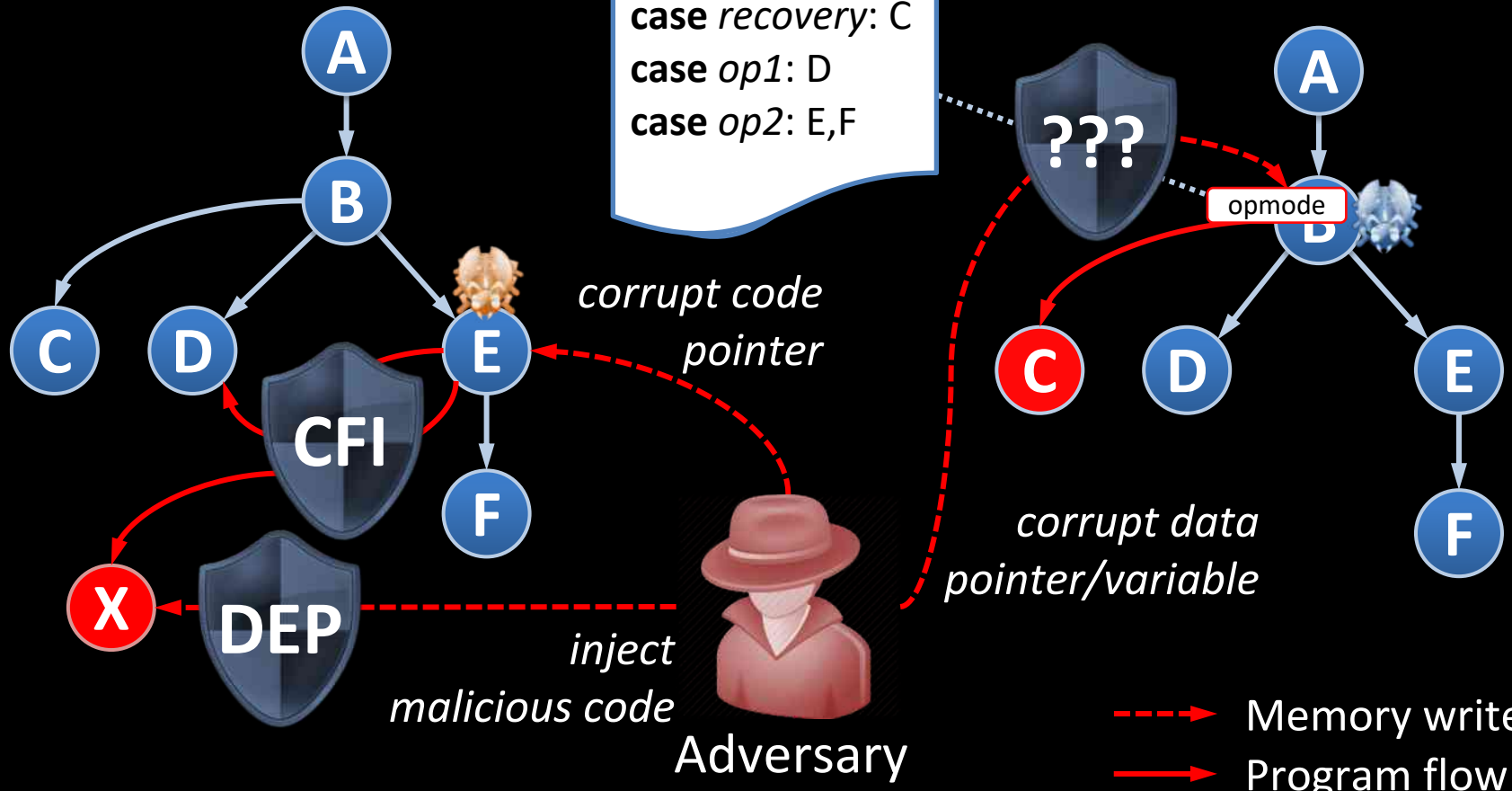
Problem Space of Zero-Day Exploits

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

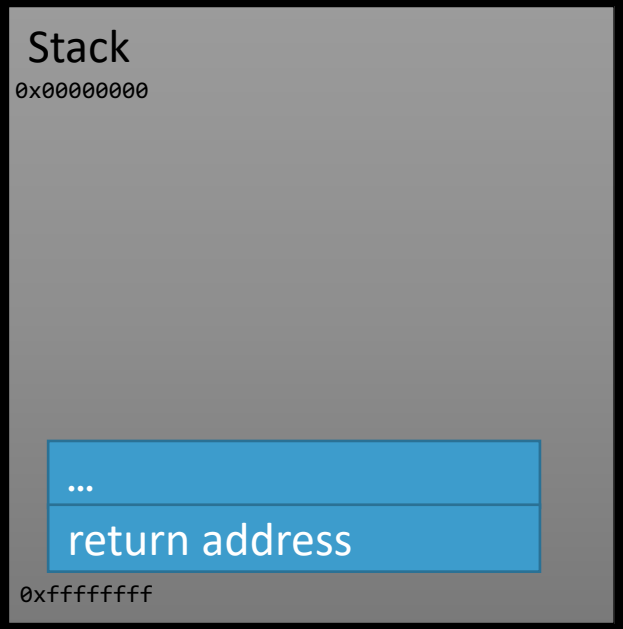
Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Control-Flow Hijacking on Example Code

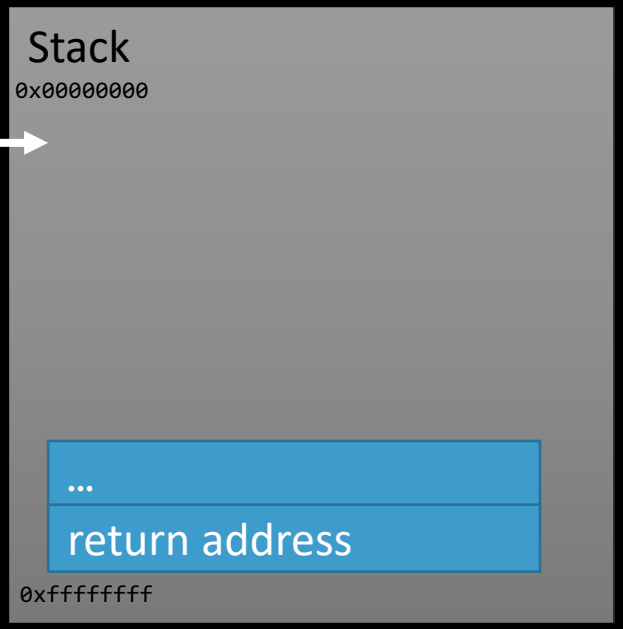
```
int main() {
  /* ... */
  user_t user;
  /* ... */
  update_name(&user);
  /* ... */
}
```



Control-Flow Hijacking on Example Code

```
int main() {
  /* ... */
  user_t user;
  /* ... */
  update_name(&user);
  /* ... */
}
```

Stack allocated

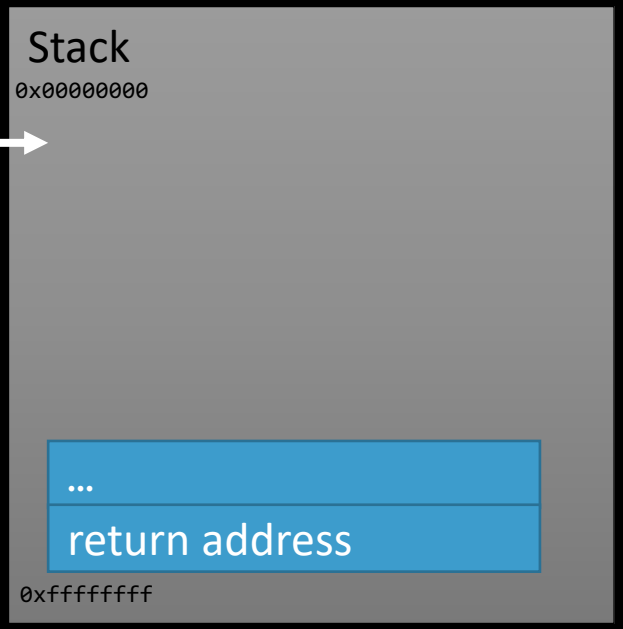



```
typedef struct {
  char name[32];
  unsigned id;
} user_t;
```

Control-Flow Hijacking on Example Code

```
int main() {
  /* ... */
  user_t user;
  /* ... */
  update_name(&user);
  /* ... */
}
```

Stack allocated

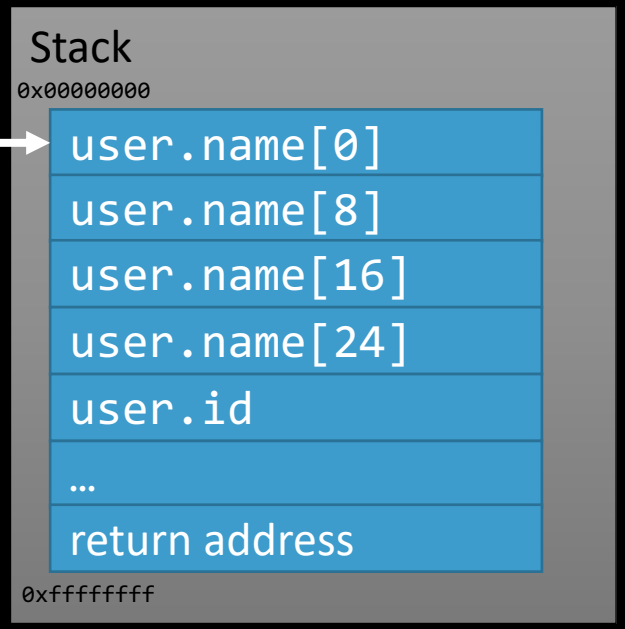



```
typedef struct {
  char name[32];
  unsigned id;
} user_t;
```

Control-Flow Hijacking on Example Code

```c
int main() {
  /* ... */
  user_t user;
  /* ... */
  update_name(&user);
  /* ... */
}
```

Stack allocated

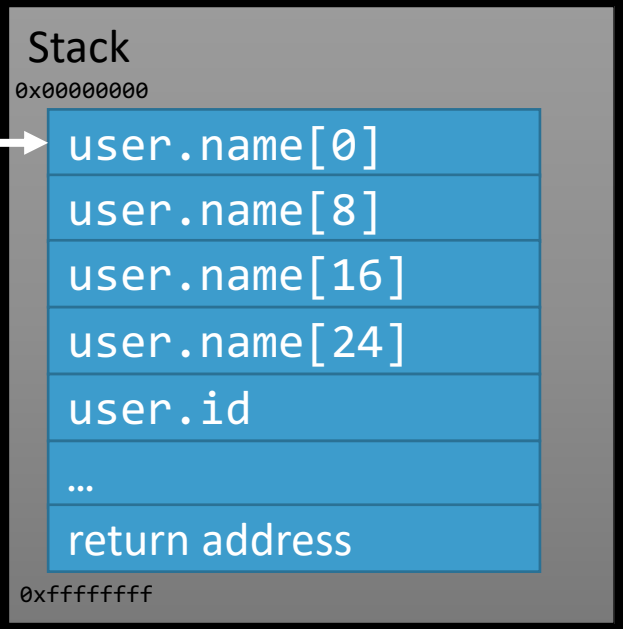



```c
typedef struct {
  char name[32];
  unsigned id;
} user_t;
```

Control-Flow Hijacking on Example Code

```
int main() {
  /* ... */
  user_t user;
  /* ... */
  update_name(&user);
  /* ... */
}
```

