Everything you code can and will be re-used against you: On run-time attacks and defenses

instead of

SYSTEM FAILURE

Ahmad-Reza Sadeghi,
Technische Universität Darmstadt,
Intel Collaborative Research Institute for Collaborative & Resilient Autonomous Systems

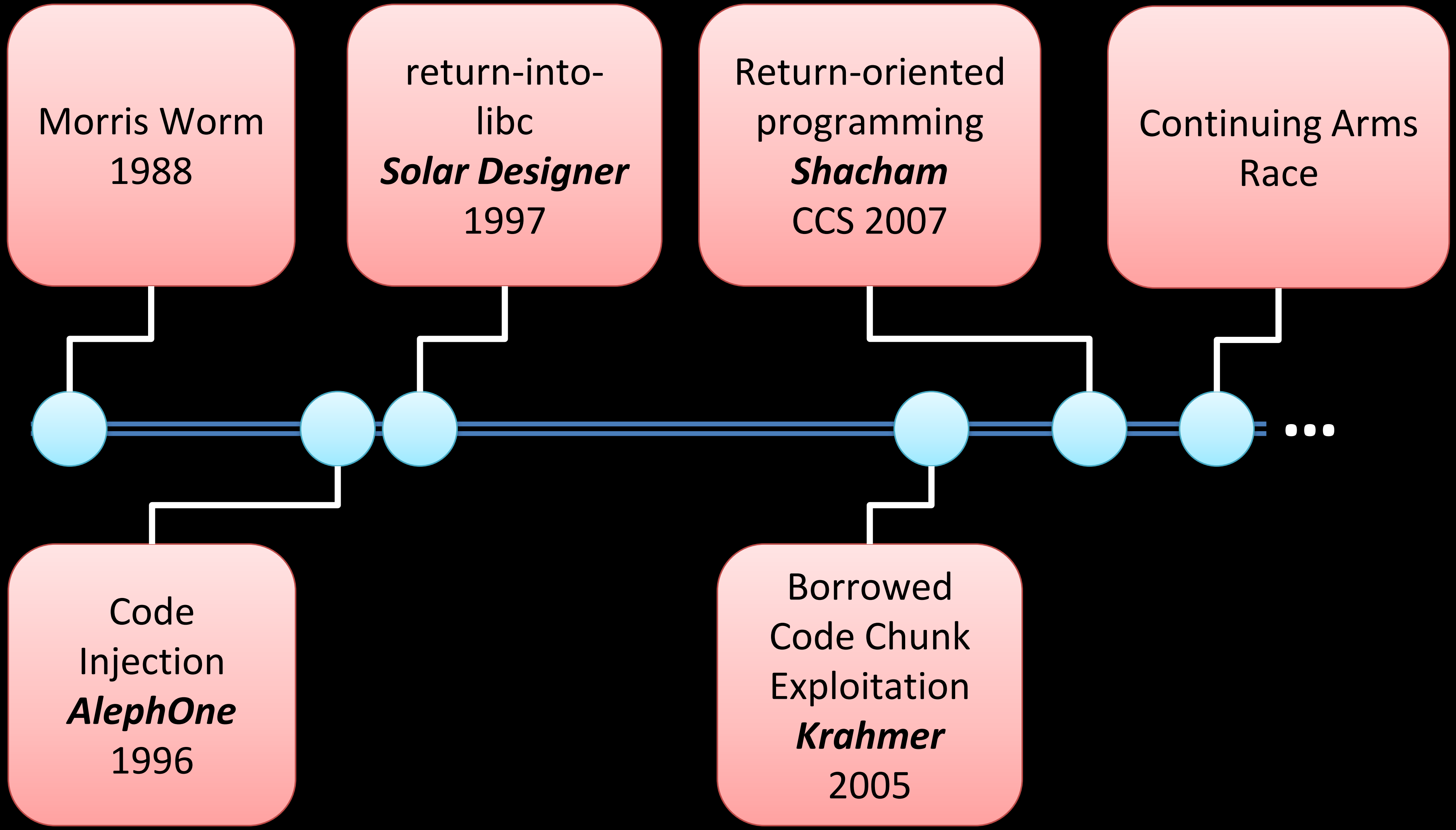
Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

Collaborators: Acknowledgement

- Luca Davi, Essen-Duisburg University, GE
- Ferdinand Brasser, Tommaso Frassetto, Christopher Liebchen, TU Darmstadt, GE
- N. Asokan, Aalto, FI
- Fabian Monrose, Kevin Snow, UNC Chapel Hill, US
- Hovav Shacham, UCSD, US
- Per Larsen, Steven Crane, Andrei Homescu, Gene Tsudik, Michael Franz, UCI, US
- Thorsten Holz, Bochum University, GE
- Felix Shuster, Microsoft Research, UK
- Yier Jin, Dean Sullivan, Orlando Arias, UCF, US
- Patrick Kroebel, Matthias Schunter, Intel Labs
- Georg Koppen, Tor Project

Motivation

Three Decades of Runtime Attacks



Morris Worm
1988

Code
Injection
AlephOne
1996

return-into-libc
Solar Designer
1997

Borrowed
Code Chunk
Exploitation
Krahmer
2005

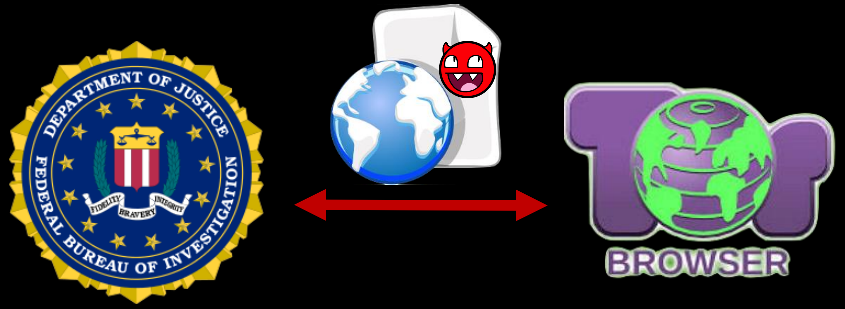
Return-oriented
programming
Shacham
CCS 2007

Continuing Arms
Race

Recent Attacks

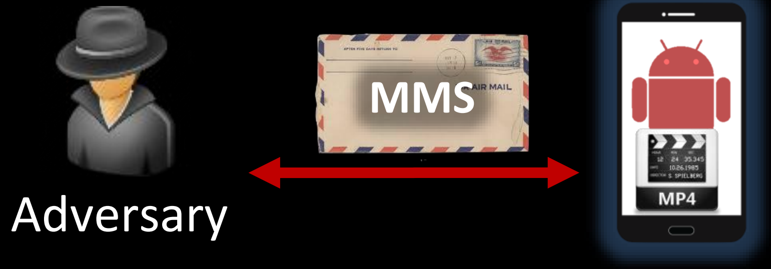
Attacks on Tor Browser [2013]

FBI Admits It Controlled Tor Servers Behind Mass Malware Attack.



Stagefright [Drake, BlackHat 2015]

These issues in Stagefright code critically expose 95% of Android devices, an estimated 950 million devices

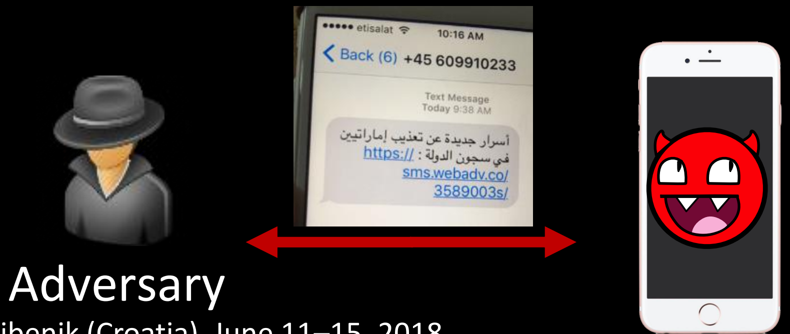


Cisco Router Exploit [2018]

*Million CISCO ASA Firewalls potentially vulnerable to attacks (*XML parsing vuln.*)*

The Million Dollar Dissident [2016]

Government targeted human rights defender with a chain of zero-day exploits to infect his iPhone with spyware.



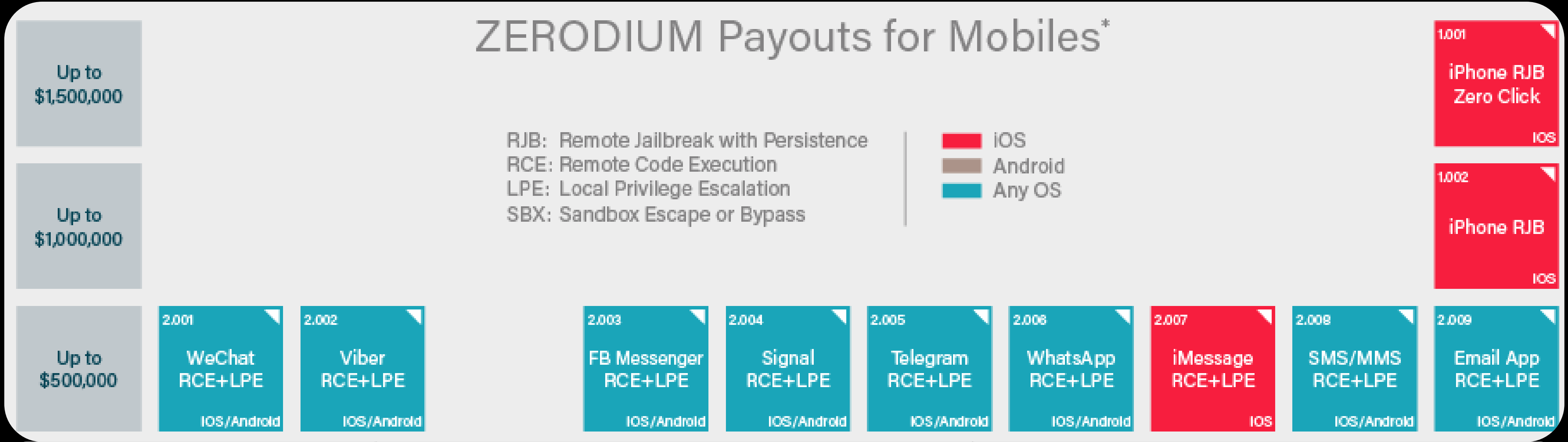
etisalat 10:16 AM

Back (6) +45 609910233

Text Message
Today 9:38 AM

أسرار جديدة عن تعذيب إماراتيين
في سجون الدولة : https://
sms.webadv.co/
3589003s/

Exploit Acquisition



ZERODIUM Payouts for Mobiles*

RJB: Remote Jailbreak with Persistence
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass

🟥 iOS
🟫 Android
🟦 Any OS

Up to $1,500,000

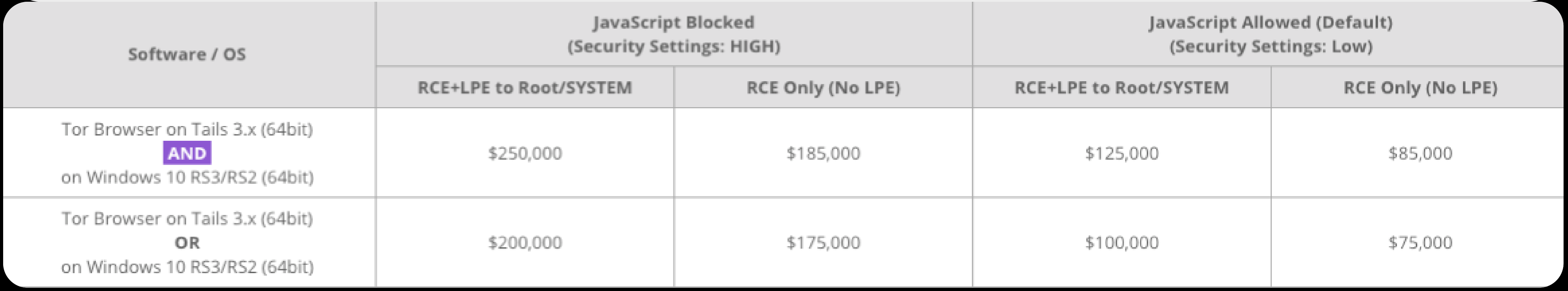
1.001 iPhone RJB Zero Click iOS

Up to $1,000,000

1.002 iPhone RJB iOS

Up to $500,000

2.001 WeChat RCE+LPE iOS/Android
2.002 Viber RCE+LPE iOS/Android
2.003 FB Messenger RCE+LPE iOS/Android
2.004 Signal RCE+LPE iOS/Android
2.005 Telegram RCE+LPE iOS/Android
2.006 WhatsApp RCE+LPE iOS/Android
2.007 iMessage RCE+LPE iOS
2.008 SMS/MMS RCE+LPE iOS/Android
2.009 Email App RCE+LPE iOS/Android

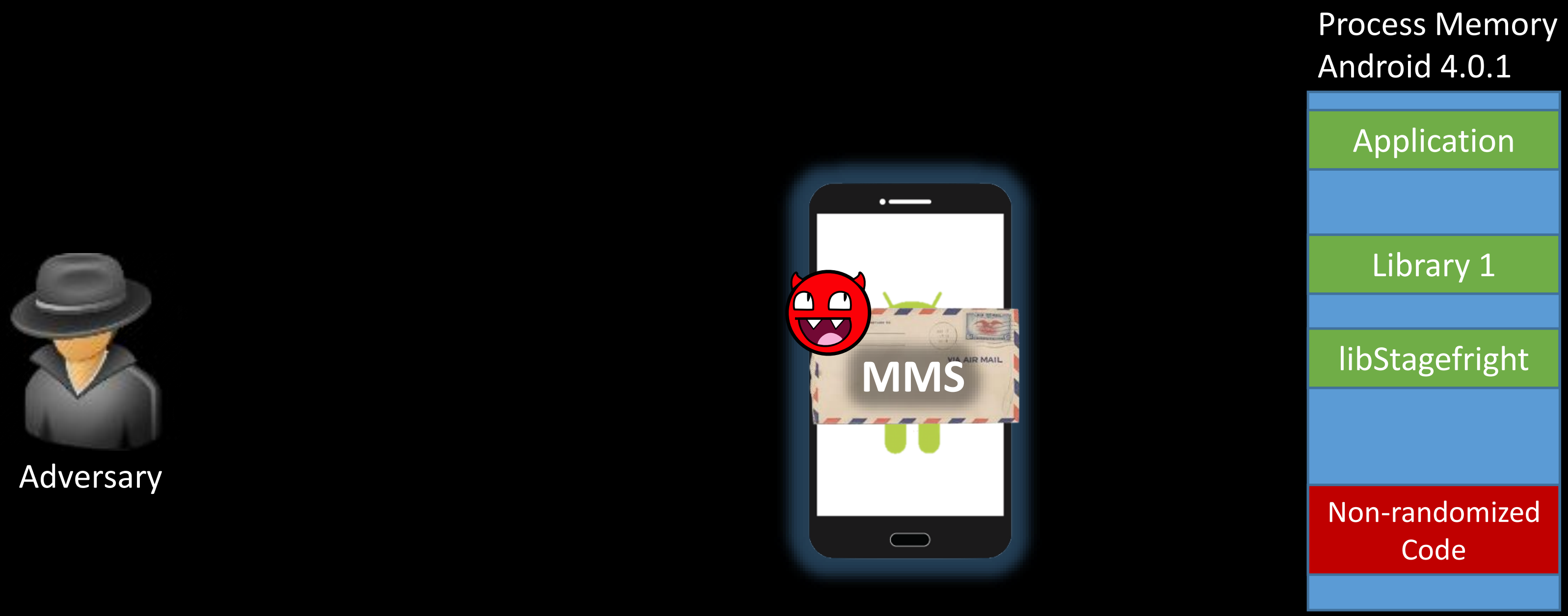


Software / OS	JavaScript Blocked (Security Settings: HIGH)		JavaScript Allowed (Default) (Security Settings: Low)	
	RCE+LPE to Root/SYSTEM	RCE Only (No LPE)	RCE+LPE to Root/SYSTEM	RCE Only (No LPE)
Tor Browser on Tails 3.x (64bit) AND on Windows 10 RS3/RS2 (64bit)	$250,000	$185,000	$125,000	$85,000
Tor Browser on Tails 3.x (64bit) OR on Windows 10 RS3/RS2 (64bit)	$200,000	$175,000	$100,000	$75,000

Remote Android Vulnerability
Case: Stagefright

Adversary
MMS

Remote Android Vulnerability
Case: Stagefright



Adversary

MMS

Process Memory
Android 4.0.1

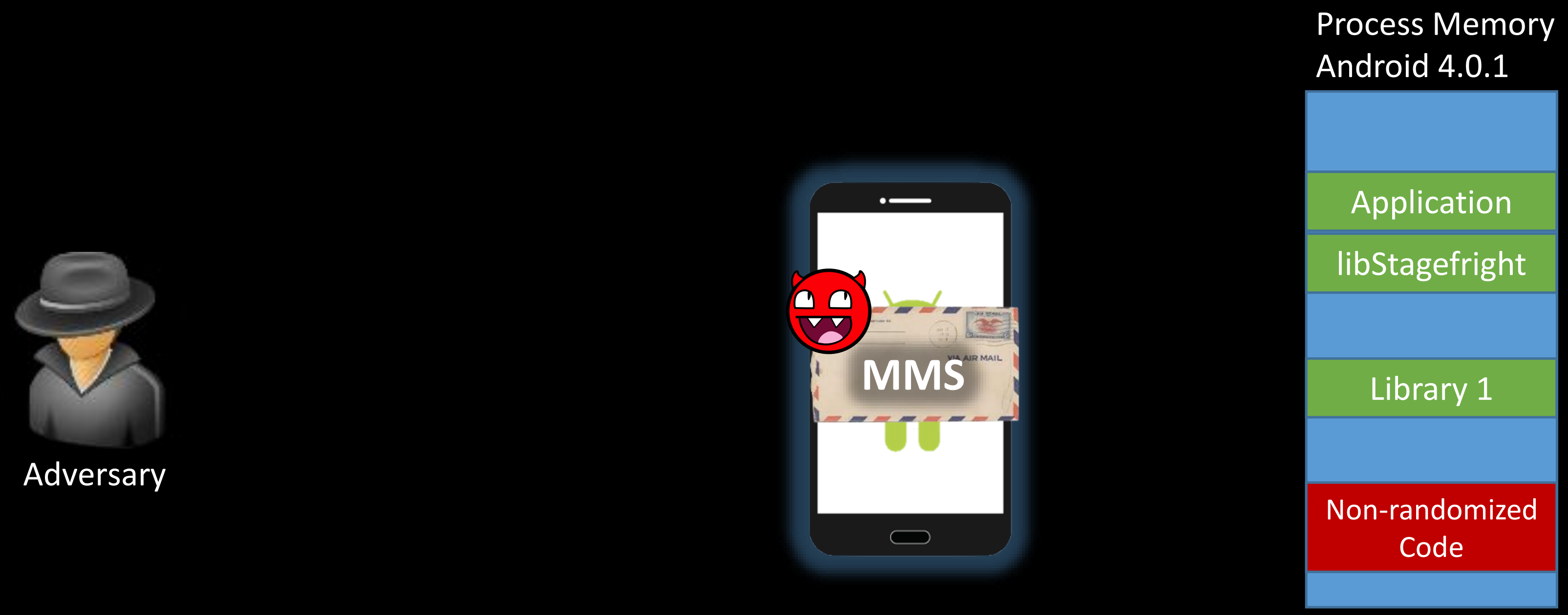
Application

Library 1

libStagefright

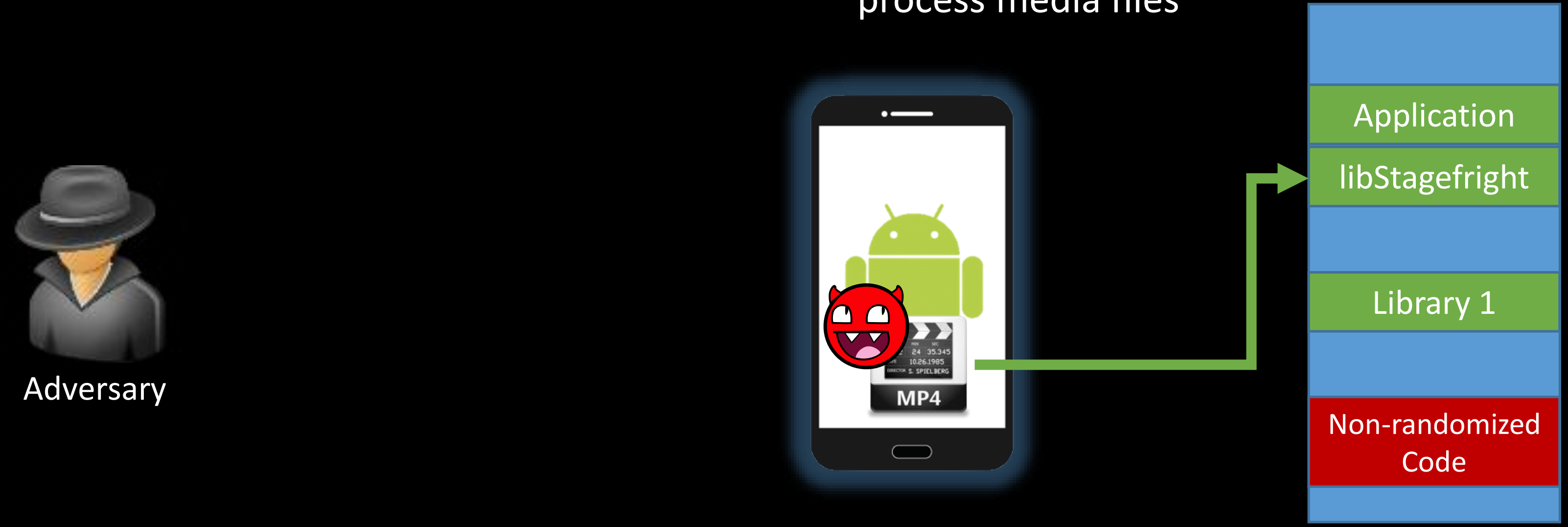
Non-randomized Code

Remote Android Vulnerability
Case: Stagefright



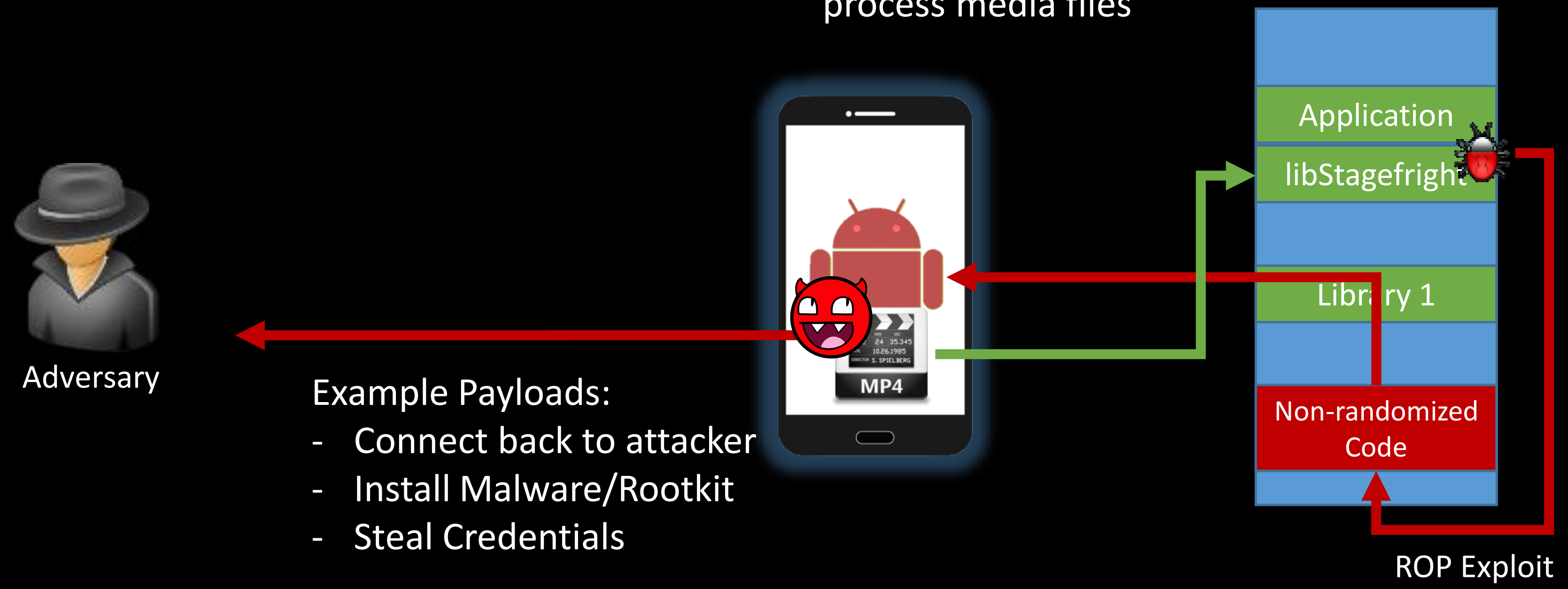
Remote Android Vulnerability
Case: Stagefright

libStagefright: Native Android library that can be used by Apps to process media files



Remote Android Vulnerability Case: Stagefright

libStagefright: Native Android library that can be used by Apps to process media files

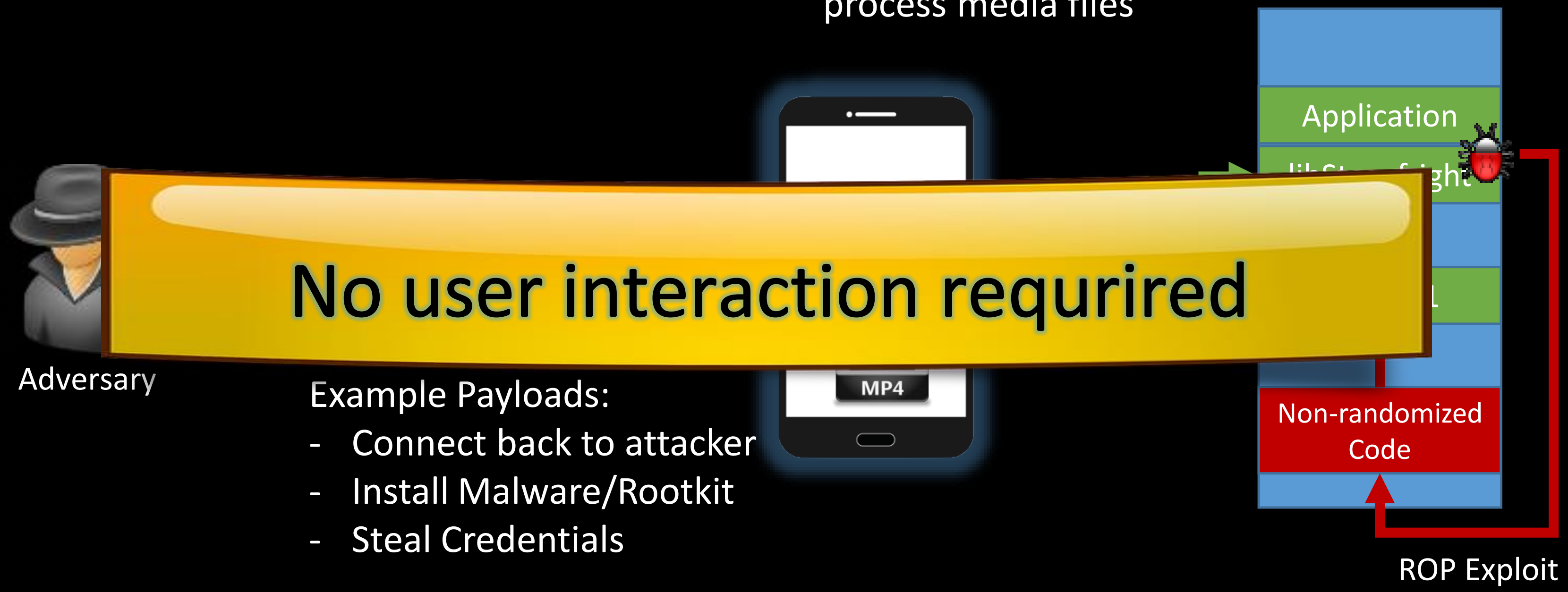


Example Payloads:

- Connect back to attacker
- Install Malware/Rootkit
- Steal Credentials

Remote Android Vulnerability Case: Stagefright

libStagefright: Native Android library that can be used by Apps to process media files

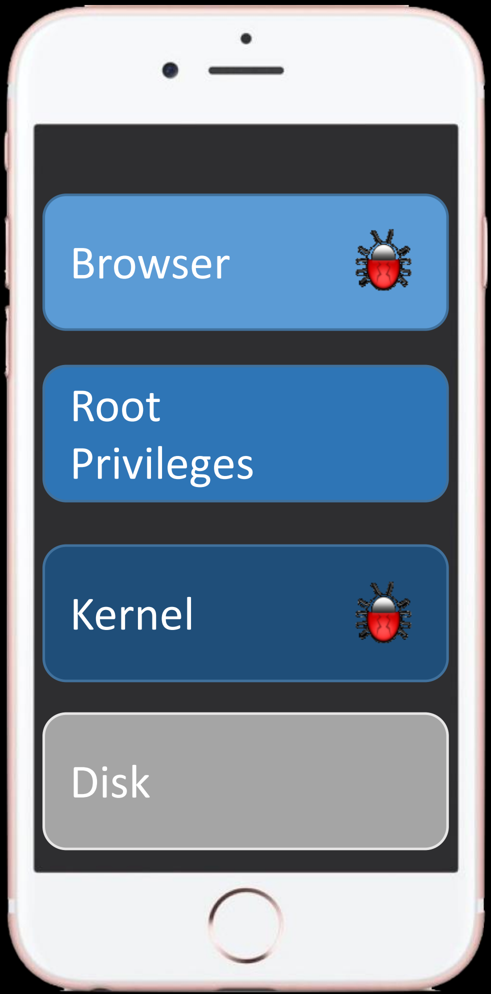


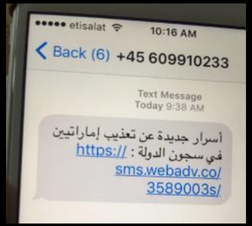
No user interaction requrired

Example Payloads:

- Connect back to attacker
- Install Malware/Rootkit
- Steal Credentials

Prevalence of Exploits
Case: Pegasus vs UAE Dissident





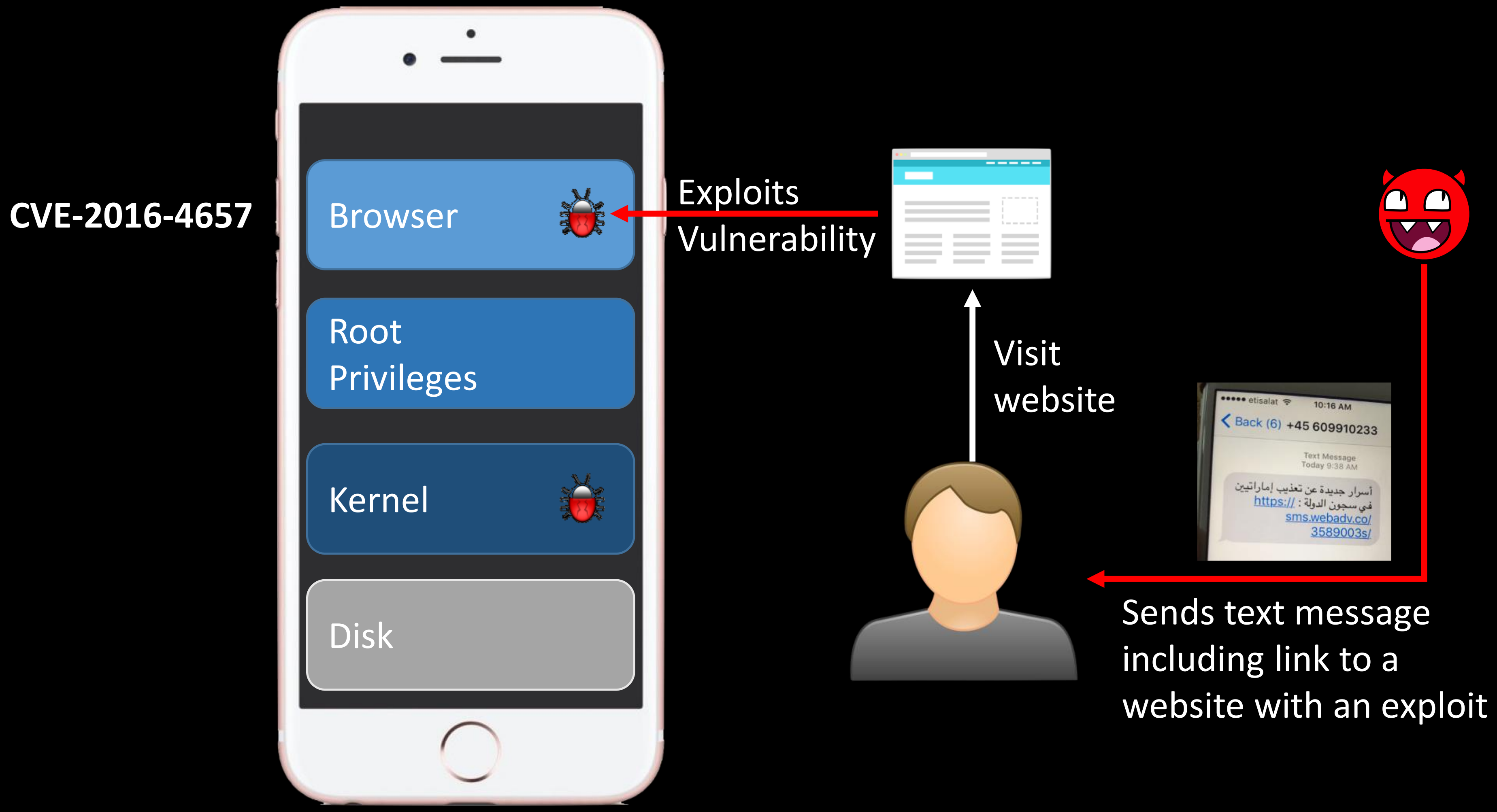
+45 609910233

Text Message
Today 9:38 AM

أسرار جديدة عن تعذيب إماراتيين
في سجون الدولة : https://
sms.webadv.co/
3589003s/

Sends text message including link to a website with an exploit

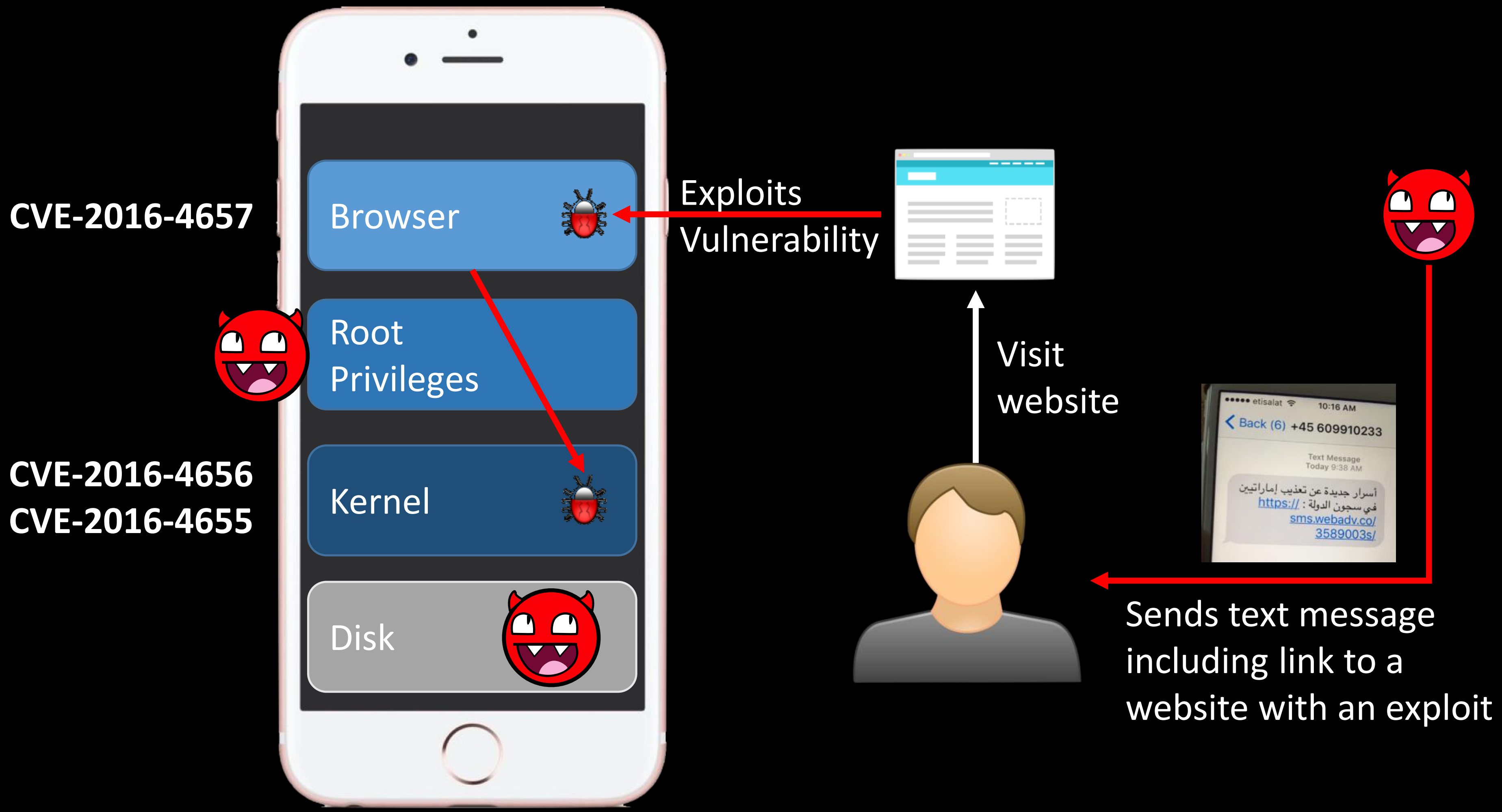
Prevalence of Exploits
Case: Pegasus vs UAE Dissident



••••• etisalat 10:16 AM
< Back (6) +45 609910233
Text Message
Today 9:38 AM
أسرار جديدة عن تعذيب إماراتيين
في سجون الدولة : https://
sms.webadv.co/
3589003s/

Sends text message including link to a website with an exploit

Prevalence of Exploits
Case: Pegasus vs UAE Dissident



••••• etisalat 10:16 AM

Back (6) +45 609910233

Text Message
Today 9:38 AM

أسرار جديدة عن تعذيب إماراتيين
في سجون الدولة : https://
sms.webadv.co/
3589003s/

Sends text message including link to a website with an exploit

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 includes a ROP detection engine
- Microsoft Control Flow Guard (CFG) in Windows 10
- Google‘s compiler extension VTV (Virtual Table Verification)
- Intel’s Hardware Extension CET (Control-flow Enforcement Technology)

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

Can either be bypassed, or may not be sufficiently effective

[Davi et al, Blackhat2014], [Liebchen et al CCS2015], [Schuster, et al S&P2015]

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 includes a ROP detection engine
- Microsoft Control Flow Guard (CFG) in Windows 10
- Google‘s compiler extension VTV (Virtual Table Verification)
- Intel’s Hardware Extension CET (Control-flow Enforcement Technology)

Hot Topic of Research

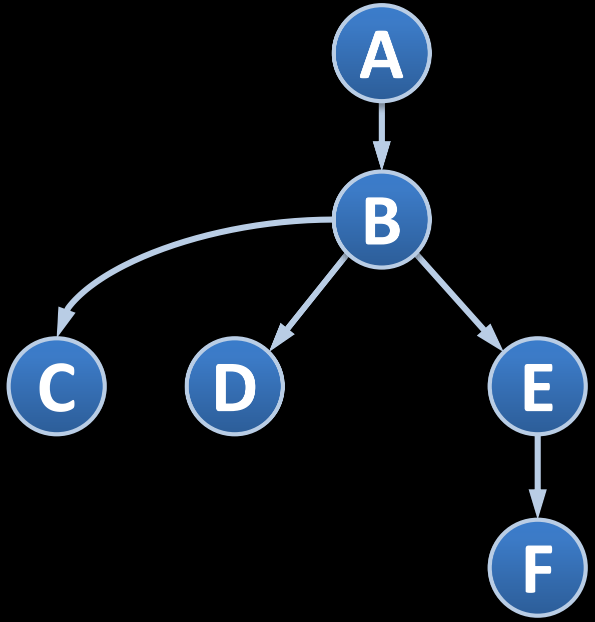
- A large body of recent literature on attacks and defenses

The whole story .....