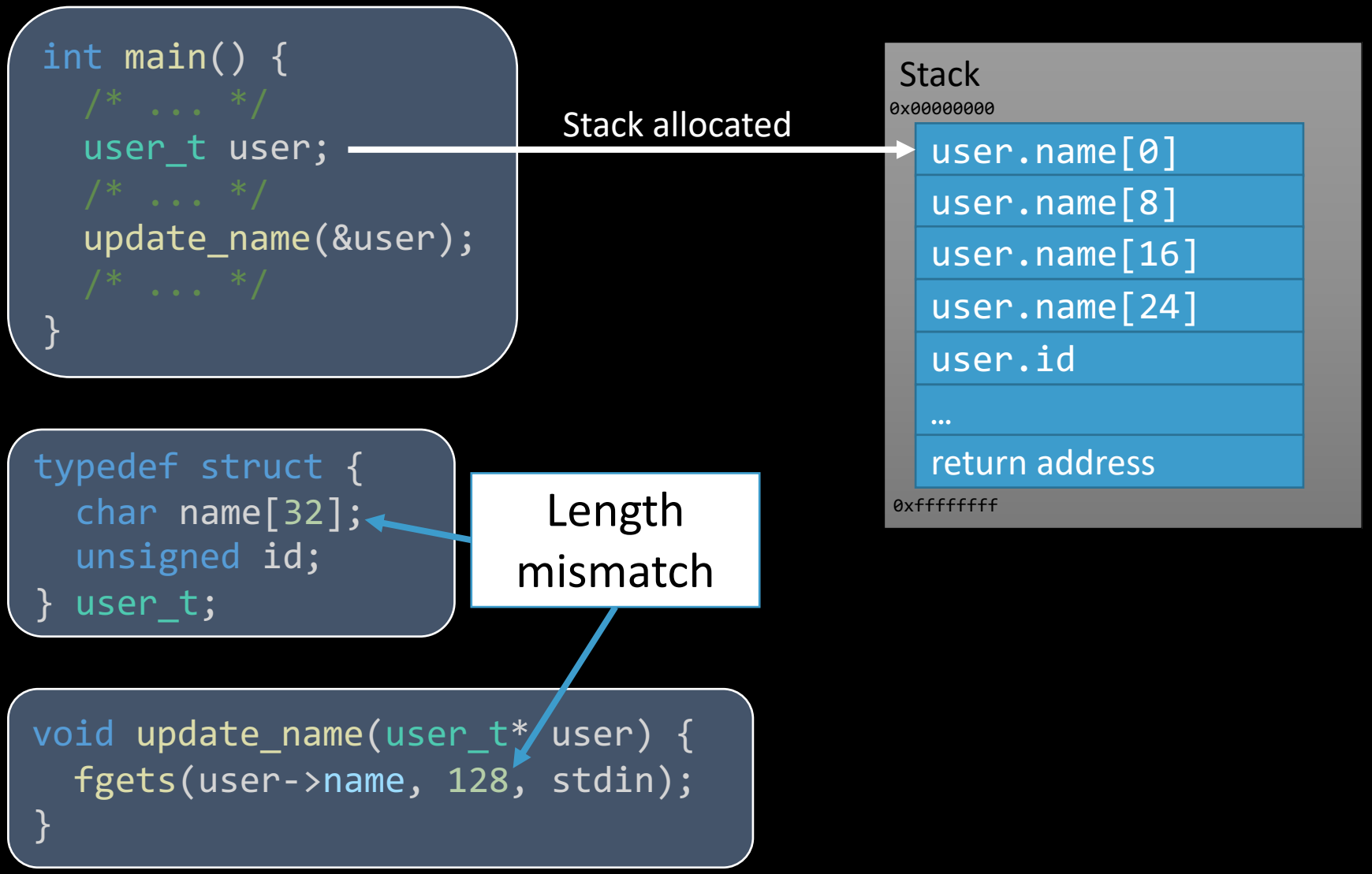
Stack allocated



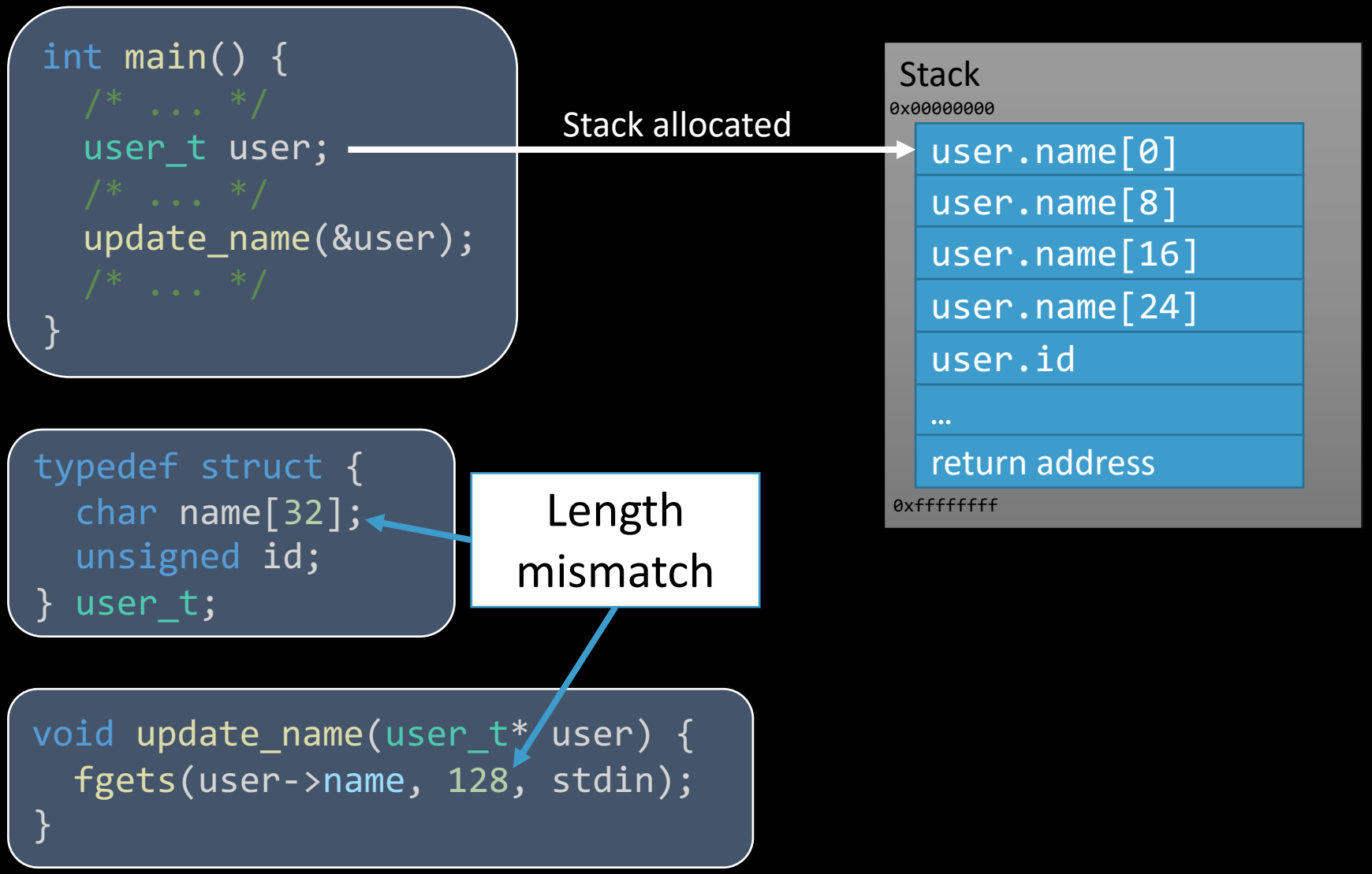

```
typedef struct {
  char name[32];
  unsigned id;
} user_t;
```

```
void update_name(user_t* user) {
  fgets(user->name, 128, stdin);
}
```

Control-Flow Hijacking on Example Code



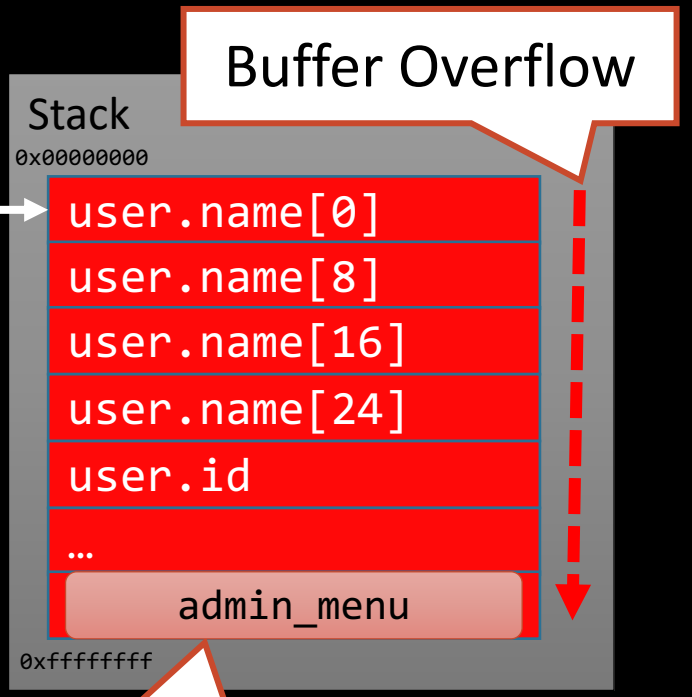
Control-Flow Hijacking on Example Code



Control-Flow Hijacking on Example Code

```
int main() {
  /* ... */
  user_t user;
  /* ... */
  update_name(&user);
  /* ... */
}
```

Stack allocated




```
typedef struct {
  char name[32];
  unsigned id;
} user_t;
```

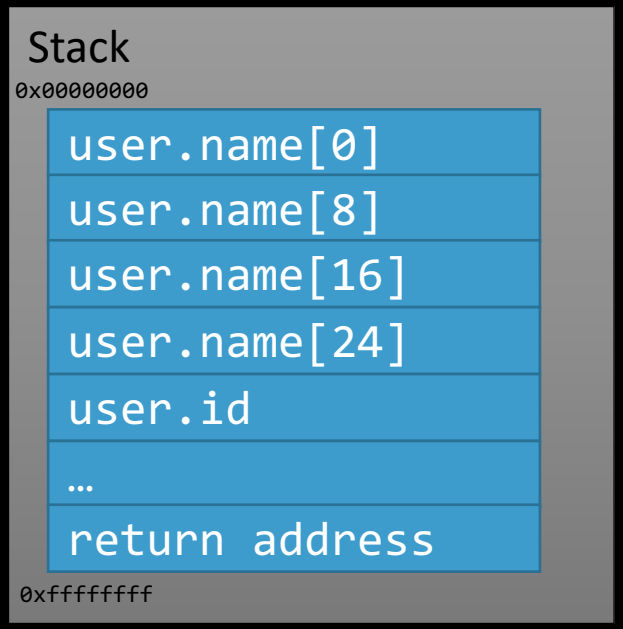
Length mismatch

```
void update_name(user_t* user) {
  fgets(user->name, 128, stdin);
}
```

Return to a different address.
Attacker redirects to target function.

Data-Only Exploit on Example Code

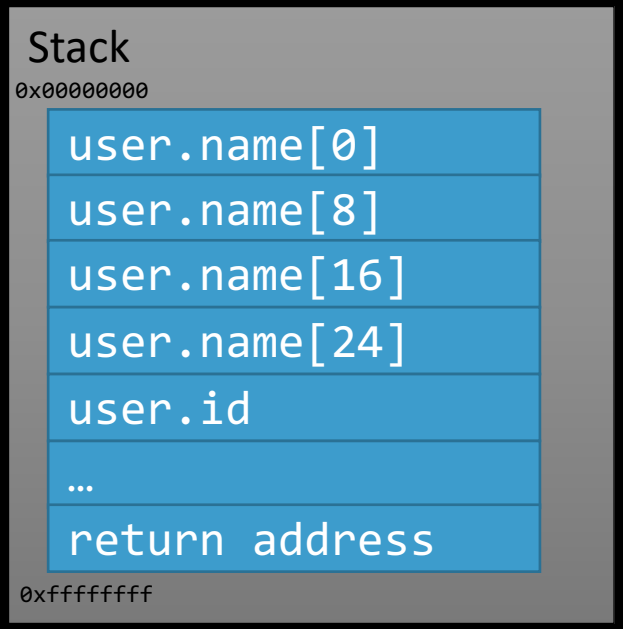
```
int main() {
  /* ... */
  update_name(&user);
  /* ... */
  admin_access(&user);
  /* ... */
}
```



Data-Only Exploit on Example Code

```
int main() {
  /* ... */
  update_name(&user);
  /* ... */
  admin_access(&user);
  /* ... */
}
```

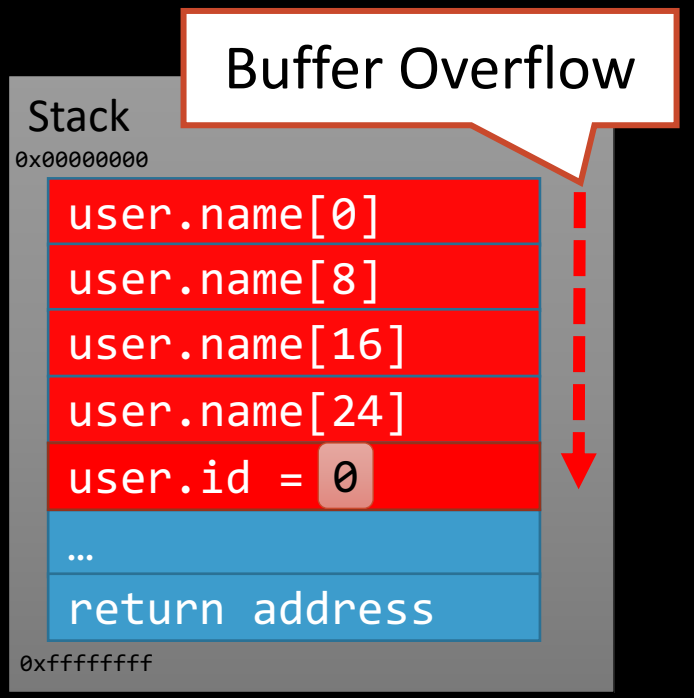
```
void update_name(user_t* user) {
  fgets(user->name, 128, stdin);
}
```



Data-Only Exploit on Example Code

```
int main() {
  /* ... */
  update_name(&user);
  /* ... */
  admin_access(&user);
  /* ... */
}
```

```
void update_name(user_t* user) {
  fgets(user->name, 128, stdin);
}
```

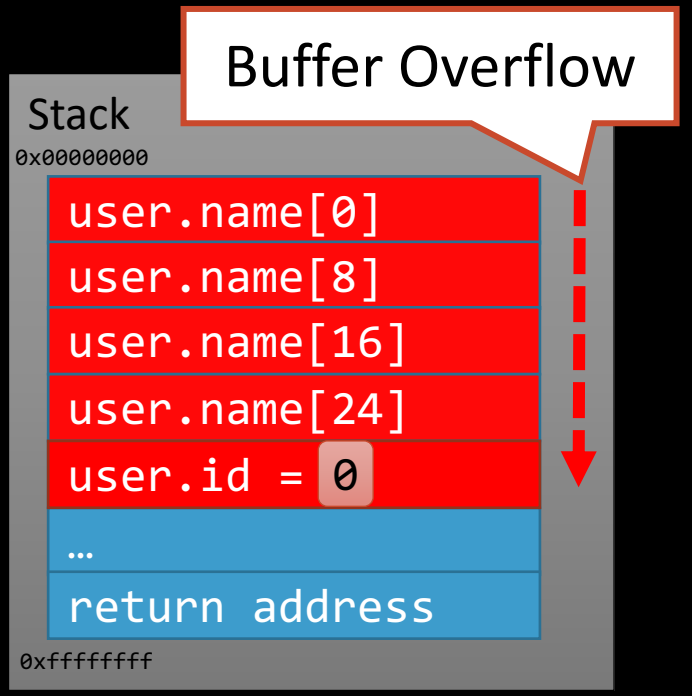


Data-Only Exploit on Example Code

```
int main() {
  /* ... */
  update_name(&user);
  /* ... */
  admin_access(&user);
  /* ... */
}
```