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

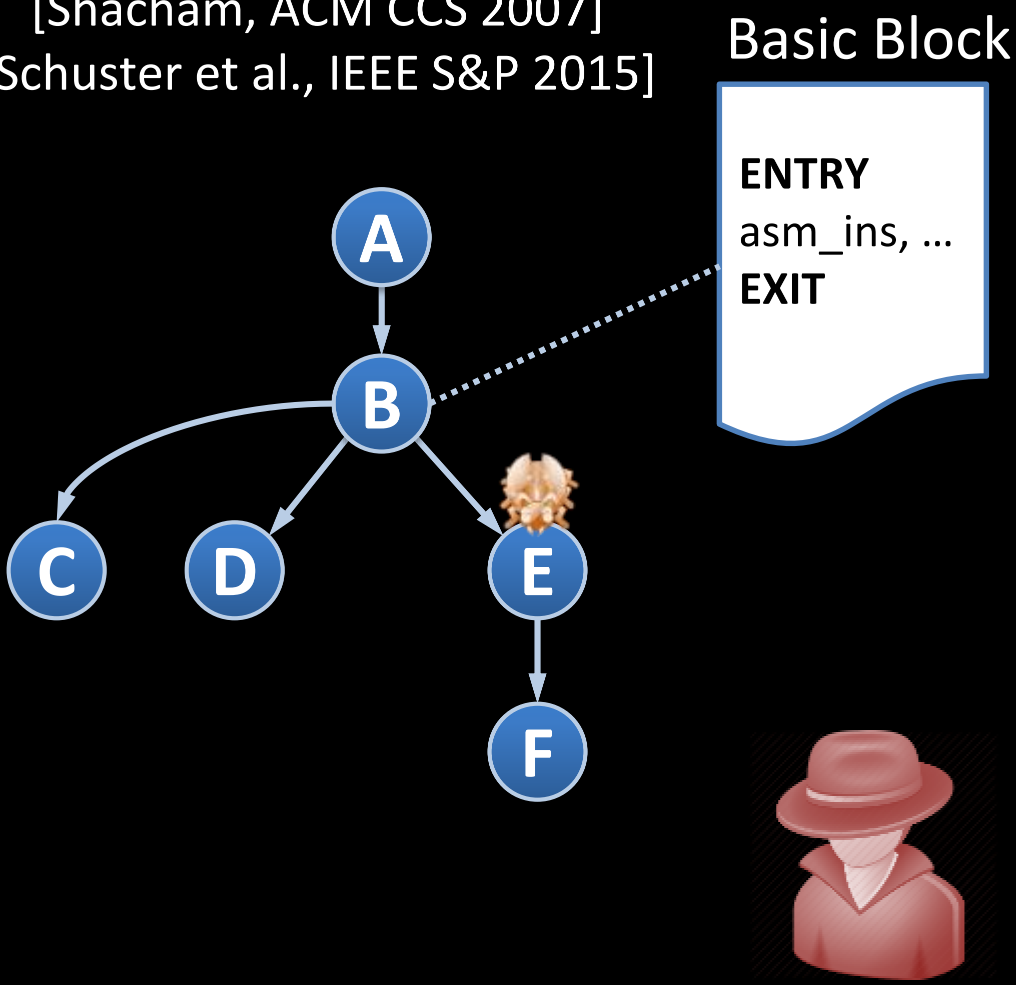
Problem Space of Zero-Day Exploits

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]



Adversary

- - -→ Memory write
——→ Program flow

Problem Space of Zero-Day Exploits

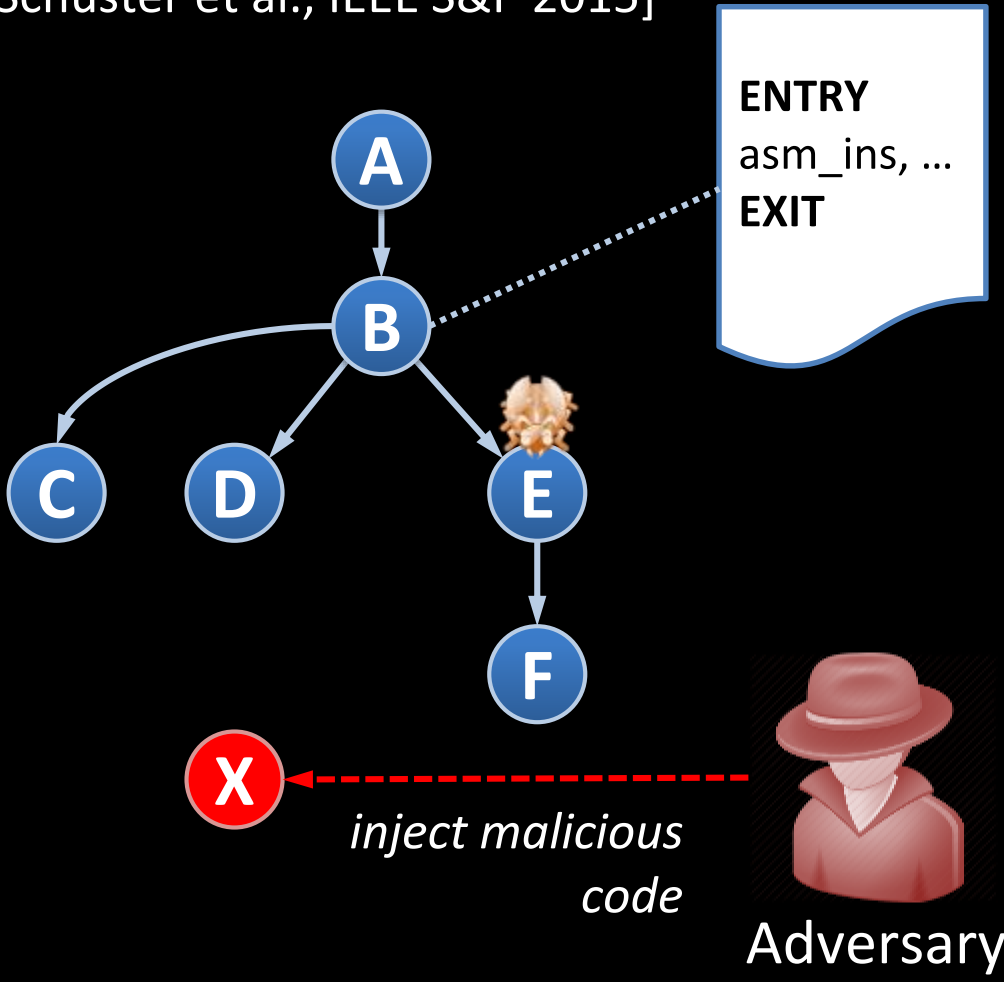
Control-Flow Attack

[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



- - -▶ Memory write

──▶ Program flow

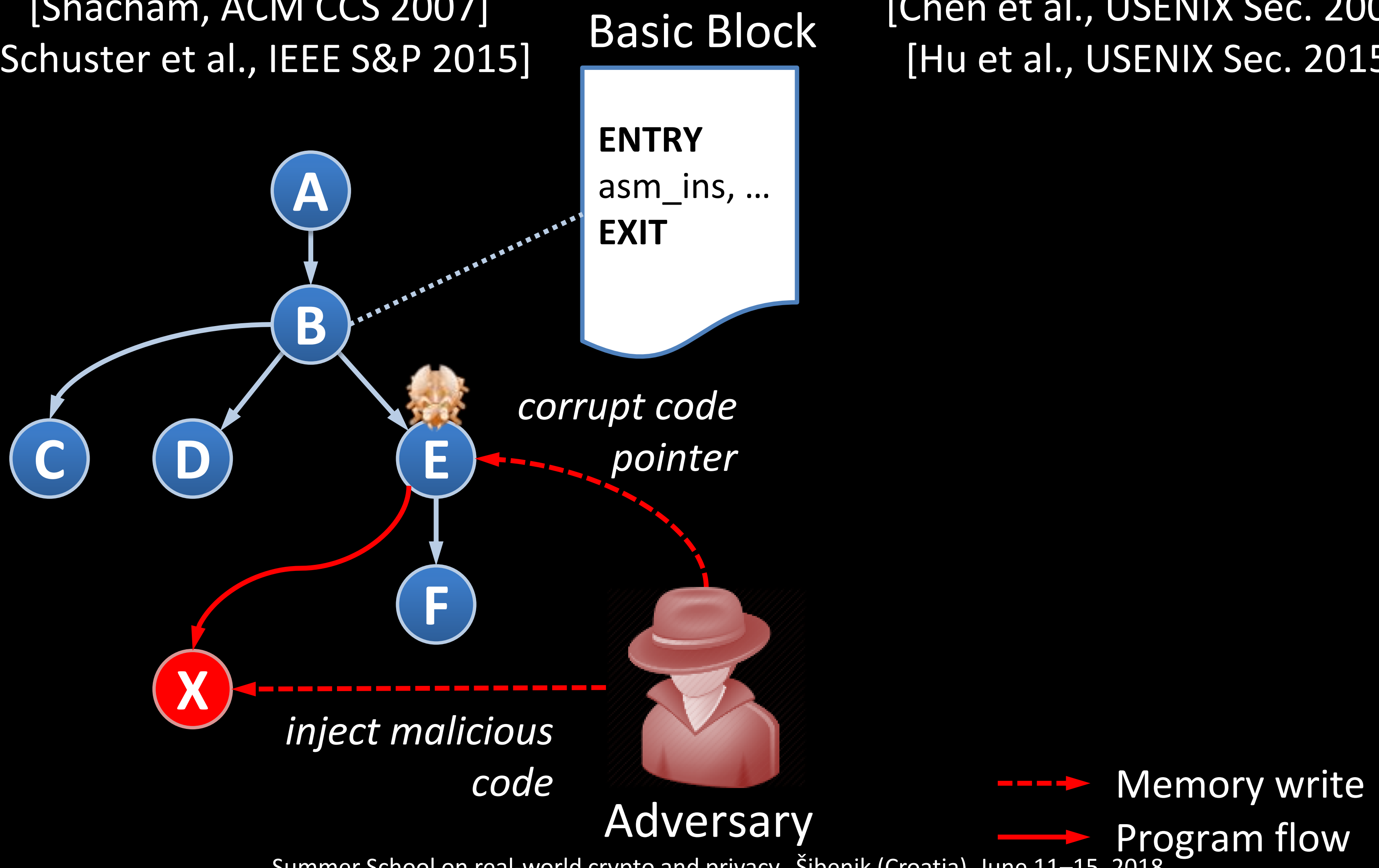
Problem Space of Zero-Day Exploits

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]



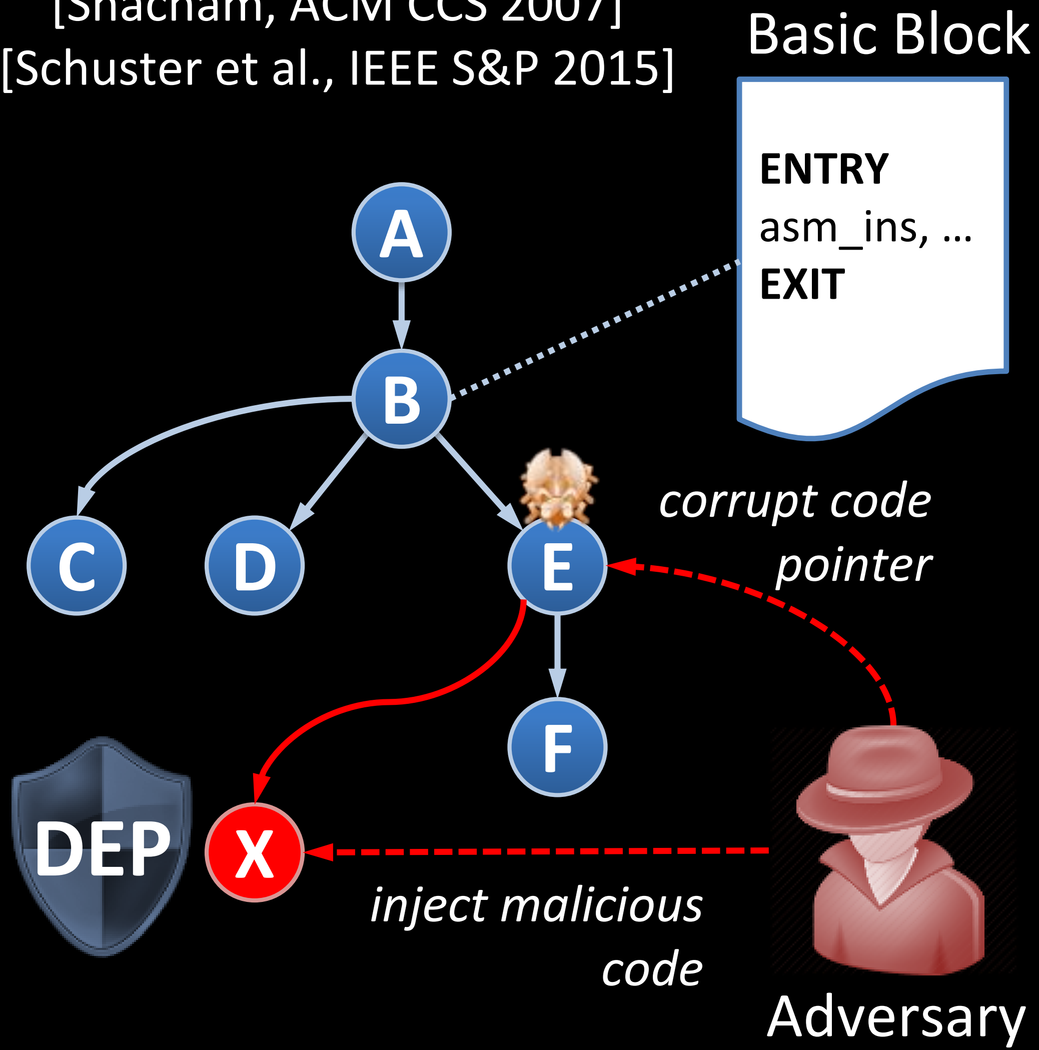
Problem Space of Zero-Day Exploits

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]



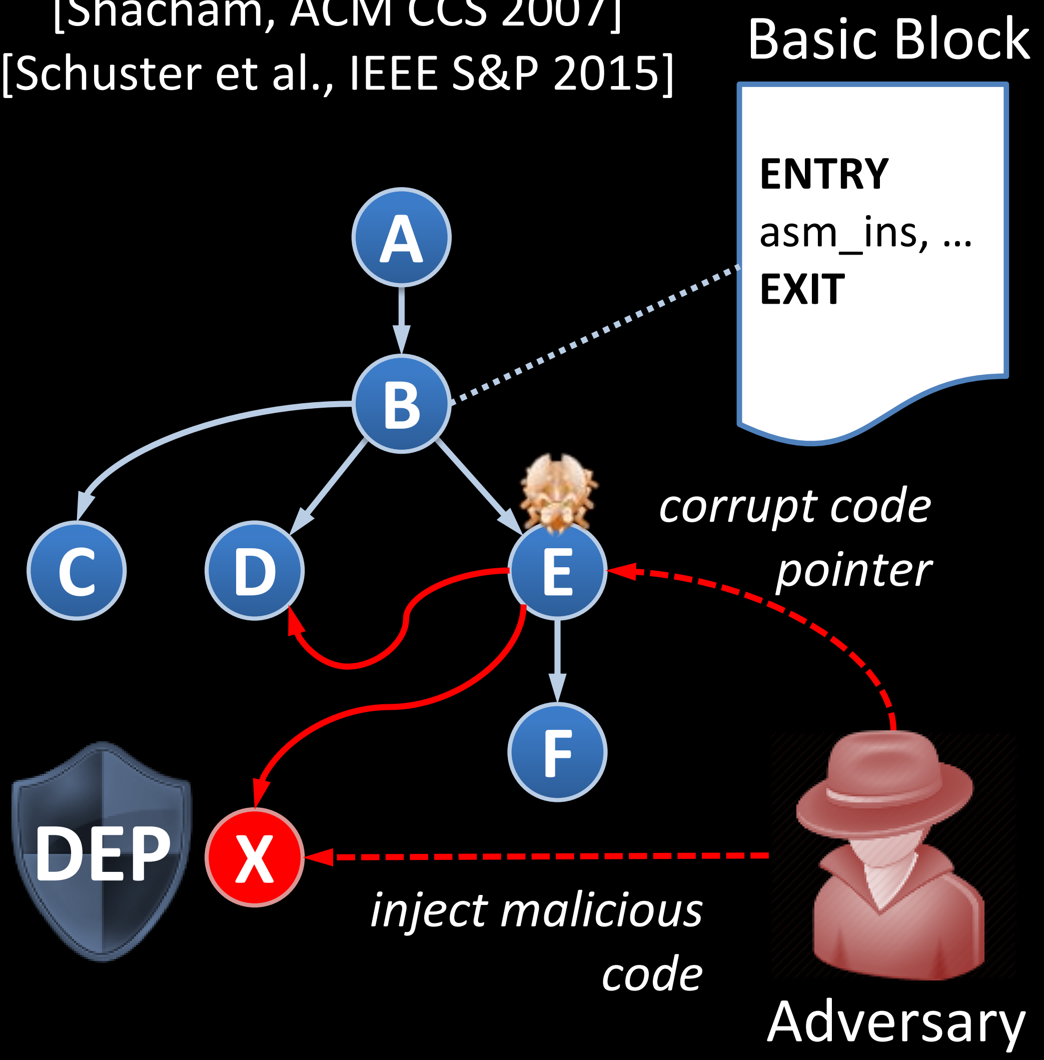
Problem Space of Zero-Day Exploits

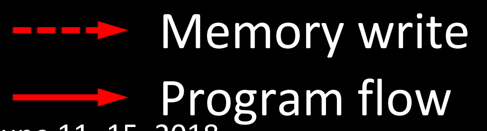
Control-Flow Attack

[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]

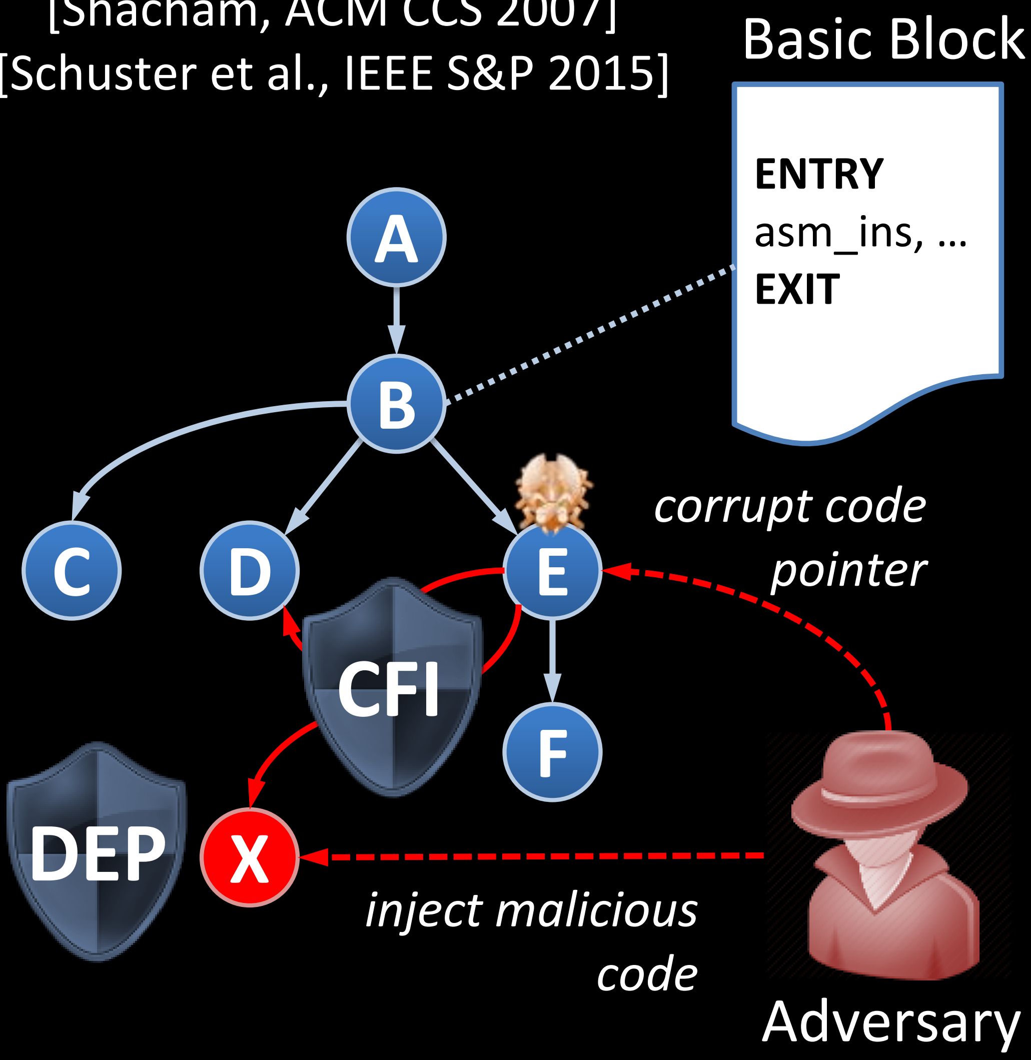




Problem Space of Zero-Day Exploits

Control-Flow Attack

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



Basic Block

ENTRY
asm_ins, …
EXIT

corrupt code pointer

inject malicious code

Adversary

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

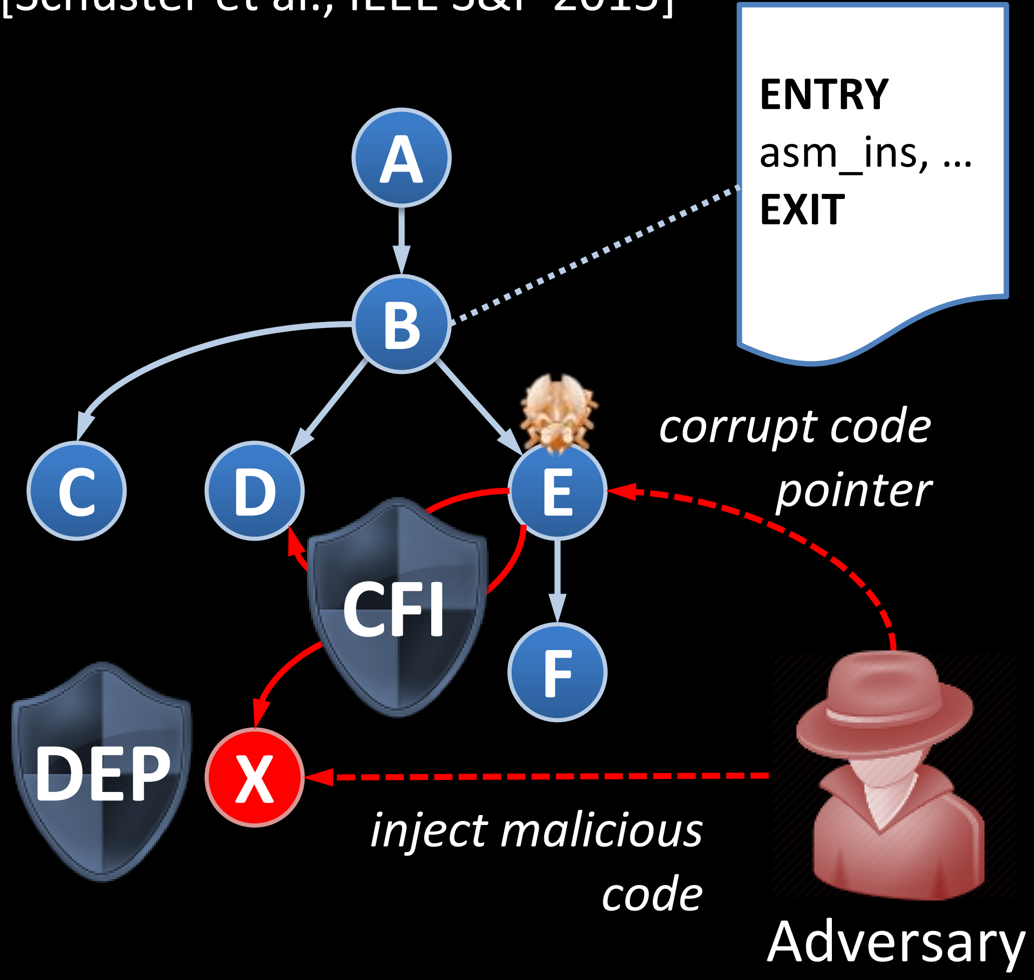
Control-Flow Attack

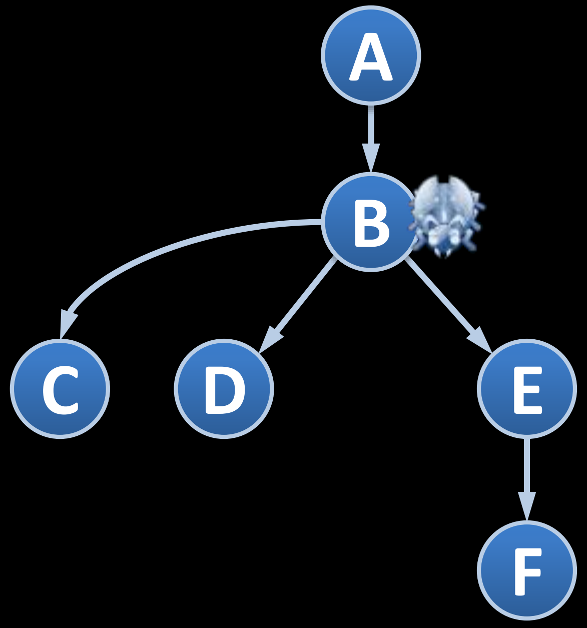
[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

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

Basic Block





⇢ Memory write

→ Program flow

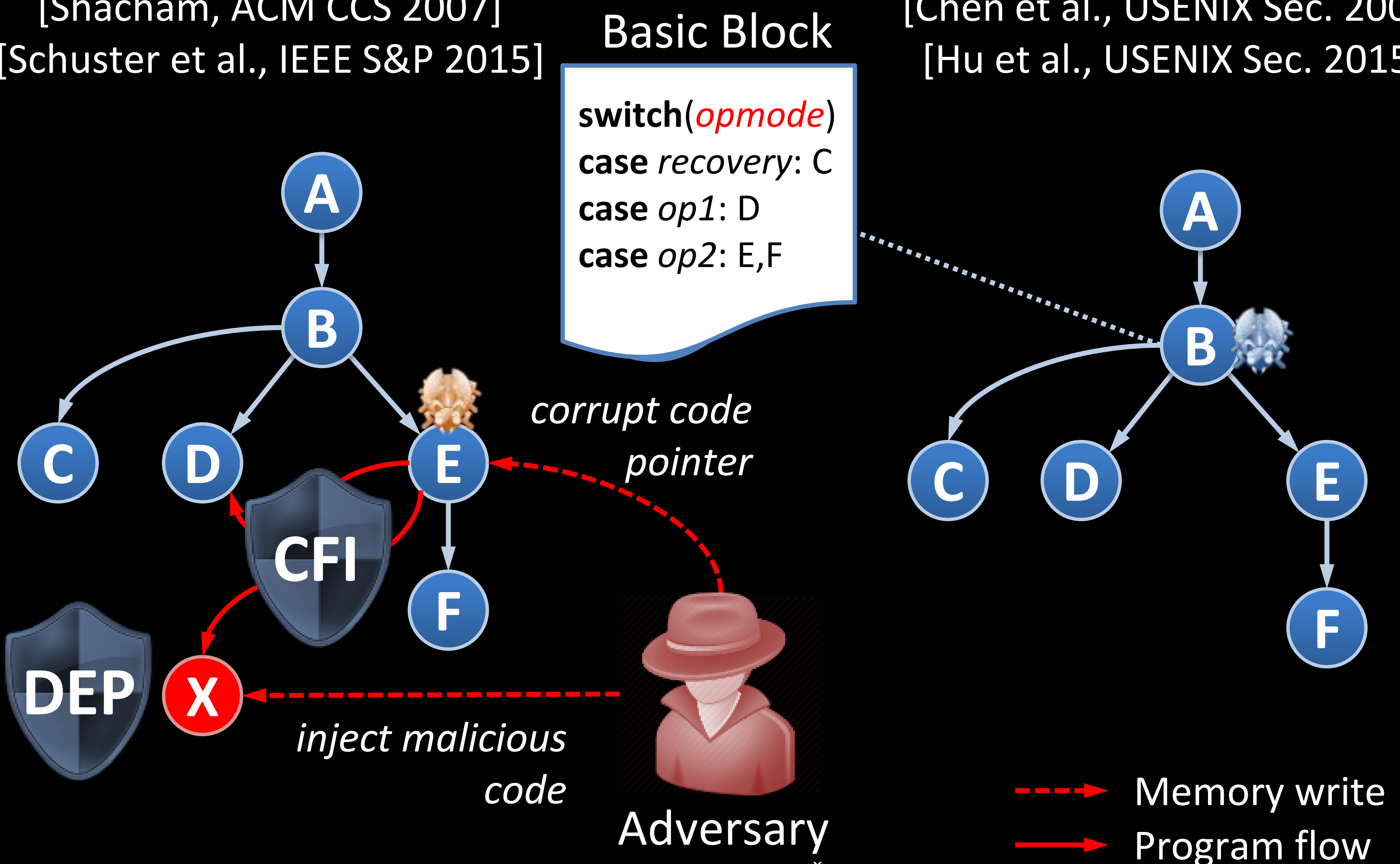
Problem Space of Zero-Day Exploits

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

```
switch(opmode)
case recovery: C
case op1: D
case op2: E,F
```

Problem Space of Zero-Day Exploits

Control-Flow Attack

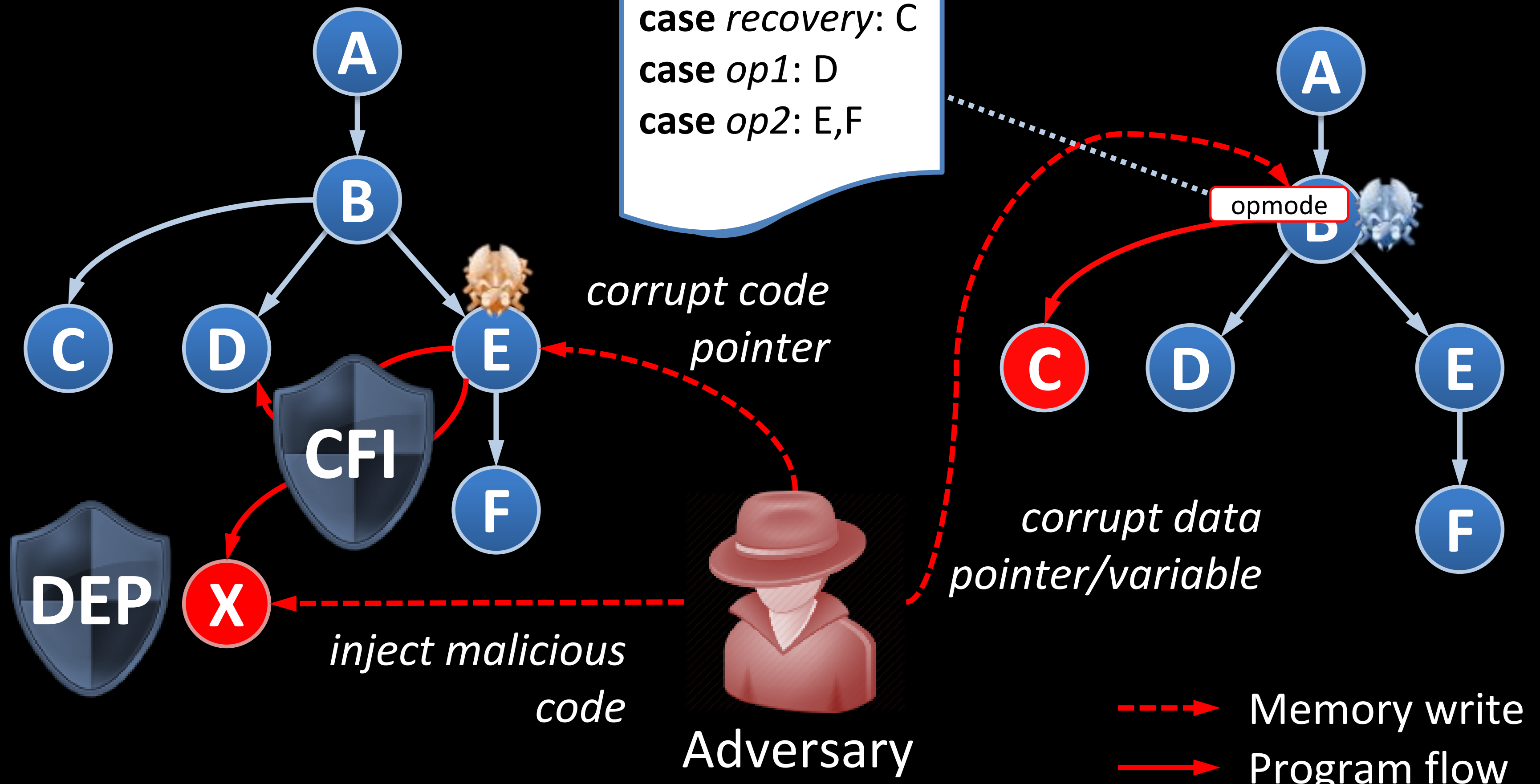
[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

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

Basic Block

```
switch(opmode)
case recovery: C
case op1: D
case op2: E,F
```



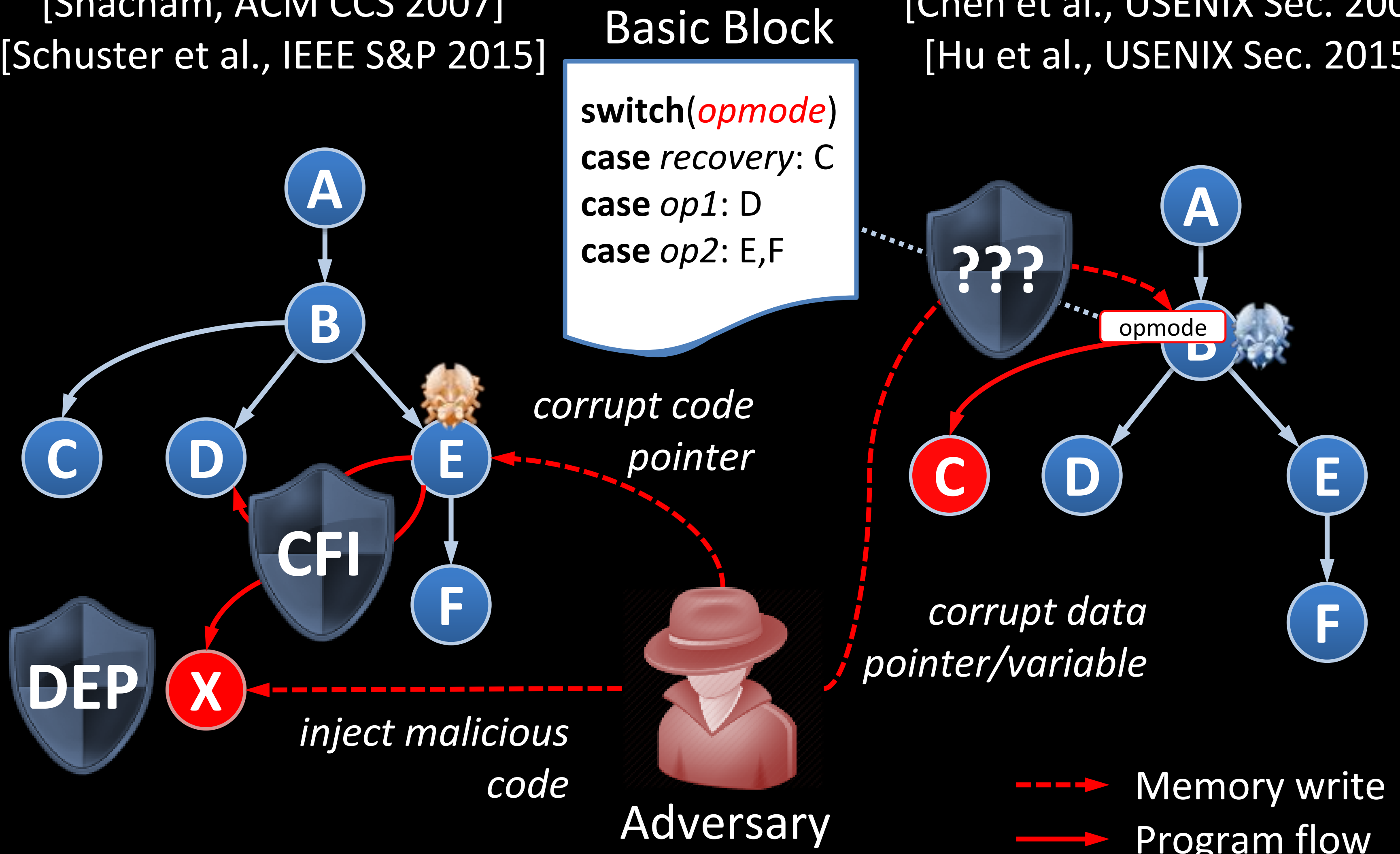
Problem Space of Zero-Day Exploits

Control-Flow Attack

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

Non-Control-Data Attack

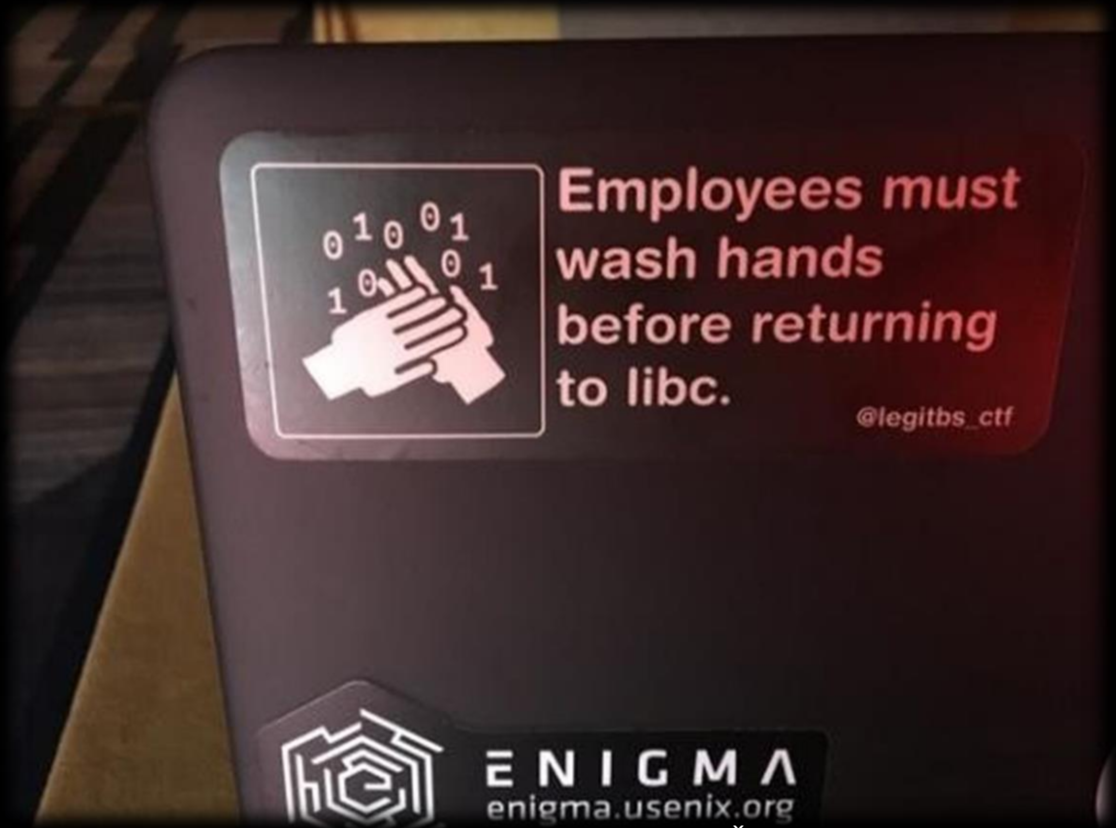
[Chen et al., USENIX Sec. 2005]
[Hu et al., USENIX Sec. 2015]



Basic Block

```
switch(opmode)
case recovery: C
case op1: D
case op2: E,F
```

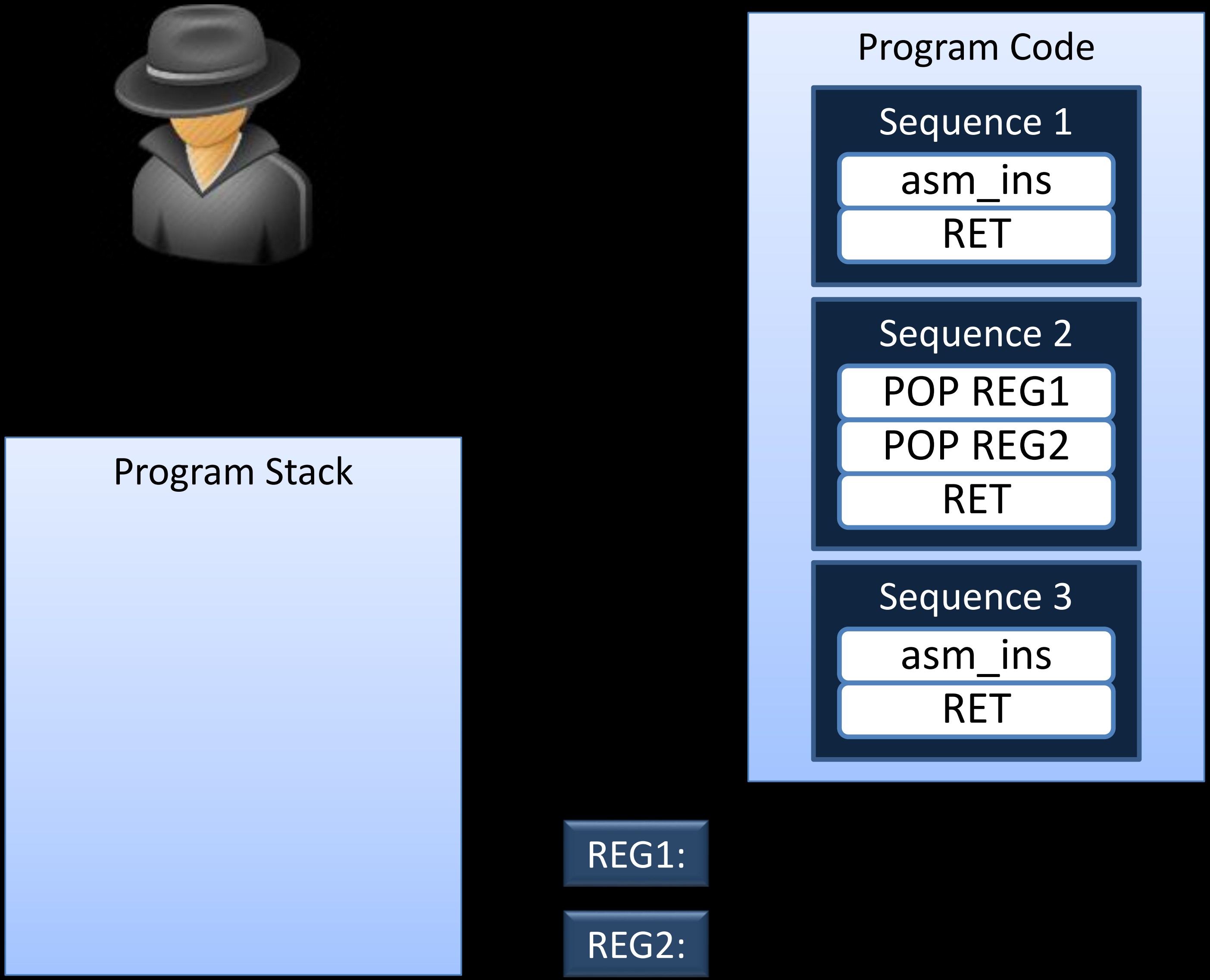
Return-oriented Programing (ROP): Prominent Code-Reuse Attack



ROP: Basic Ideas/Steps

- Use small instruction sequences
- Instruction sequences have length 2 to 5
- All sequences end with a return instruction, or an indirect jump/call
- Instruction sequences chained together as gadgets
- Gadget perform particular task, e.g., load, store, xor, or branch
- Attacks launched by combining gadgets
- Generalization of return-to-libc

ROP Attack Technique: Overview



Program Code

Sequence 1

asm_ins

RET

Sequence 2

POP REG1

POP REG2

RET

Sequence 3

asm_ins

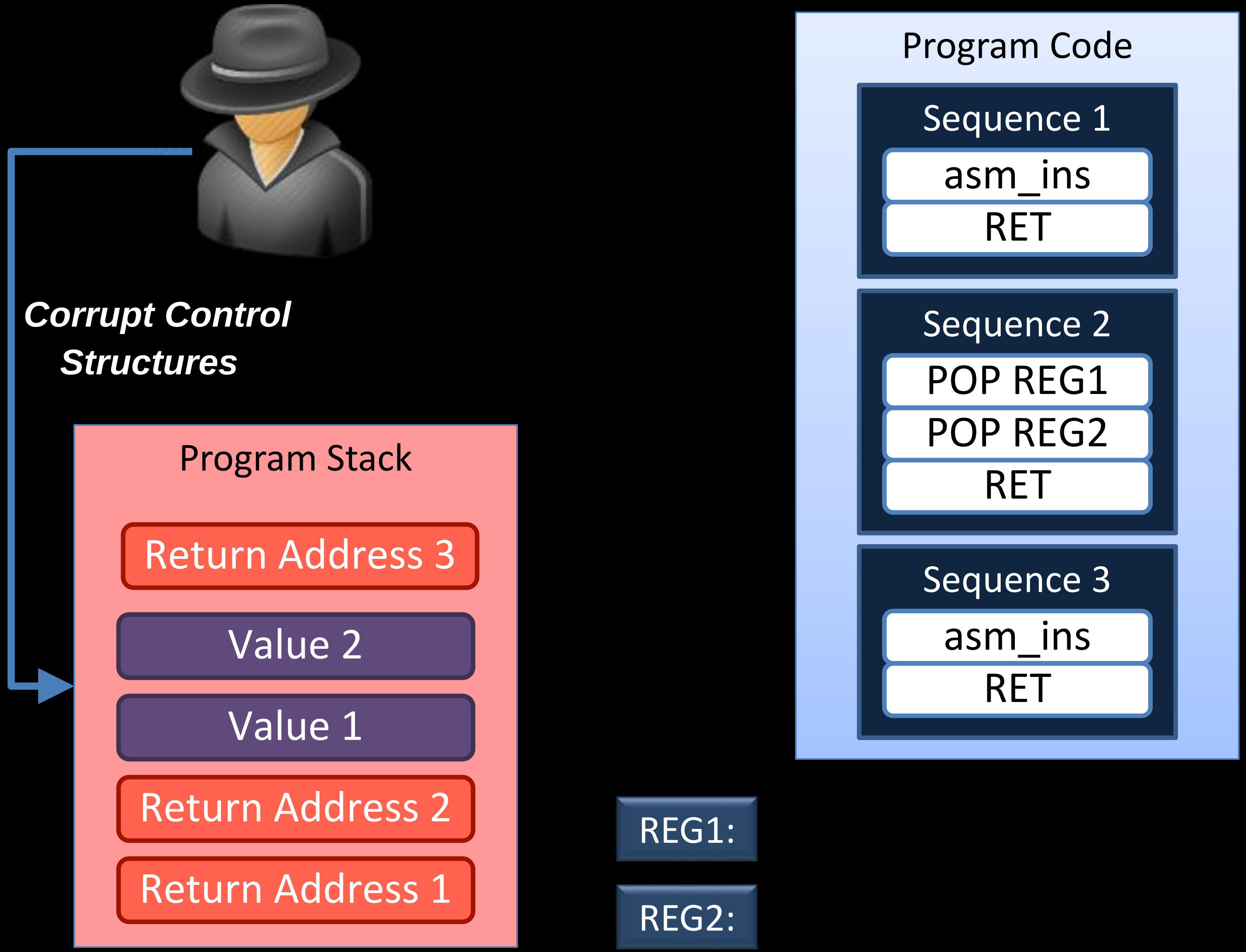
RET

Program Stack

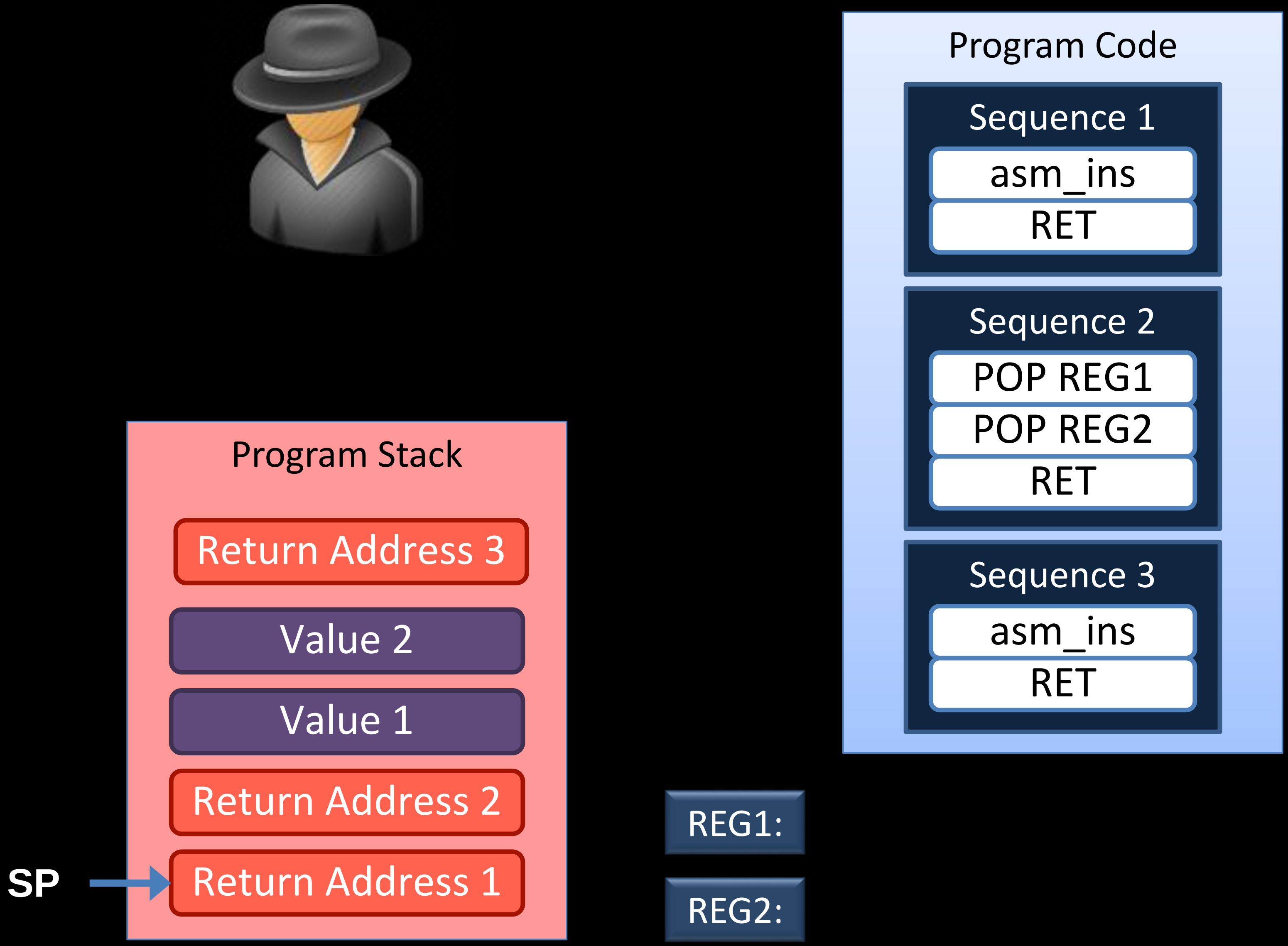
REG1:

REG2:

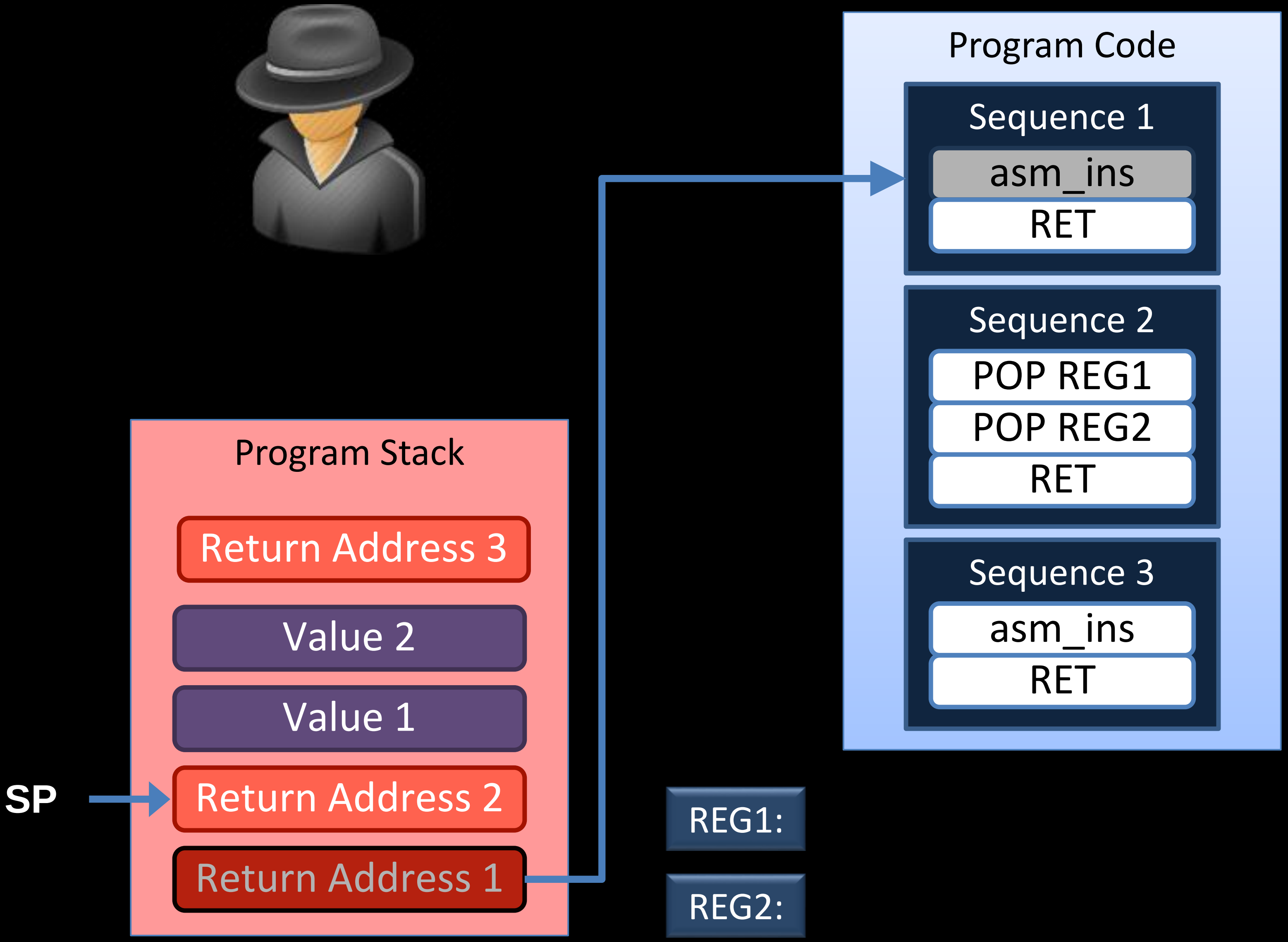
ROP Attack Technique: Overview



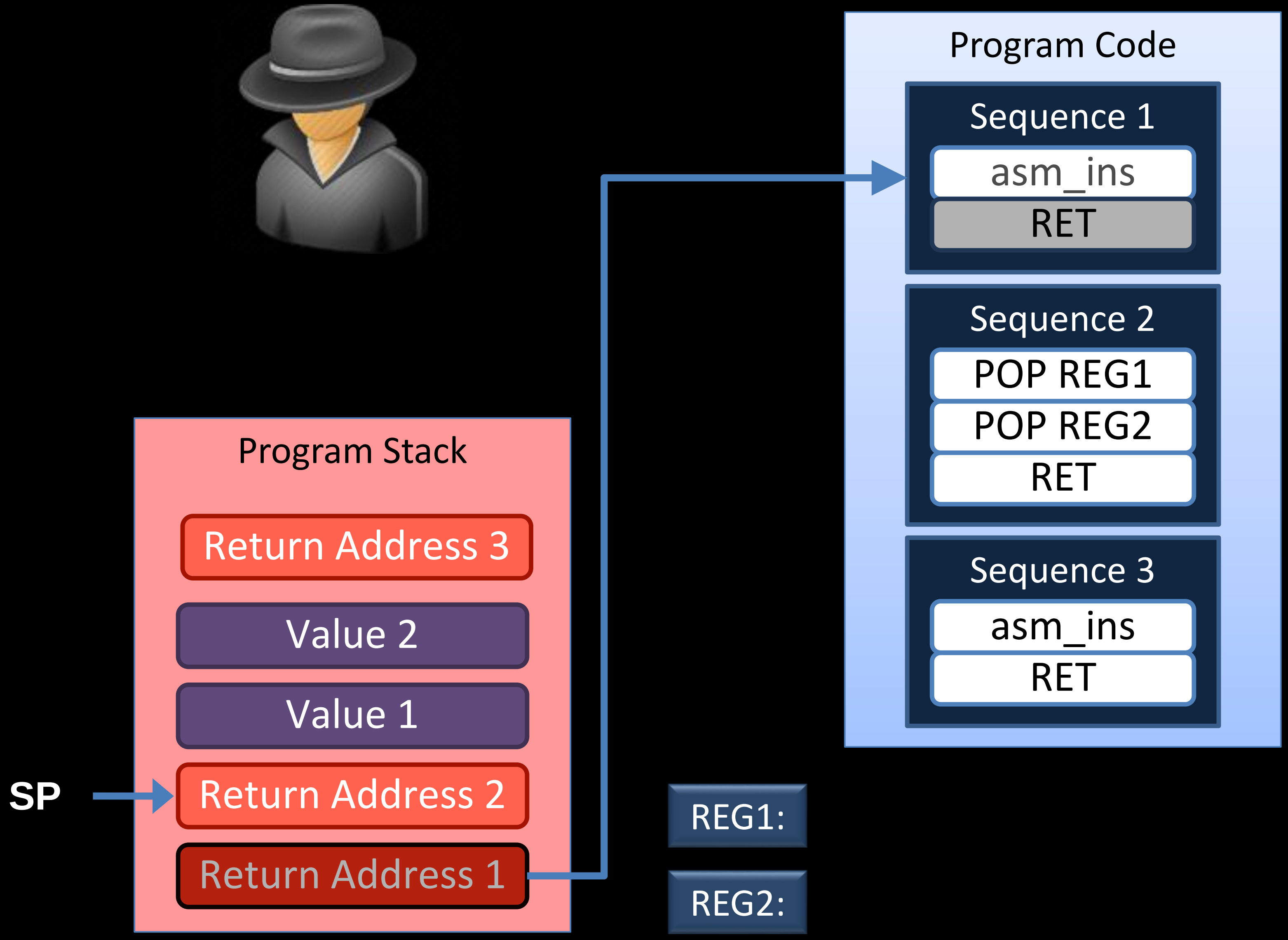
ROP Attack Technique: Overview



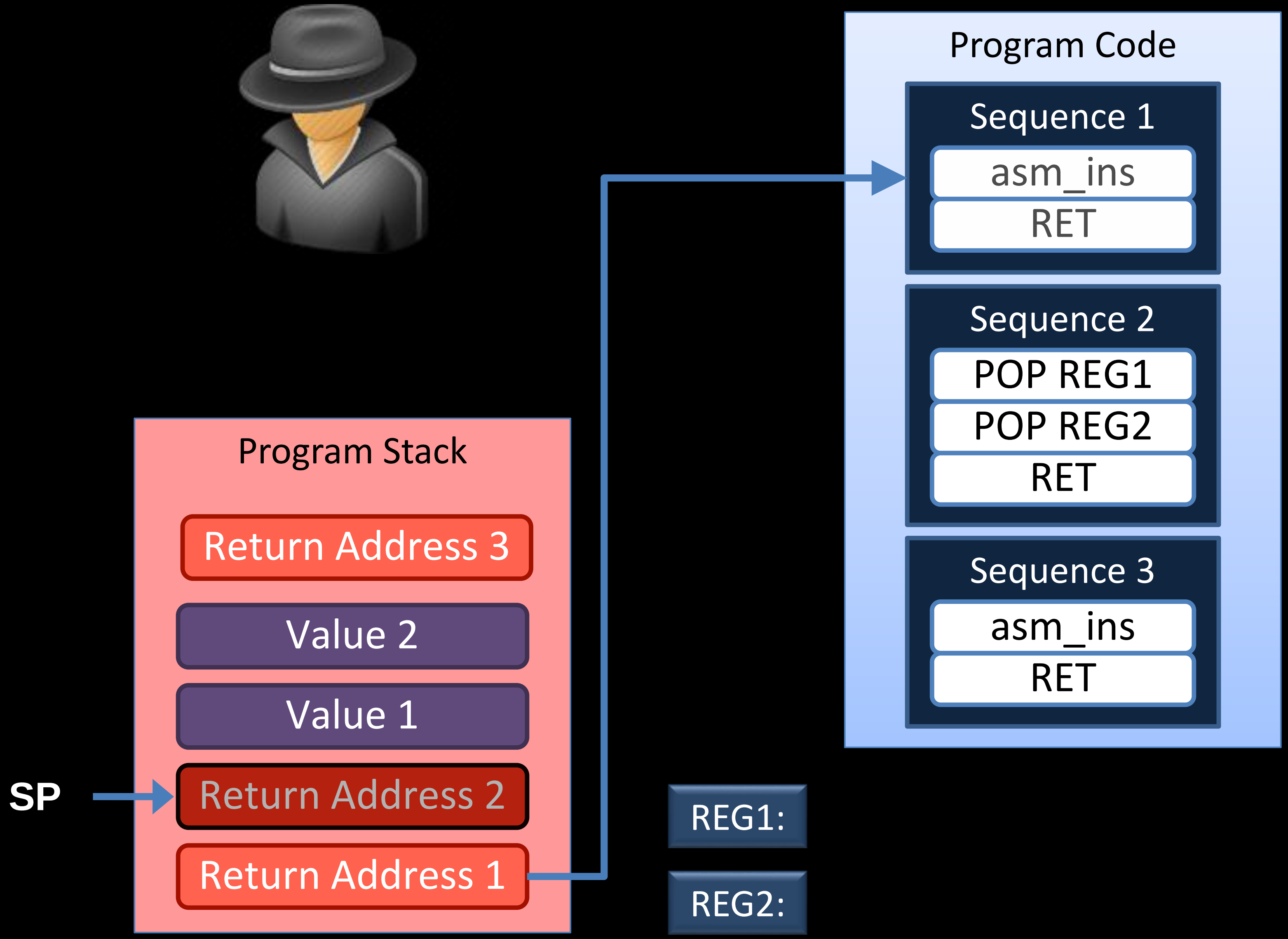
ROP Attack Technique: Overview



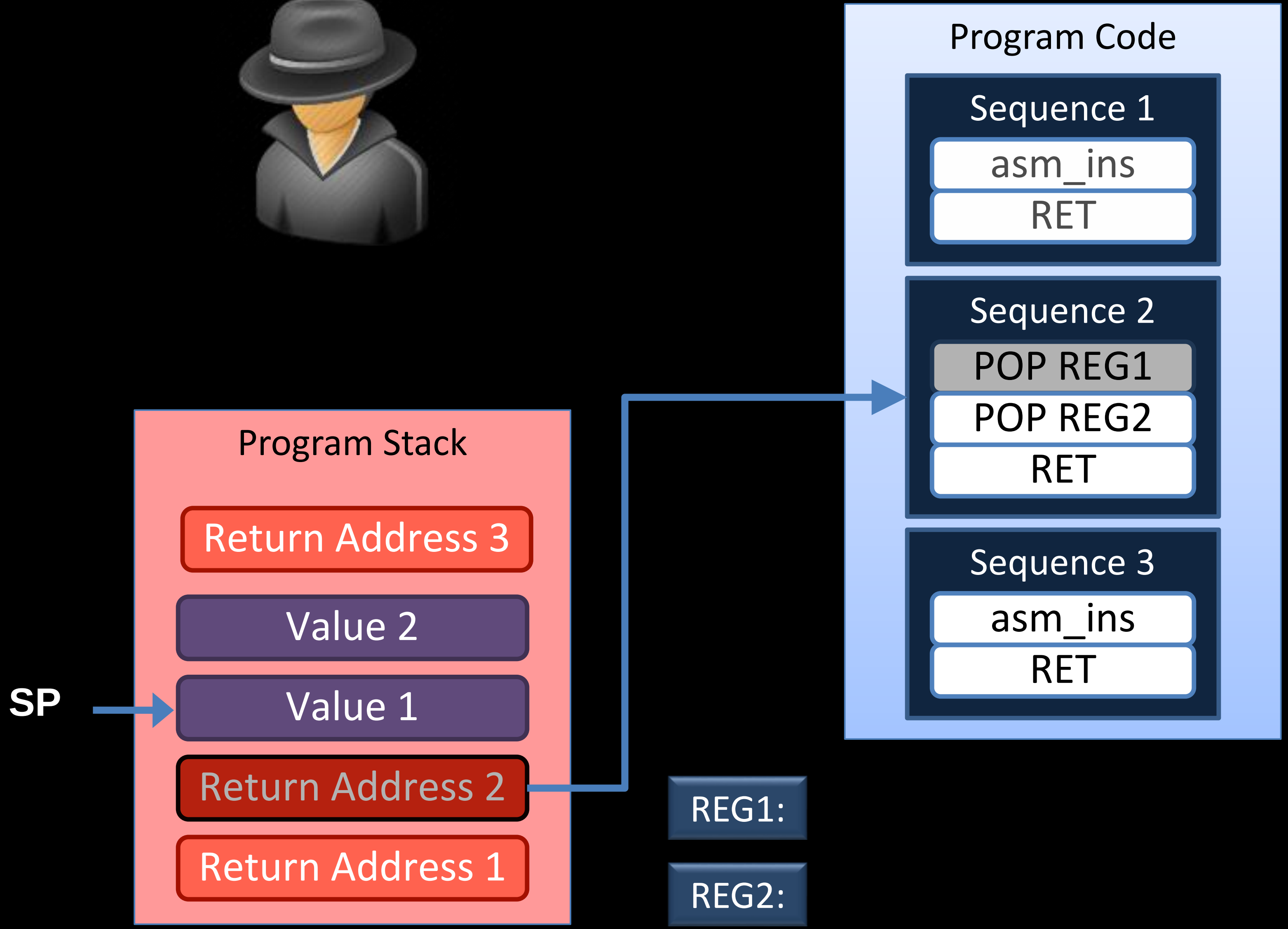
ROP Attack Technique: Overview



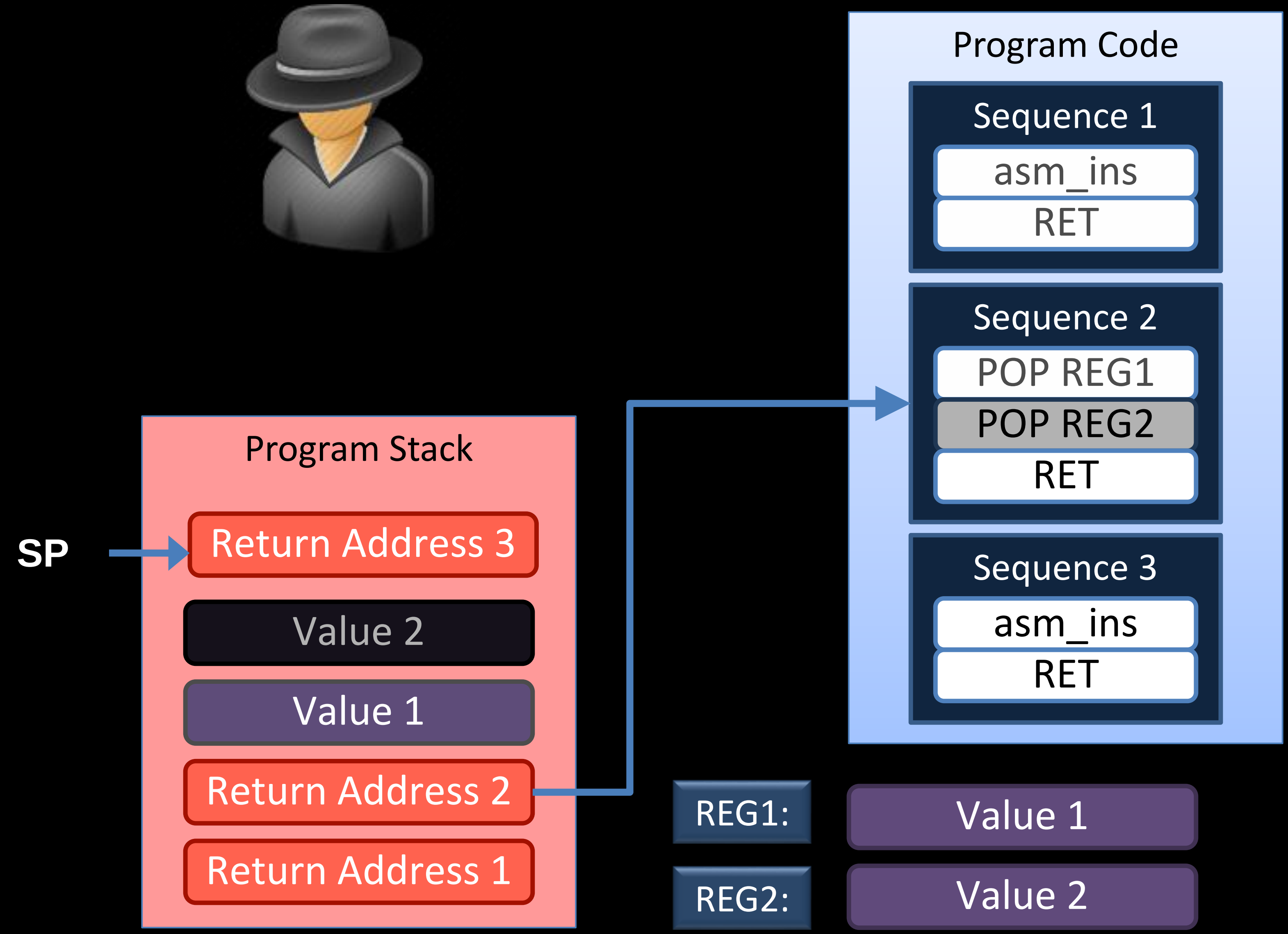
ROP Attack Technique: Overview



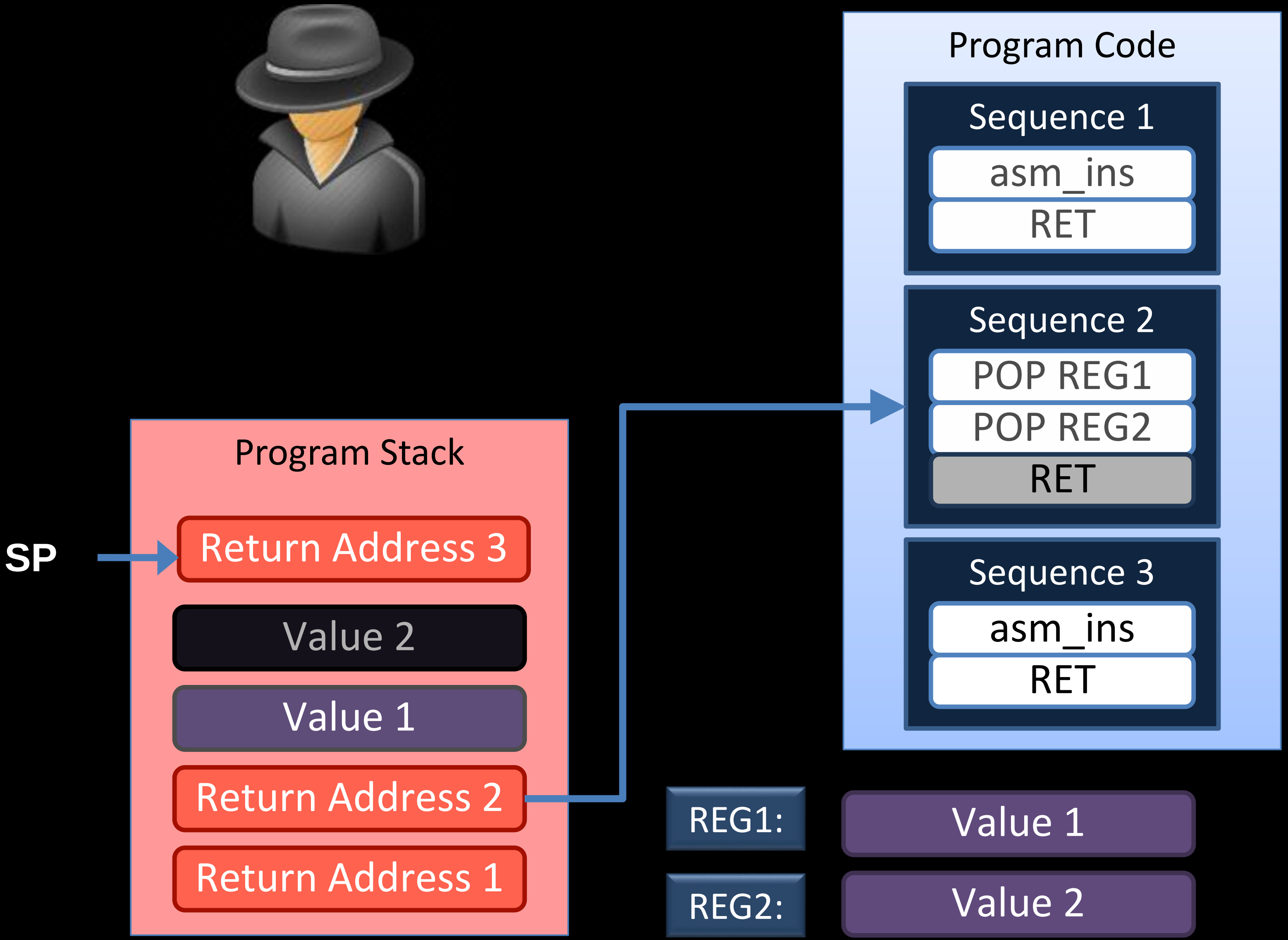
ROP Attack Technique: Overview



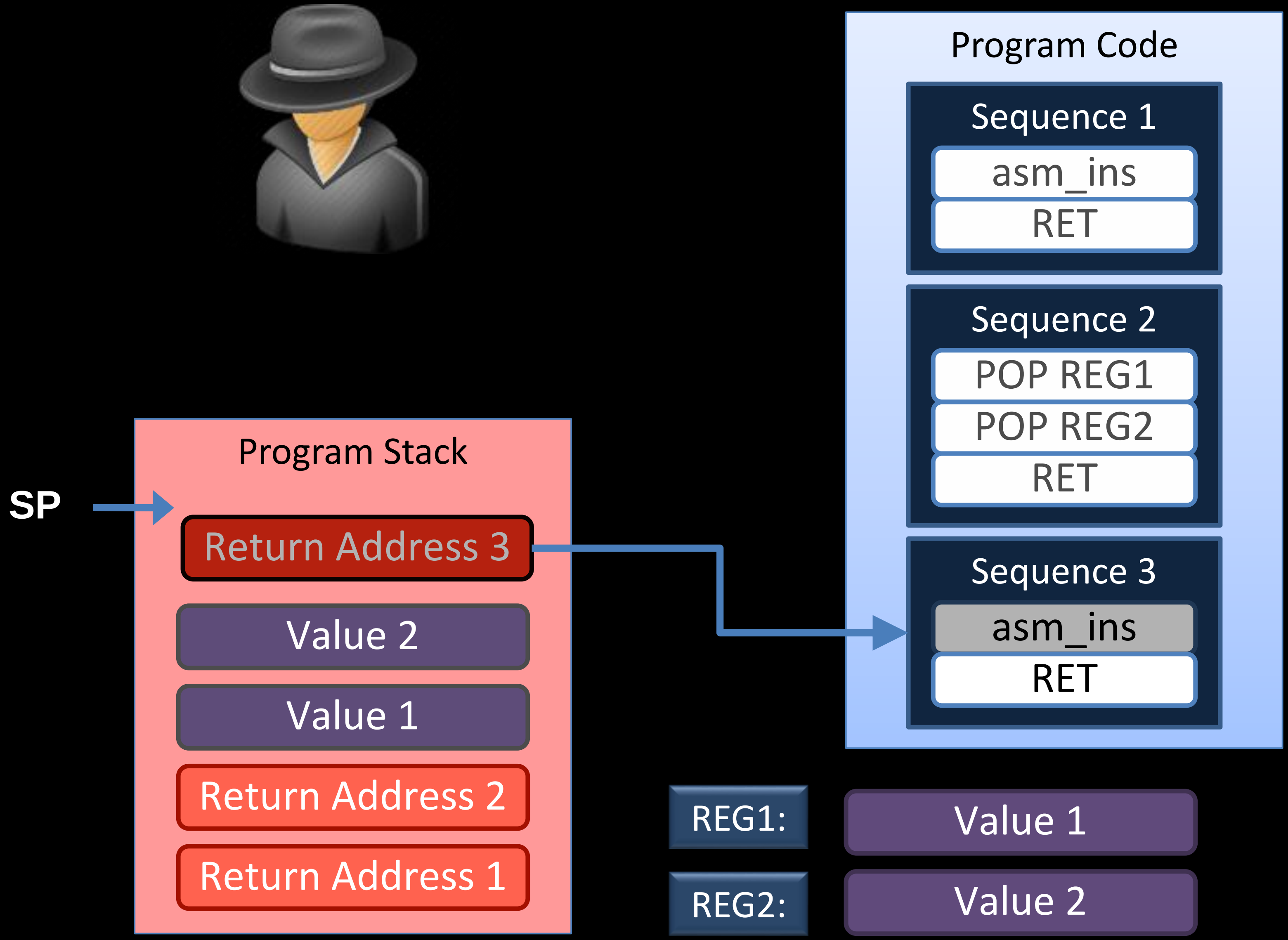
ROP Attack Technique: Overview



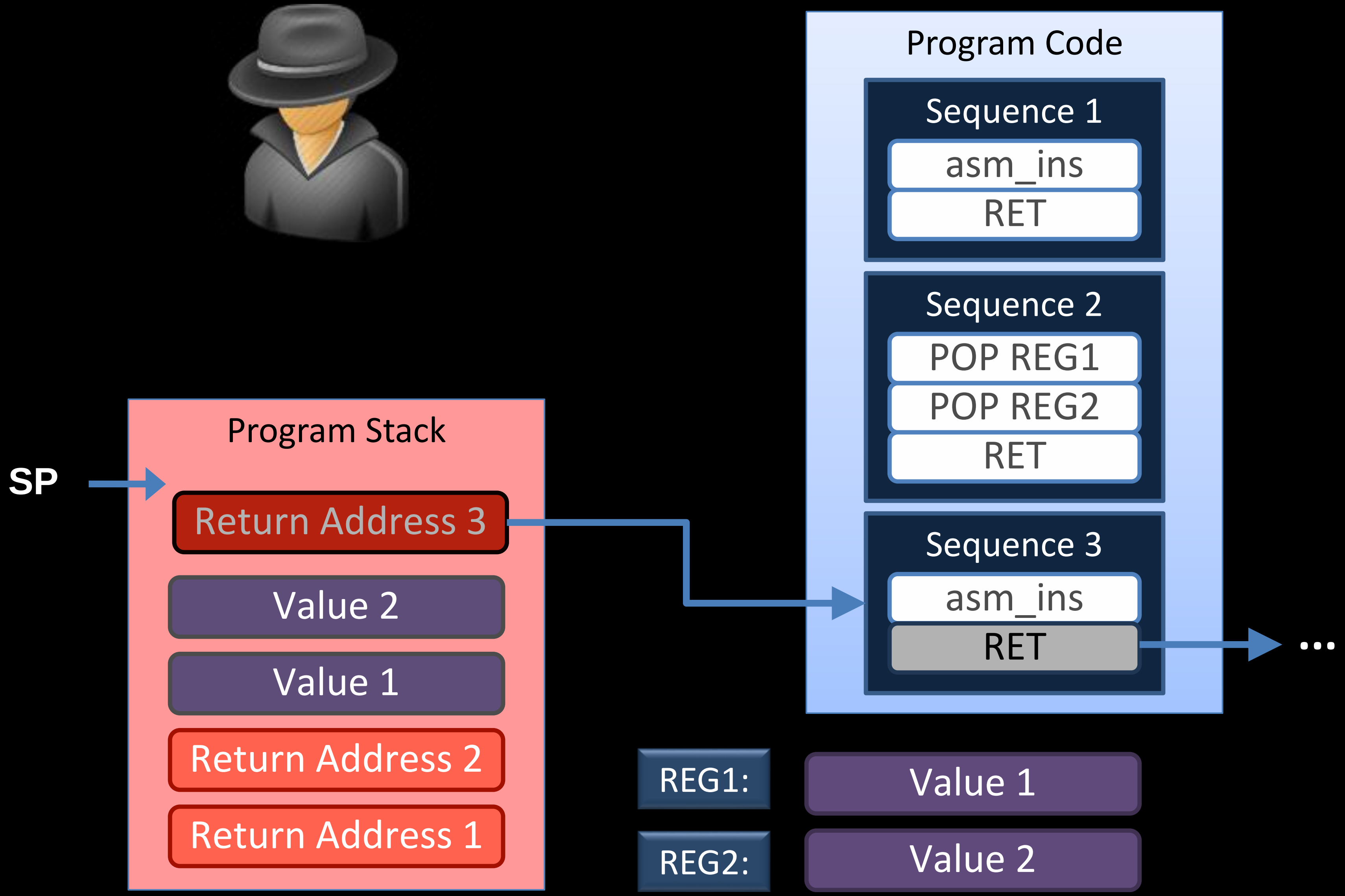
ROP Attack Technique: Overview



ROP Attack Technique: Overview



ROP Attack Technique: Overview



SP

Program Stack

Return Address 3

Value 2

Value 1

Return Address 2

Return Address 1

Program Code

Sequence 1

asm_ins

RET

Sequence 2

POP REG1

POP REG2

RET

Sequence 3

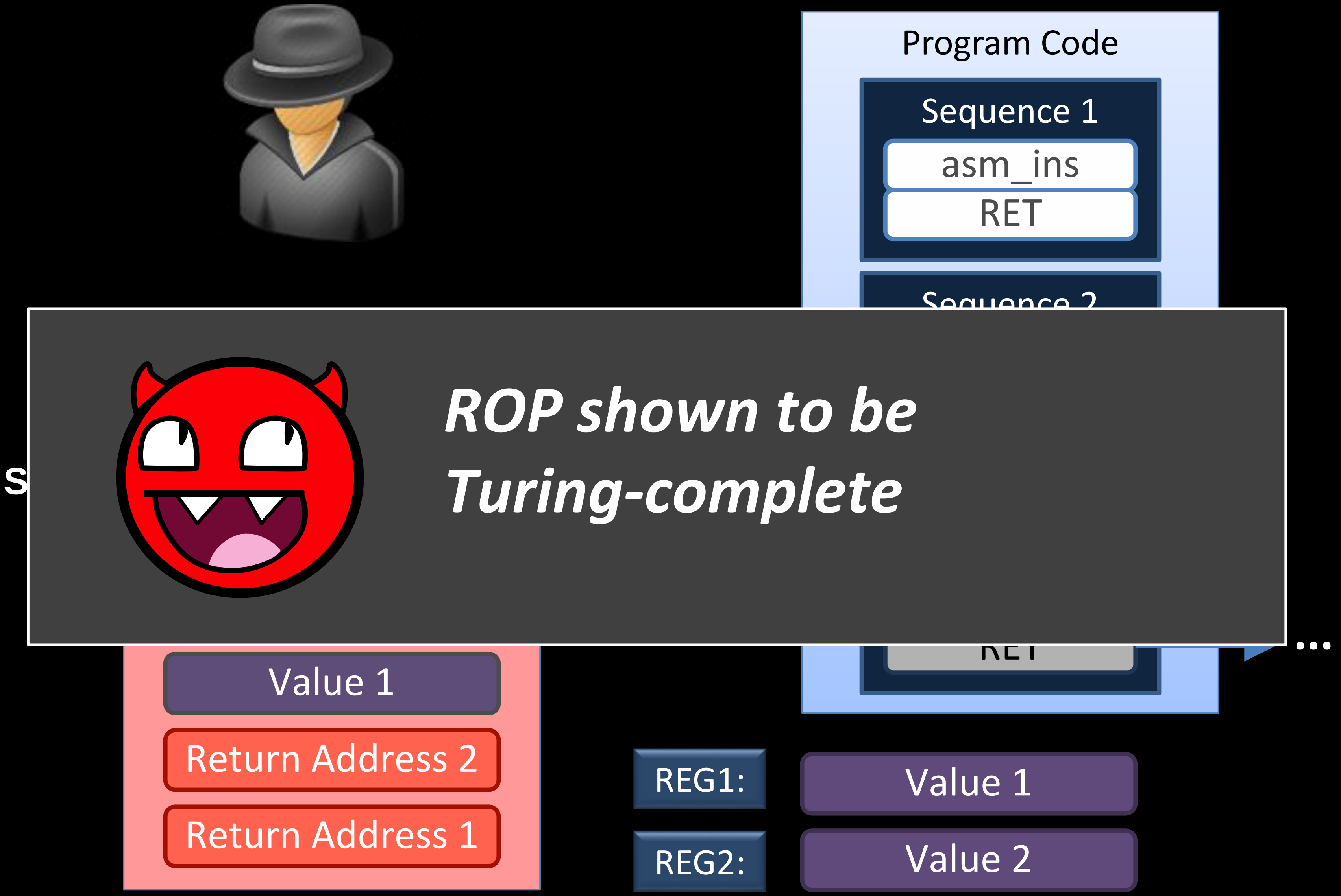
asm_ins

RET

…

REG1:	Value 1
REG2:	Value 2

ROP Attack Technique: Overview

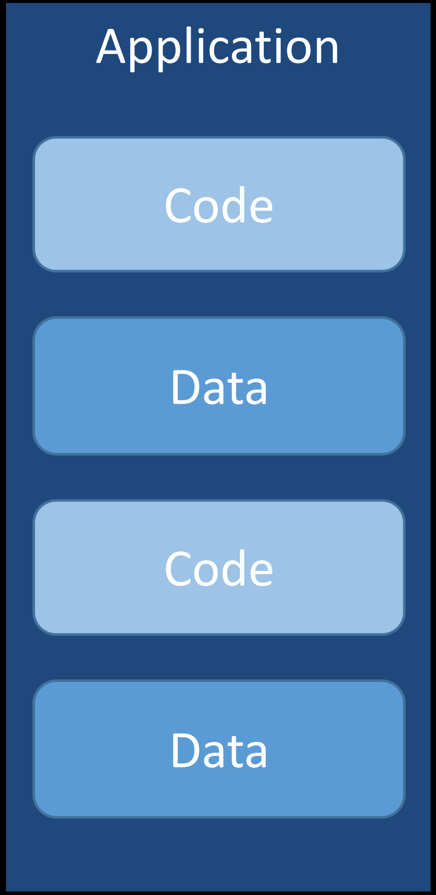


ROP shown to be Turing-complete

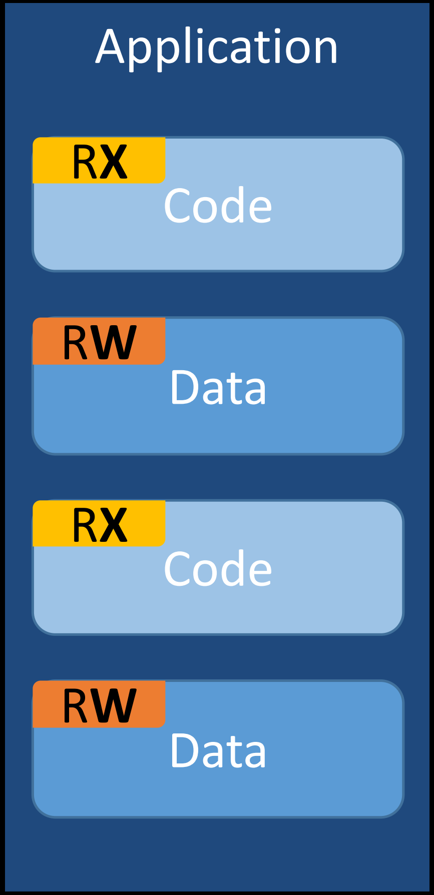
Code Injection vs. Code Reuse

- Code Injection – *Adding a new node to the CFG*
  - Adversary can execute arbitrary malicious code
    - open a remote console (classical shellcode)
    - exploit further vulnerabilities in the OS kernel to install a virus or a backdoor
- Code Reuse – *Adding a new path to the CFG*
  - Adversary is limited to the code nodes that are available in the CFG
  - Requires reverse-engineering and static analysis of the code base of a program