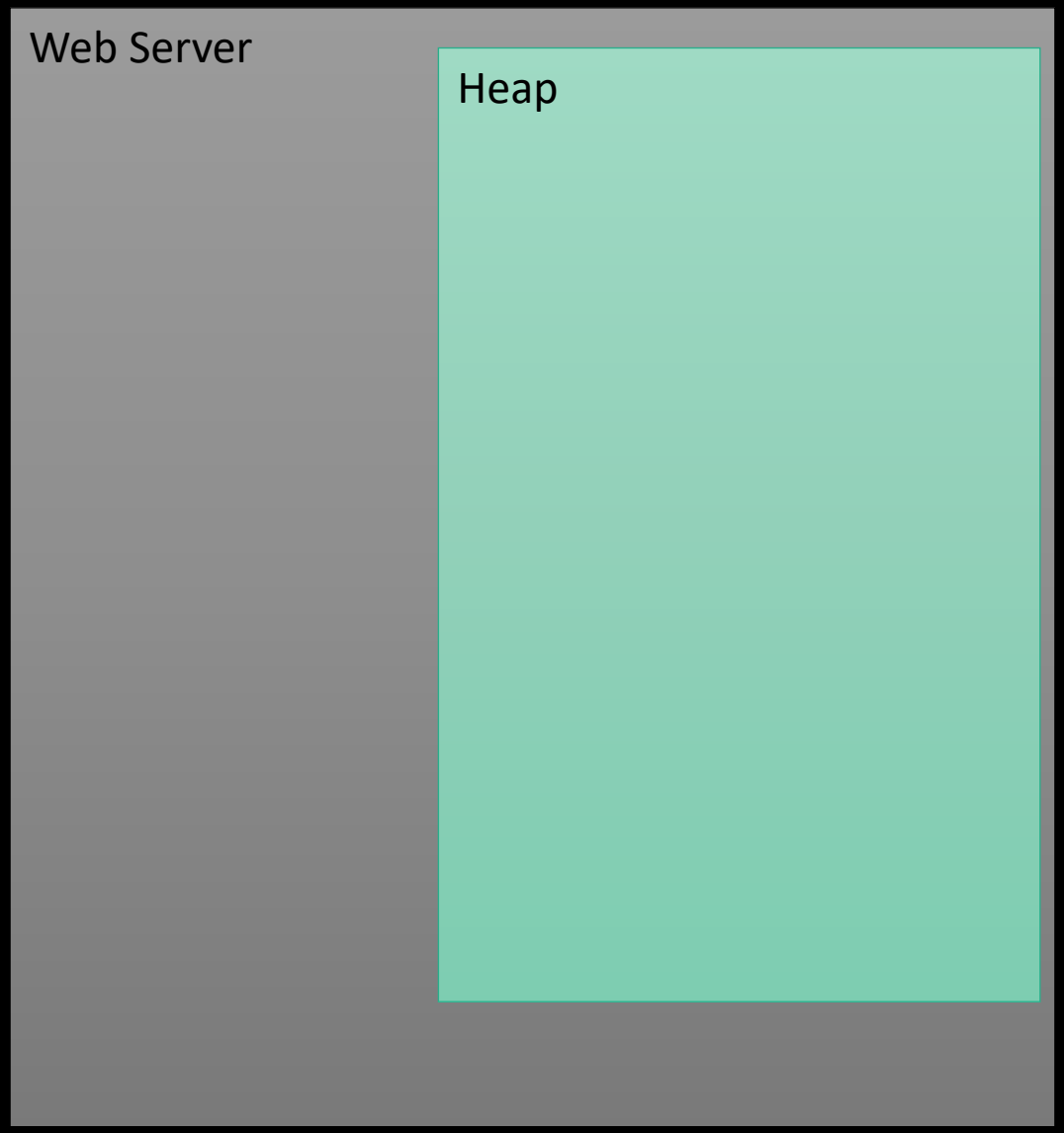
```
void update_name(user_t* user) {
  fgets(user->name, 128, stdin);
}
```

```
void admin_access(user_t* user) {
  if (user->id == 0) {
    admin_menu();
  } else {
    error(); }}
```

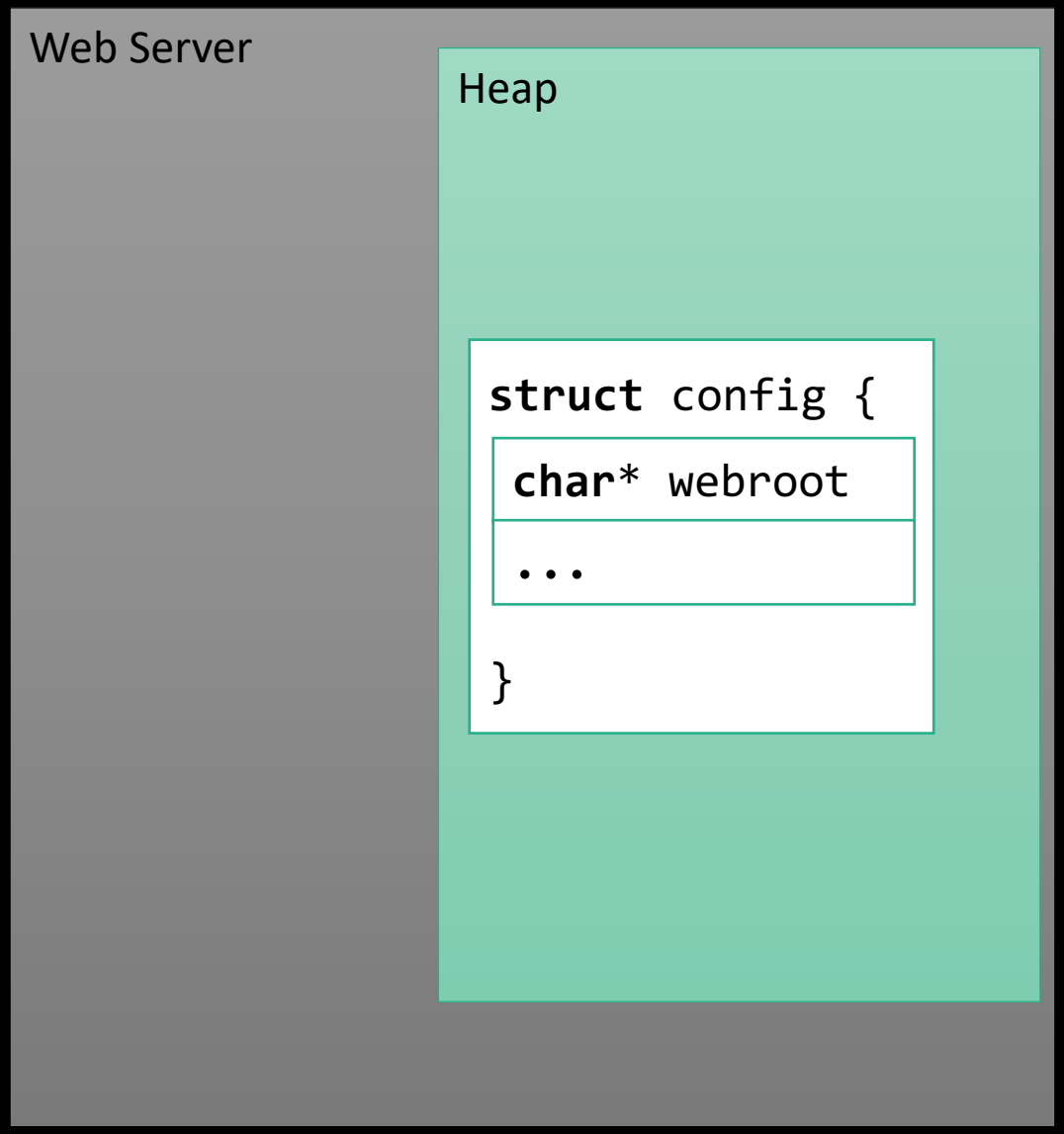


Same effect – control-data left untouched

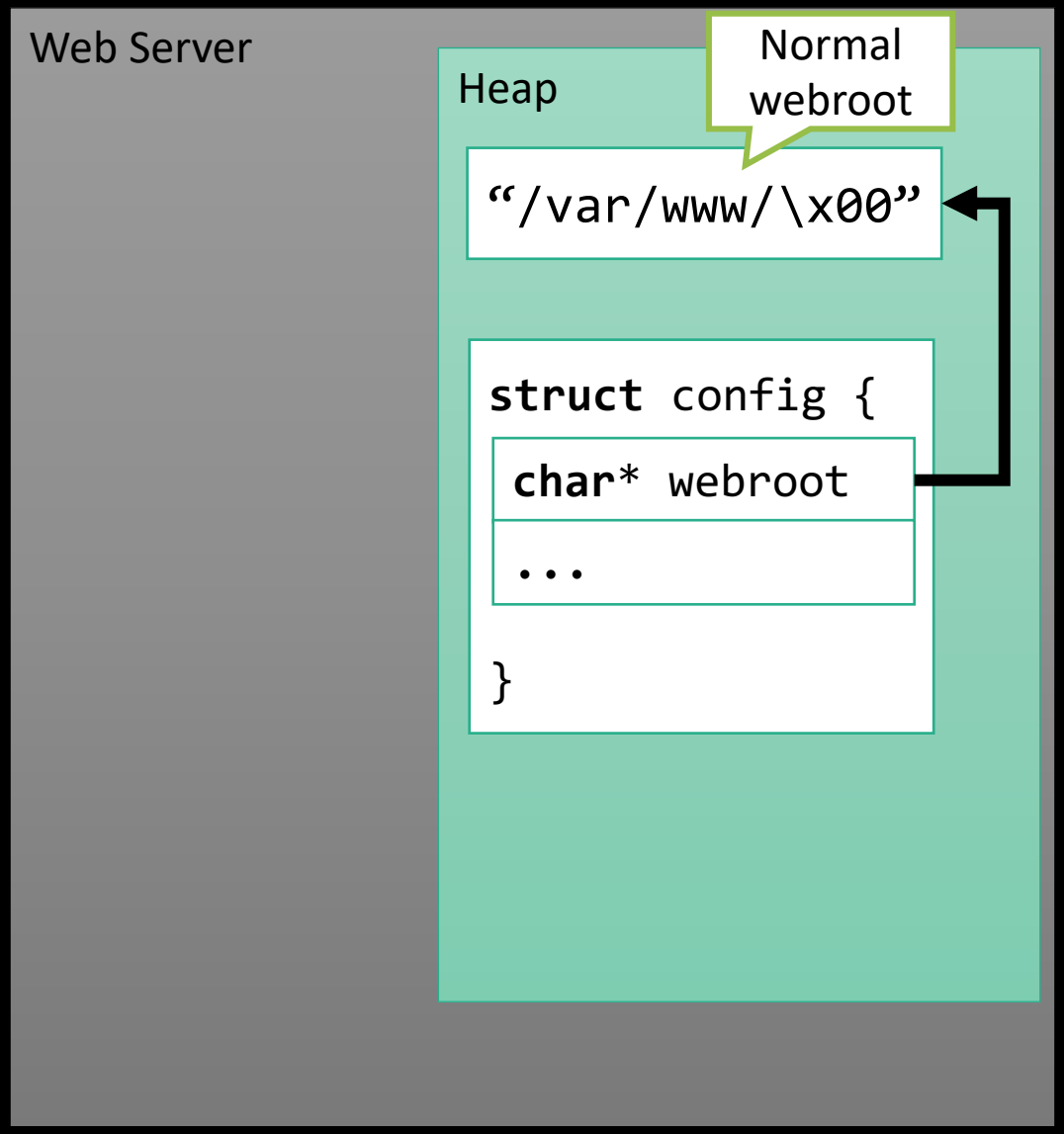
Corrupt Configuration Data



Corrupt Configuration Data

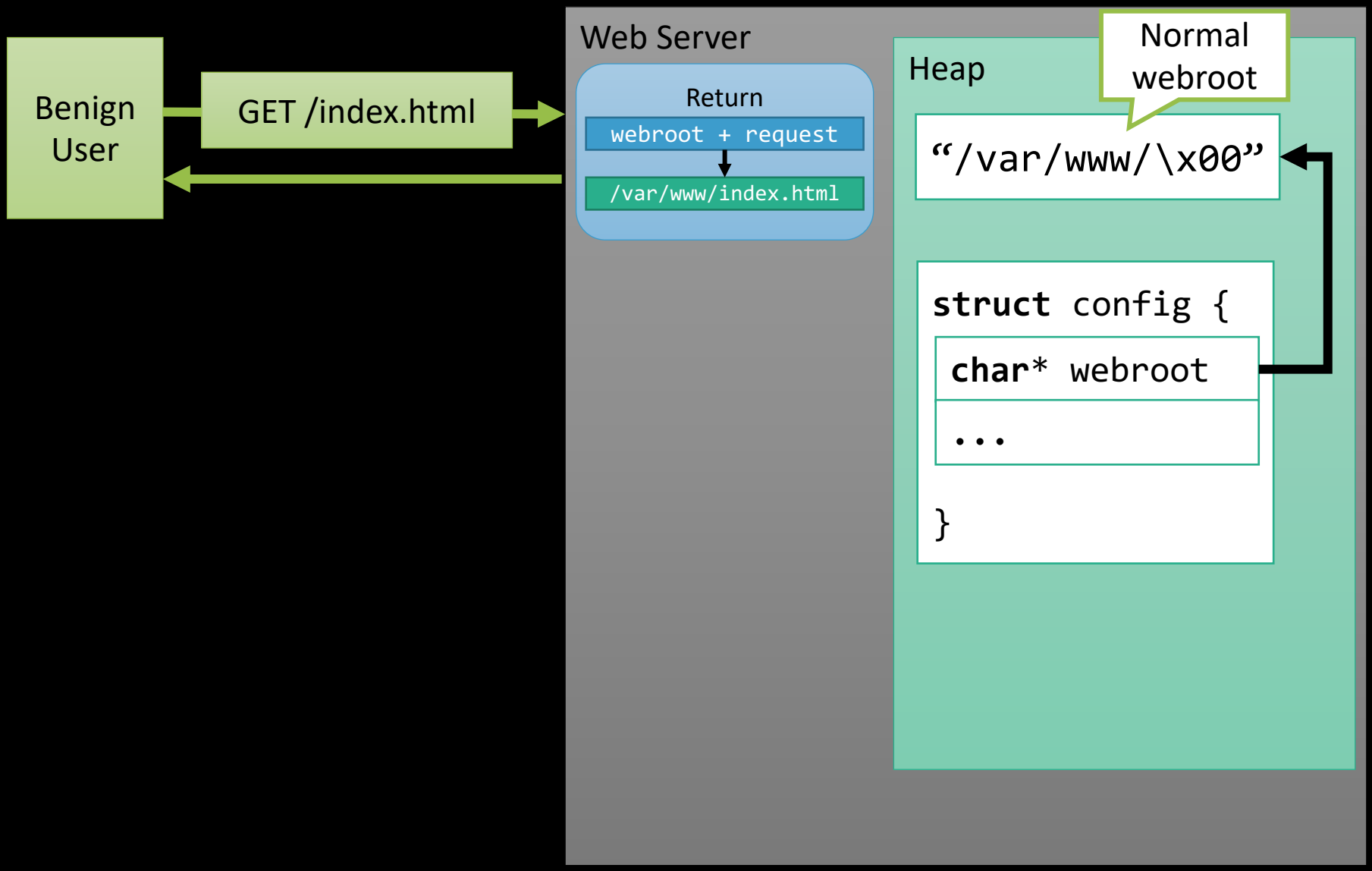


Corrupt Configuration Data

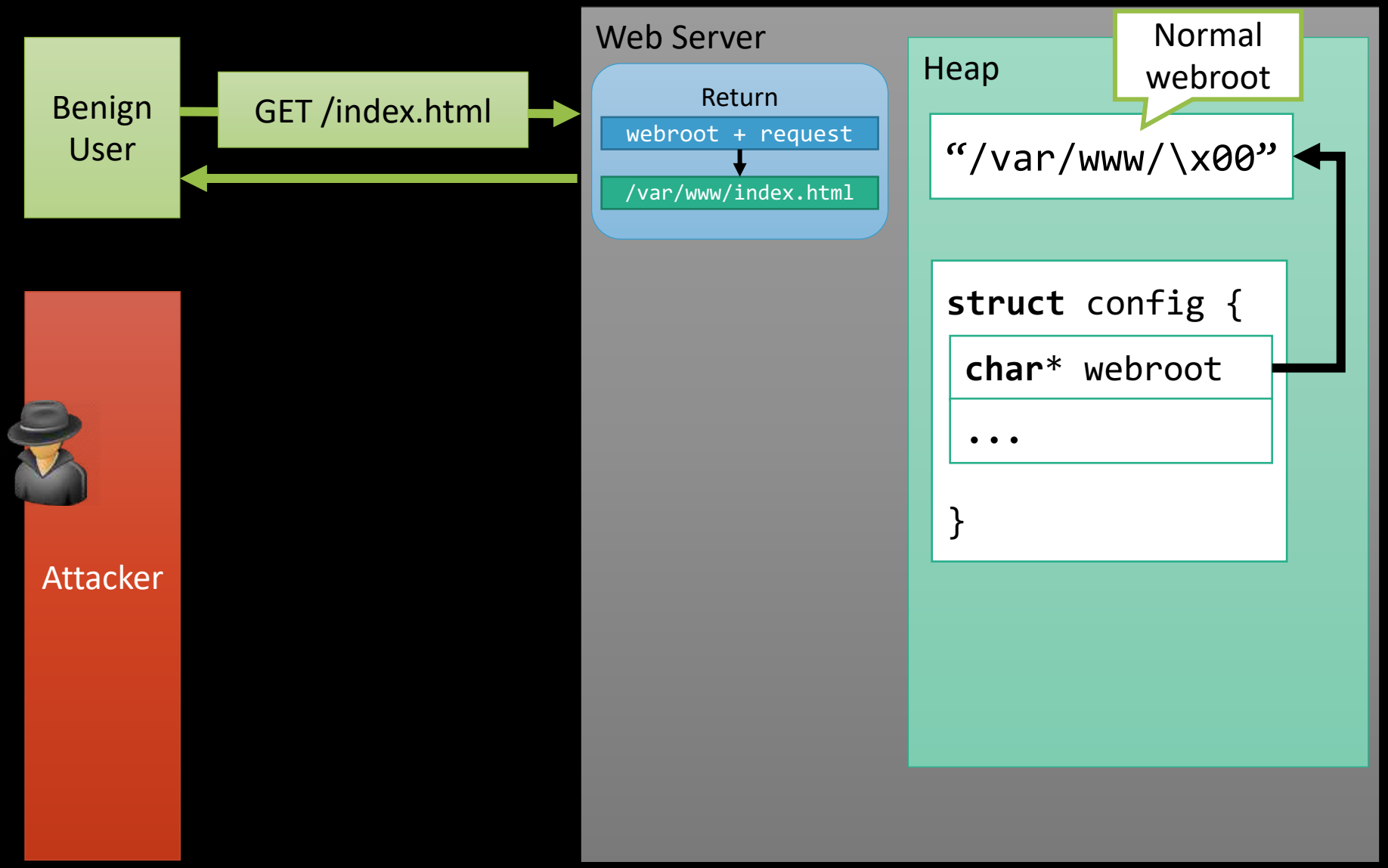


Corrupt Configuration Data

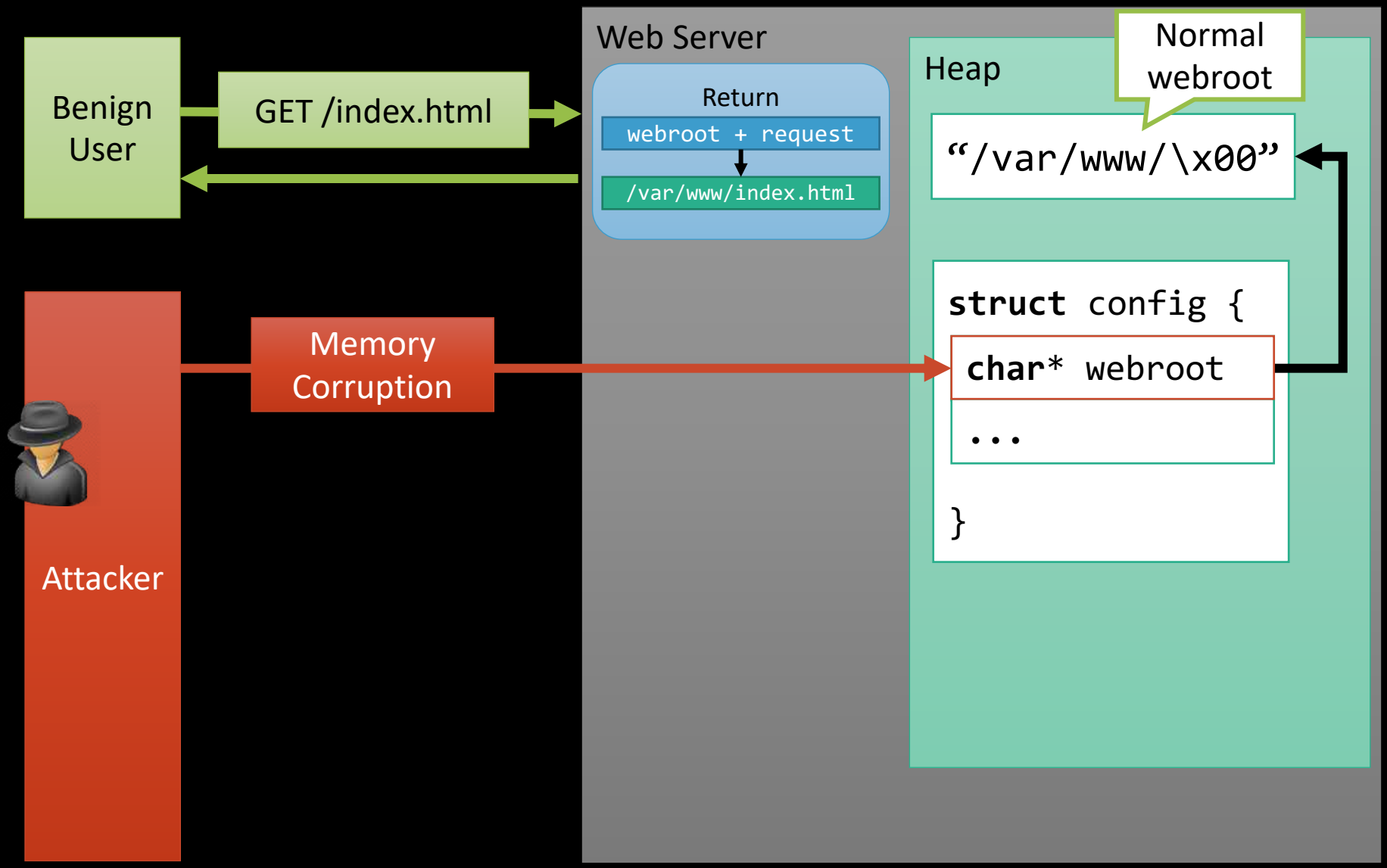
Corrupt Configuration Data



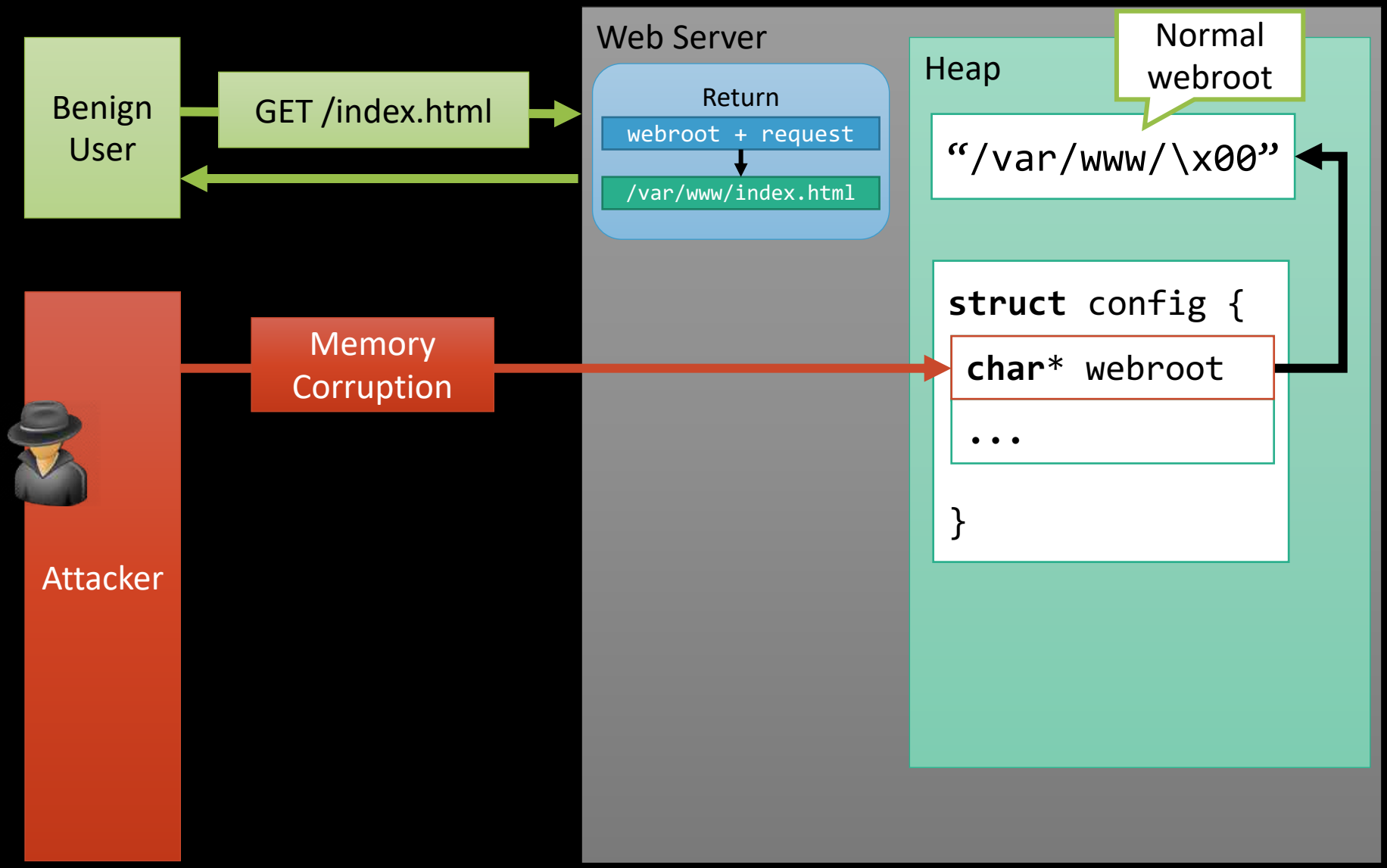