Threat Model: Code-reuse Attacks

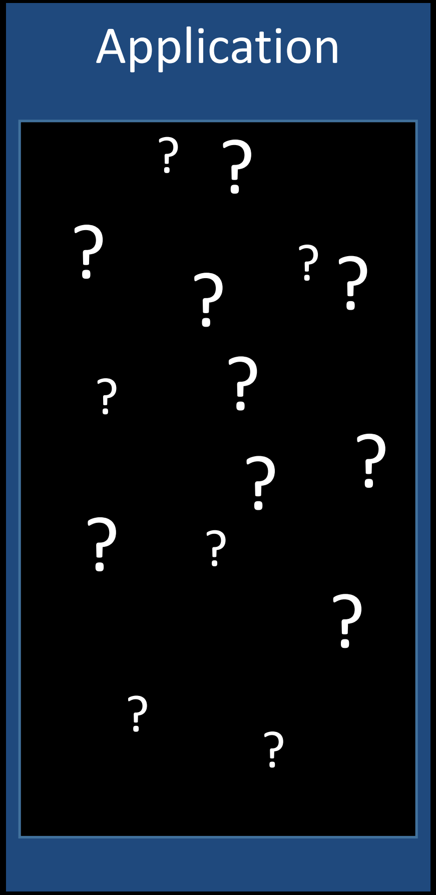


Threat Model: Code-reuse Attacks



1 Writable ⊕ Executable

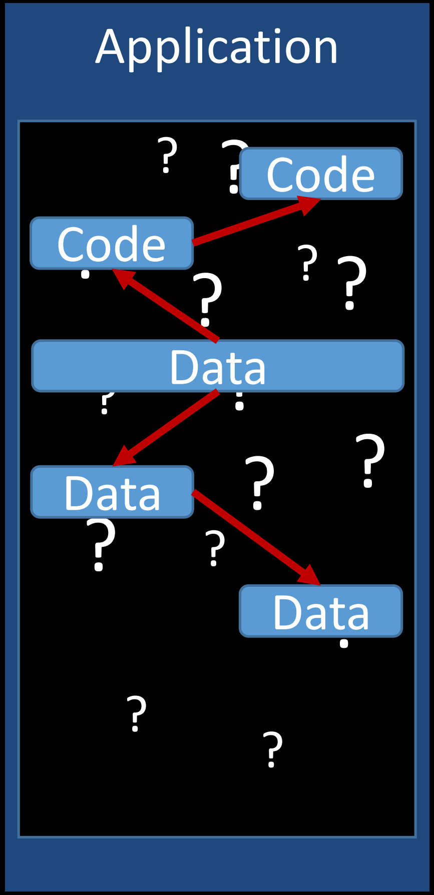
Threat Model: Code-reuse Attacks



1 Writable ⊕ Executable

2 Opaque Memory Layout

Threat Model: Code-reuse Attacks

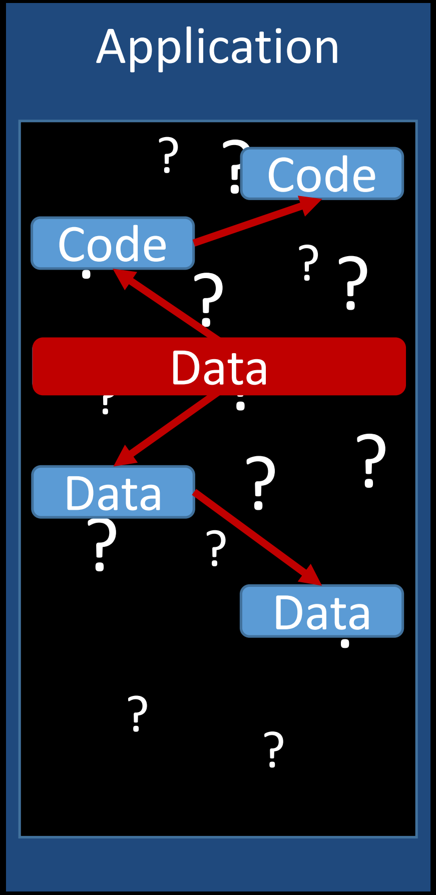


1 Writable ⊕ Executable

2 Opaque Memory Layout

3 Disclose readable Memory

Threat Model: Code-reuse Attacks



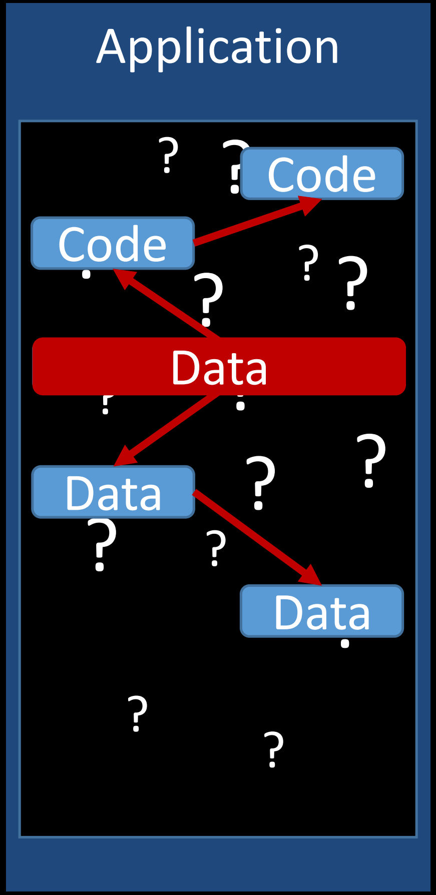
1 Writable ⊕ Executable

2 Opaque Memory Layout

3 Disclose readable Memory

4 Manipulate writable Memory

Threat Model: Code-reuse Attacks



1 Writable ⊕ Executable

2 Opaque Memory Layout

3 Disclose readable Memory

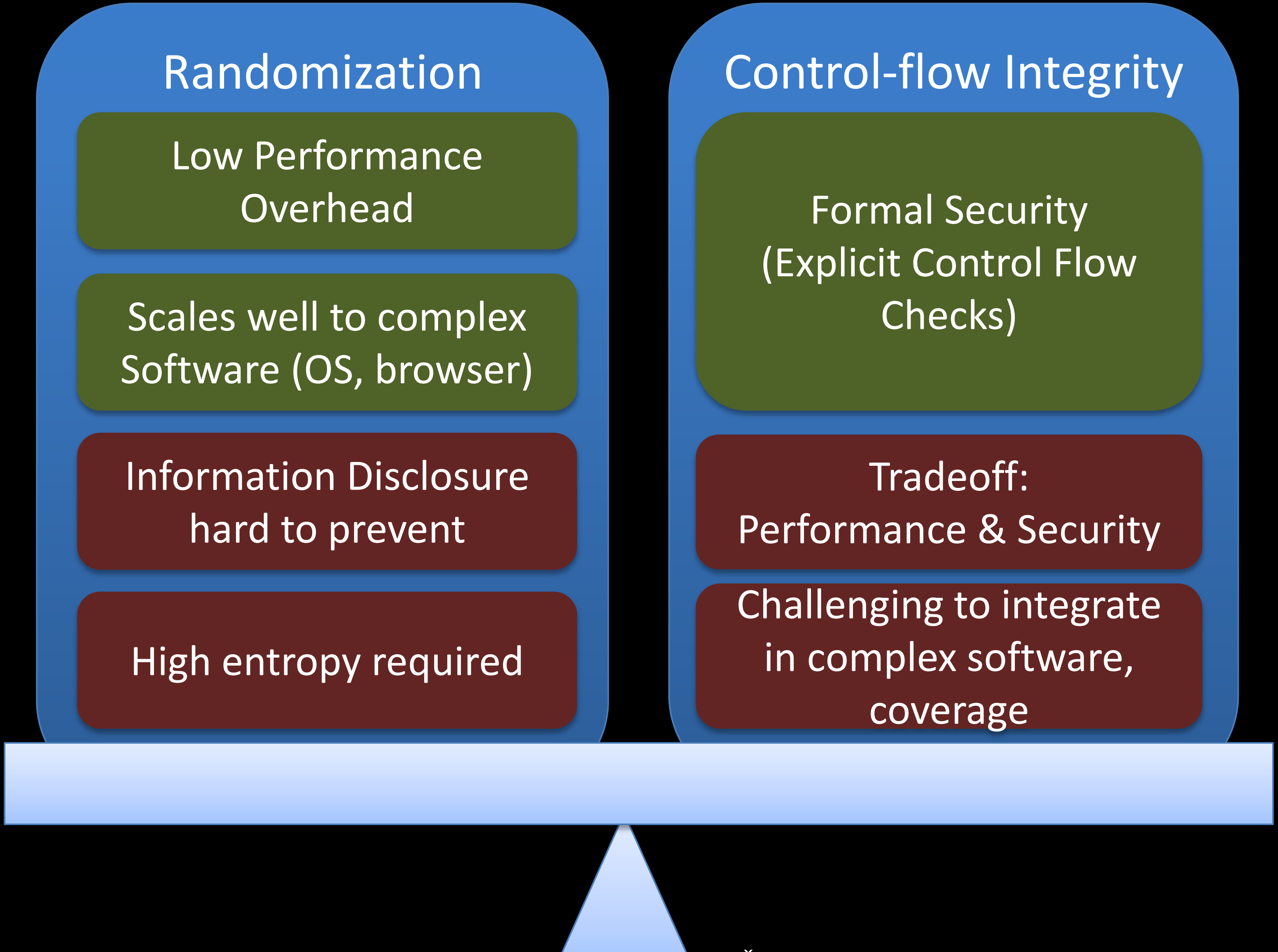
4 Manipulate writable Memory

5 Computing Engine

Main Defenses against Code Reuse

1. Code Randomization
2. Control-Flow Integrity (CFI)

Randomization vs. CFI



Randomization

- Low Performance Overhead
- Scales well to complex Software (OS, browser)
- Information Disclosure hard to prevent
- High entropy required

Control-flow Integrity

- Formal Security (Explicit Control Flow Checks)
- Tradeoff: Performance & Security
- Challenging to integrate in complex software, coverage

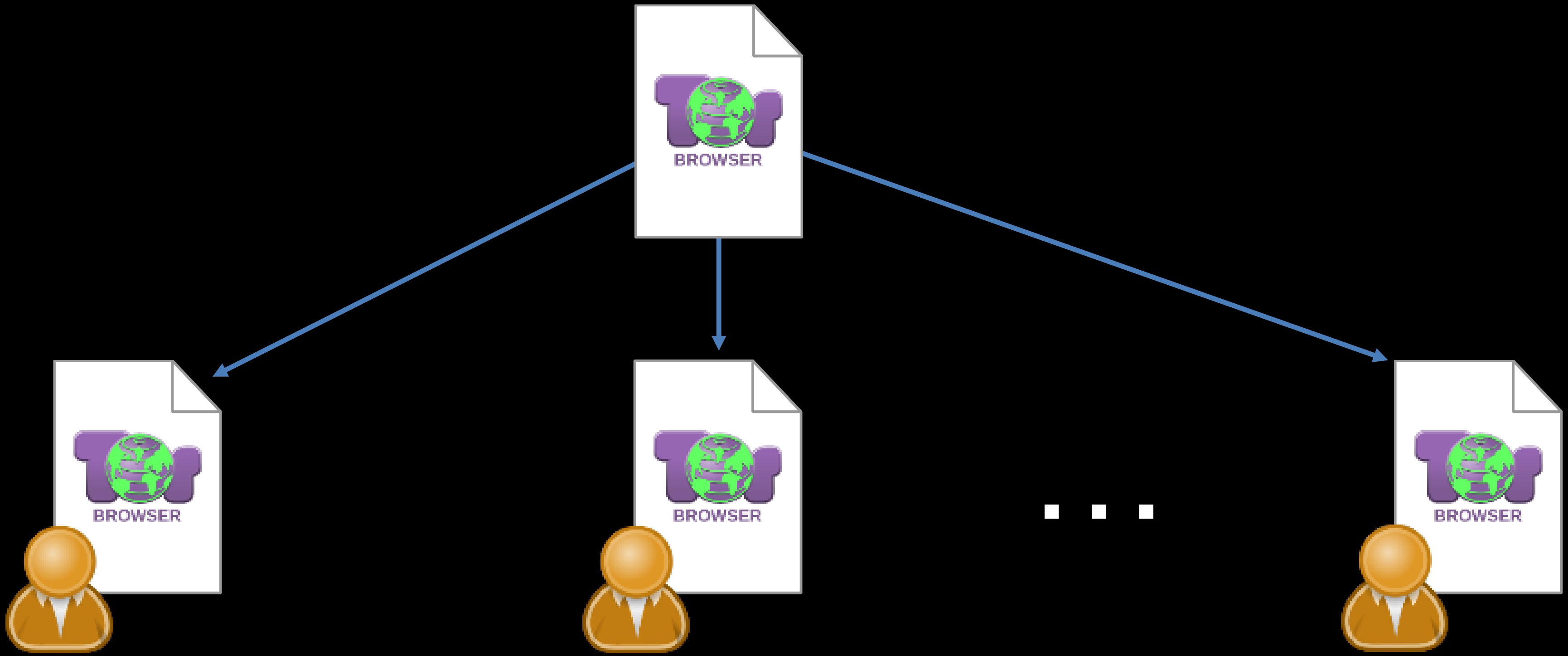
CYSEC
Cybersecurity
TU Darmstadt

Code Randomization

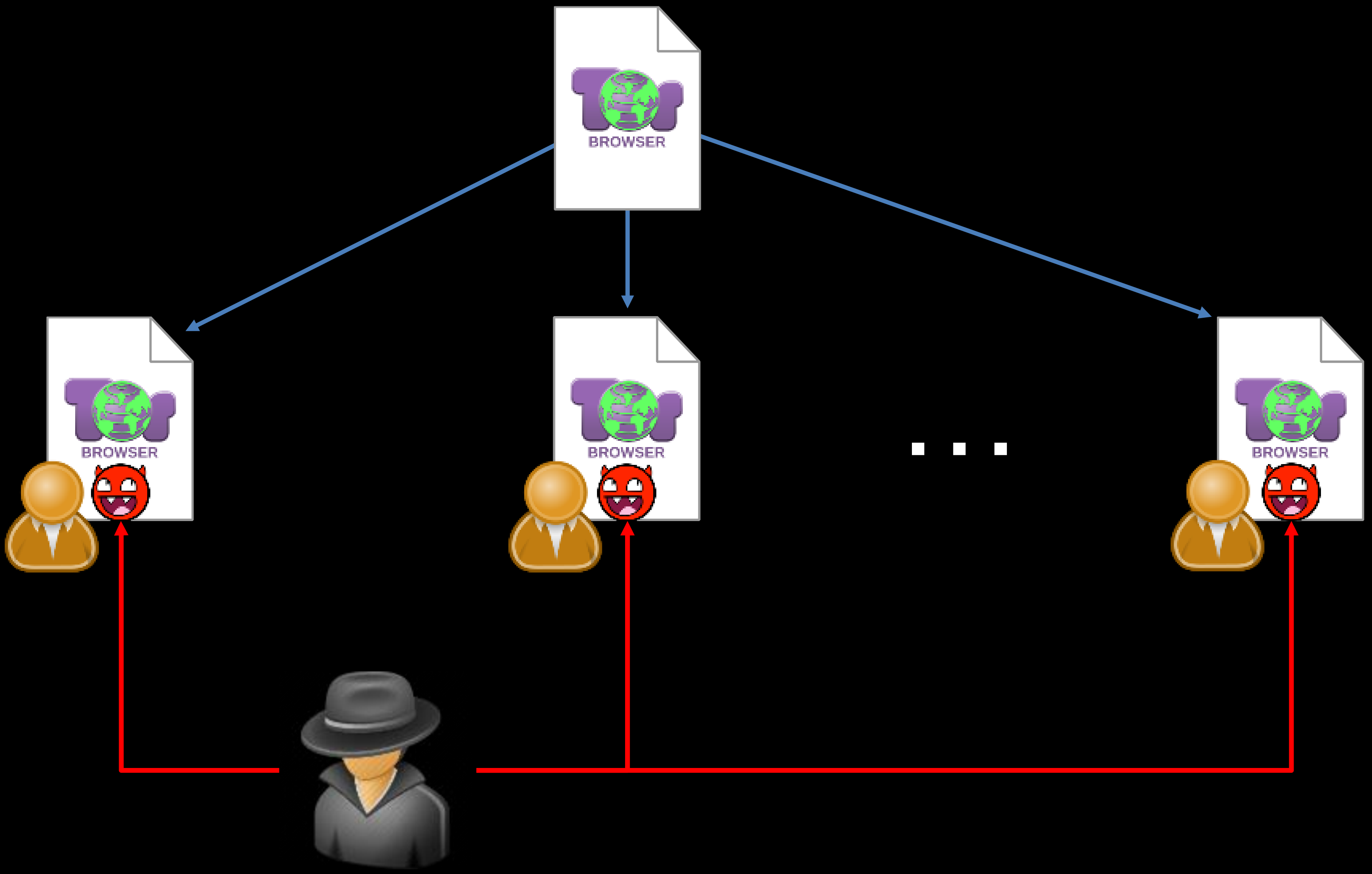
Becoming a Moving Target

Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

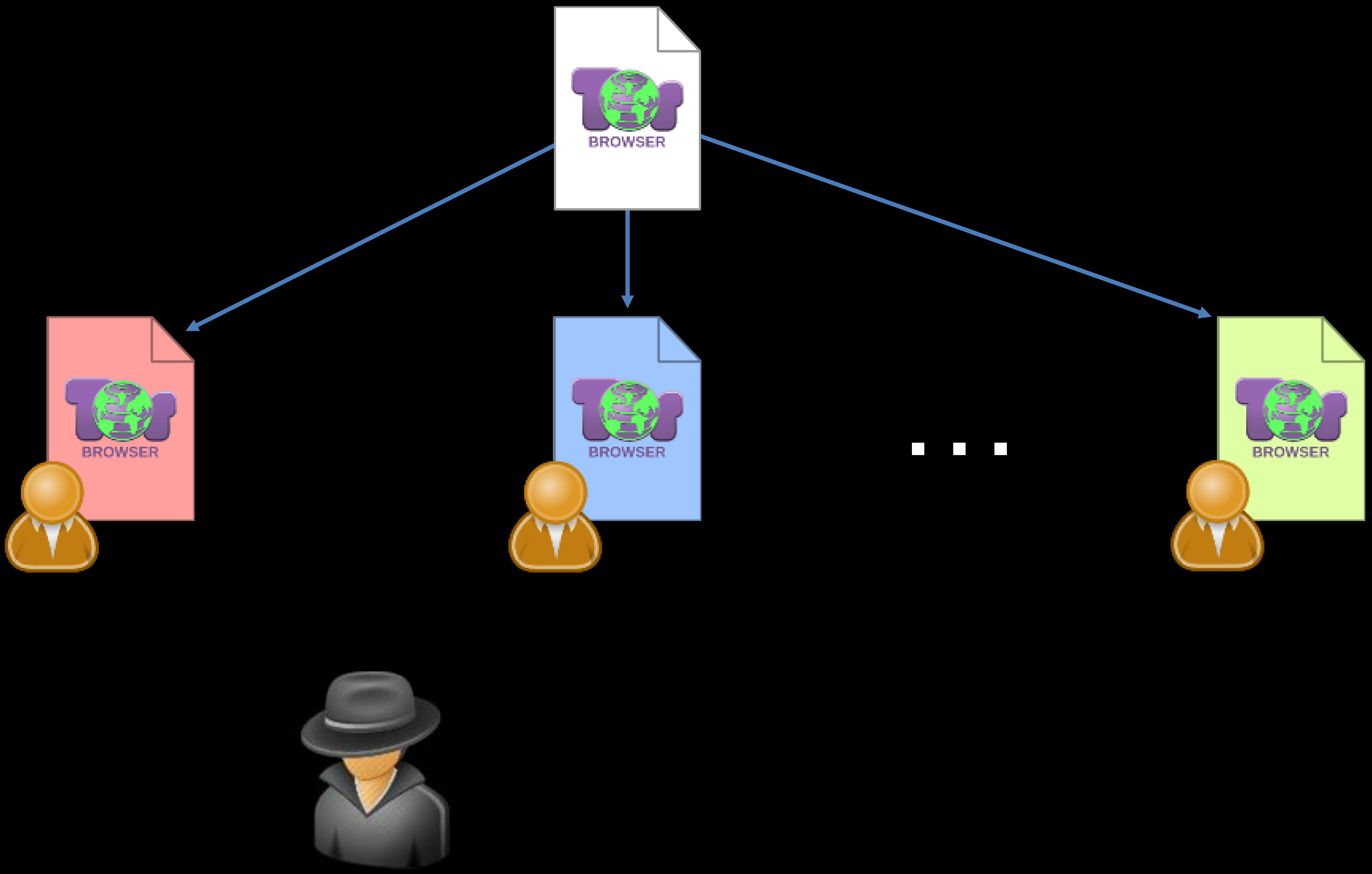
General Idea: Software Diversity



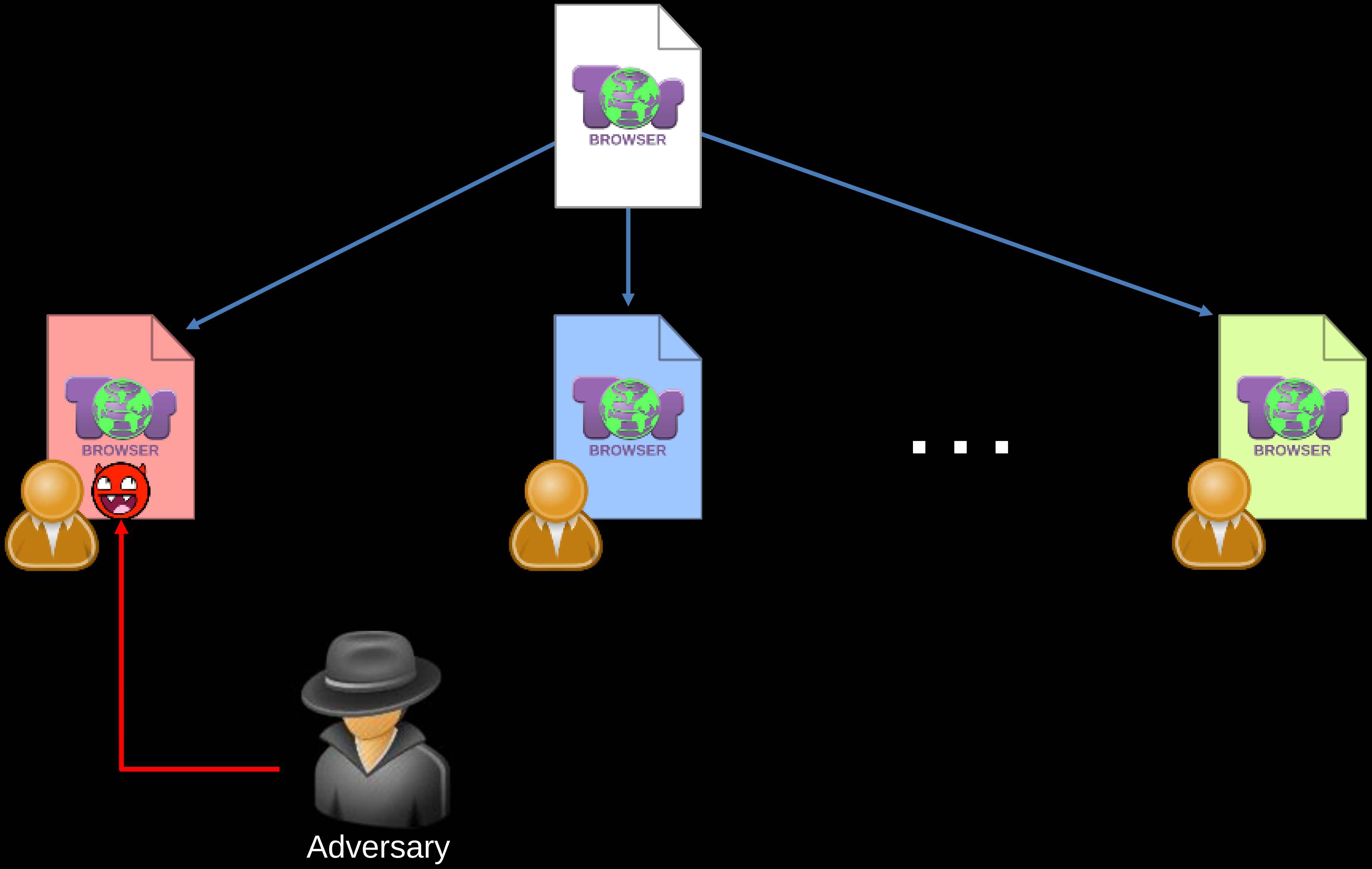
General Idea: Software Diversity



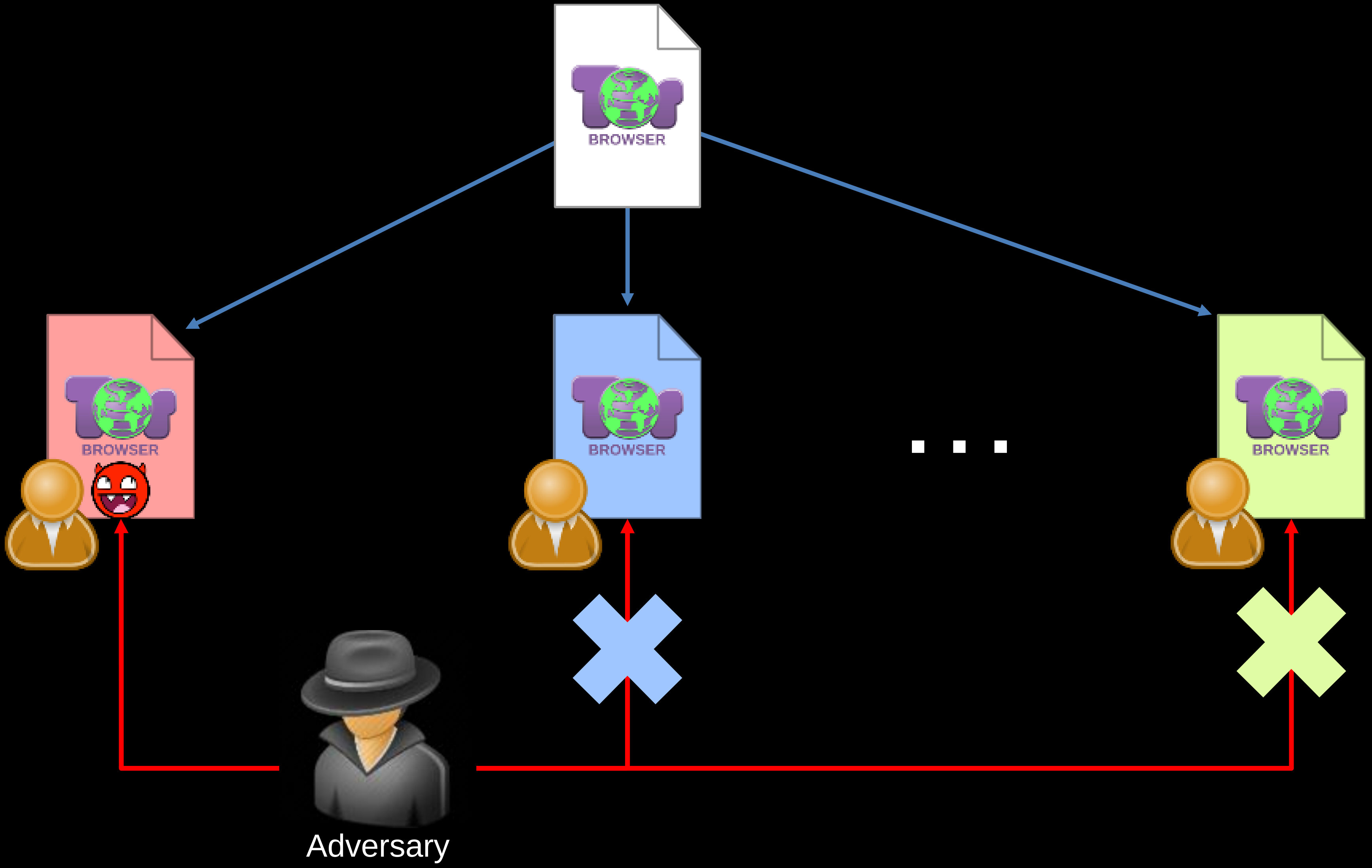
General Idea: Software Diversity



General Idea: Software Diversity

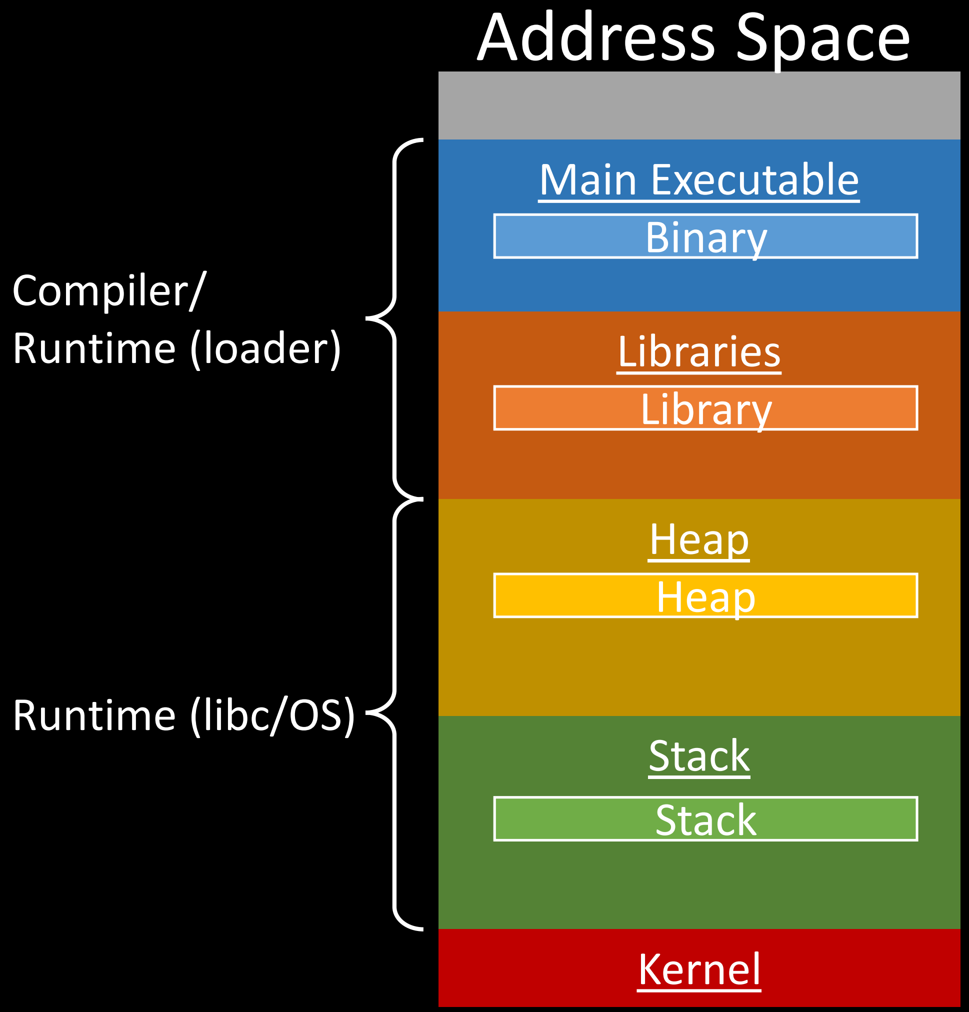


General Idea: Software Diversity



Address Space Layout Randomization (ASLR)

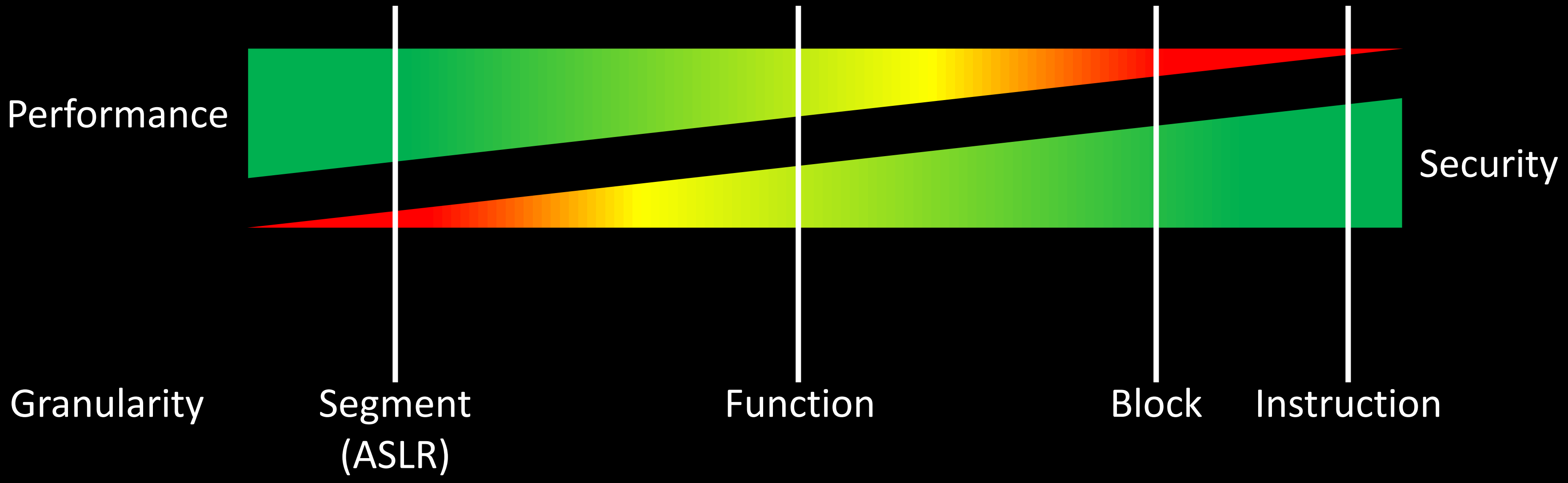
- Supported by all main operating systems
  - Windows, macOS, Linux, Android, iOS
- Randomizes
  - Code (main executable, libraries)
  - Data (stack, heap)



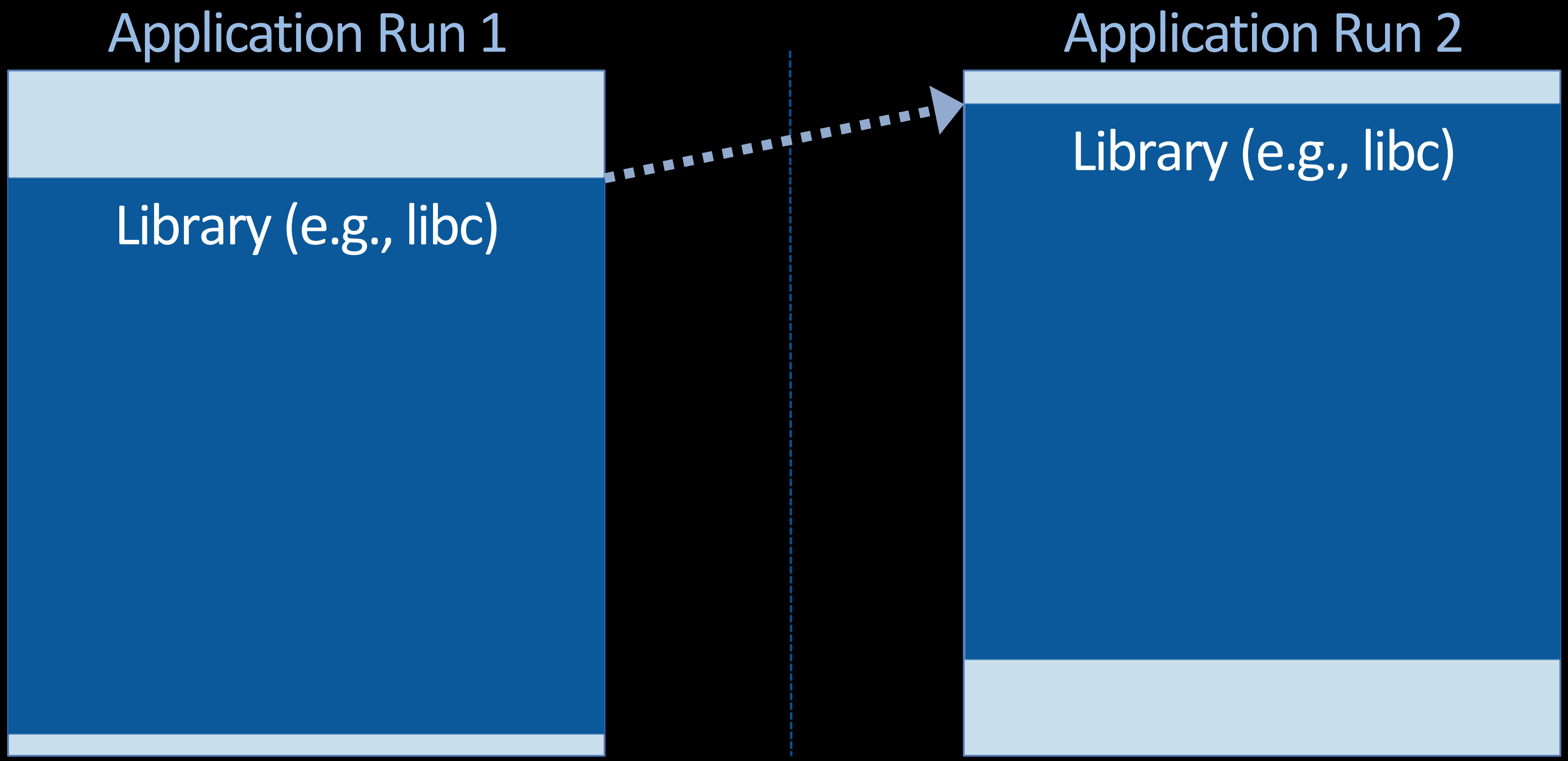
Address Space Layout Randomization in Practice

	Region	Windows 10 (Visual Studio 2015)	macOS 10.12.3 (clang 8.0)	Ubuntu 16.04.1 (gcc 5.4)
Program restart	Binary	0x7FF765C91070	0x00010b23ce80	0x0000004005d6
	Library	0x7FFB46458940	0x7fffddd8c180	0x7fbca76a5800
	Stack	0x0015962FFC90	0x7fff549c3a80	0x7ffd36967a08
	Heap (small)	0x01B5559BE230	0x7ff524502740	0x00000200c010
	Heap (big)	0x01B5559BE3F0	0x7ff524801000	0x00000200c030
	Binary	0x7FF765C91070	0x0001030fbe80	0x0000004005d6
	Library	0x7FFB46458940	0x7fffddd8c180	0x7f0d76c35800
	Stack	0x009998CFFE40	0x7fff5cb04a80	0x7fffad7736a8
	Heap (small)	0x0246CF58C190	0x7fc969402760	0x000000c04010
	Heap (big)	0x0246CF58E7F0	0x7fc969801000	0x000000c04030
Reboot	Binary	0x7FF60A871070	0x000106059e80	0x0000004005d6
	Library	0x7FFA9ACC8940	0x7fffbae1a180	0x7f5c6cb1b800
	Stack	0x0038AF74FDE0	0x7fff59ba6a60	0x7ffdd1a26848
	Heap (small)	0x0158059EC490	0x7f8fe9402740	0x00000257e010
	Heap (big)	0x0158059EE7F0	0x7f8fe9801000	0x00000257e030

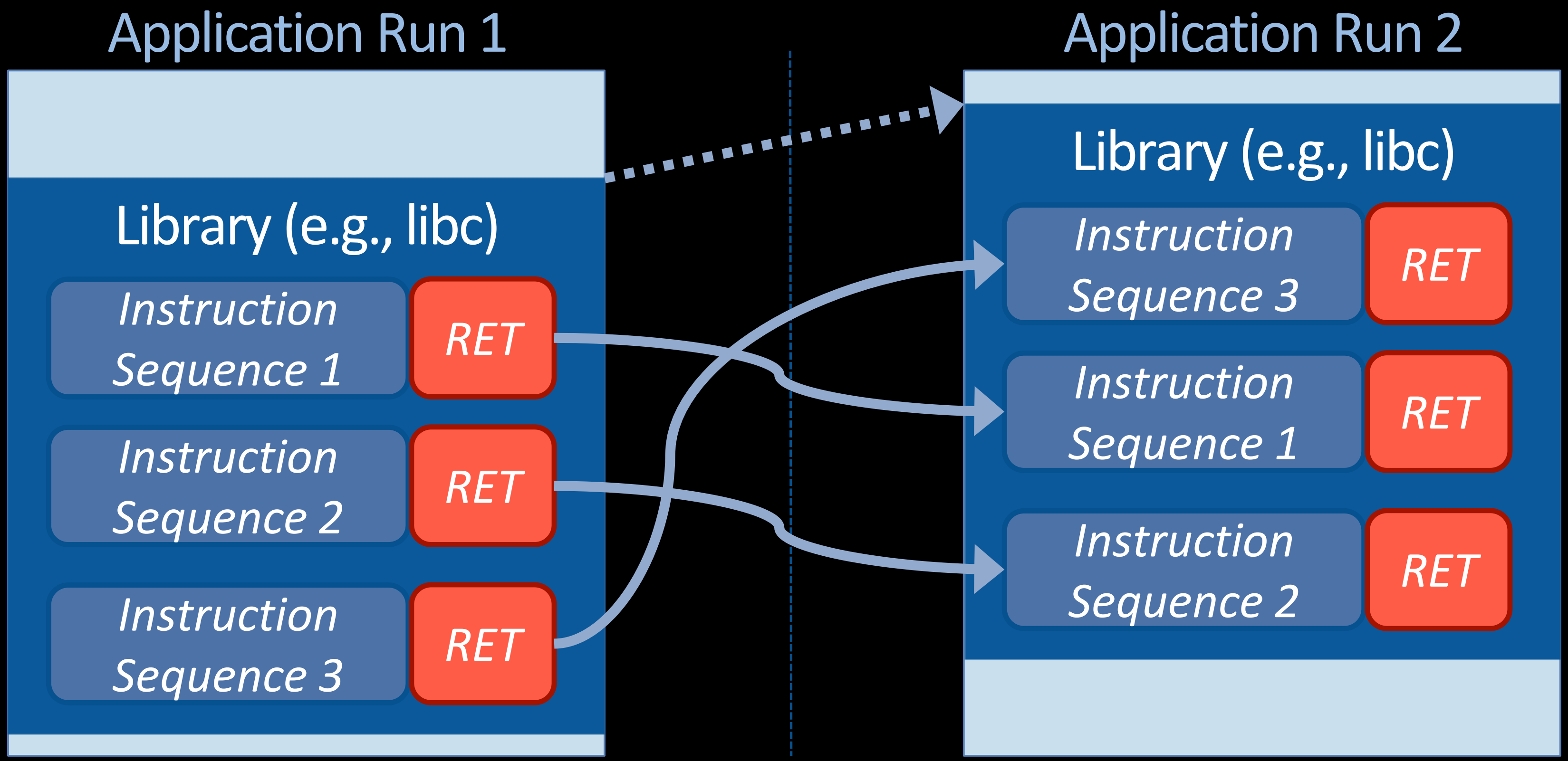
Randomization Granularity



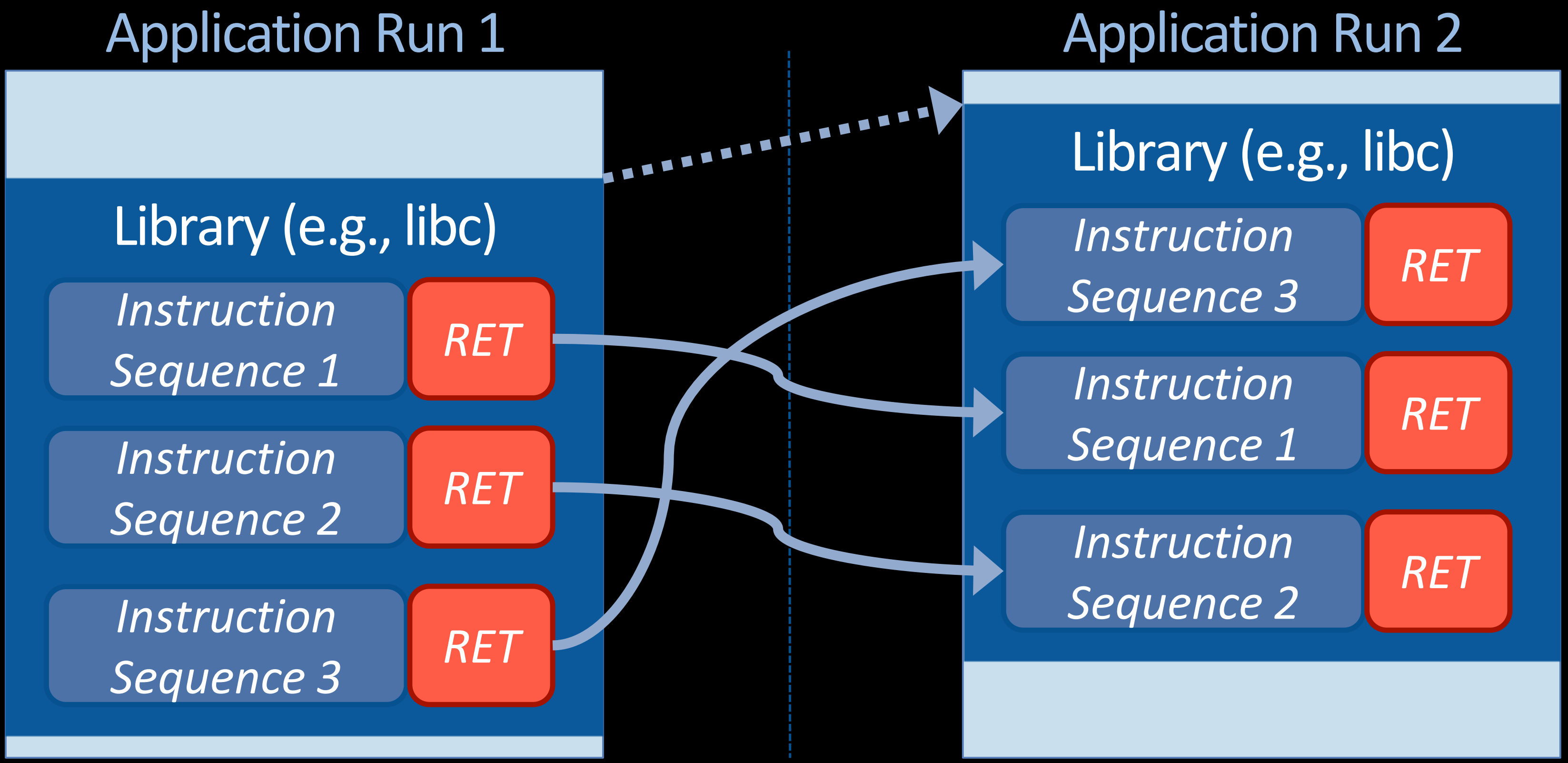
Fine-Grained ASLR



Fine-Grained ASLR



Fine-Grained ASLR



- Instruction reordering/substitution within a BBL
  ORP [Pappas et al., IEEE S&P 2012]
- Randomizing each instruction‘s location:
  ILR [Hiser et al., IEEE S&P 2012]
- Permutation of BBLs:
  STIR [Wartell et al., CCS 2012] & **XIFER** [with Davi et al., AsiaCCS 2013]

Randomization Vulnerable to Memory Leakage

Direct memory disclosure

- Pointer leakage on code pages
- e.g., direct call and jump instruction

Indirect memory disclosure

- Pointer leakage on data pages such as stack or heap
- e.g., return addresses, function pointers, pointers in vTables

JIT-ROP:
Bypassing Randomization via Direct Memory Disclosure

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

IEEE Security and Privacy 2013, and Blackhat 2013

Kevin Z. Snow, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen,
Fabian Monrose, Ahmad-Reza Sadeghi

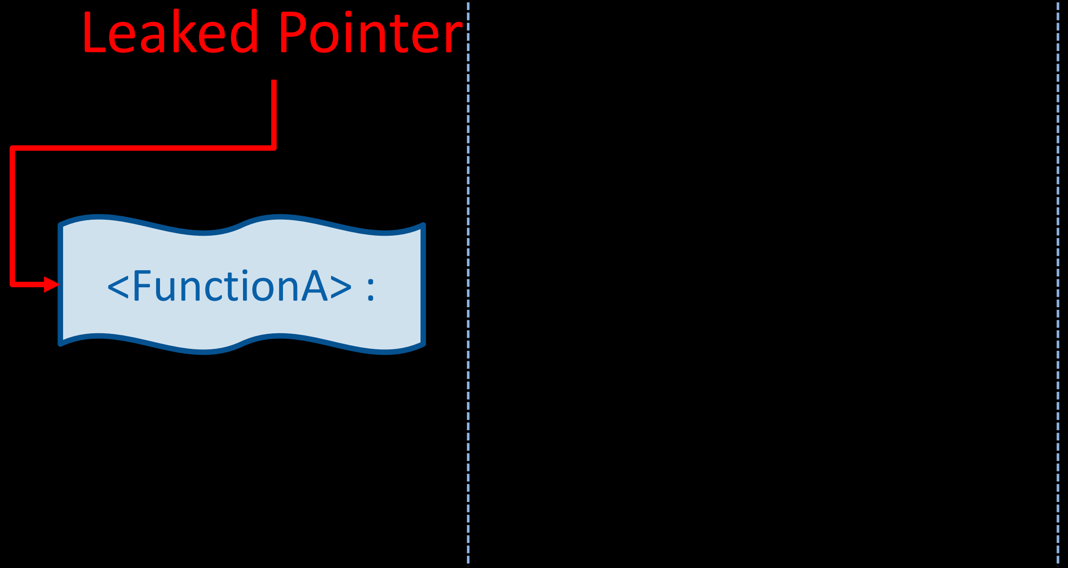
Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

Just-In-Time ROP:
Direct Memory Disclosure

1. Undermines fine-grained ASLR
2. Shows memory disclosures are far more damaging than believed
3. Can be instantiated with real-world exploit

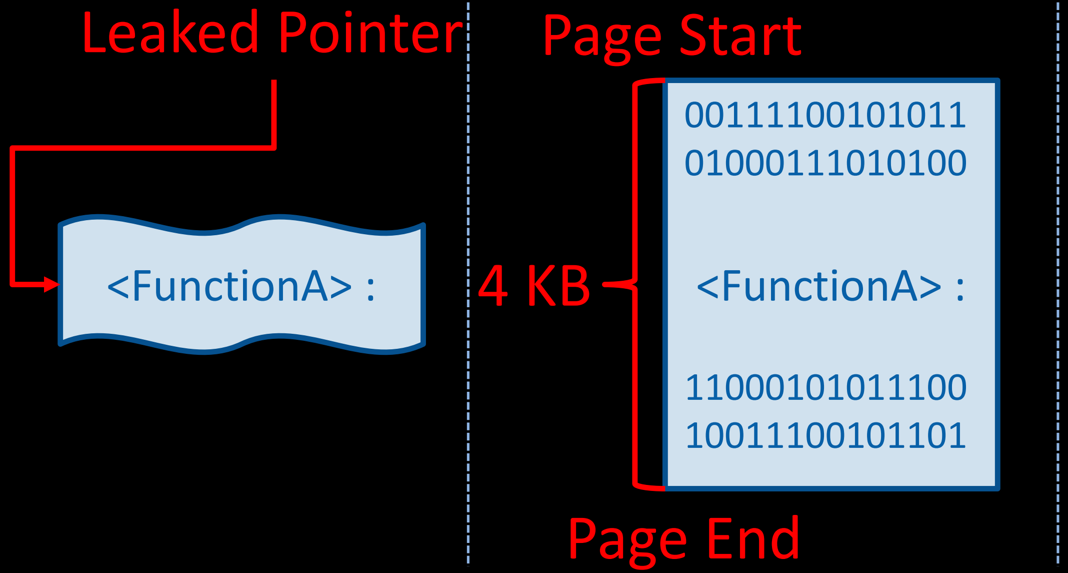
Key Insight and Observation

- Goal: Exploit a memory disclosure
  - Leak of a single address leading to leak of entire memory pages
- Observations
  - Leaked address will reside in a 4KB aligned memory page
  - Determine the page boundaries and disassemble the 4 KB page
  - Disassembled page contains references to other pages



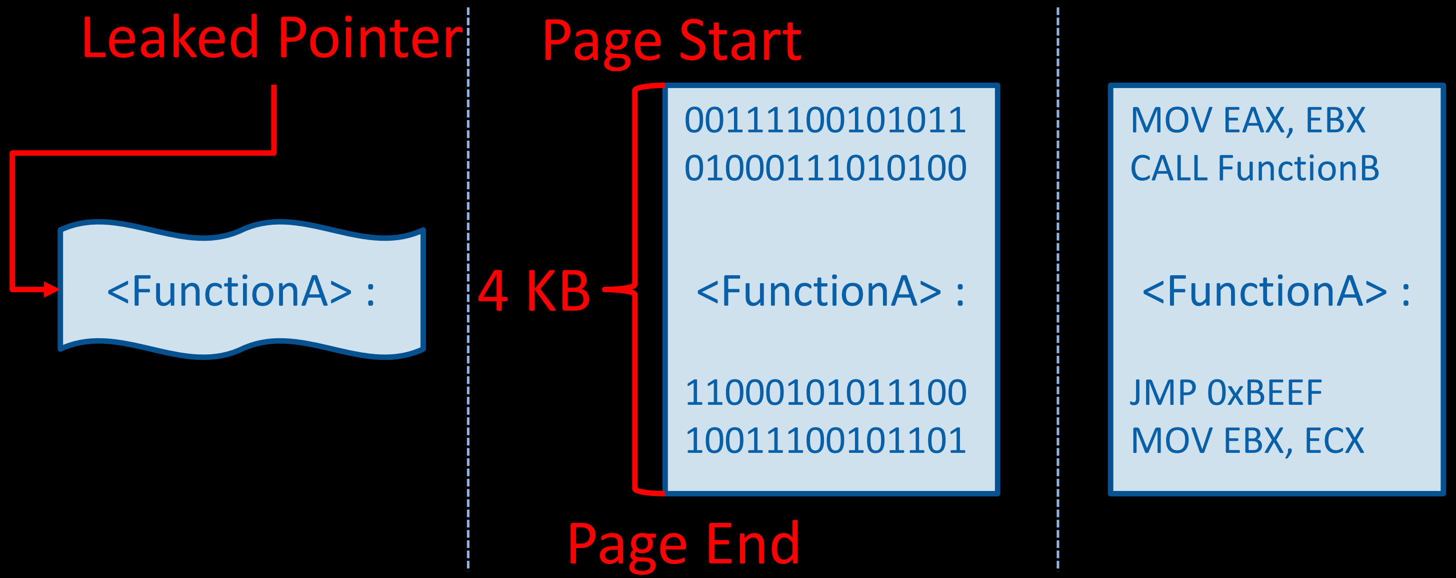
Key Insight and Observation

- Goal: Exploit a memory disclosure
  - Leak of a single address leading to leak of entire memory pages
- Observations
  - Leaked address will reside in a 4KB aligned memory page
  - Determine the page boundaries and disassemble the 4 KB page
  - Disassembled page contains references to other pages



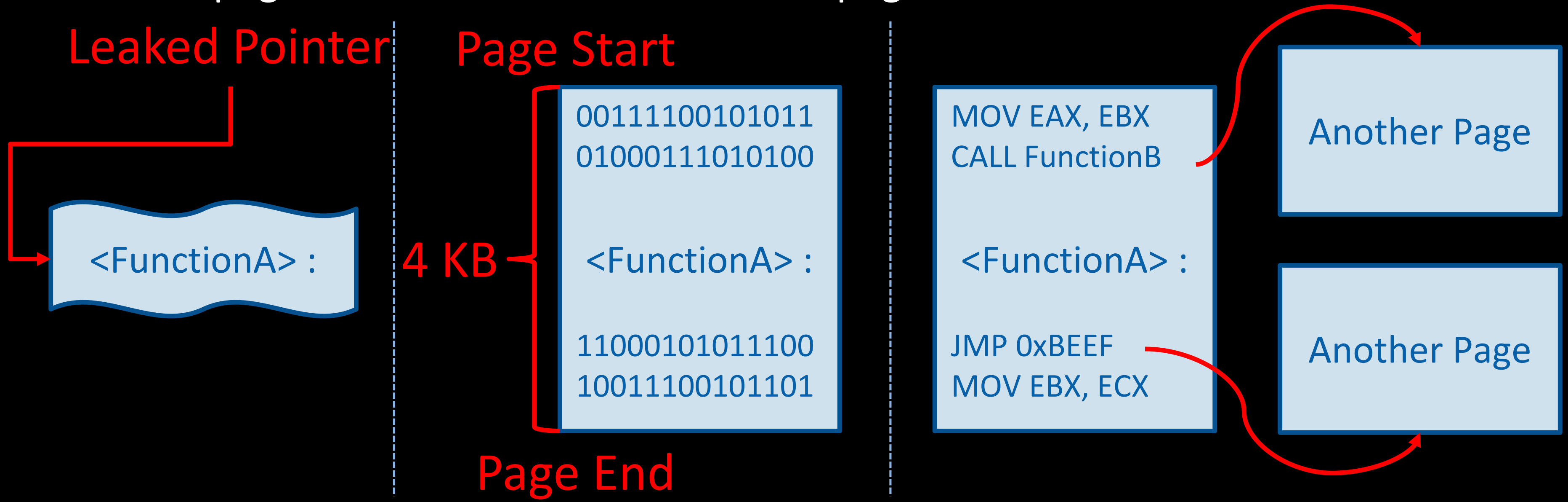
Key Insight and Observation

- Goal: Exploit a memory disclosure
  - Leak of a single address leading to leak of entire memory pages
- Observations
  - Leaked address will reside in a 4KB aligned memory page
  - Determine the page boundaries and disassemble the 4 KB page
  - Disassembled page contains references to other pages



Key Insight and Observation

- Goal: Exploit a memory disclosure
  - Leak of a single address leading to leak of entire memory pages
- Observations
  - Leaked address will reside in a 4KB aligned memory page
  - Determine the page boundaries and disassemble the 4 KB page
  - Disassembled page contains references to other pages



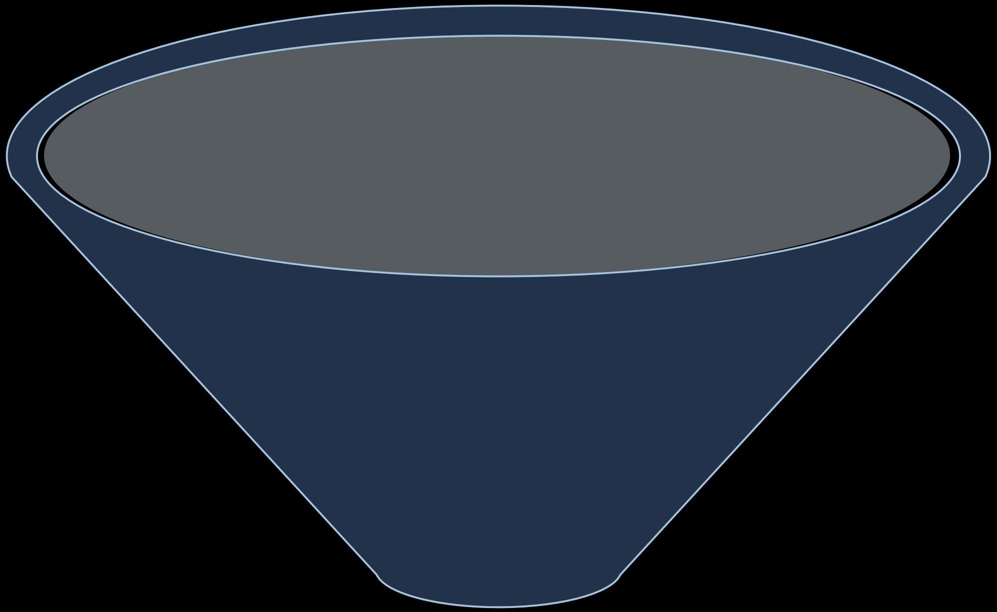
Gadget Finding and Payload Generation

Page 1

```
MOV EAX, EBX
RET
 …
SUB EAX, EBX
RET
 …
MOV EAX, (EBX)
RET
```

Page 2

```
INT 0x80

 …

 …

MOV ESP, EAX
RET
```



Gadget Pool

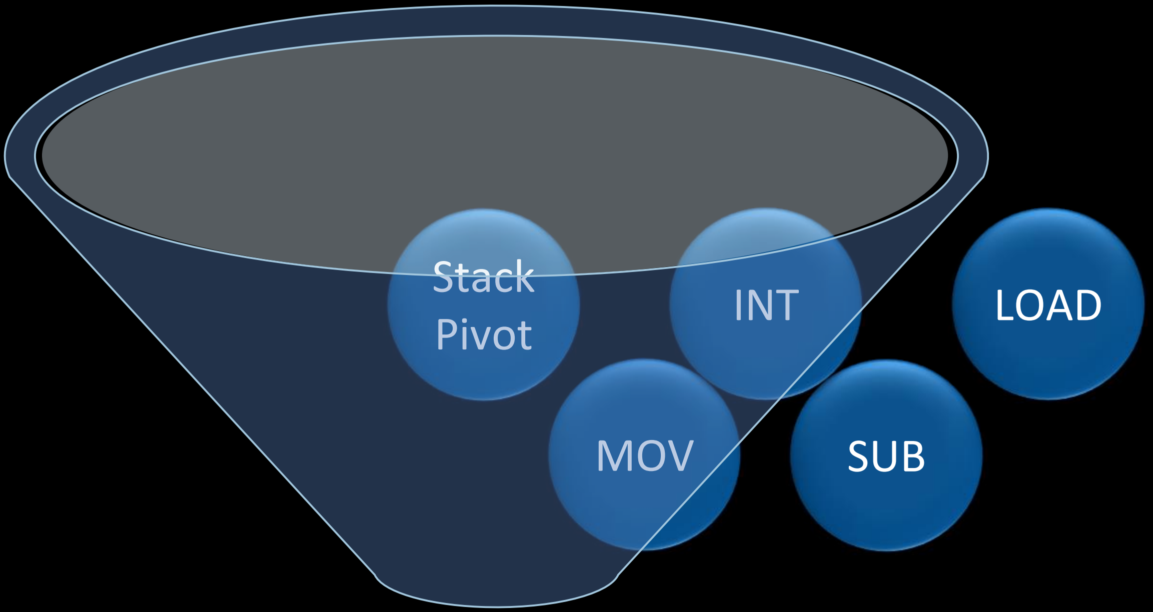
Gadget Finding and Payload Generation

Page 1

```
MOV EAX, EBX
RET
 ...
SUB EAX, EBX
RET
 ...
MOV EAX, (EBX)
RET
```

Page 2

```
INT 0x80


 ...

 ...

MOV ESP, EAX
RET
```



Gadget Pool

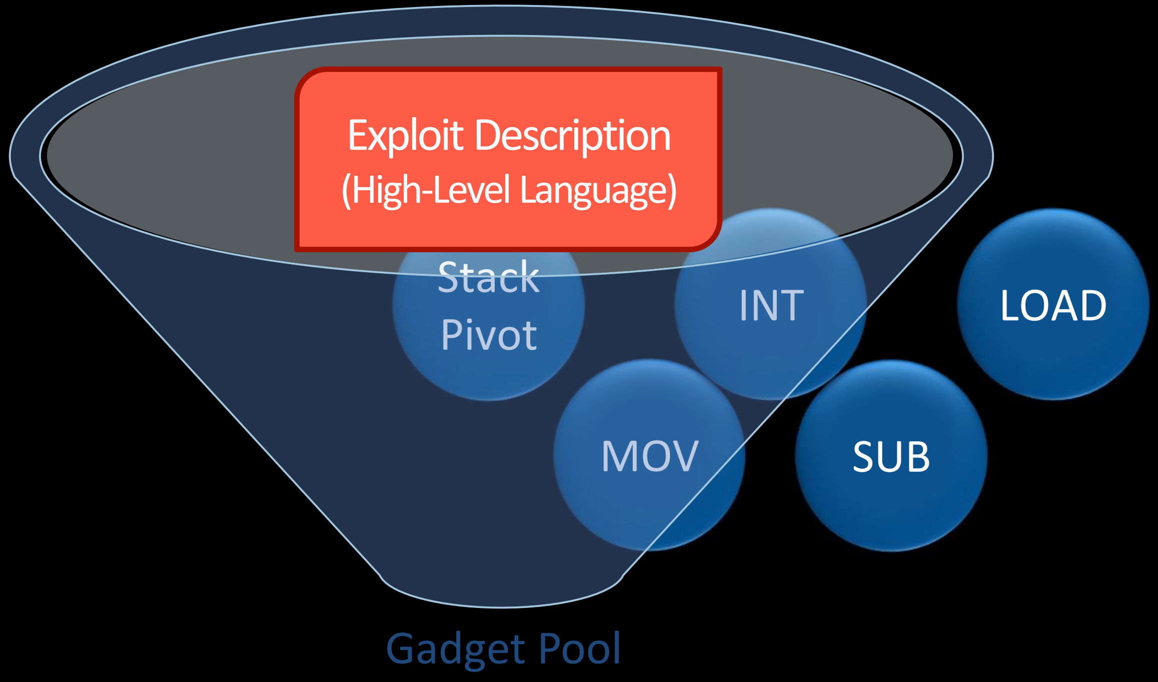
Gadget Finding and Payload Generation

Page 1

```
MOV EAX, EBX
RET
 …
SUB EAX, EBX
RET
 …
MOV EAX, (EBX)
RET
```

Page 2

```
INT 0x80


 …

 …

MOV ESP, EAX
RET
```



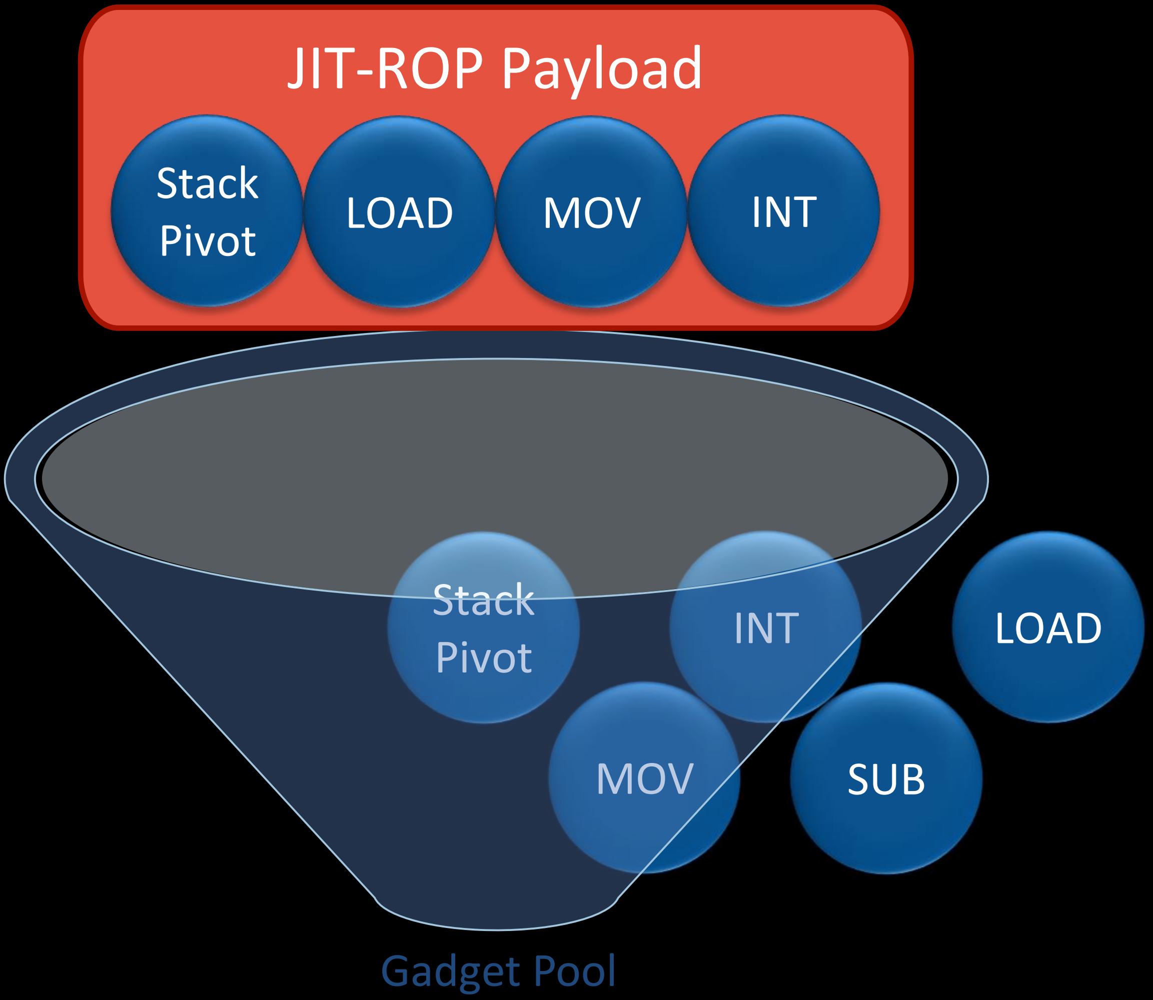
Gadget Finding and Payload Generation

Page 1

```
MOV EAX, EBX
RET
 …
SUB EAX, EBX
RET
 …
MOV EAX, (EBX)
RET
```

Page 2

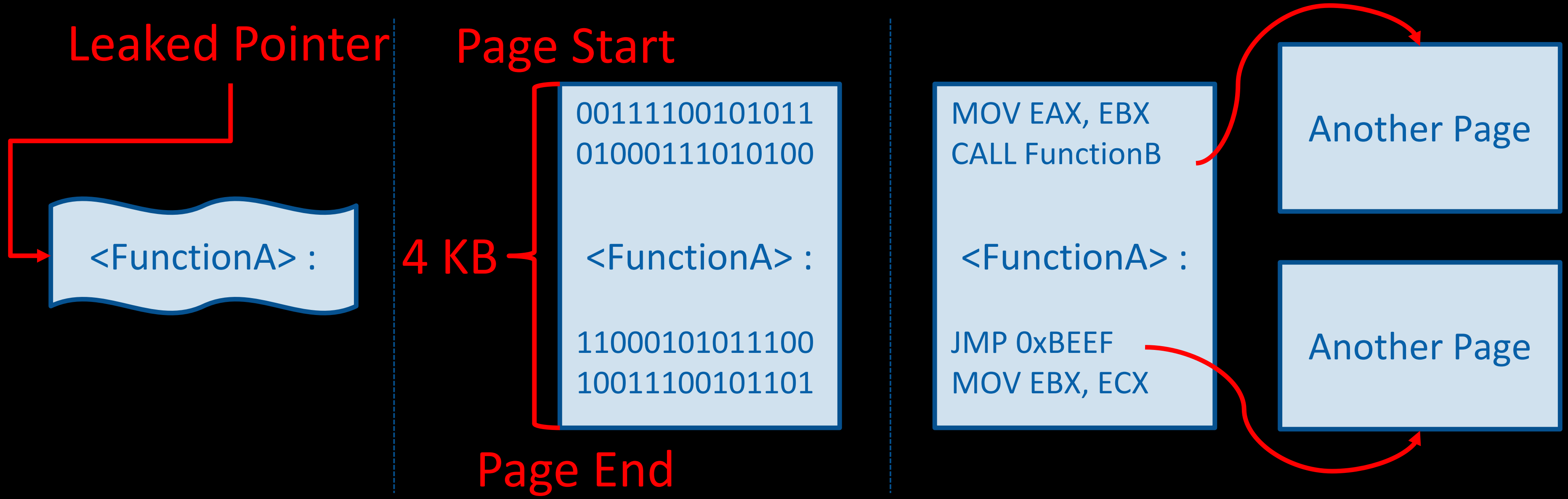
```
INT 0x80


 …

 …

MOV ESP, EAX
RET
```



Example Defense Proposals against JIT-ROP

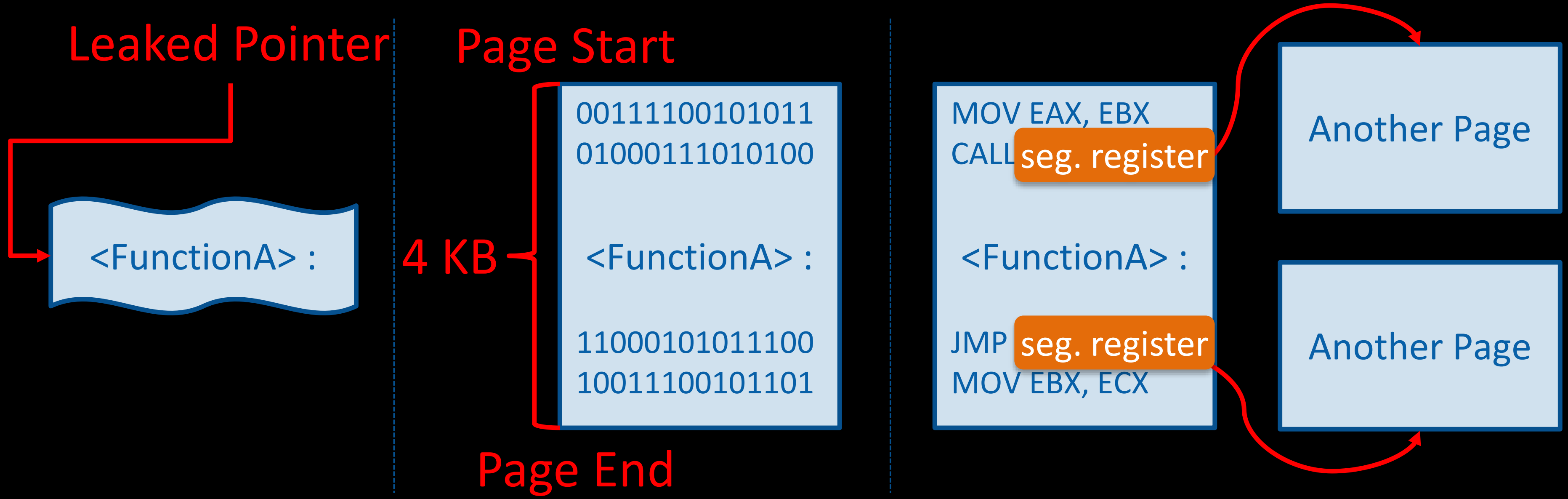
Attempt to Ptevent Direct Code Disclosure

- Oxymoron [Backes et al. USENIX‘14]
  - Tries to prevent of ROP and JIT-ROP by obfuscating direct code references
  - Unfortunately ignores information leakage from the data sector



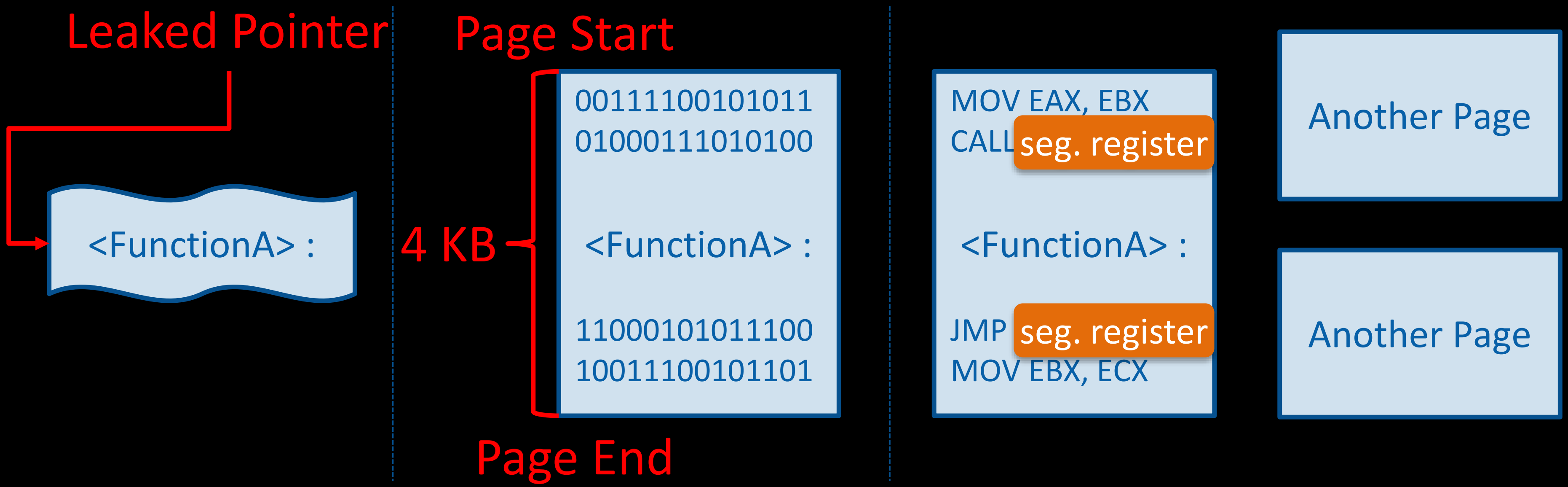
Attempt to Ptevent Direct Code Disclosure

- Oxymoron [Backes et al. USENIX'14]
  - Tries to prevent of ROP and JIT-ROP by obfuscating direct code references
  - Unfortunately ignores information leakage from the data sector



Attempt to Ptevent Direct Code Disclosure

- Oxymoron [Backes et al. USENIX‘14]
  - Tries to prevent of ROP and JIT-ROP by obfuscating direct code references
  - Unfortunately ignores information leakage from the data sector



Isomeron:
Fully bypasses Oxymoron via Indirect Memory Disclosure

Isomeron:

Code Randomization Resilient to (Just-In-Time) Return-Oriented Programming

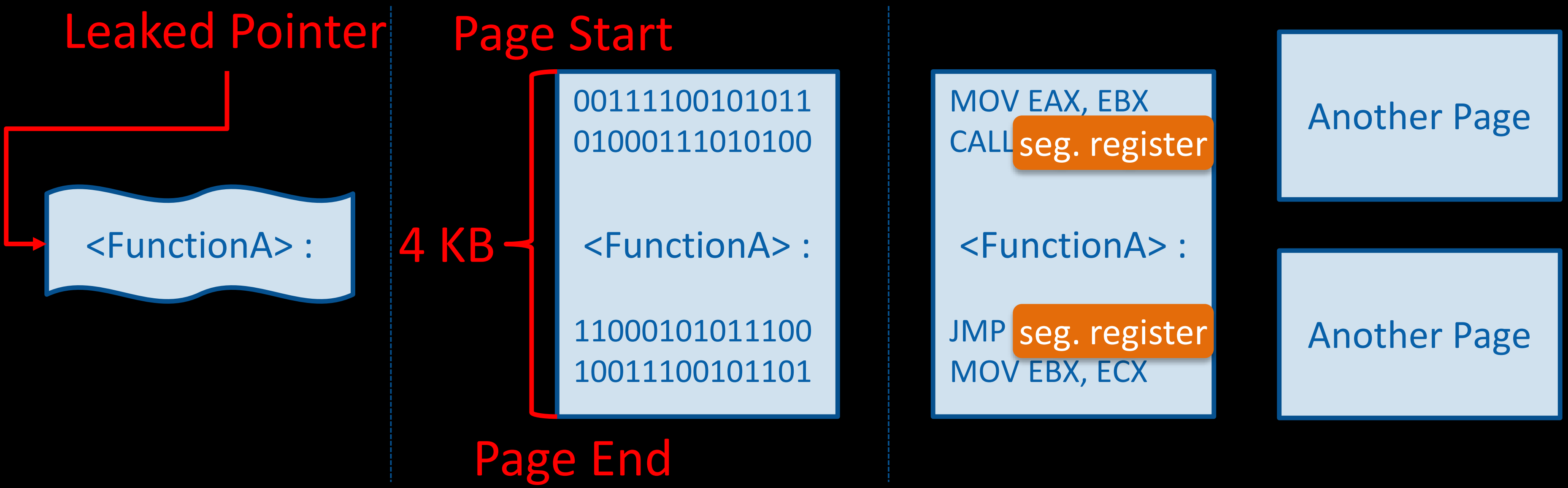
The Network and Distributed System Security Symposium (NDSS) 2015

Lucas Davi, Christopher Liebchen, Ahmad-Reza Sadeghi,

Kevin Snow, Fabian Monrose

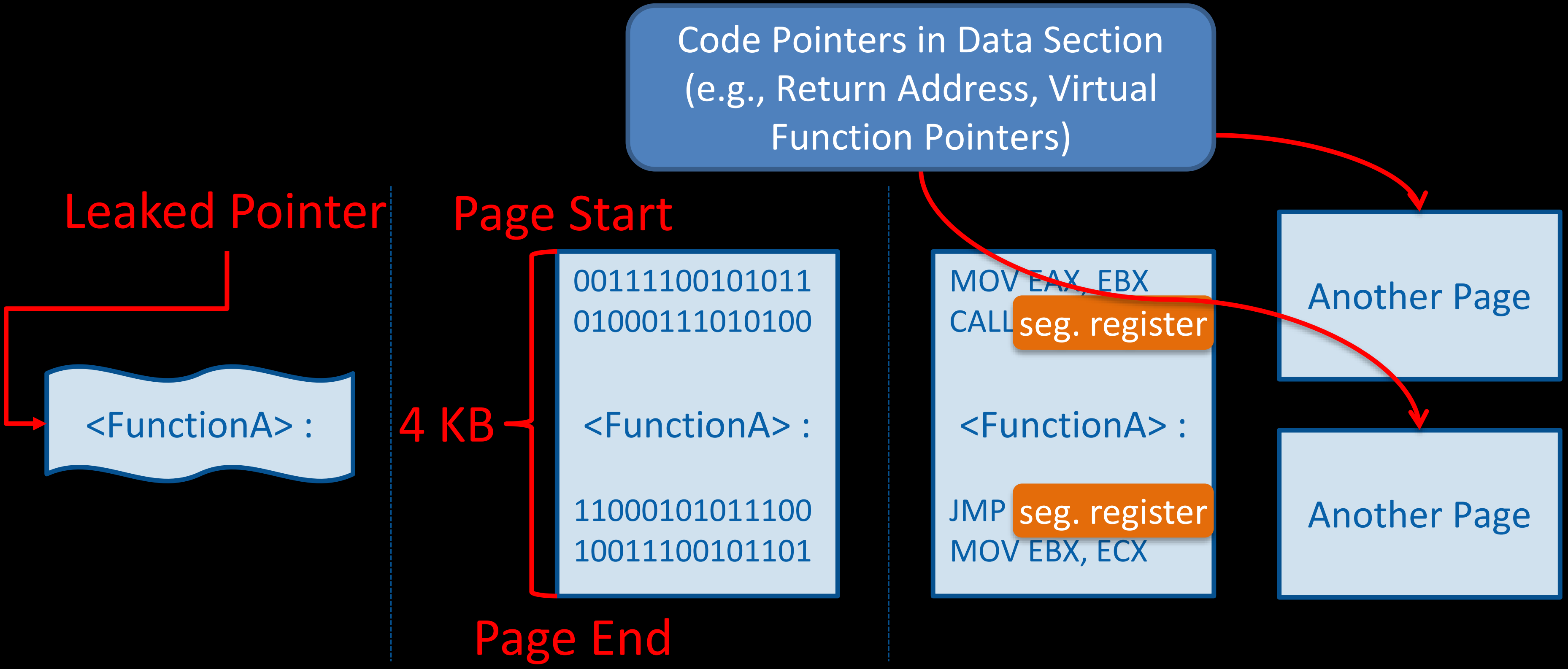
Bypass Oxymoron via Indirect Disclosure

- Isomeron [Davi et al. NDSS'15]
  - Discover code pages through code pointers in data structures



Bypass Oxymoron via Indirect Disclosure

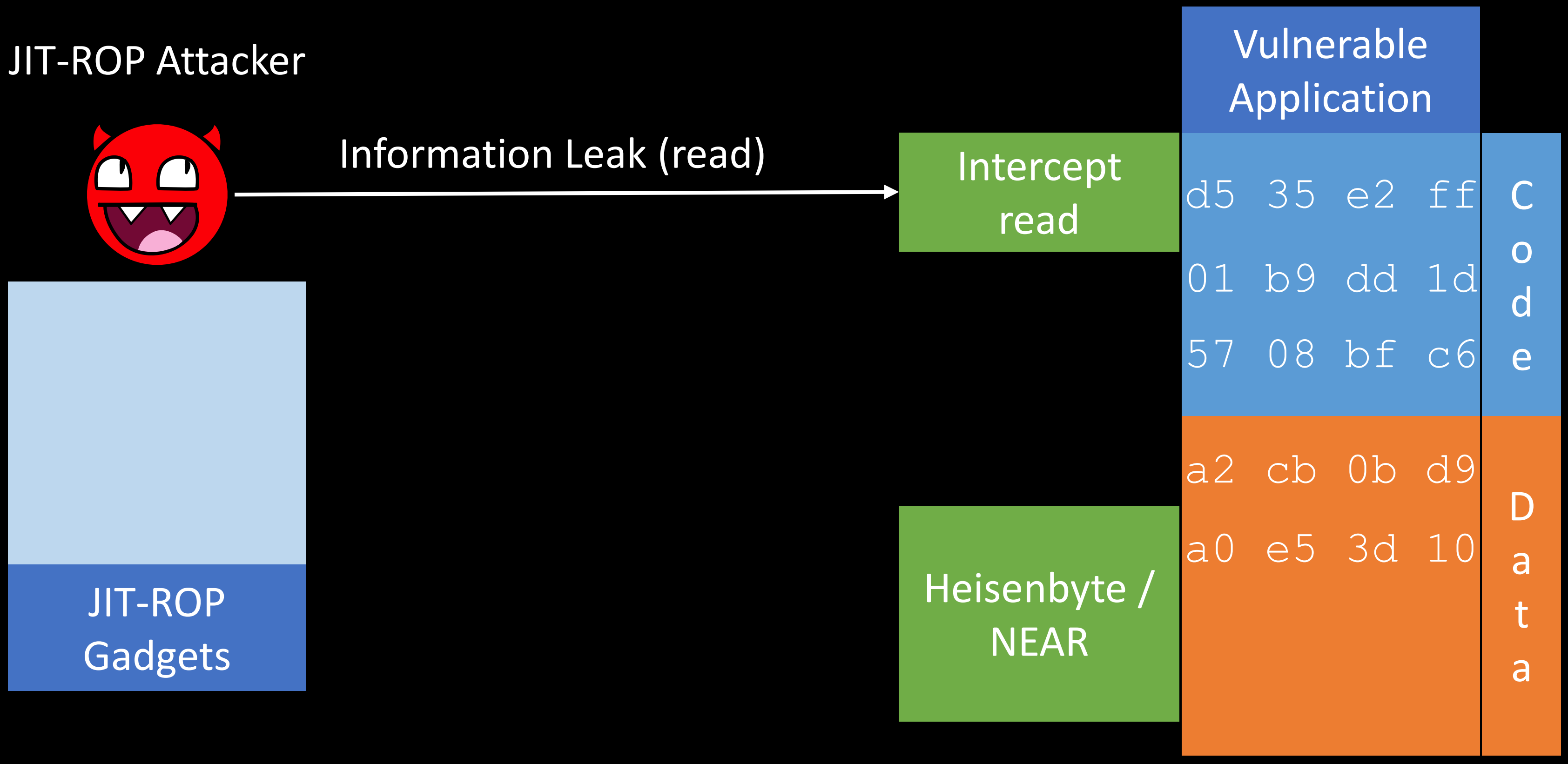
- Isomeron [Davi et al. NDSS‘15]
  - Discover code pages through code pointers in data structures