Corrupt Configuration Data



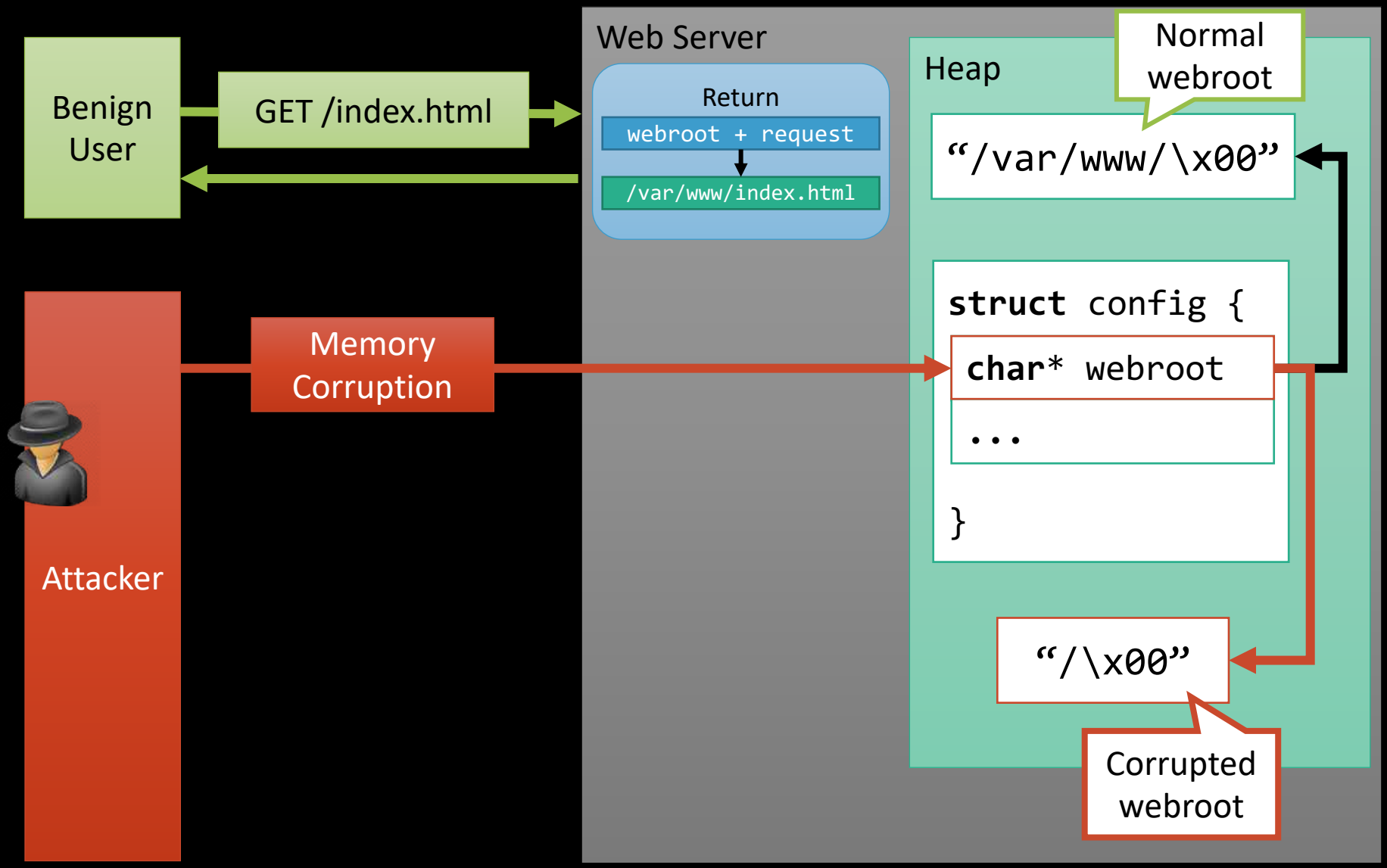
Corrupt Configuration Data



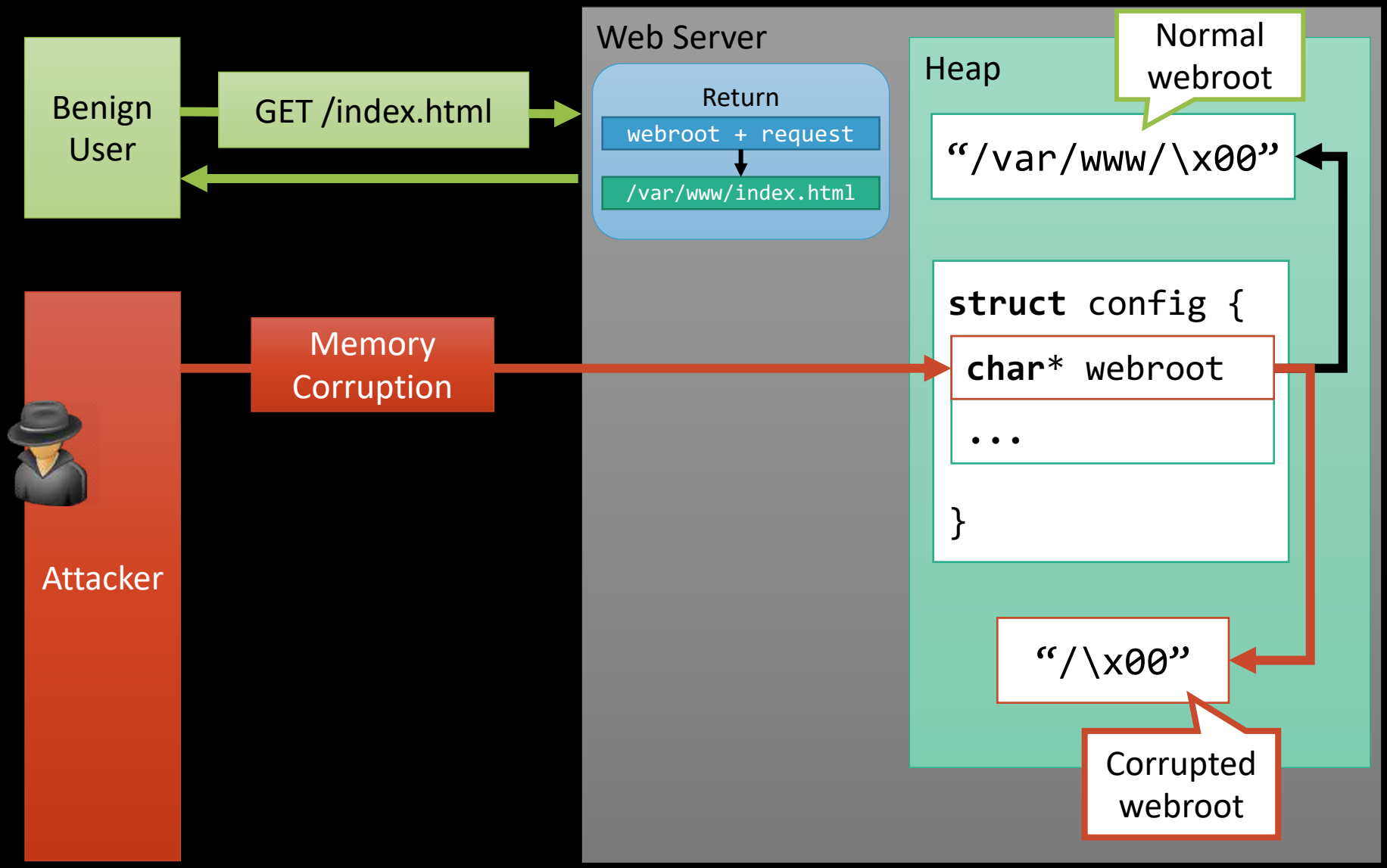
Corrupt Configuration Data



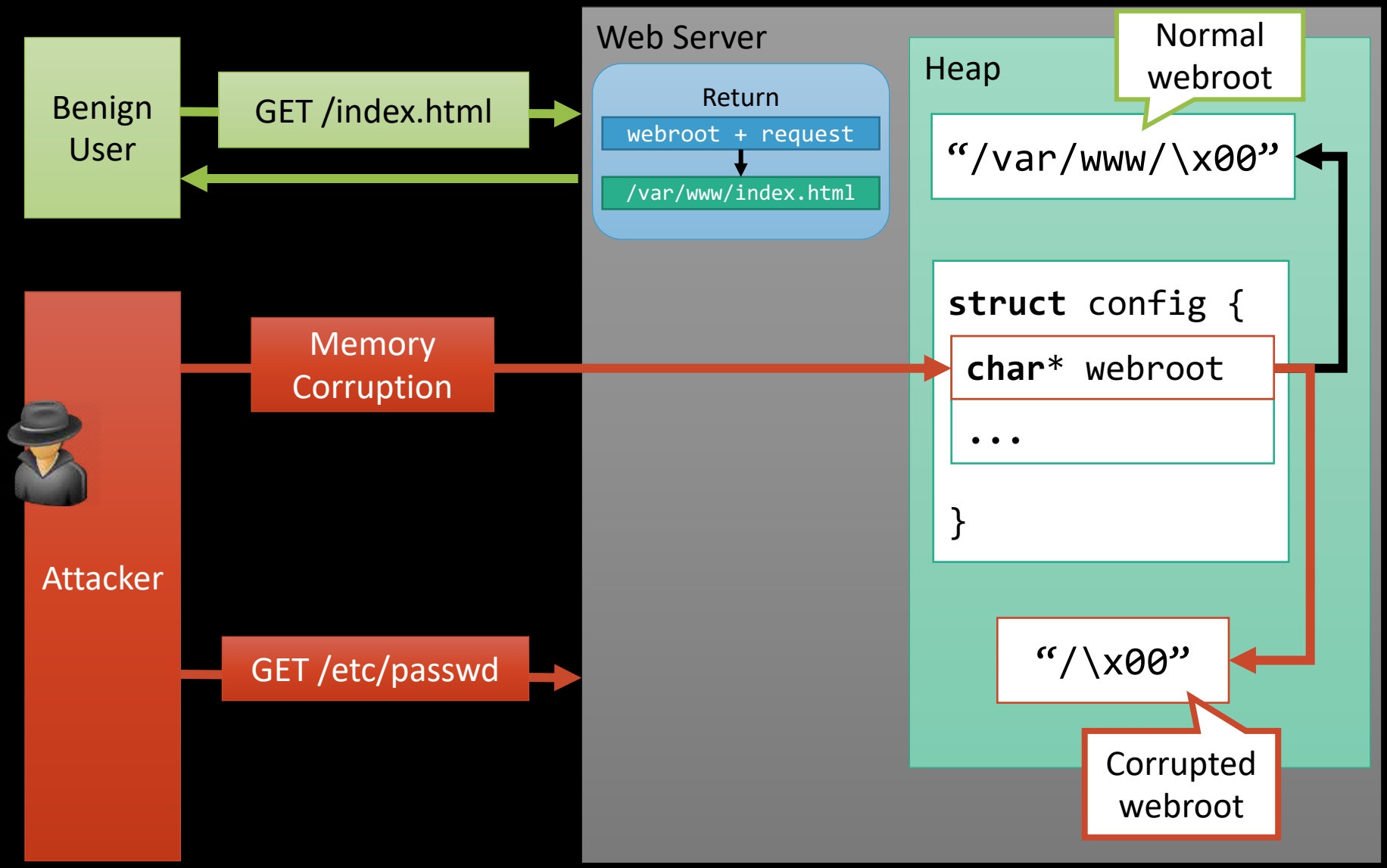
Corrupt Configuration Data



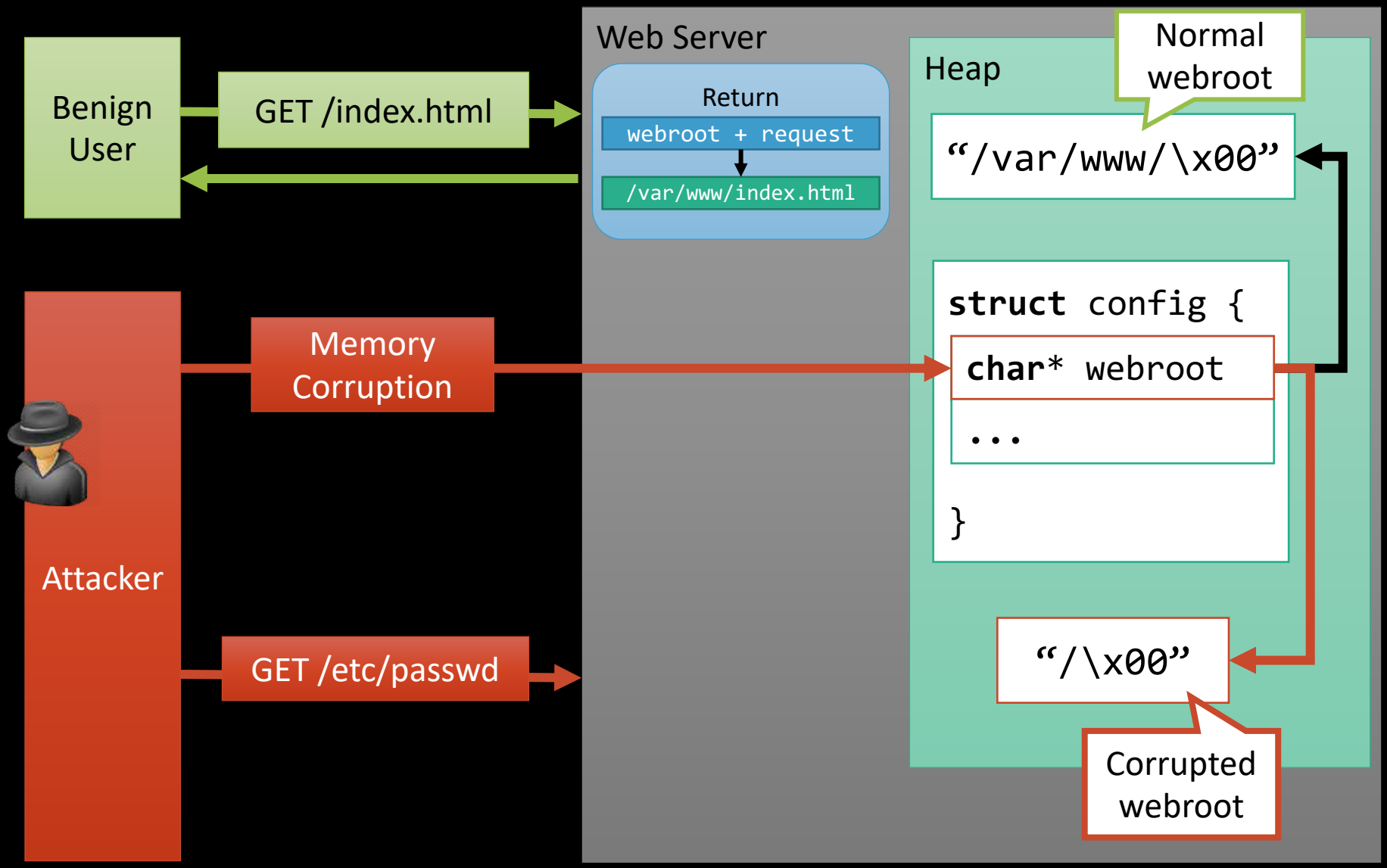
Corrupt Configuration Data



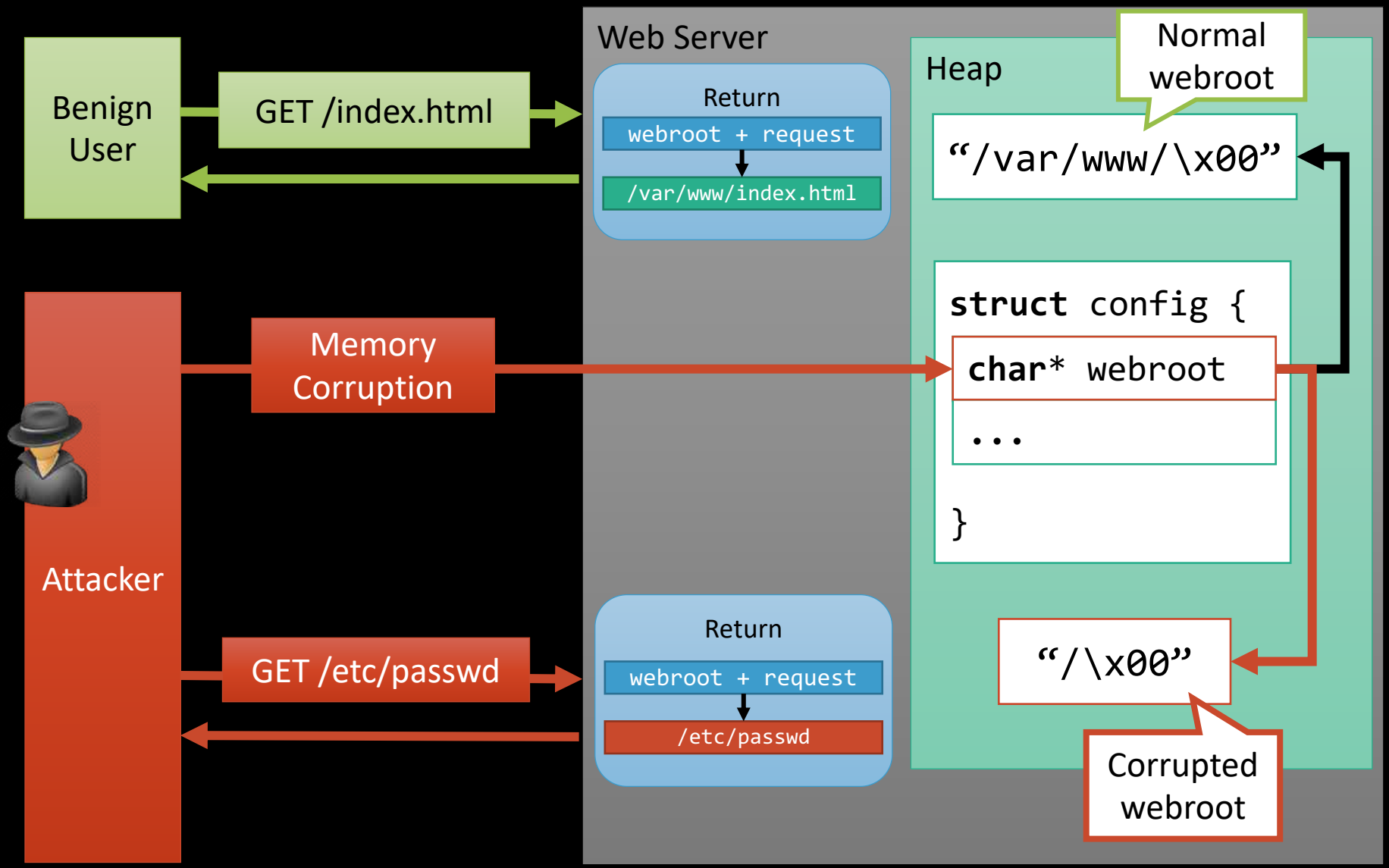
Corrupt Configuration Data



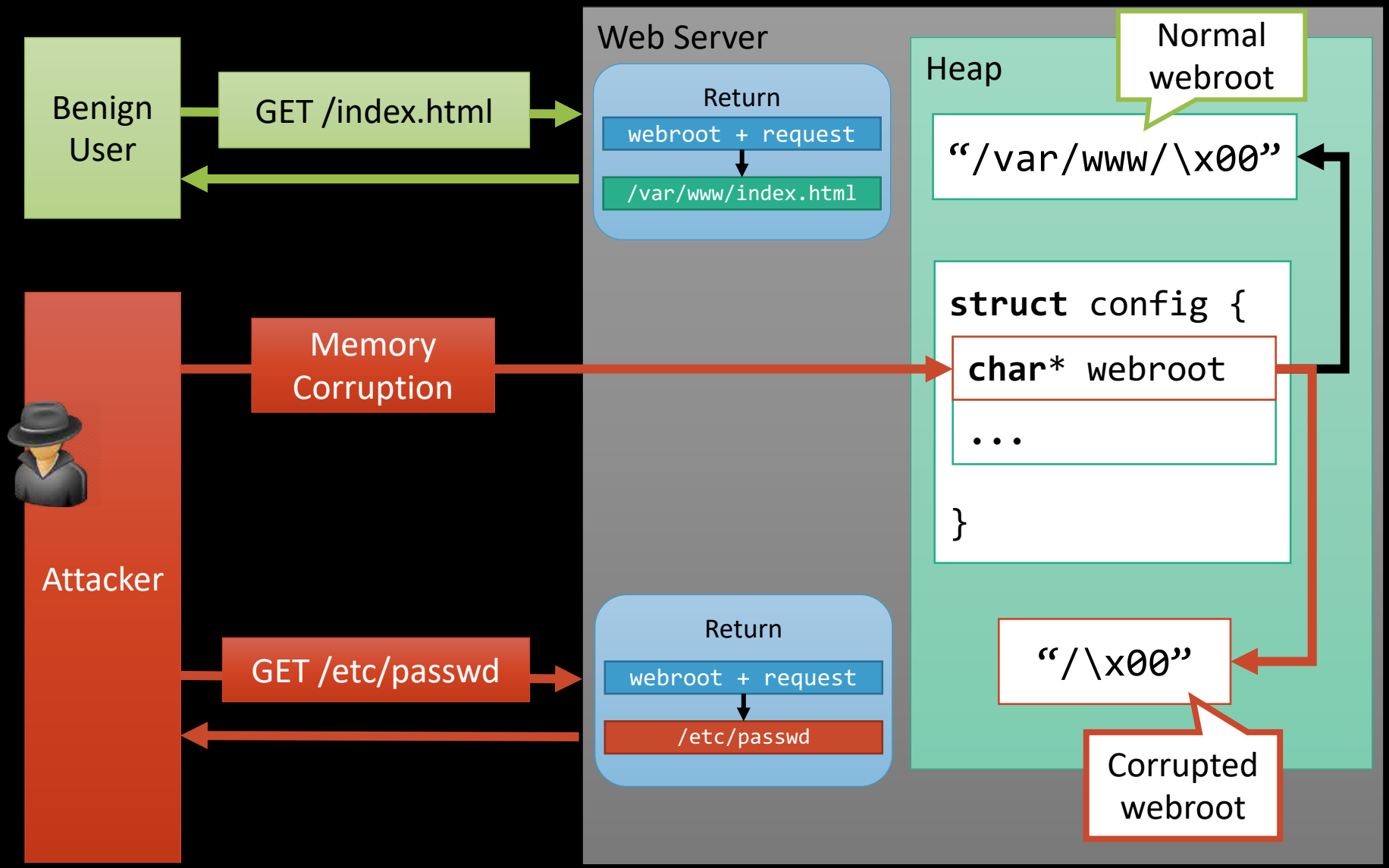
Corrupt Configuration Data



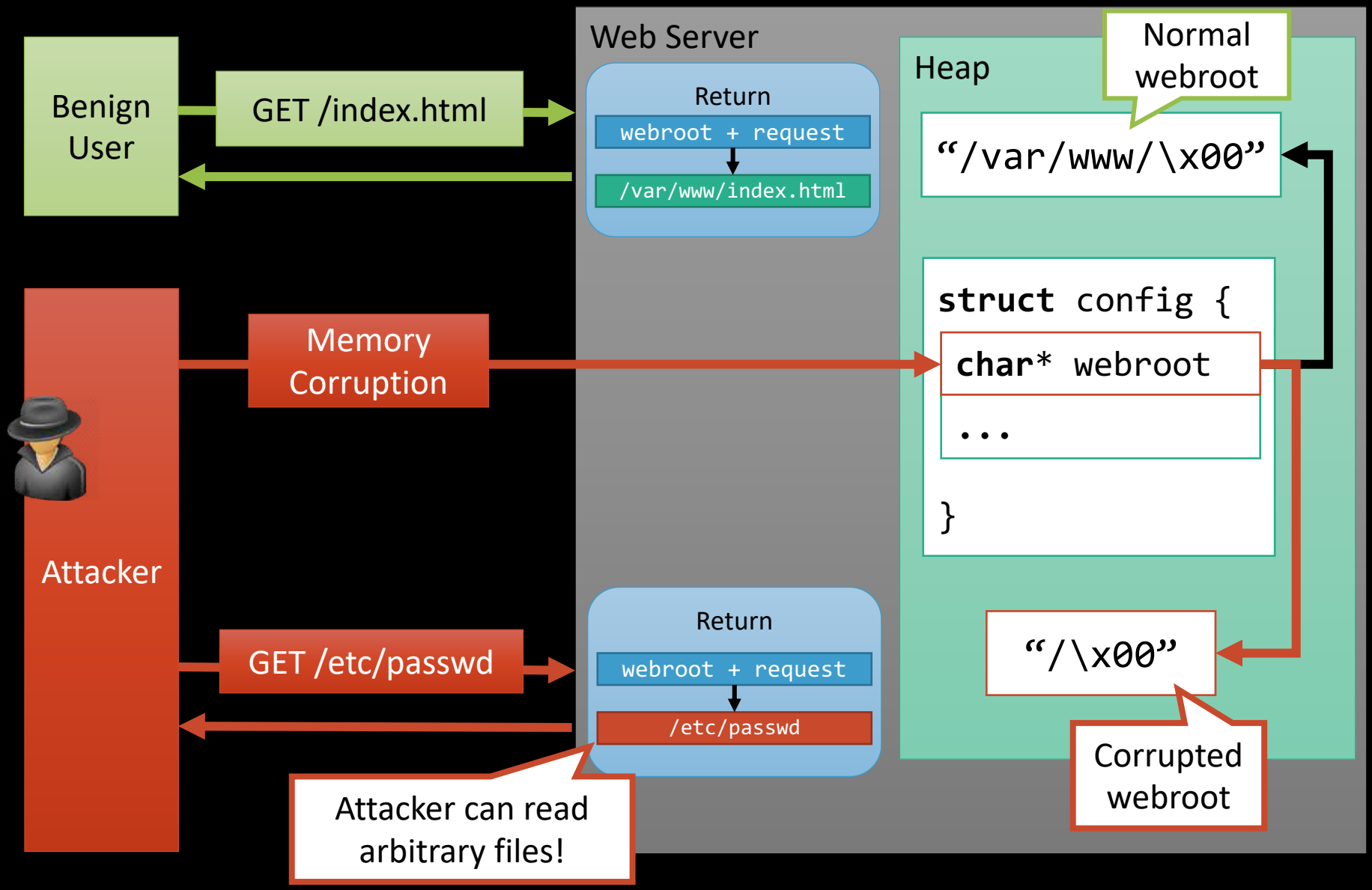
Corrupt Configuration Data



Corrupt Configuration Data



Corrupt Configuration Data



Generalization of data-only attacks: data-oriented programming [Hu et al., IEEE S&P 2016]

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

```
r0 = r0 + r1
```

DOP Virtual Registers

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

```
r0 = r0 + r1
```

Stored in target program memory.