Heisenbyte/NEAR

Heisenbyte
[Tang et al. CCS’15]

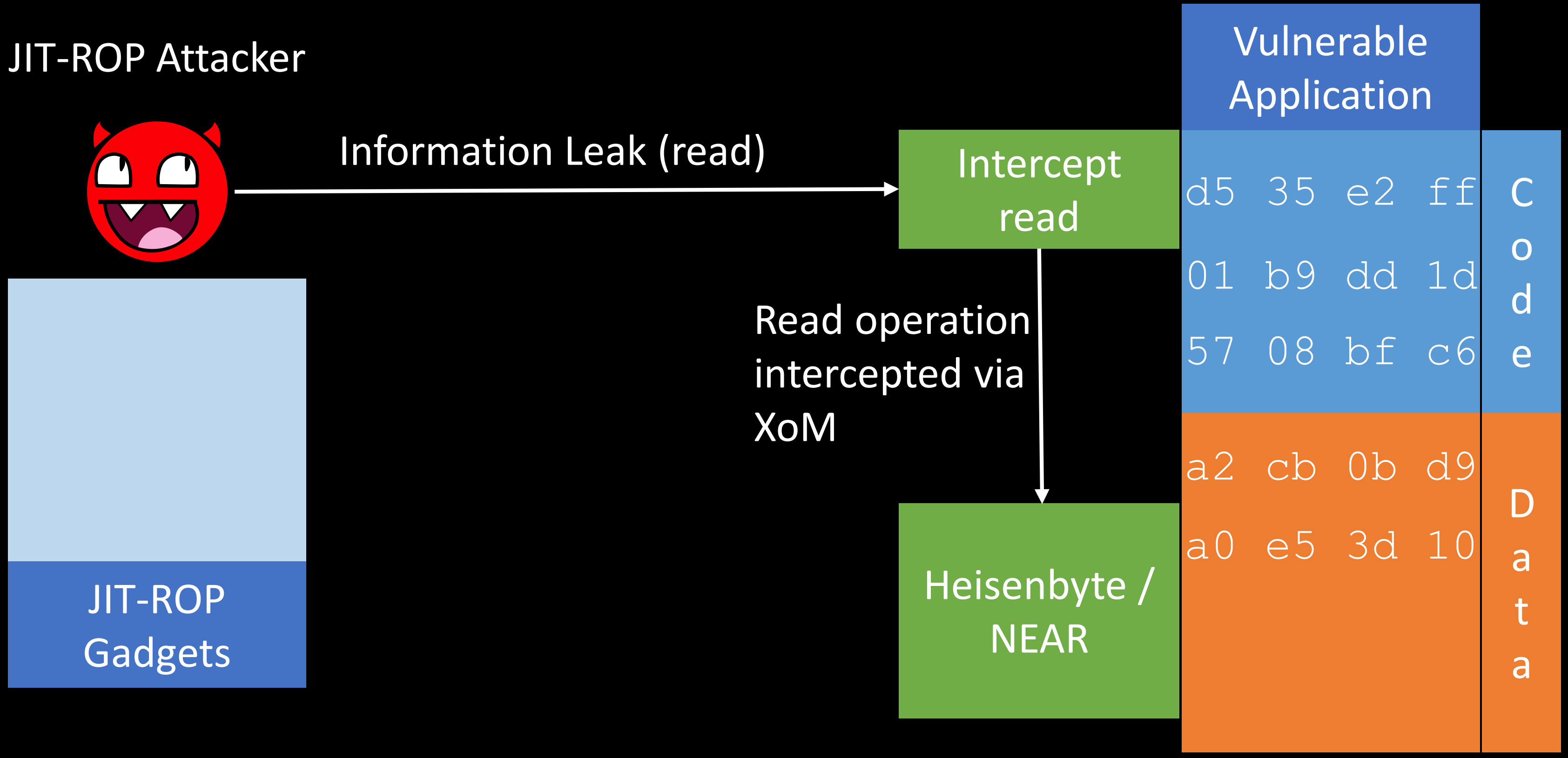
No Execute After Read (NEAR)
[Werner et al. ASIACCS’16]



Heisenbyte/NEAR

Heisenbyte
[Tang et al. CCS'15]

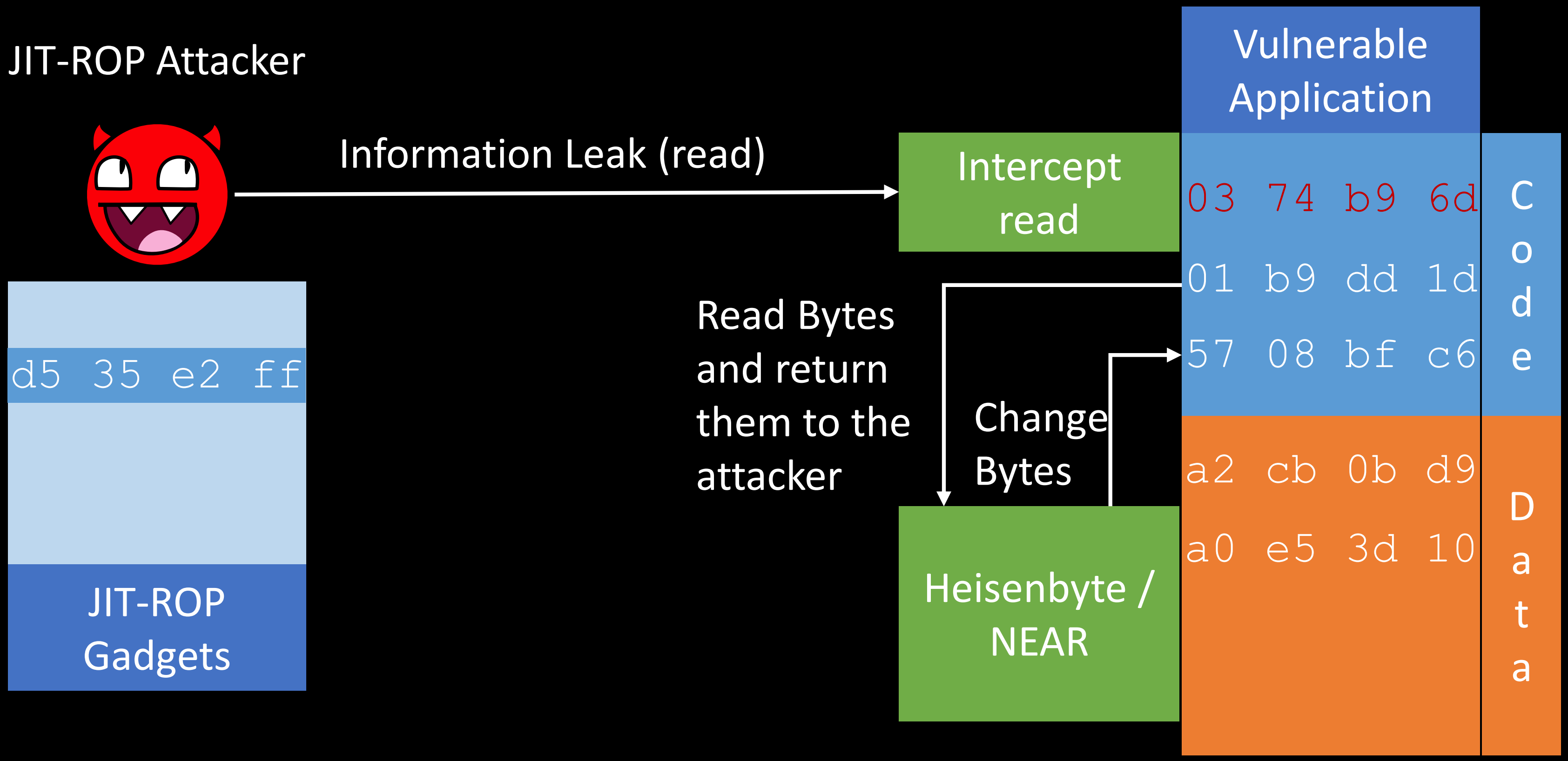
No Execute After Read (NEAR)
[Werner et al. ASIACCS'16]



Heisenbyte/NEAR

Heisenbyte
[Tang et al. CCS'15]

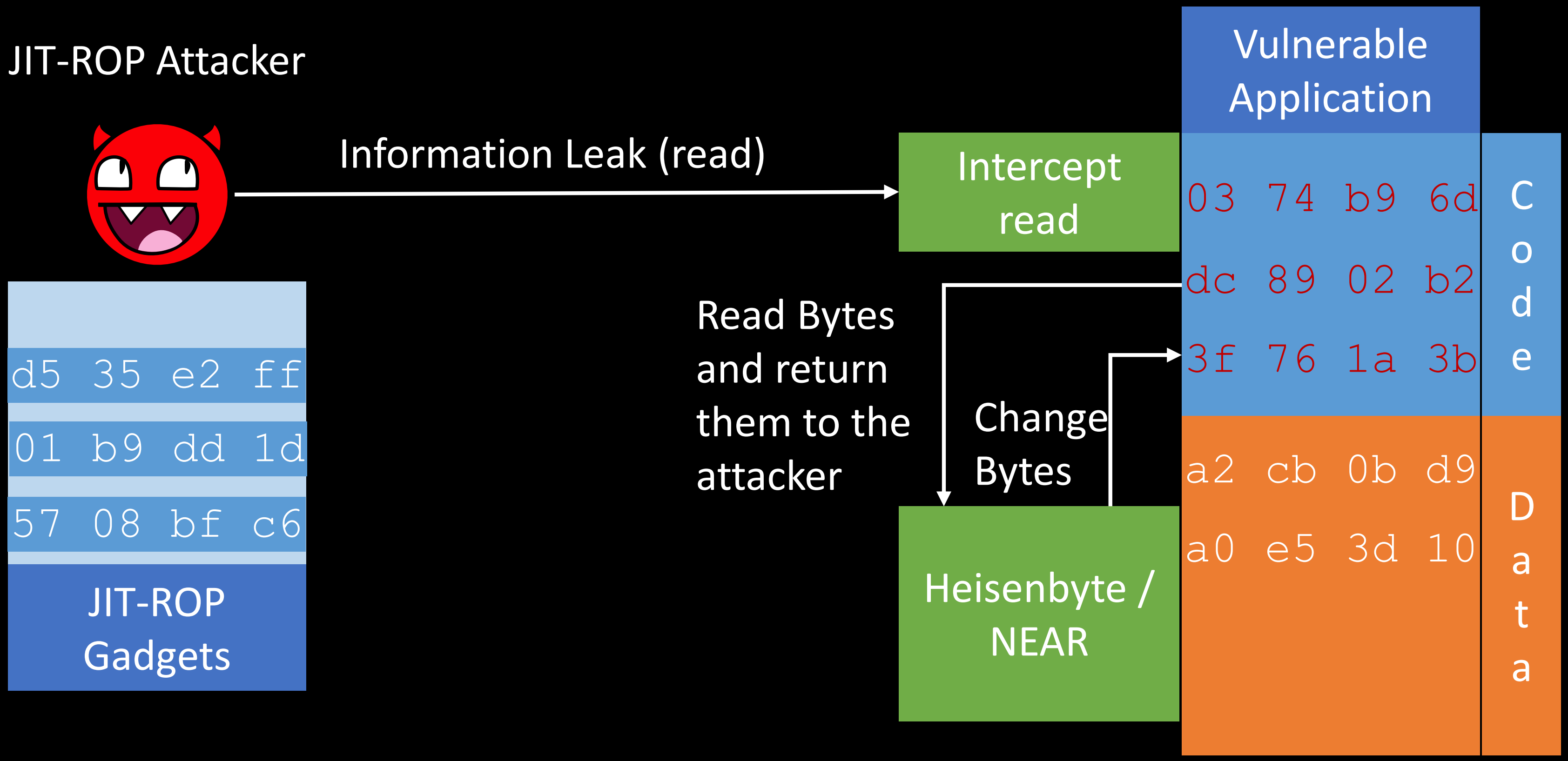
No Execute After Read (NEAR)
[Werner et al. ASIACCS'16]



Heisenbyte/NEAR

Heisenbyte
[Tang et al. CCS'15]

No Execute After Read (NEAR)
[Werner et al. ASIACCS'16]



Heisenbyte/NEAR

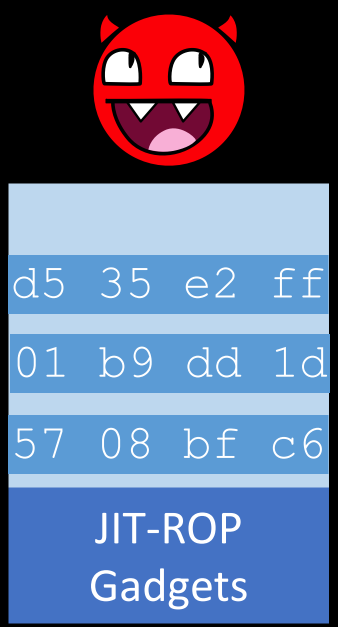
Heisenbyte

[Tang et al. CCS’15]

No Execute After Read (NEAR)

[Werner et al. ASIACCS’16]

JIT-ROP Attacker



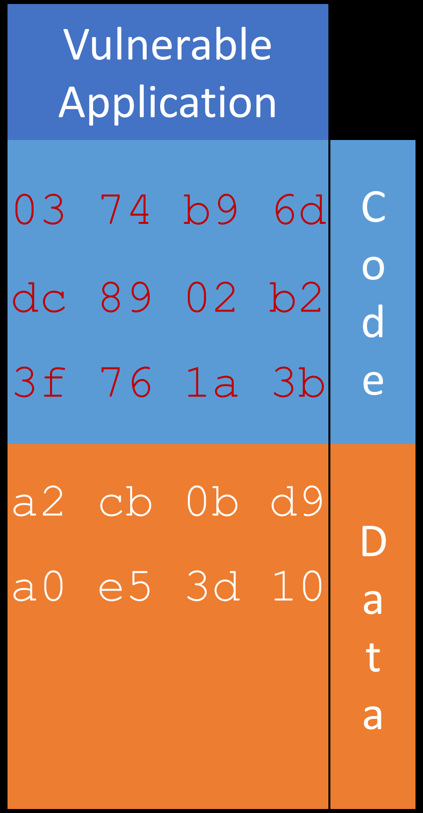
d5 35 e2 ff

01 b9 dd 1d

57 08 bf c6

JIT-ROP Gadgets

- Leaked memory is randomized
- Assembled attack payload will likely crash the application



Vulnerable Application

03 74 b9 6d

dc 89 02 b2

3f 76 1a 3b

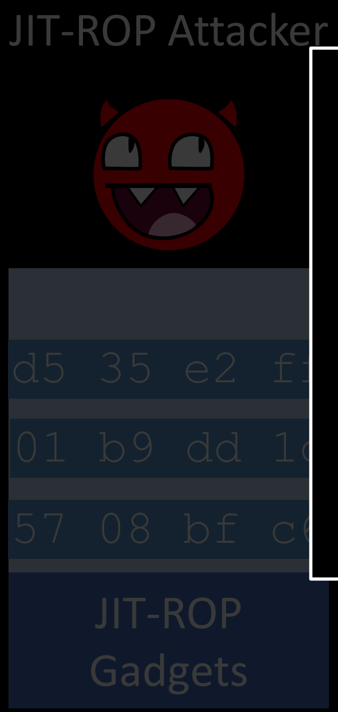
a2 cb 0b d9

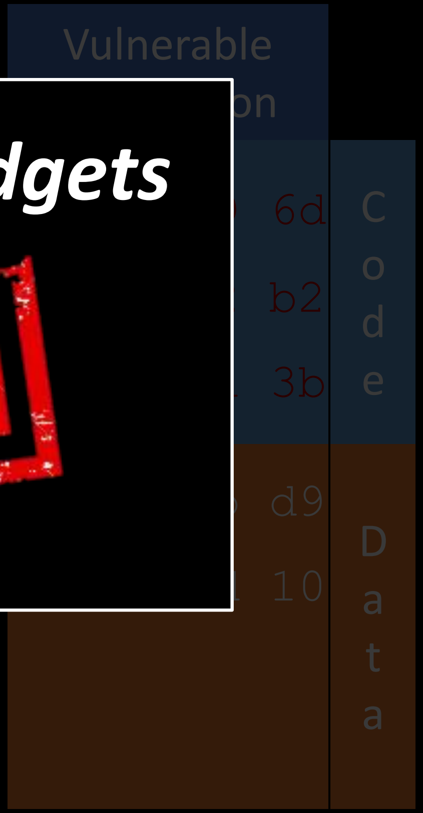
a0 e5 3d 10

Heisenbyte/NEAR

Heisenbyte
[Tang et al. CCS'15]

No Execute After Read (NEAR)
[Werner et al. ASIACCS'16]





Return to Zombie Gadgets
[Snow et al. IEEE S&P'16]

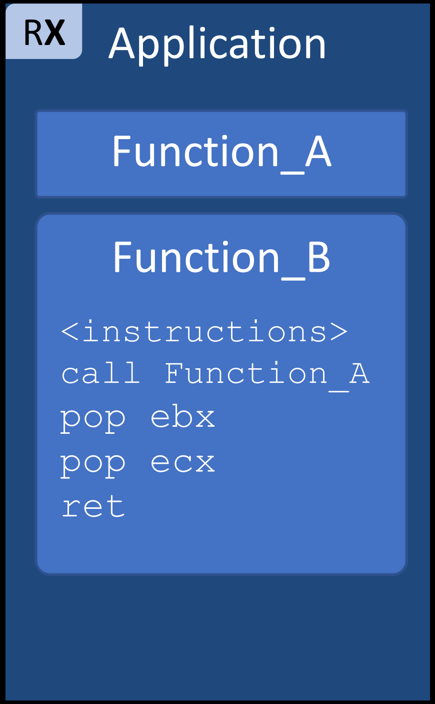
BROKEN

CYSEC
Cybersecurity
TU Darmstadt

Summary: Randomization Defense & Attacks

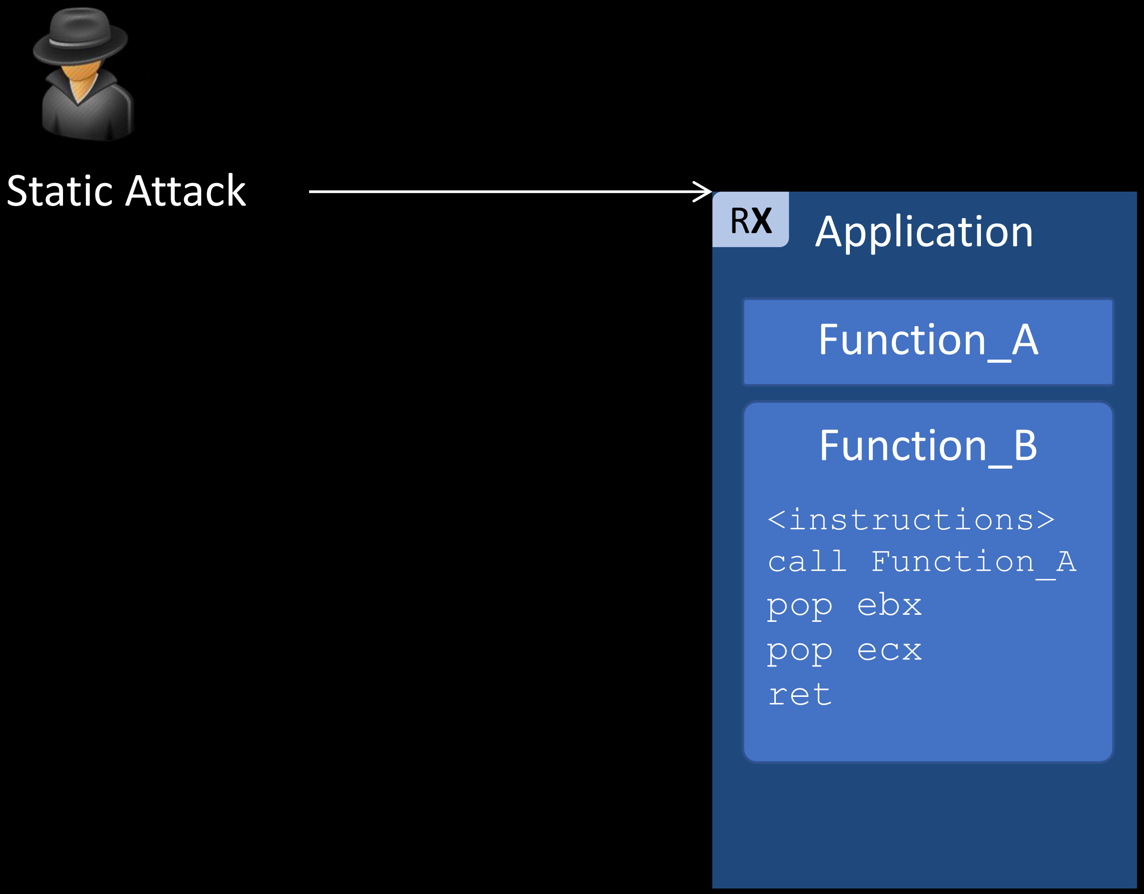
Code Randomization: Attacks & Defense Techniques

Attack Timeline



```
<instructions>
call Function_A
pop ebx
pop ecx
ret
```

Code Randomization: Attack & Defense Techniques

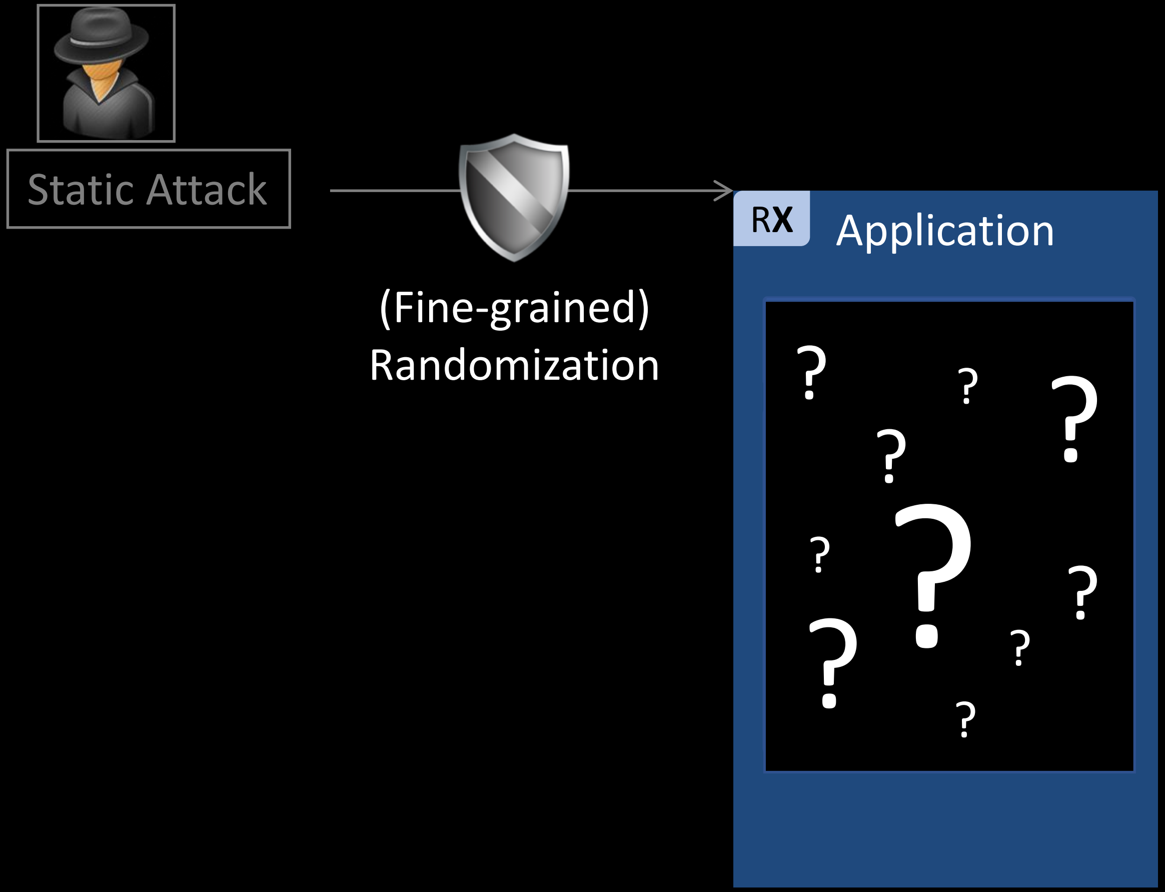


```
<instructions>
call Function_A
pop ebx
pop ecx
ret
```

Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]

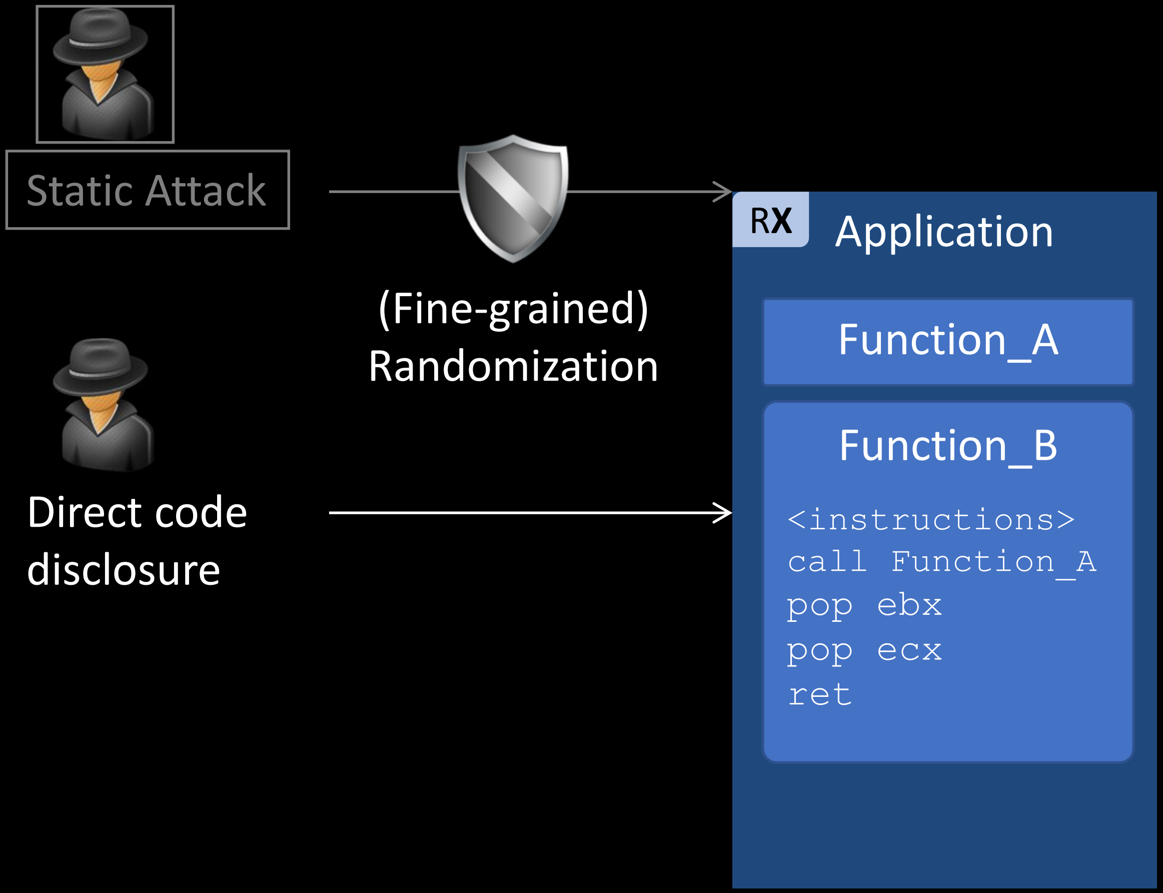
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]

Code Randomization: Attack & Defense Techniques

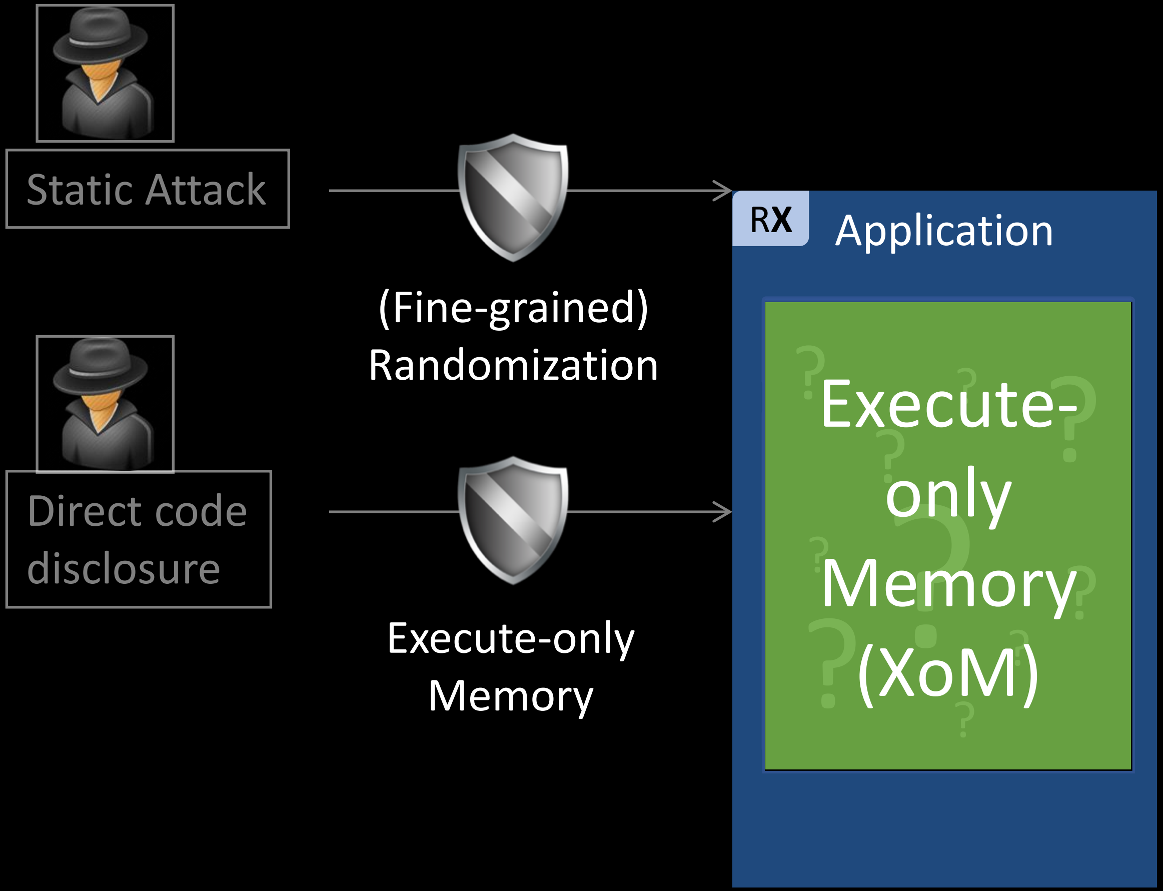


```
<instructions>
call Function_A
pop ebx
pop ecx
ret
```

Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq'97]
- Just-In-Time ROP [Snow et al. IEEE S&P'13]

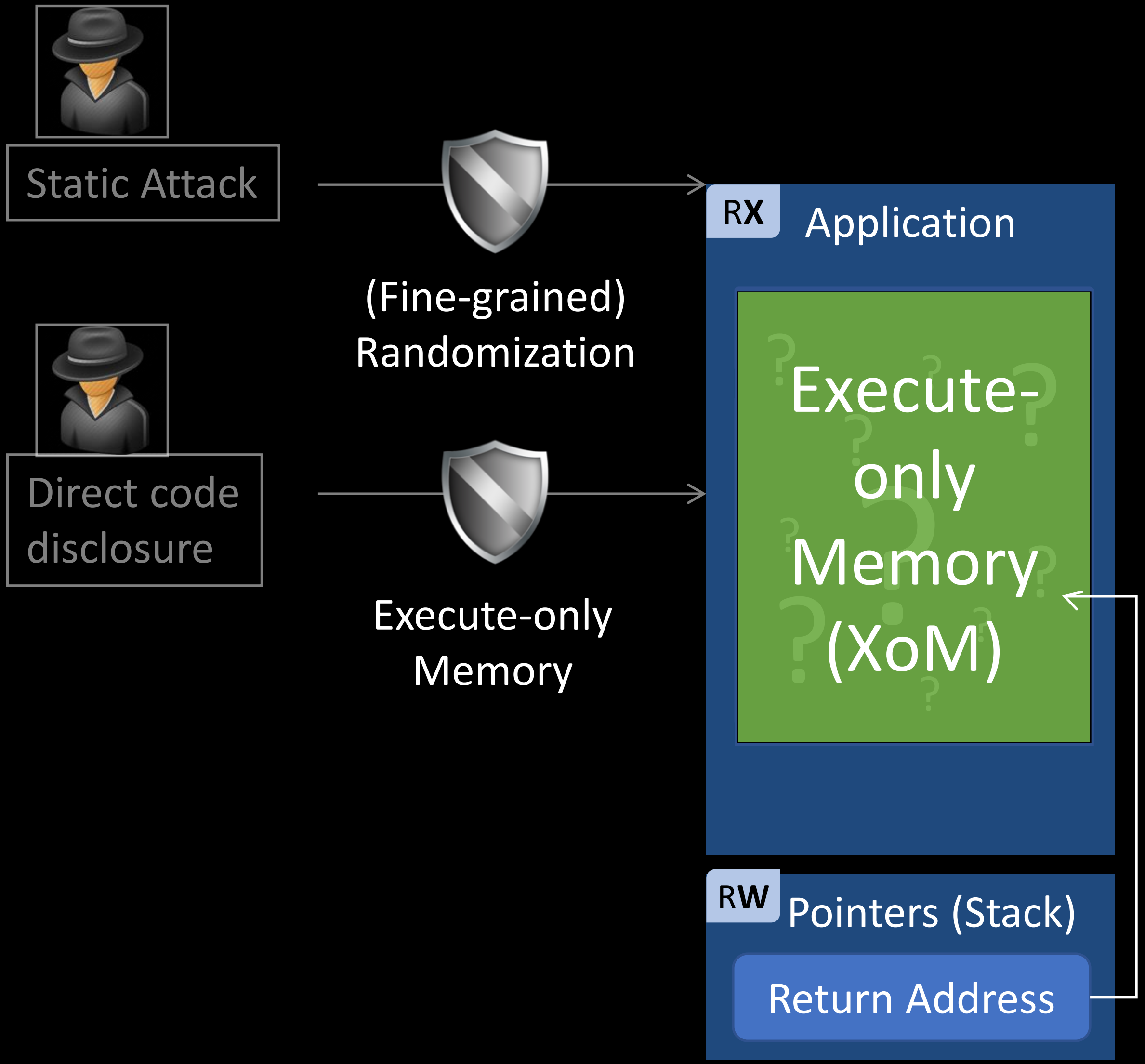
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]

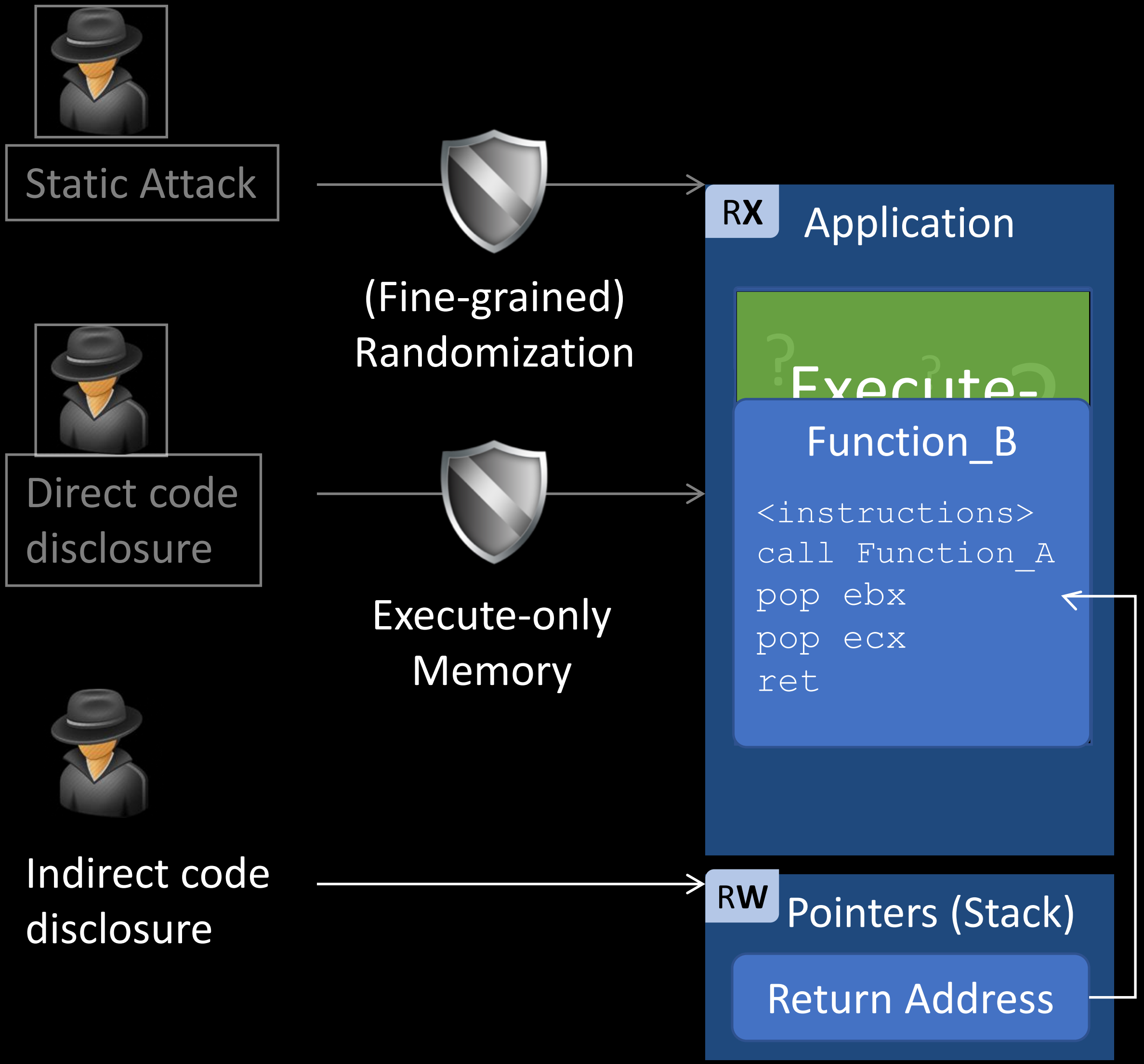
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]

Code Randomization: Attack & Defense Techniques

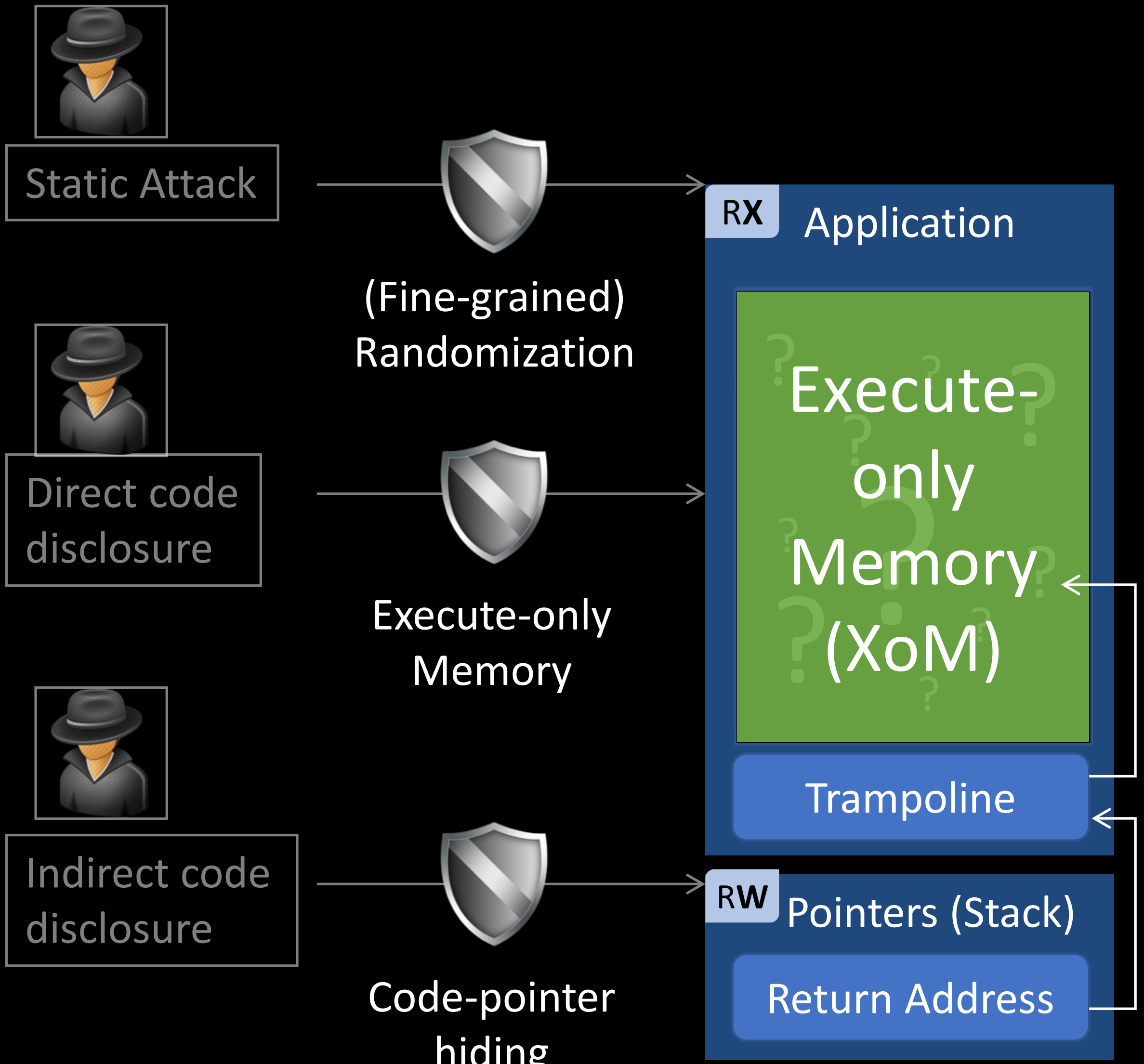


```
<instructions>
call Function_A
pop ebx
pop ecx
ret
```

Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

Code Randomization: Attack & Defense Techniques



Static Attack

(Fine-grained) Randomization

Direct code disclosure

Execute-only Memory

Indirect code disclosure

Code-pointer hiding

RX Application

Execute-only Memory (XoM)

Trampoline

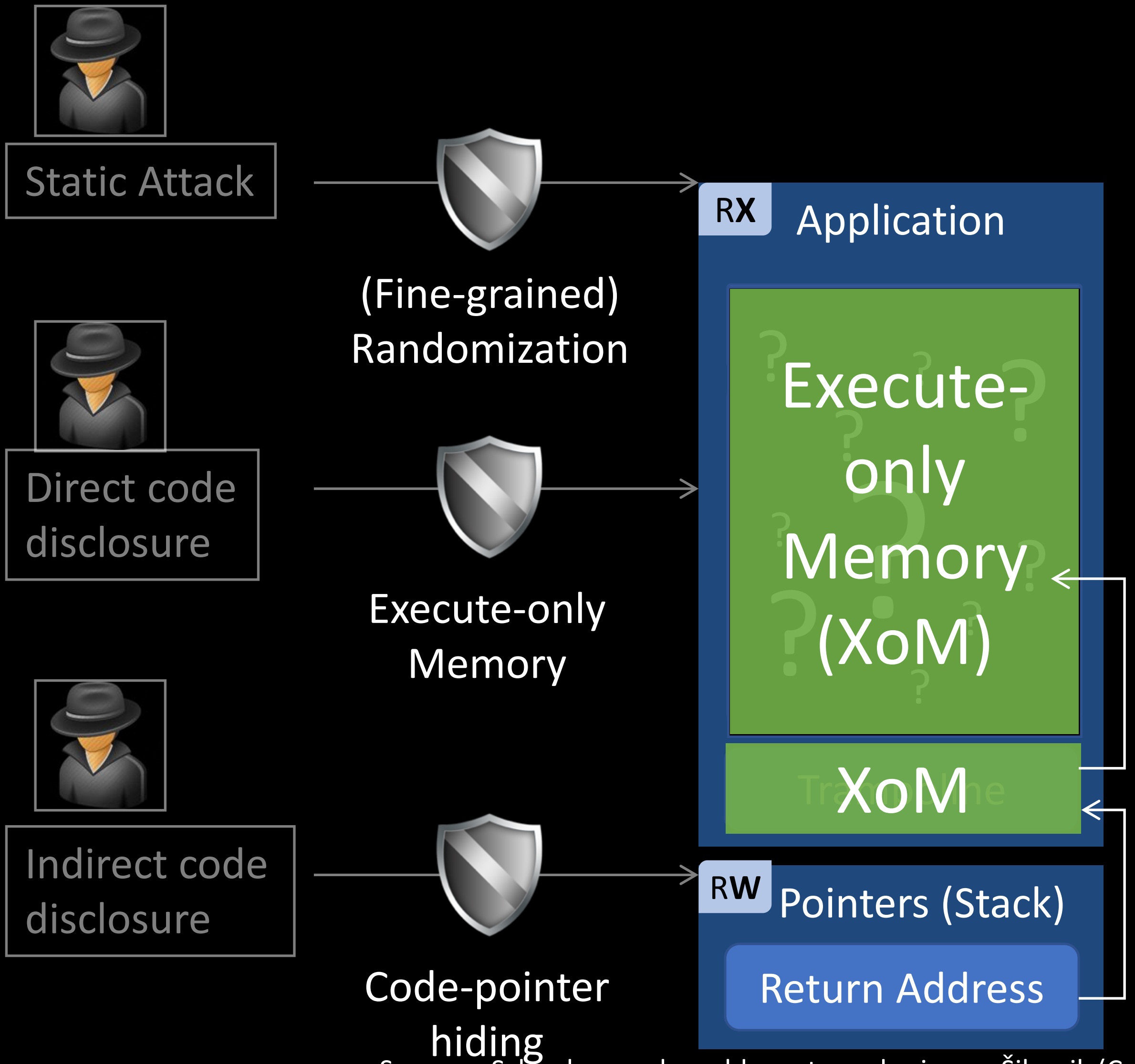
RW Pointers (Stack)

Return Address

Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

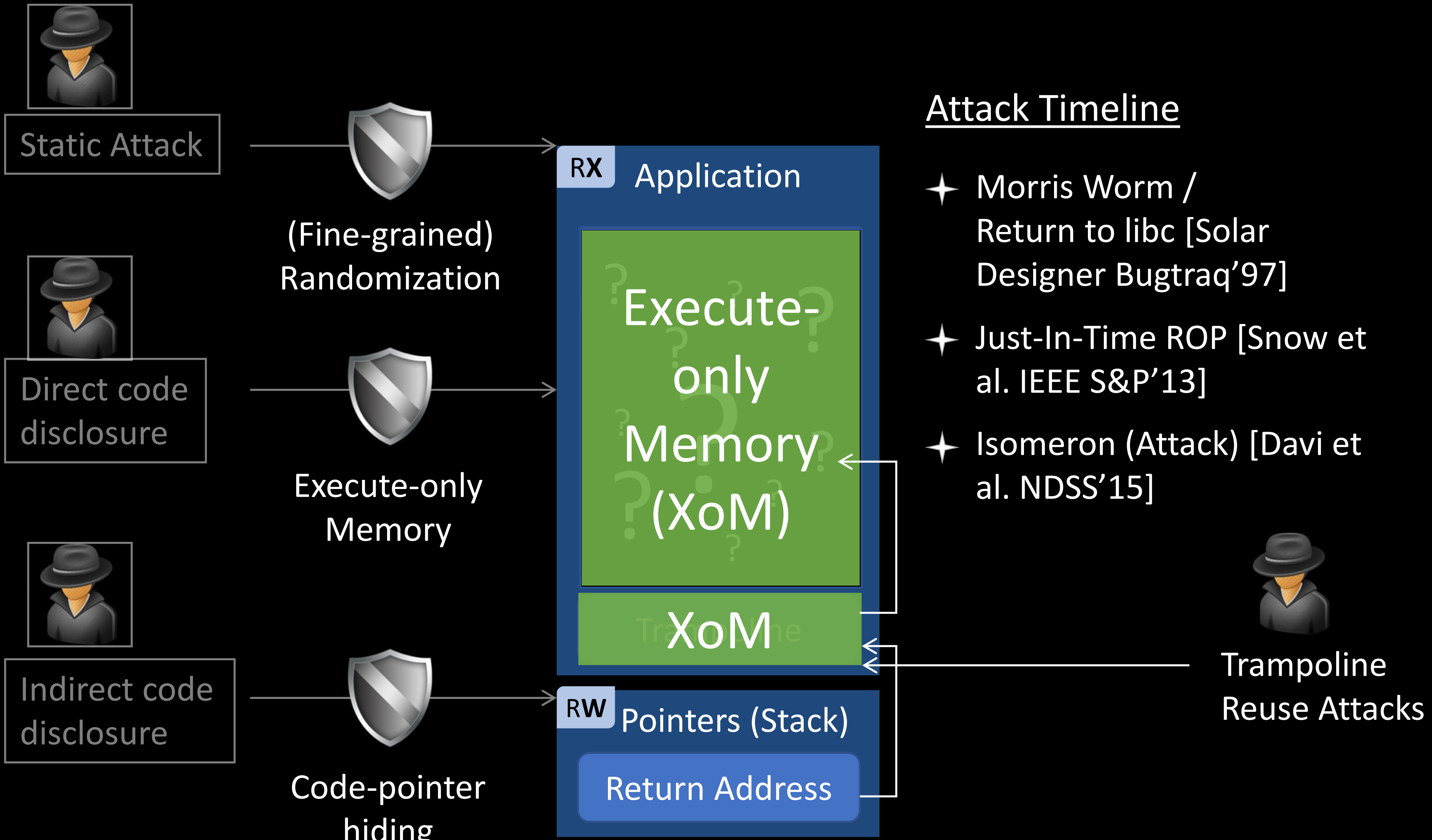
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

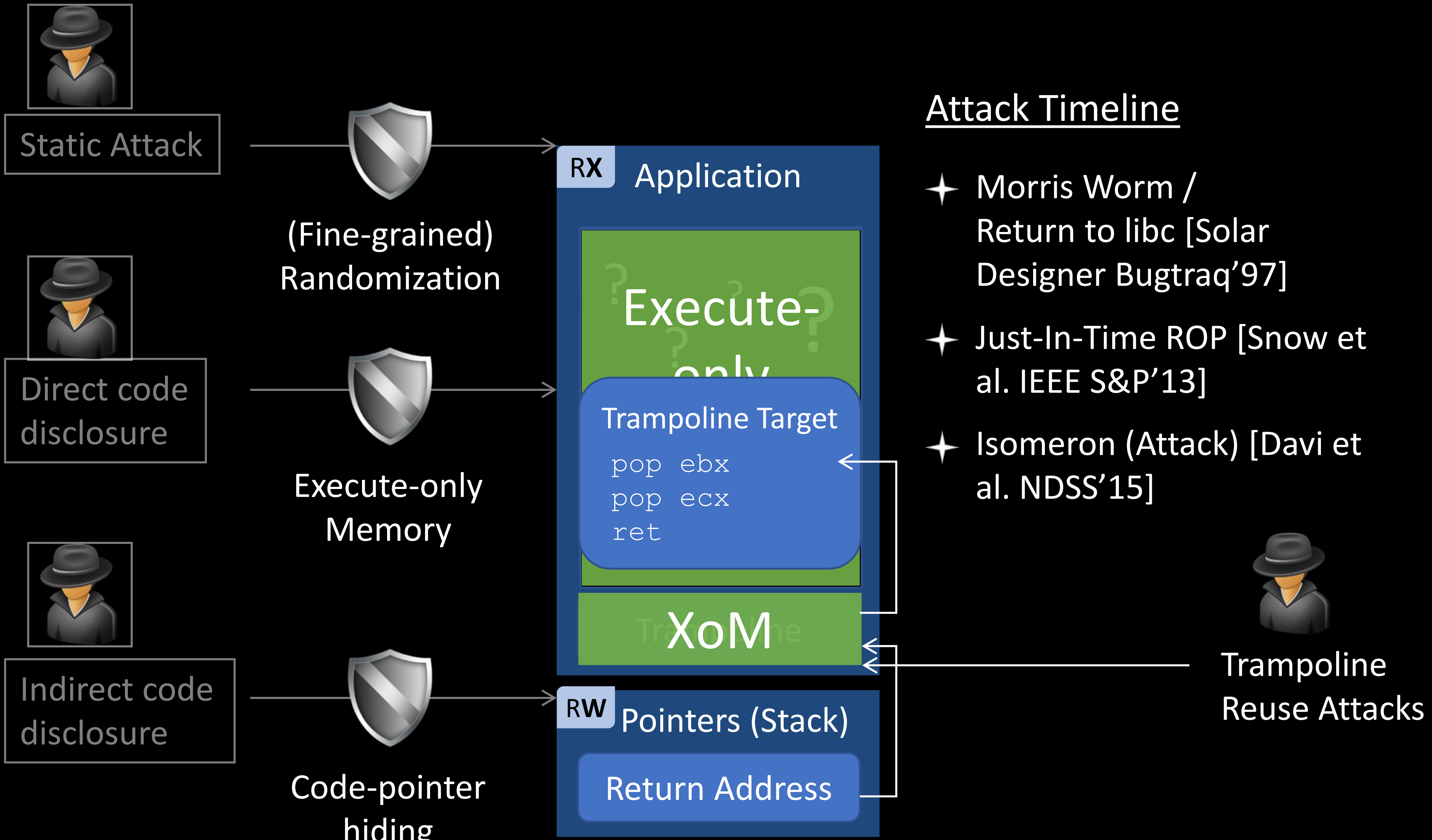
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

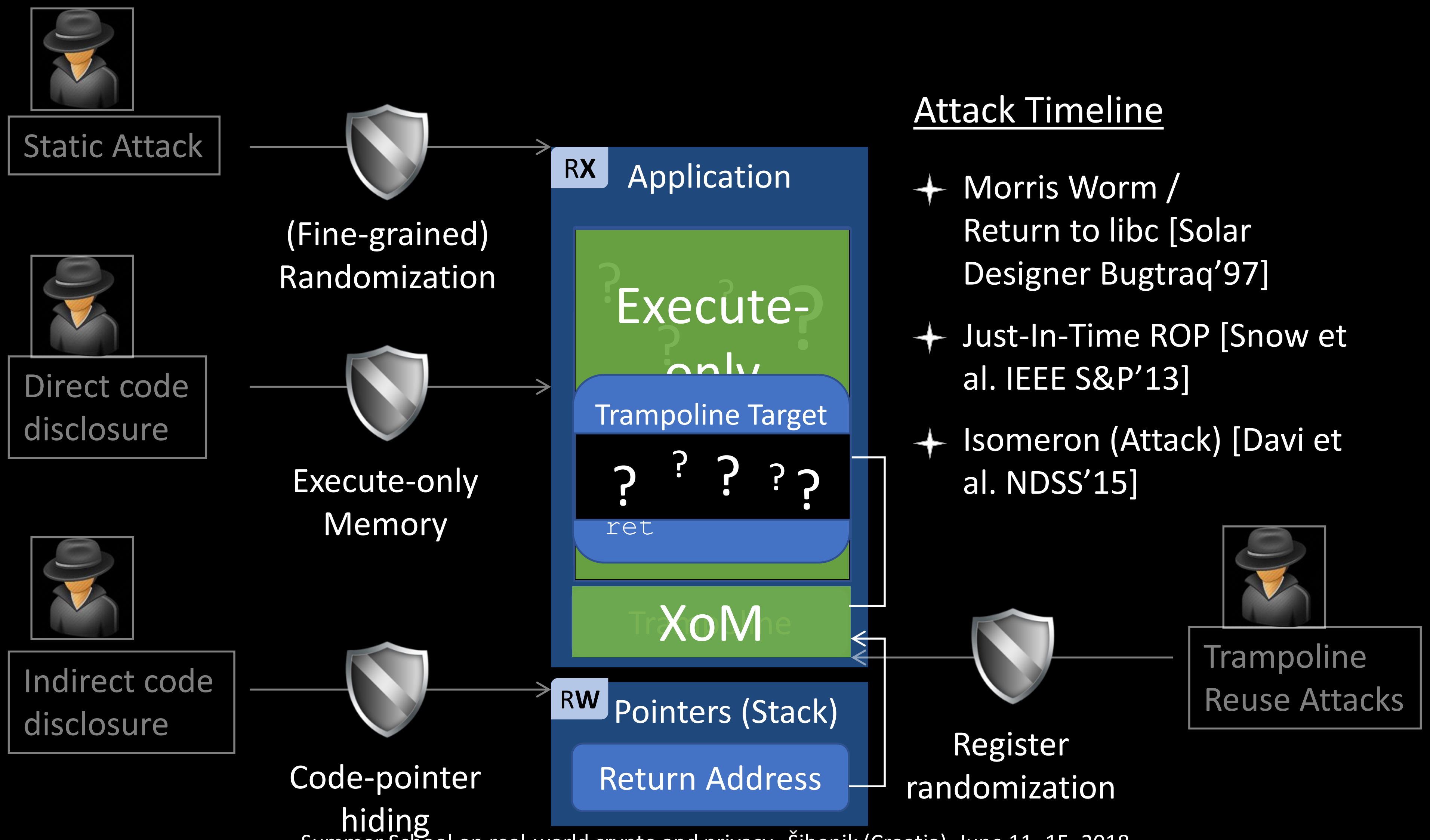
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

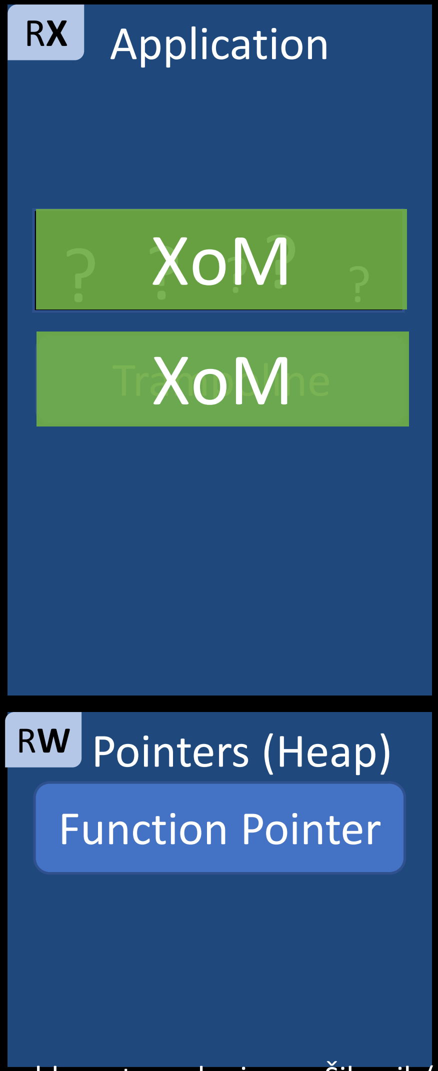
Code Randomization: Attack & Defense Techniques



Attack Timeline

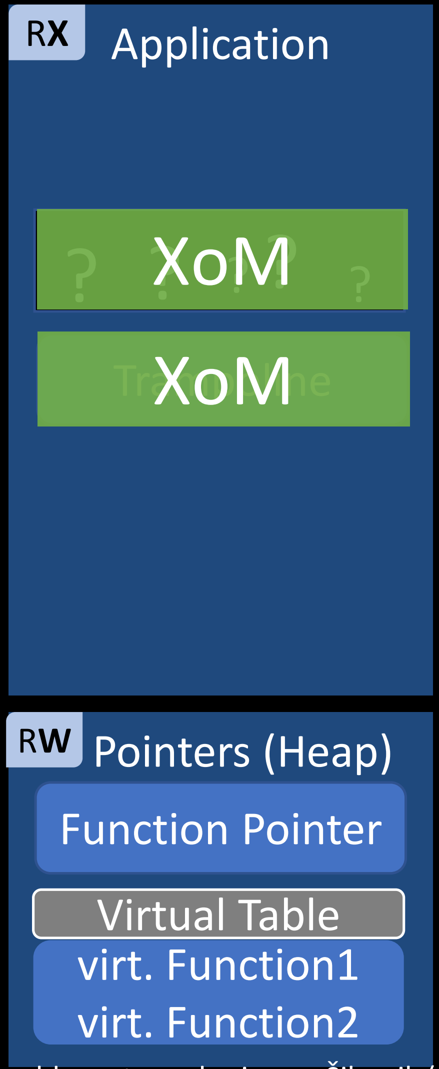
- Morris Worm / Return to libc [Solar Designer Bugtraq’97]
- Just-In-Time ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

Code Randomization: Attack & Defense Techniques



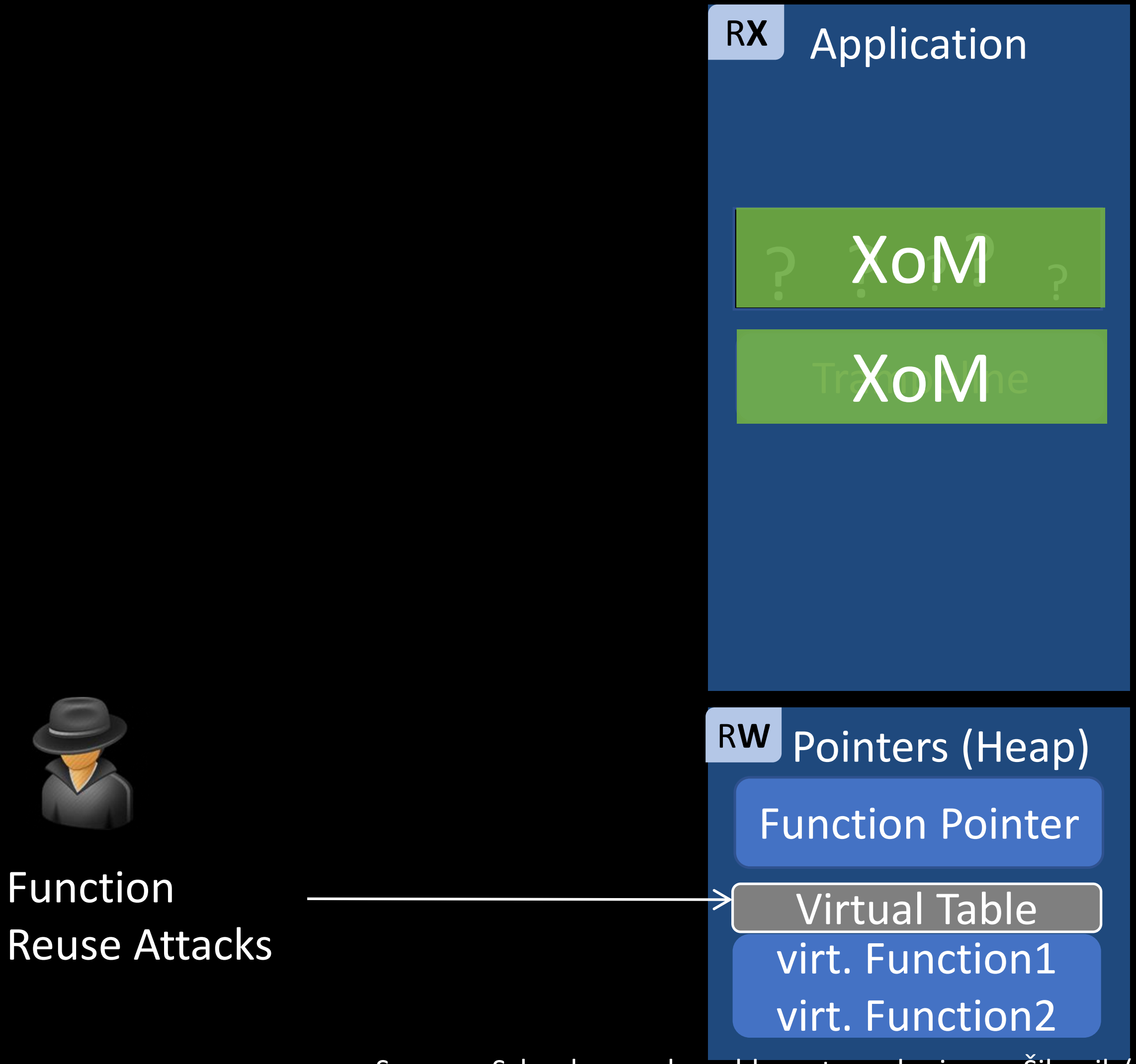
Attack Timeline

Code Randomization: Attack & Defense Techniques



<u>Attack Timeline</u>

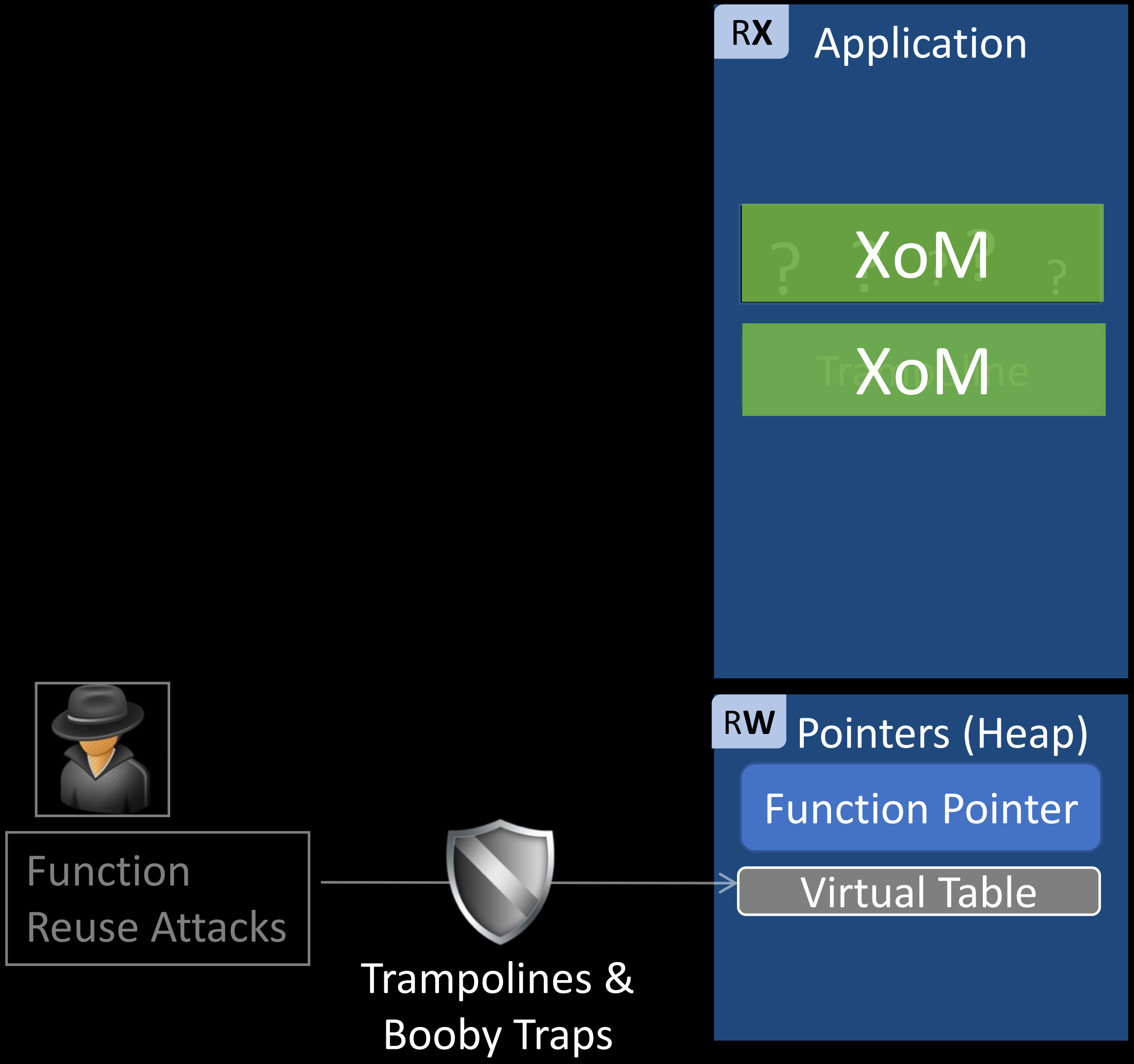
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P’15]

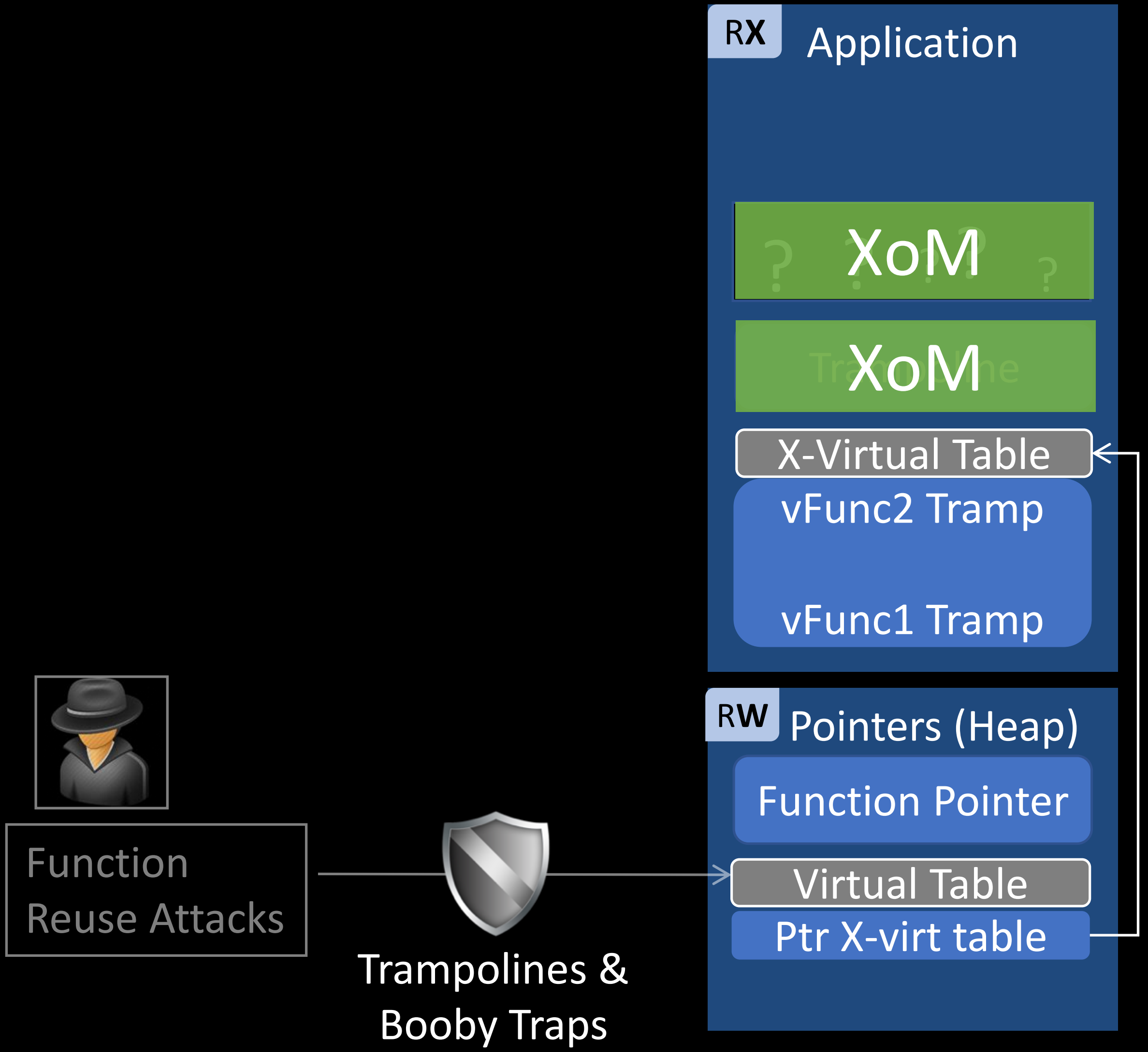
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]

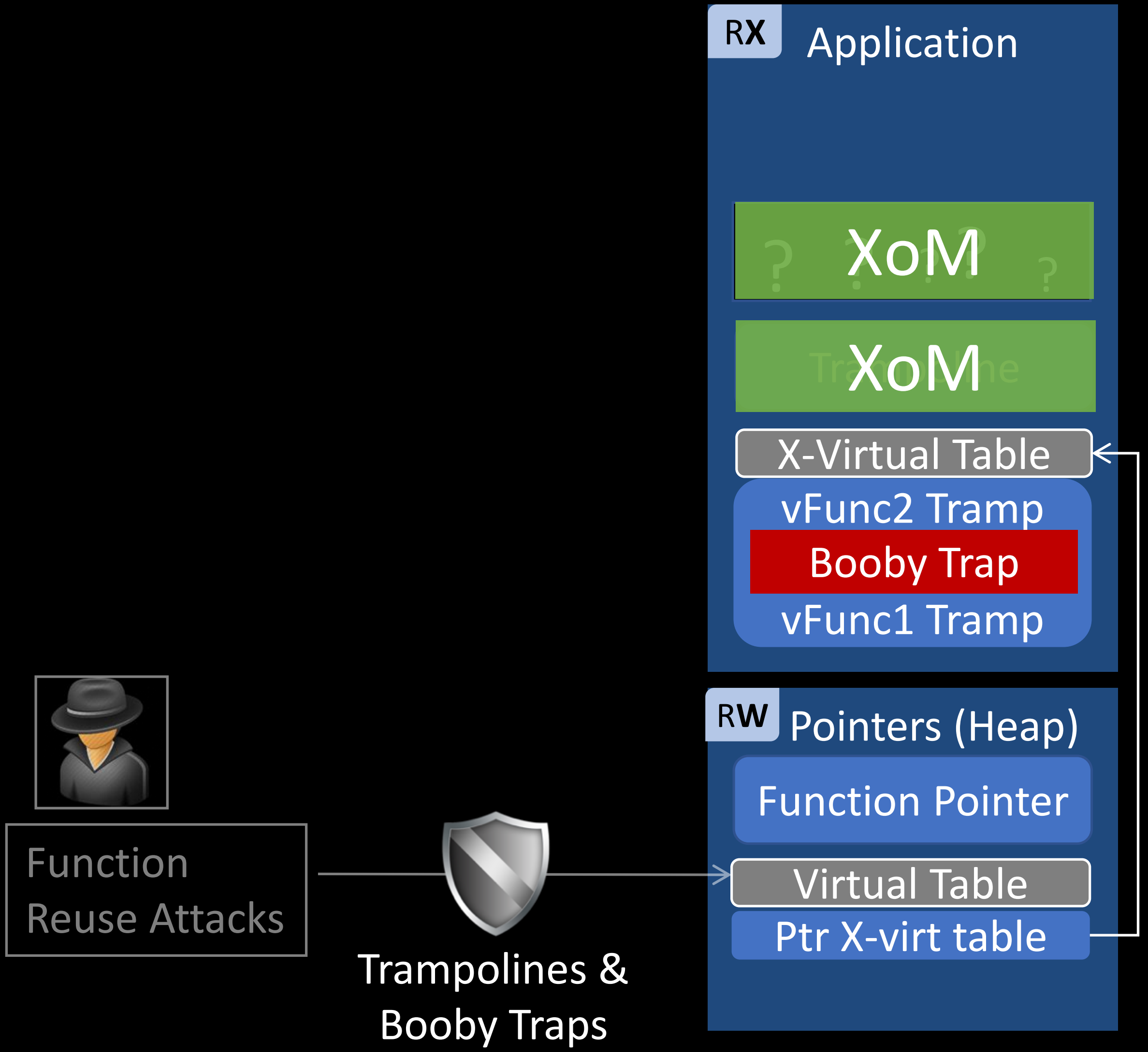
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]

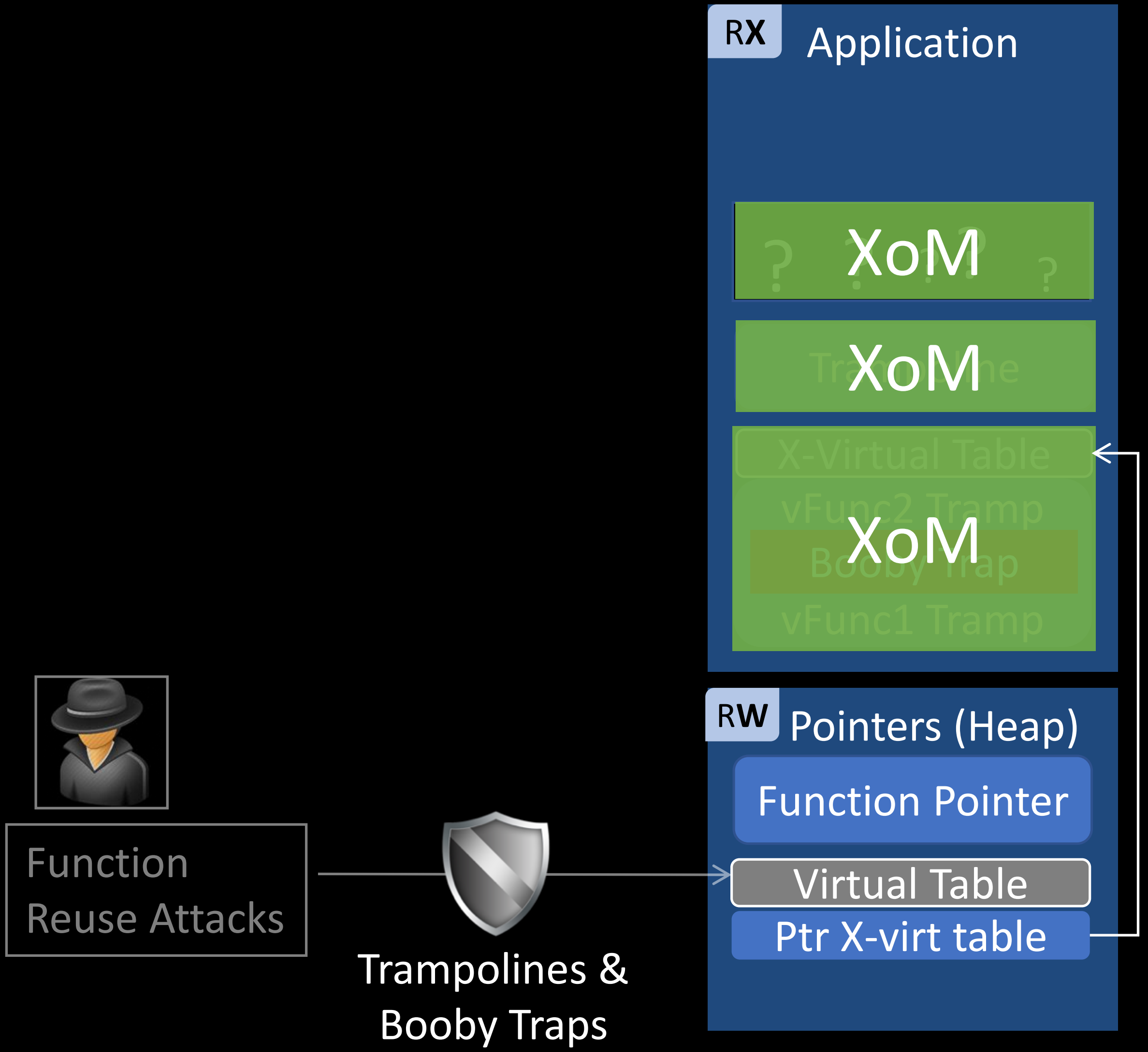
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]

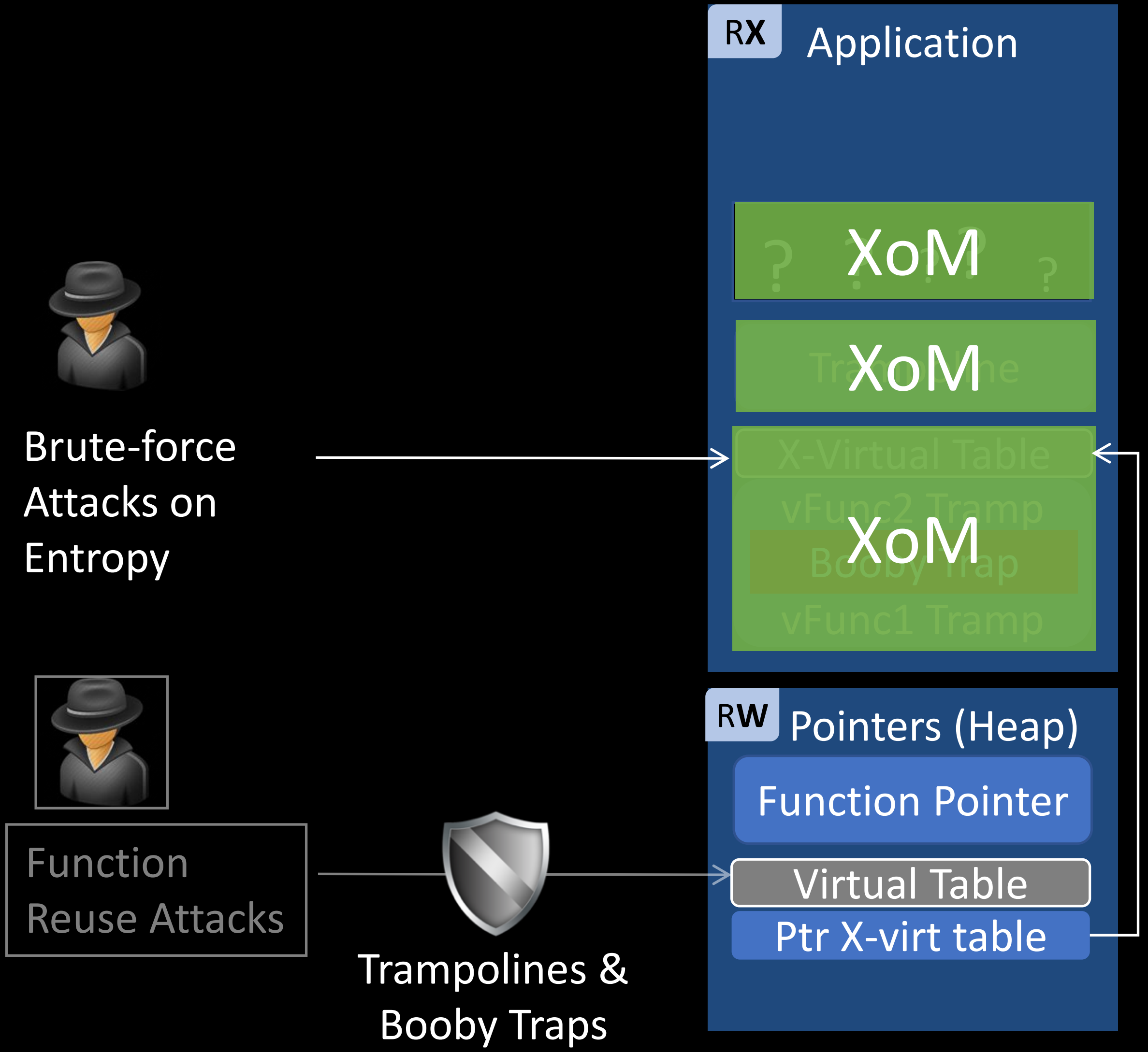
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]

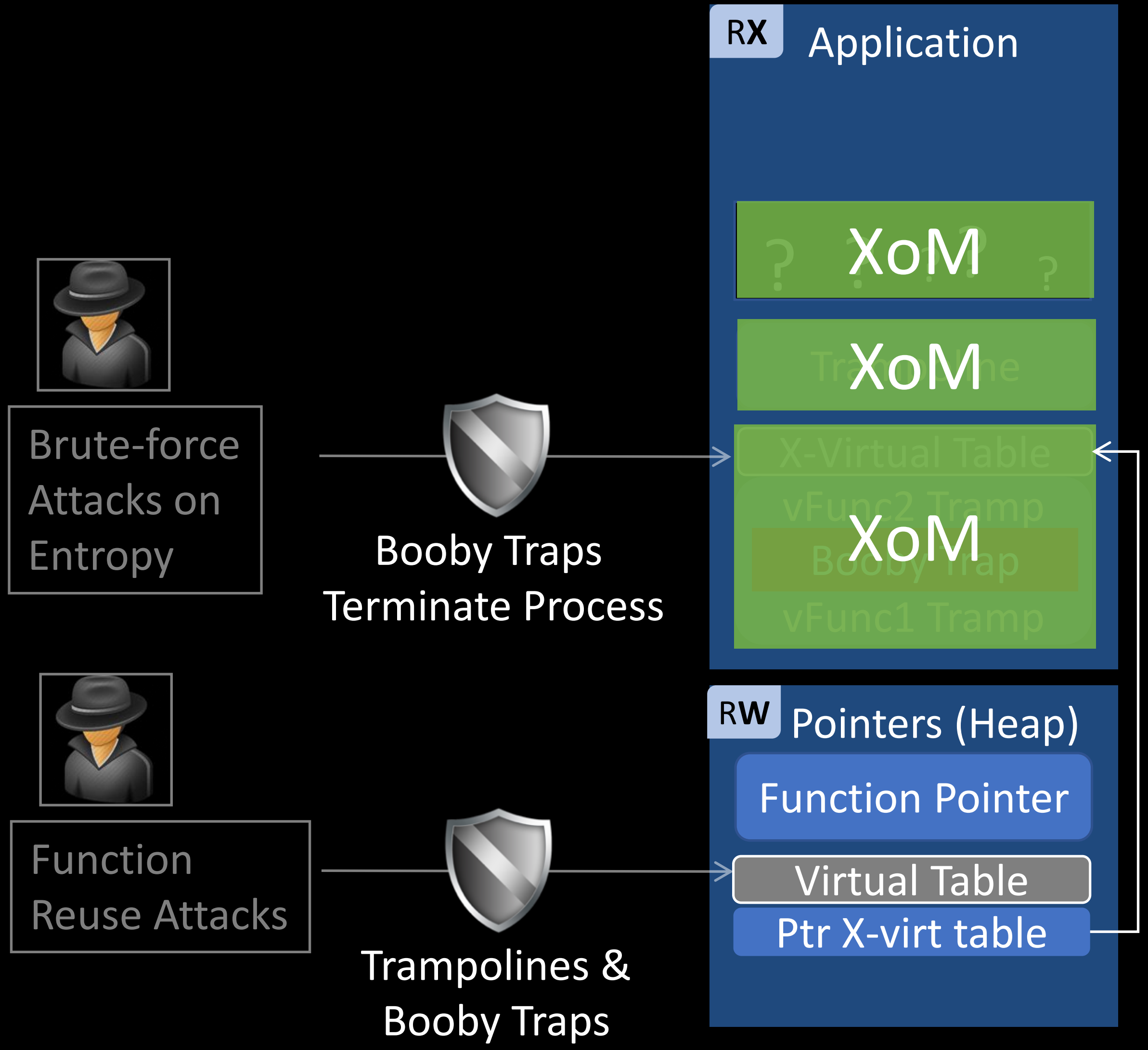
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS'16]