DOP Virtual Registers

r0	r1	…

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

```
r0 = r0 + r1
```

C code	`srv->total += *size;`

Stored in target program memory.

DOP Virtual Registers

r0	r1	…

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

`r0 = r0 + r1`

Pointer dereference

C code	`srv->total += *size;`

Stored in target program memory.

DOP Virtual Registers

r0	r1	…

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

`r0 = r0 + r1`

Pointer dereference

C code	`srv->total += *size;`

Stored in target program memory.

DOP Virtual Registers

r0	r1	…

DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]

Add Gadget

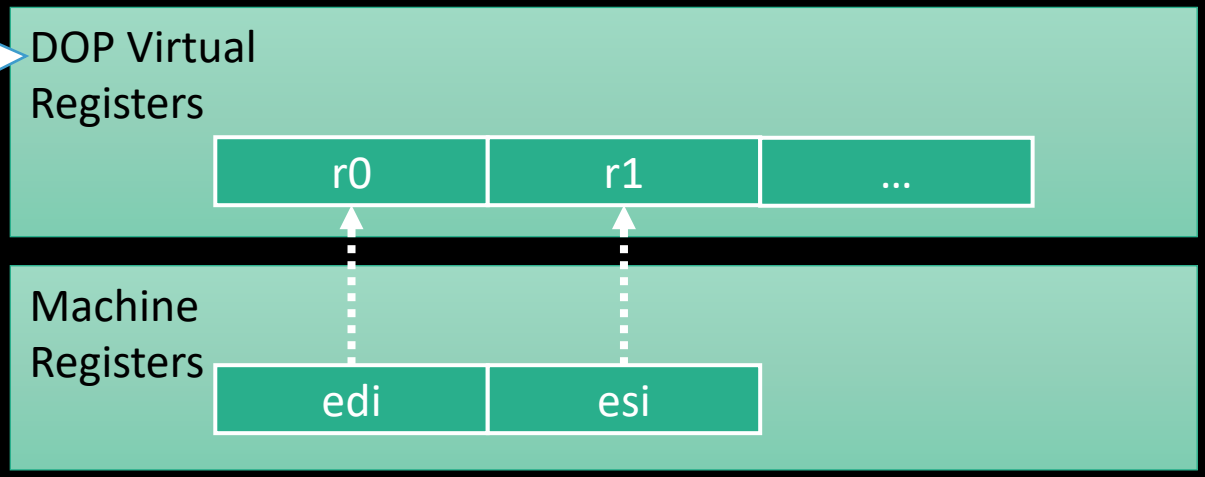
`r0 = r0 + r1`

Pointer dereference

C code	`srv->total += *size;`

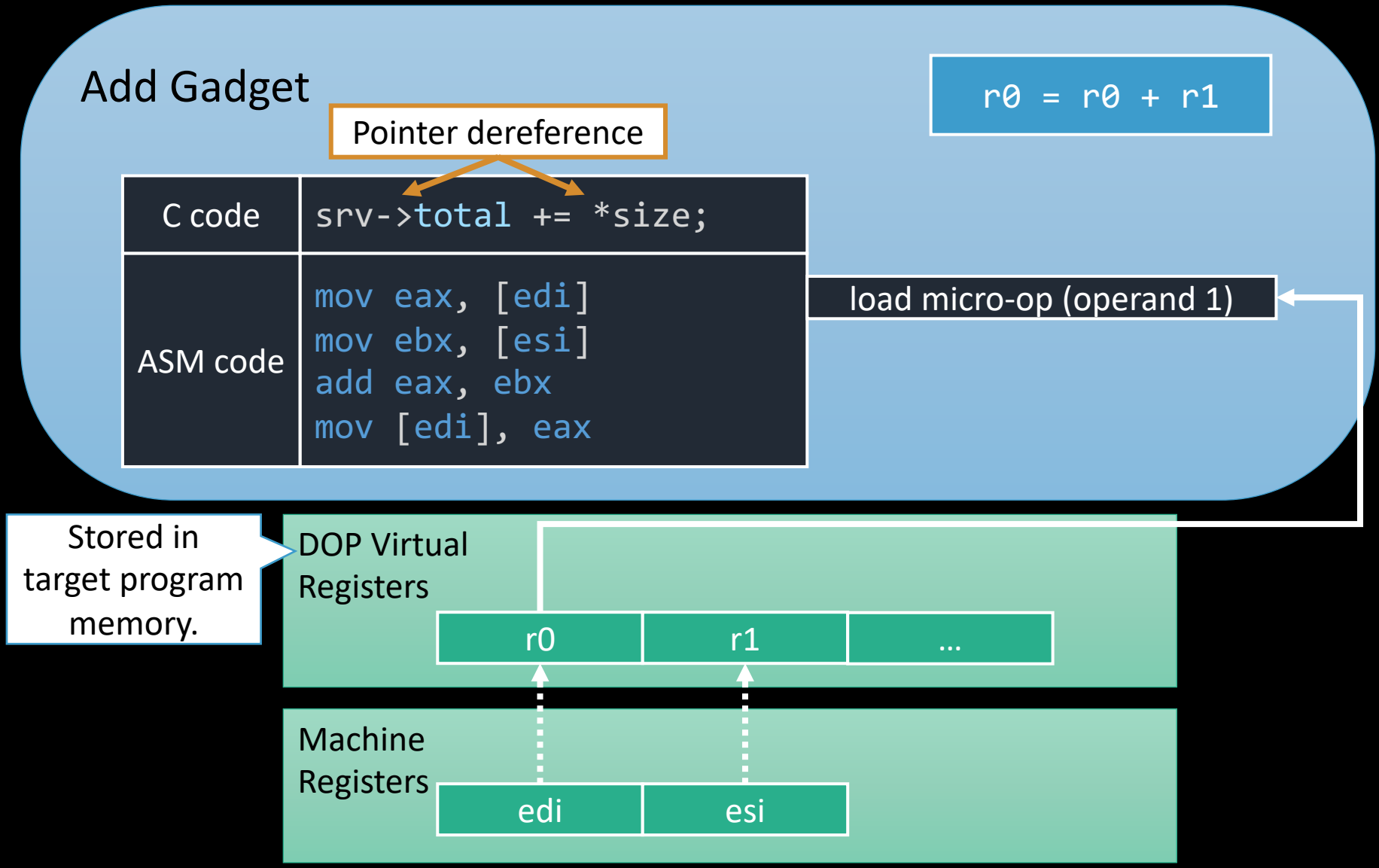
ASM code	`mov eax, [edi]`
	`mov ebx, [esi]`
	`add eax, ebx`
	`mov [edi], eax`

Stored in target program memory.



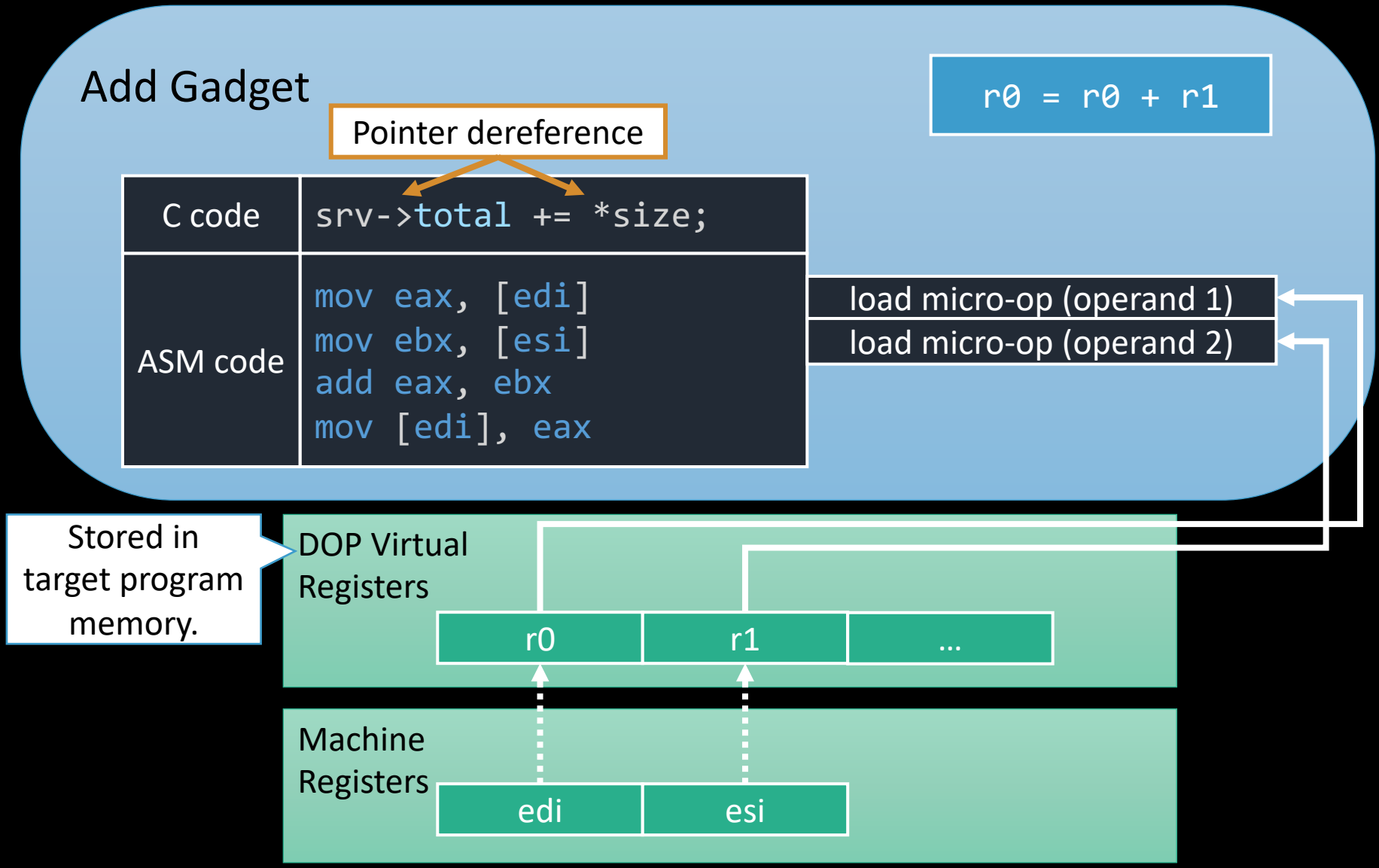
DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]



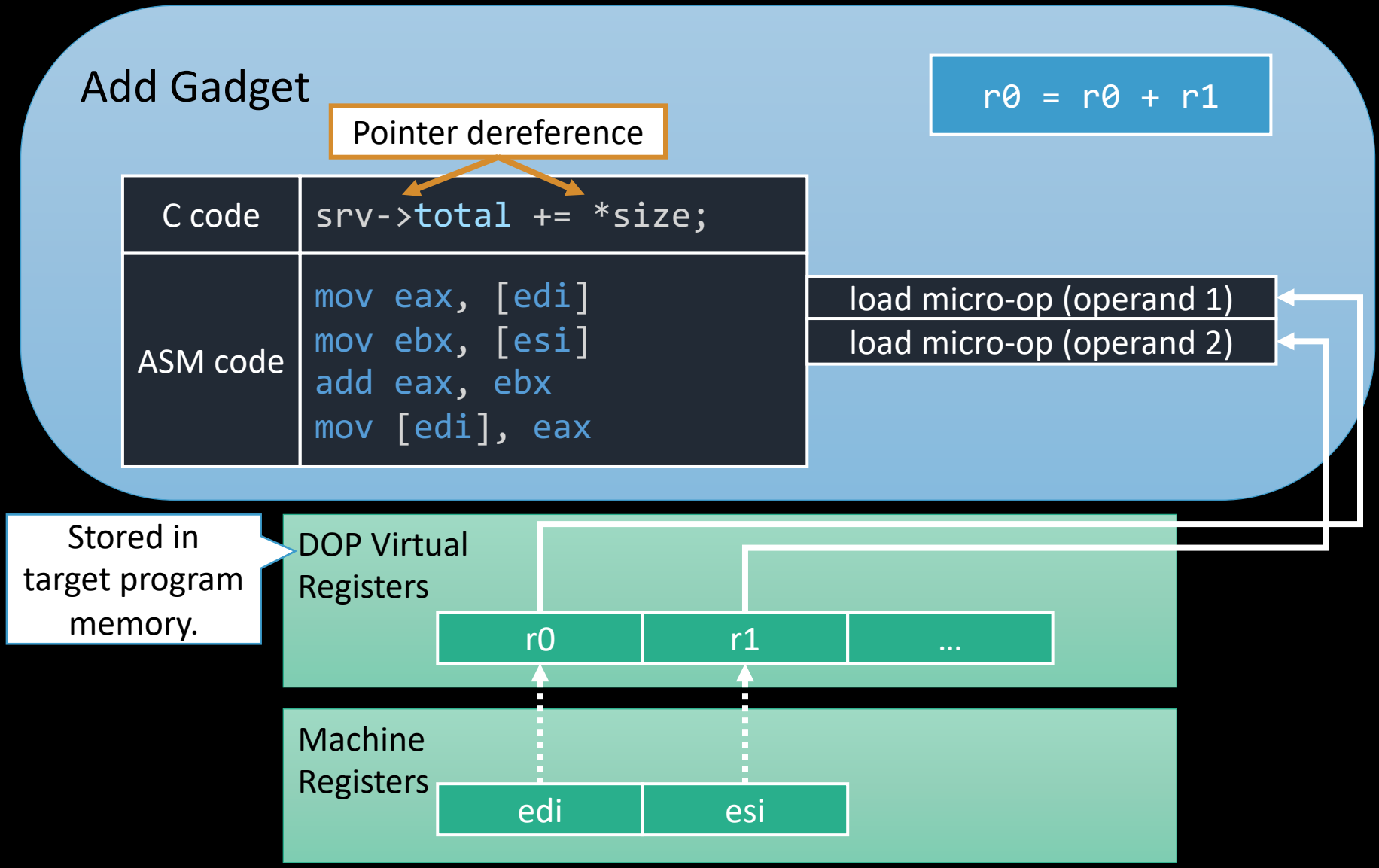
DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]



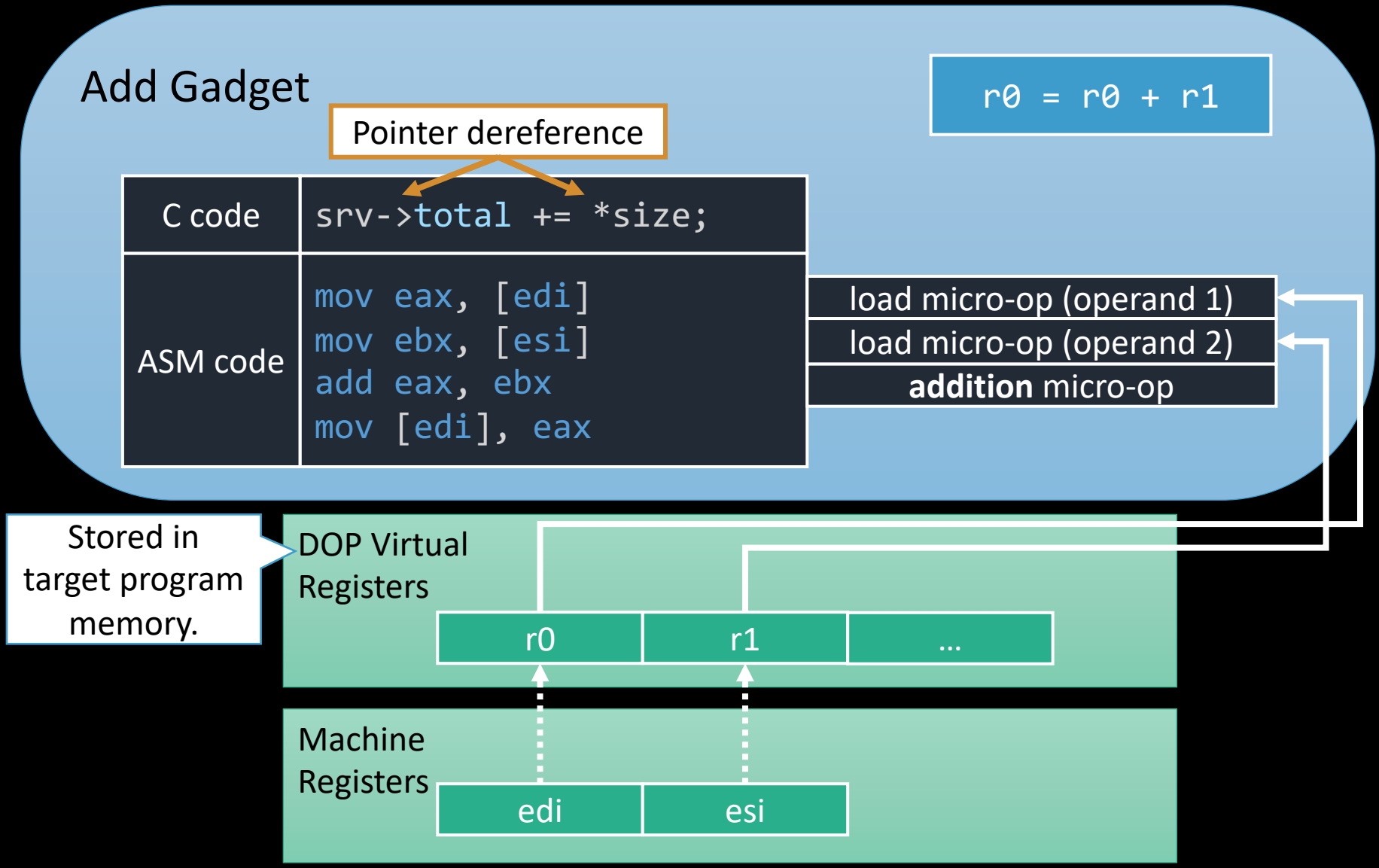
DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]



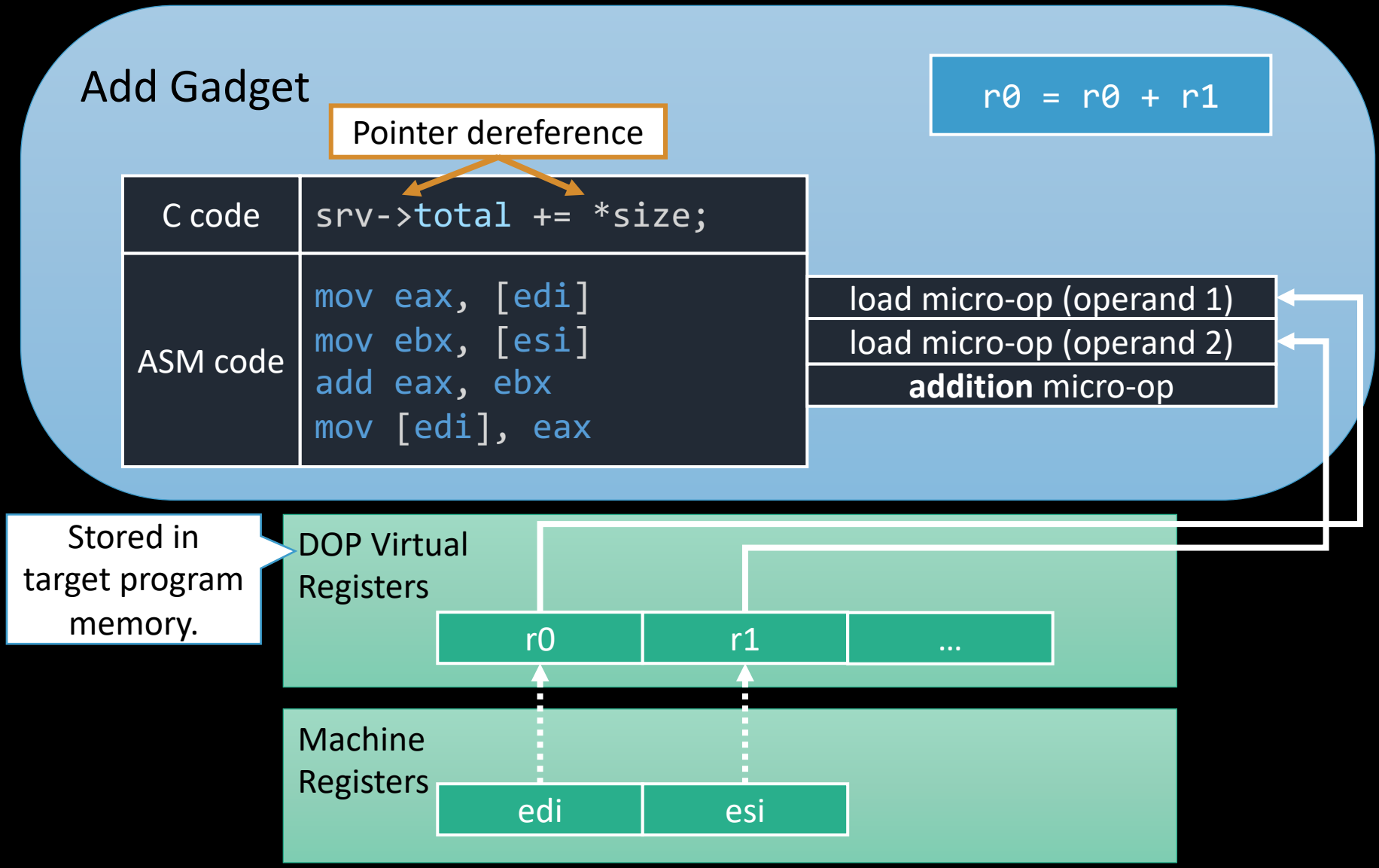
DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]



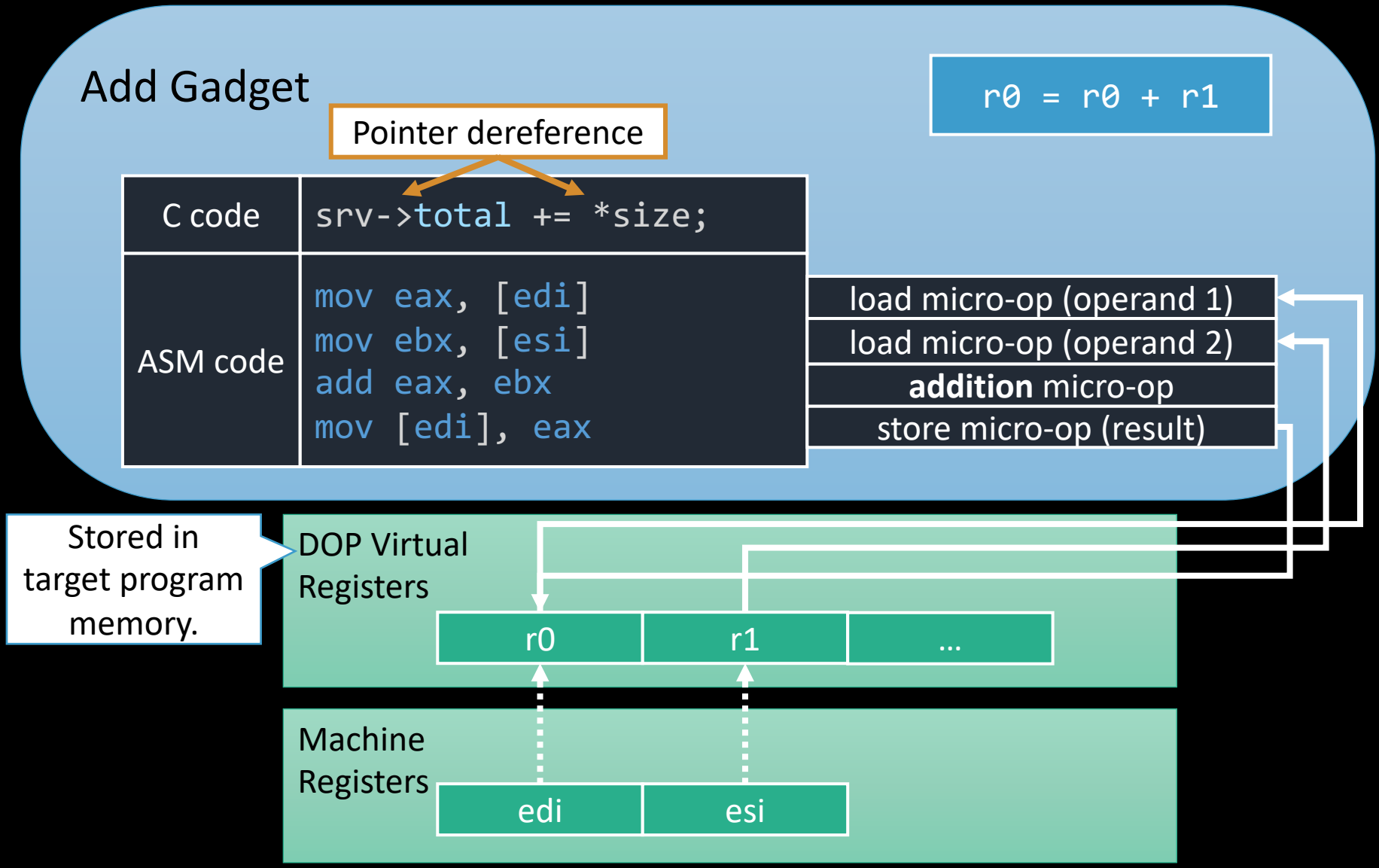
DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]



DOP Add Gadget Example

[Hu et al., IEEE S&P 2016]



Discussion on Defenses

- Possible defense techniques:
  - Data-Flow Integrity (DFI) [Castro et al., OSDI 2006]
  - Write-Integrity Testing (WIT) [Akritidis et al., IEEE S&P 2008]
- DFI and WIT suffer from similar issues as CFI
  - High performance overhead
  - Depends on the precision of the data-flow graph

Hands-On Lab

Download and install the VirtualBox VM:

http://bit.do/secsummer19

UNIVERSITÄT DUISBURG ESSEN

Open-Minded