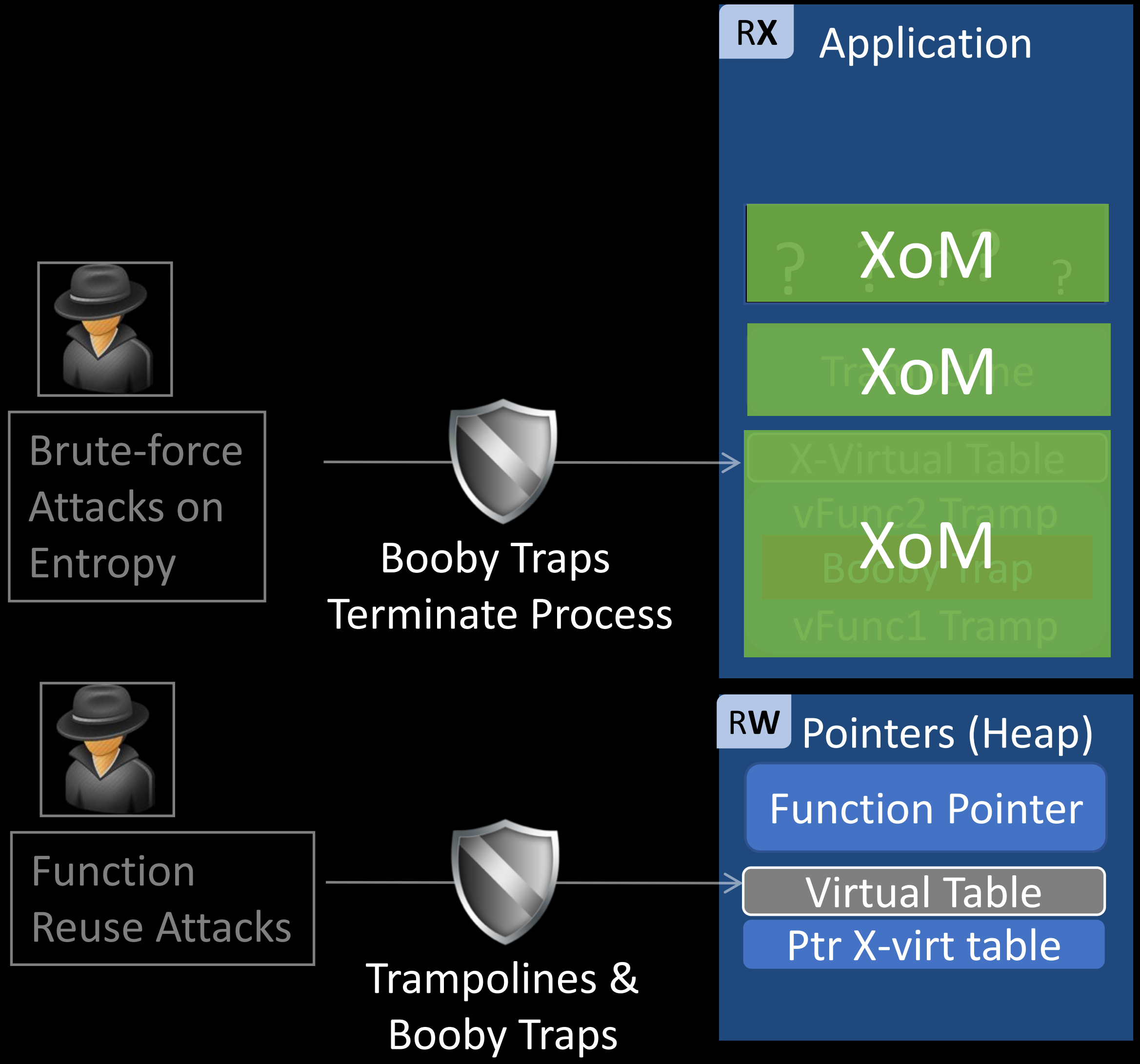
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS'16]

Code Randomization: Attack & Defense Techniques



Brute-force Attacks on Entropy

Booby Traps Terminate Process

Function Reuse Attacks

Trampolines & Booby Traps

RX Application

XoM

XoM

X-Virtual Table

vFunc2 Tramp

XoM

Booby Trap

vFunc1 Tramp

RW Pointers (Heap)

Function Pointer

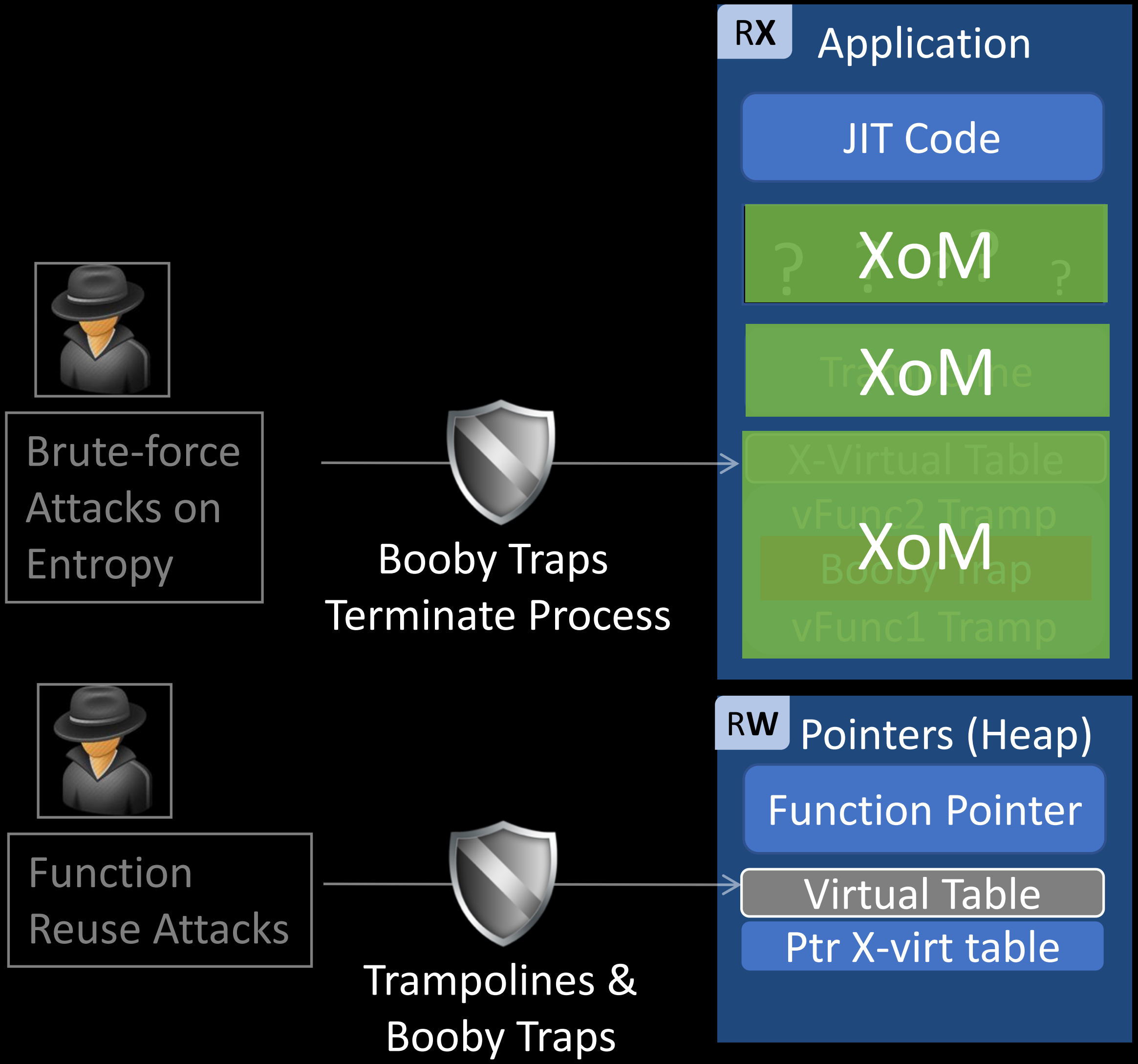
Virtual Table

Ptr X-virt table

Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P’15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS’16]

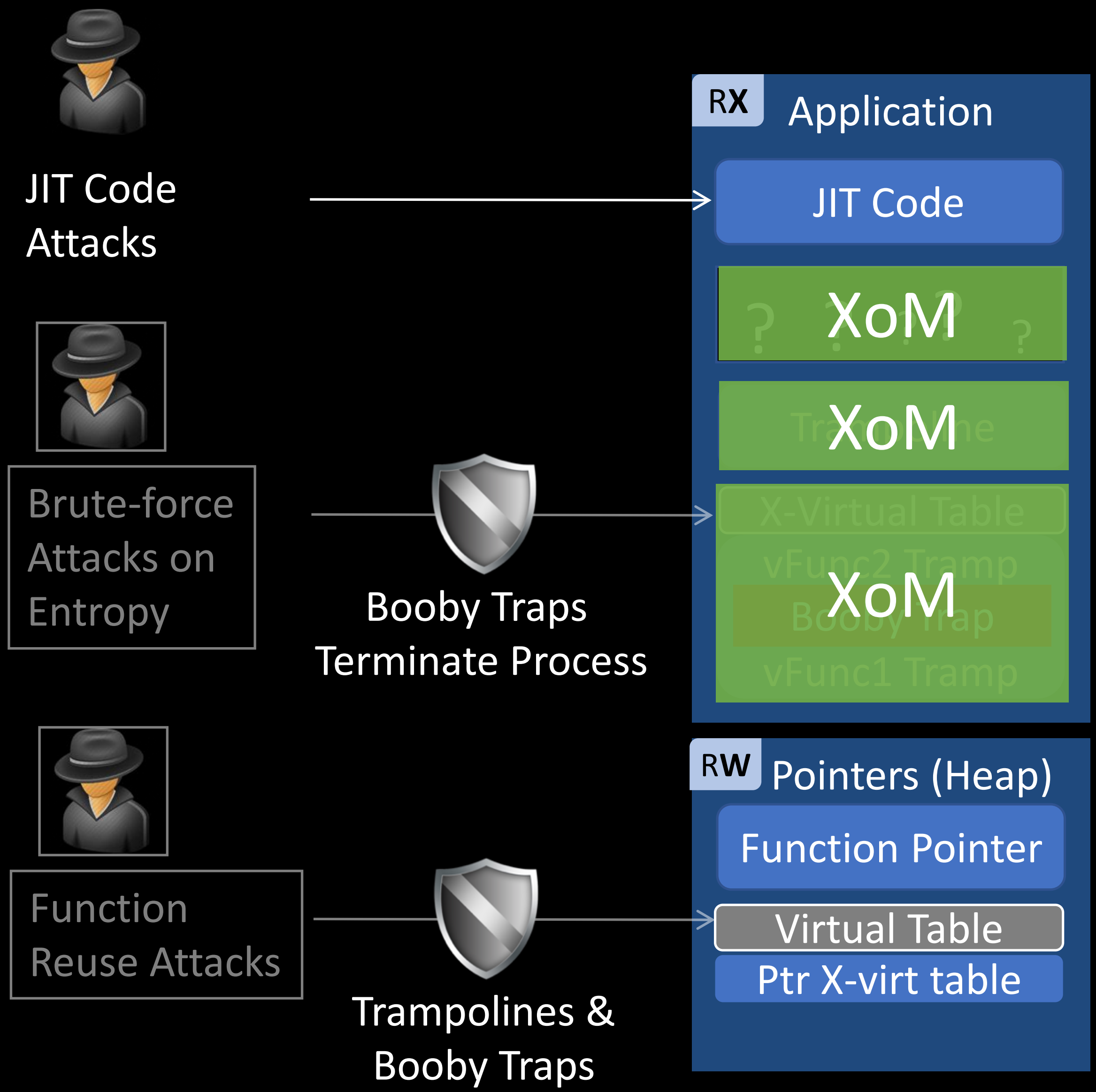
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P’15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS’16]

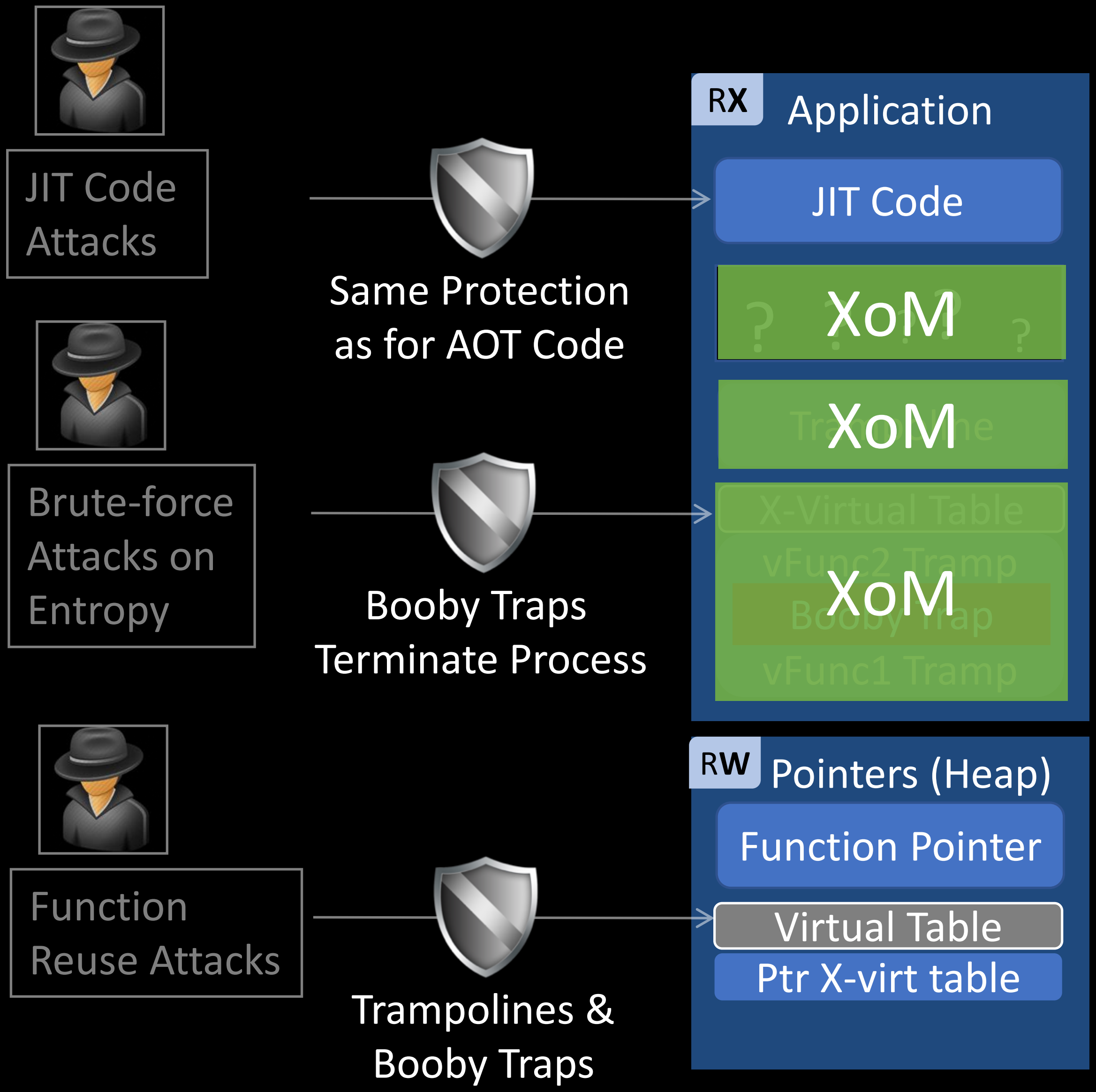
Code Randomization: Attack & Defense Techniques



<u>Attack Timeline</u>

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P’15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS’16]

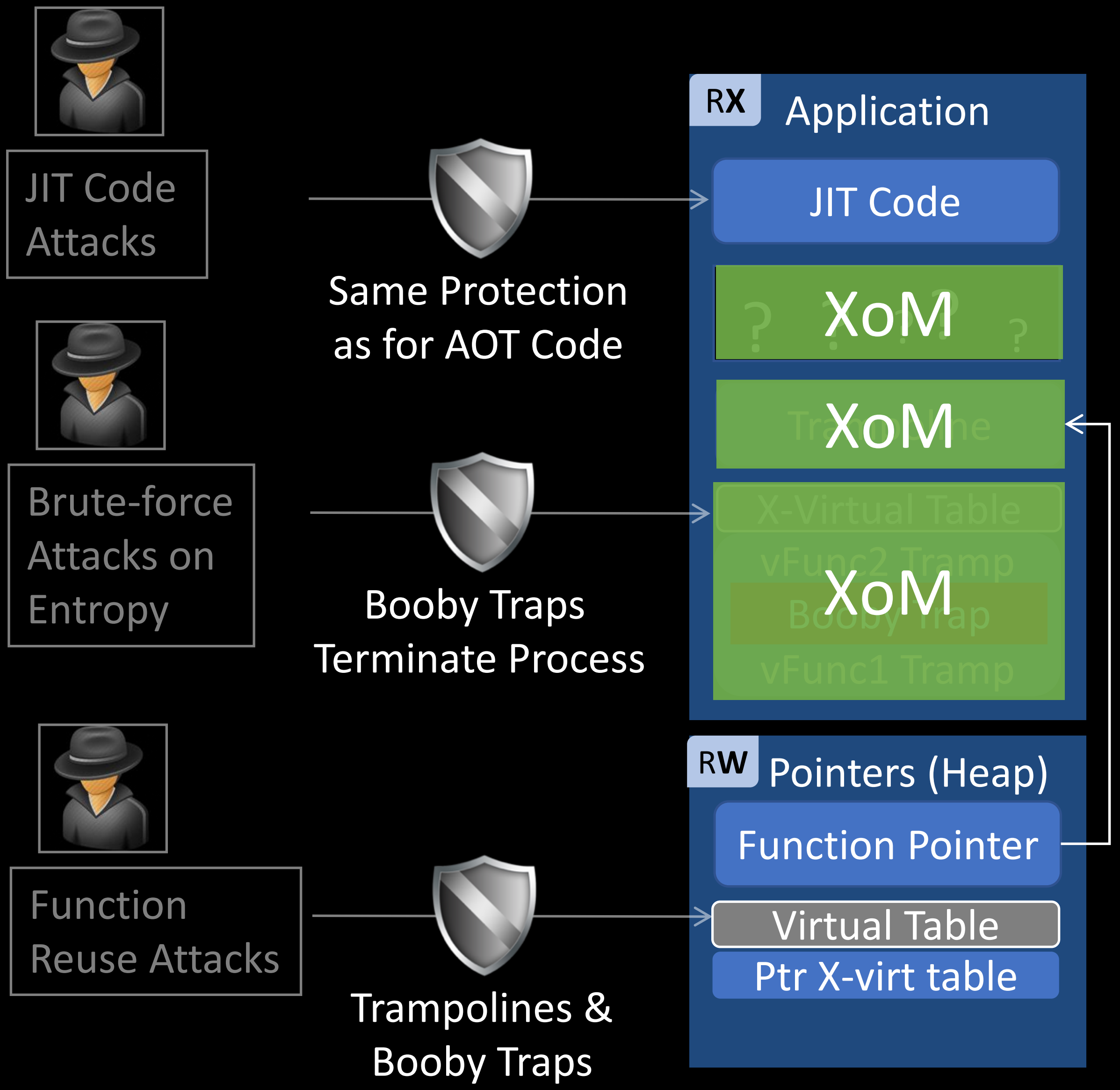
Code Randomization: Attack & Defense Techniques



<u>Attack Timeline</u>

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P’15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS’16]

Code Randomization: Attack & Defense Techniques



JIT Code Attacks

Same Protection as for AOT Code

Brute-force Attacks on Entropy

Booby Traps Terminate Process

Function Reuse Attacks

Trampolines & Booby Traps

RX Application

JIT Code

XoM

? ? ? ?

XoM

Trampoline

X-Virtual Table

vFunc2 Tramp

XoM

Booby Trap

vFunc1 Tramp

RW Pointers (Heap)

Function Pointer

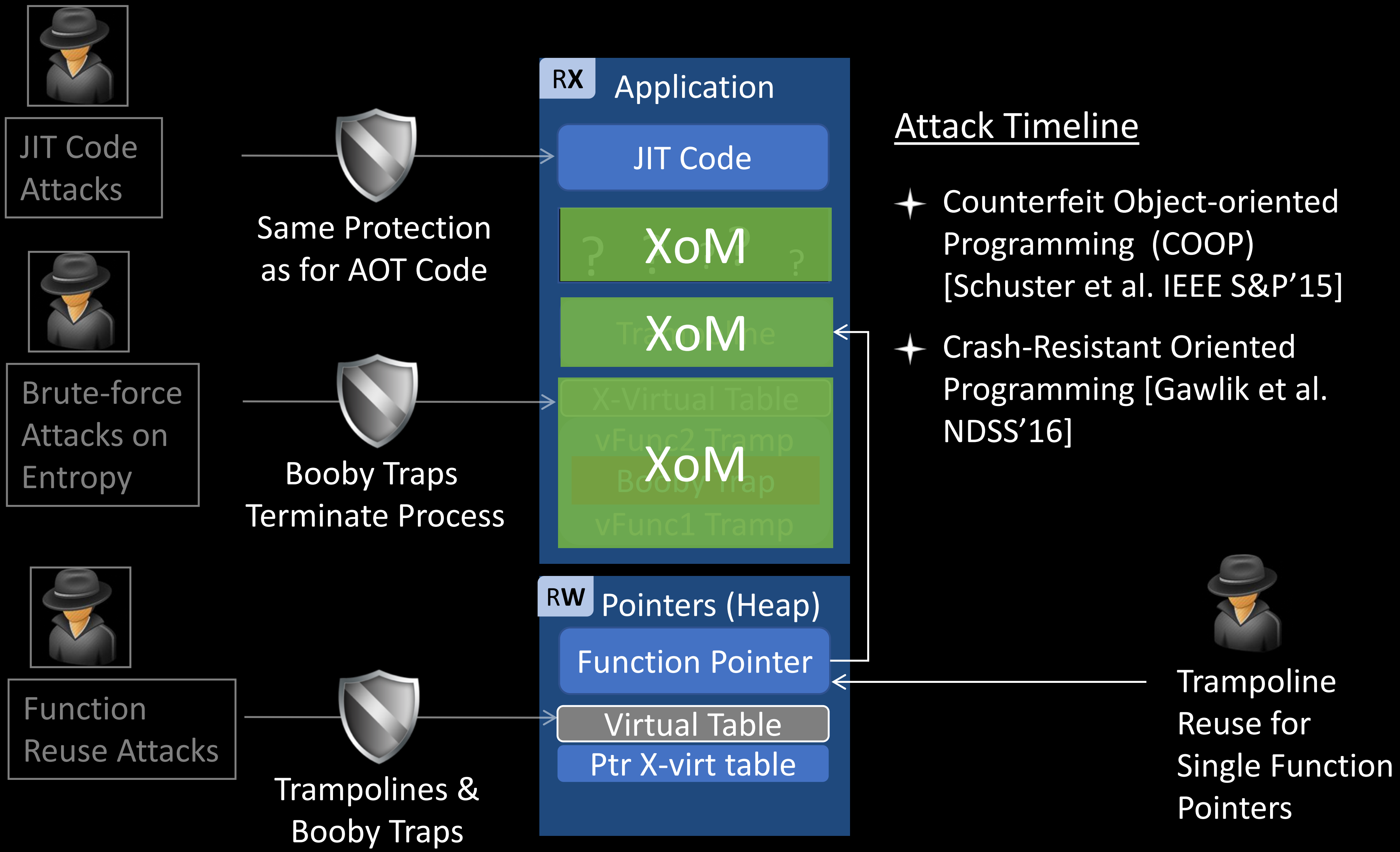
Virtual Table

Ptr X-virt table

<u>Attack Timeline</u>

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS'16]

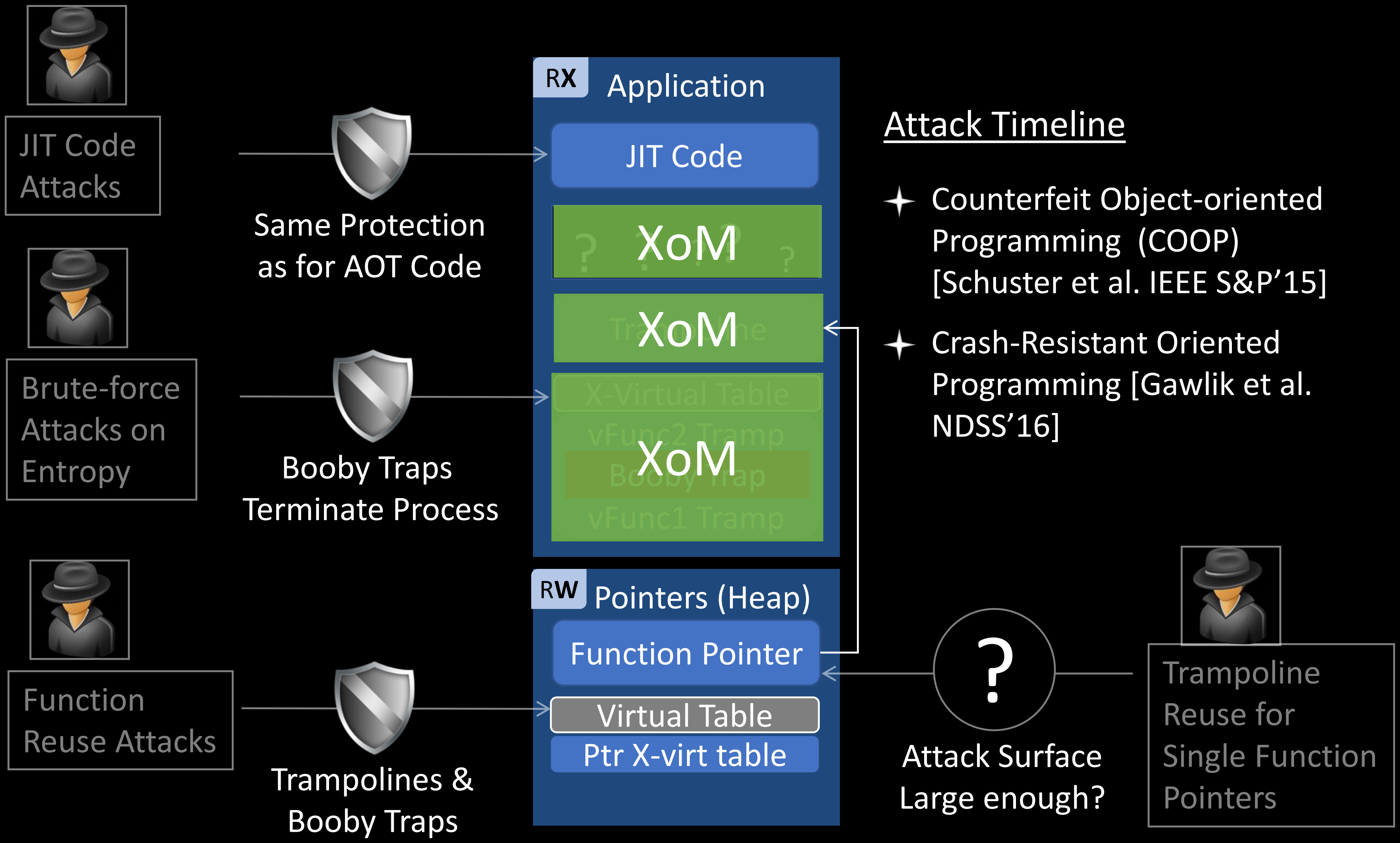
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS'16]

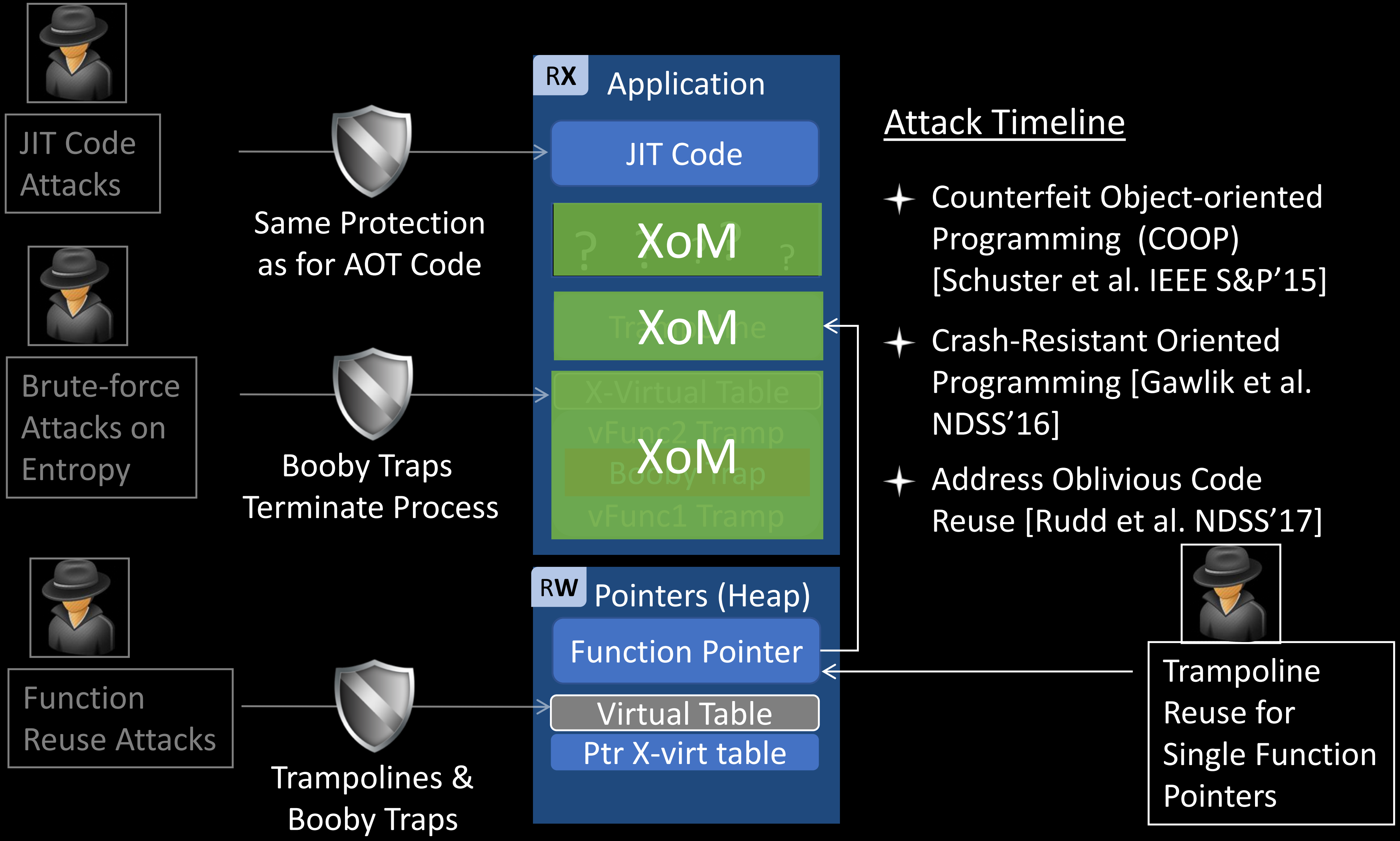
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS'16]

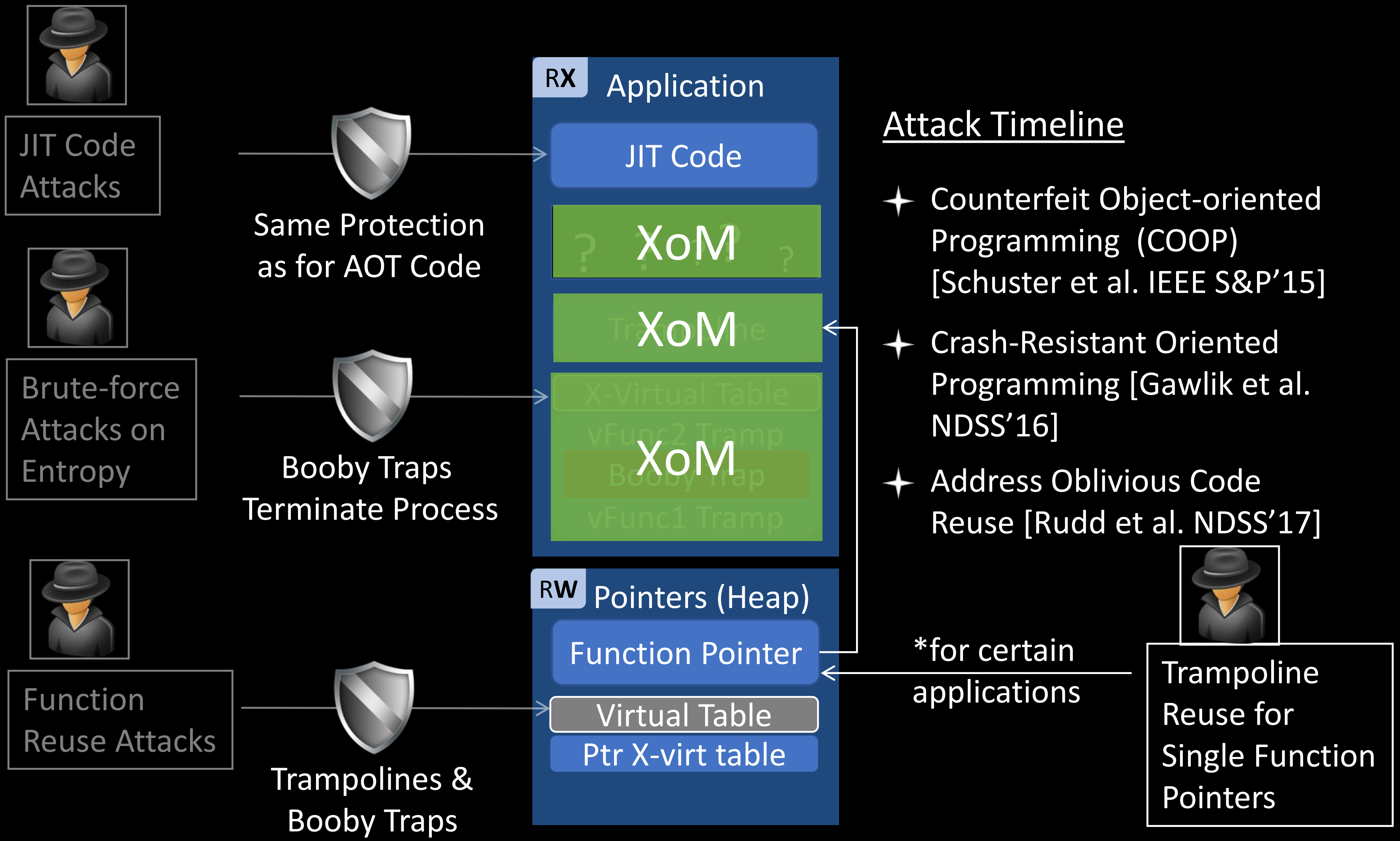
Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS'16]
- Address Oblivious Code Reuse [Rudd et al. NDSS'17]

Code Randomization: Attack & Defense Techniques



Attack Timeline

- Counterfeit Object-oriented Programming (COOP) [Schuster et al. IEEE S&P'15]
- Crash-Resistant Oriented Programming [Gawlik et al. NDSS'16]
- Address Oblivious Code Reuse [Rudd et al. NDSS'17]

Lessons Learned

- There are a lot sources for information leaks
- … but I think we got them all
- … are we good now?
- … well …

Lessons Learned

- Th matic
- … but I think we got them all
- … are we good now?
- … well …

Breaking kernel address space layout randomization with intel TSX
[Jang et al., CCS 2016]

Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR
[Evtyushkin et al., MICRO 2016]

Practical Timing Side Channel Attacks Against Kernel Space
[Hund et al., S&P 2013]

ASLR on the Line: Practical Cache Attacks on the MMU
[Gras et al., NDSS 2017]

Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR
[Gruss et al., CCS 2016]

Lessons Learned

- Th
- … but
- … are v
- … well

Brea

ch

AS

R

[Gras et al., NDSS 2017]

[Gruss et al., CCS 2016]

New Hot Topic:

Side-Channel Attacks against Randomization

How to Tackle the Problem of Information Disclosure?

Readactor: Towards Resilience to Memory Disclosure

Readactor:
Practical Code Randomization Resilient to Memory Disclosure
IEEE Security and Privacy 2015

Stephen Crane, Christopher Liebchen, Andrei Homescu, Lucas Davi, Per Larsen,
Ahmad-Reza Sadeghi, Stefan Brunthaler, Michael Franz

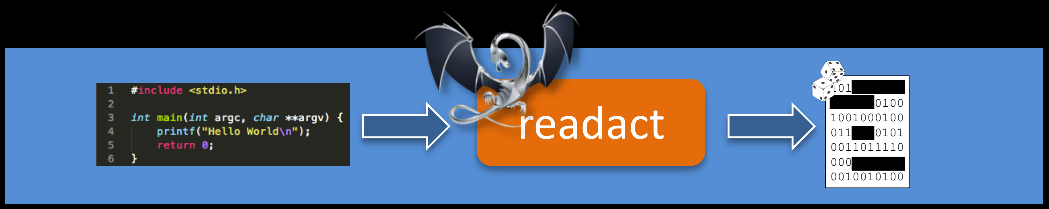
Objectives

secure	prevent code reuse + memory disclosure
comprehensive	ahead of time + JIT
practical	real browsers
fast	Less than 6% overhead

Readactor++: Architecture

Compile time

Compiler

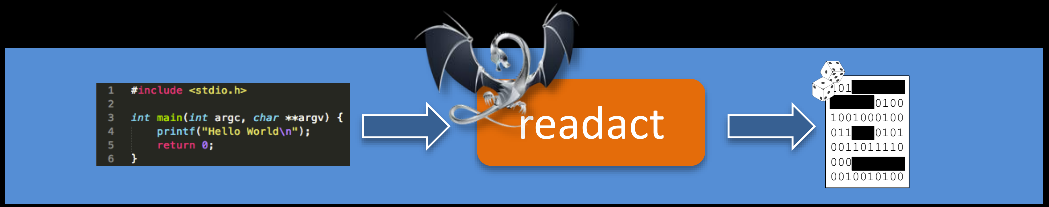


```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

Readactor++: Architecture

Compile time

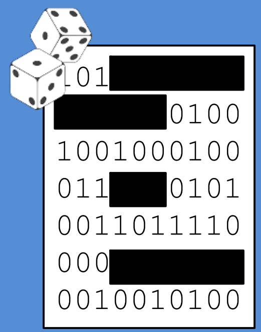
Compiler



```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

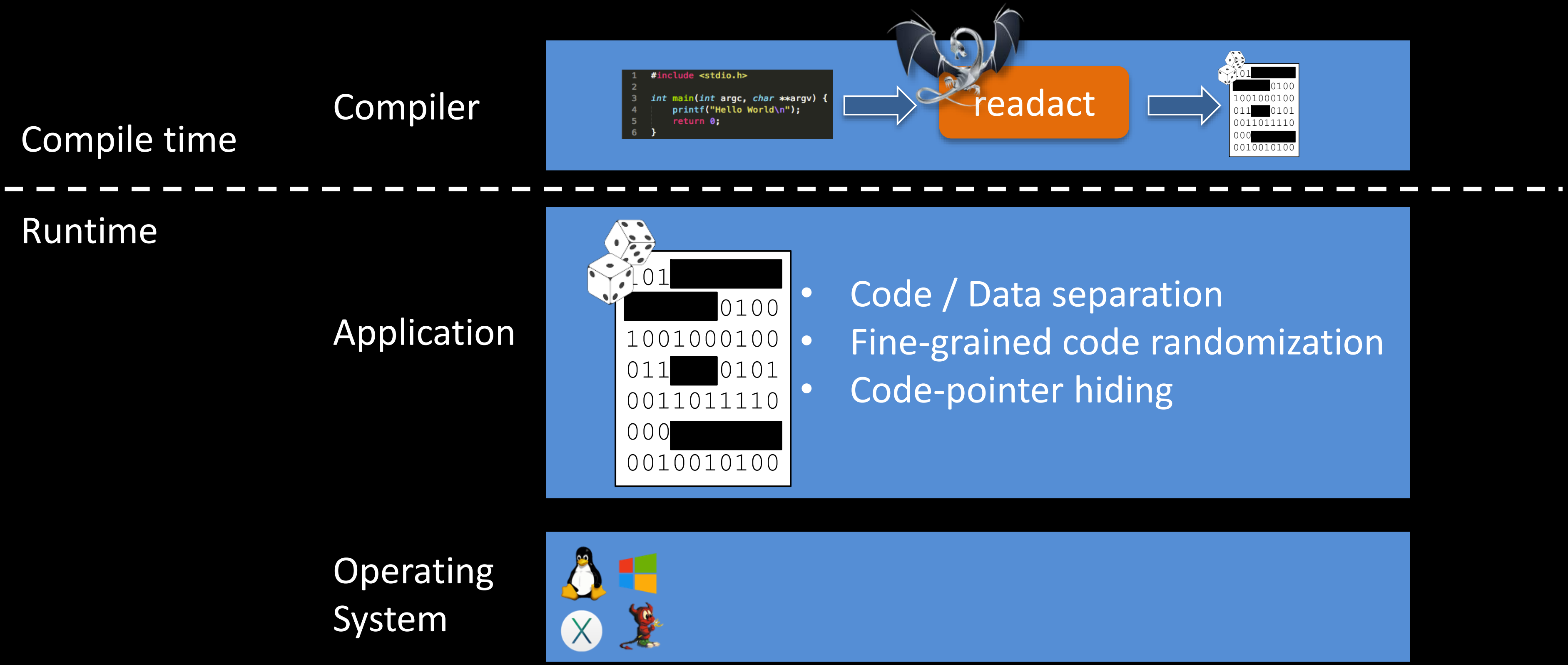
Runtime

Application



- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding

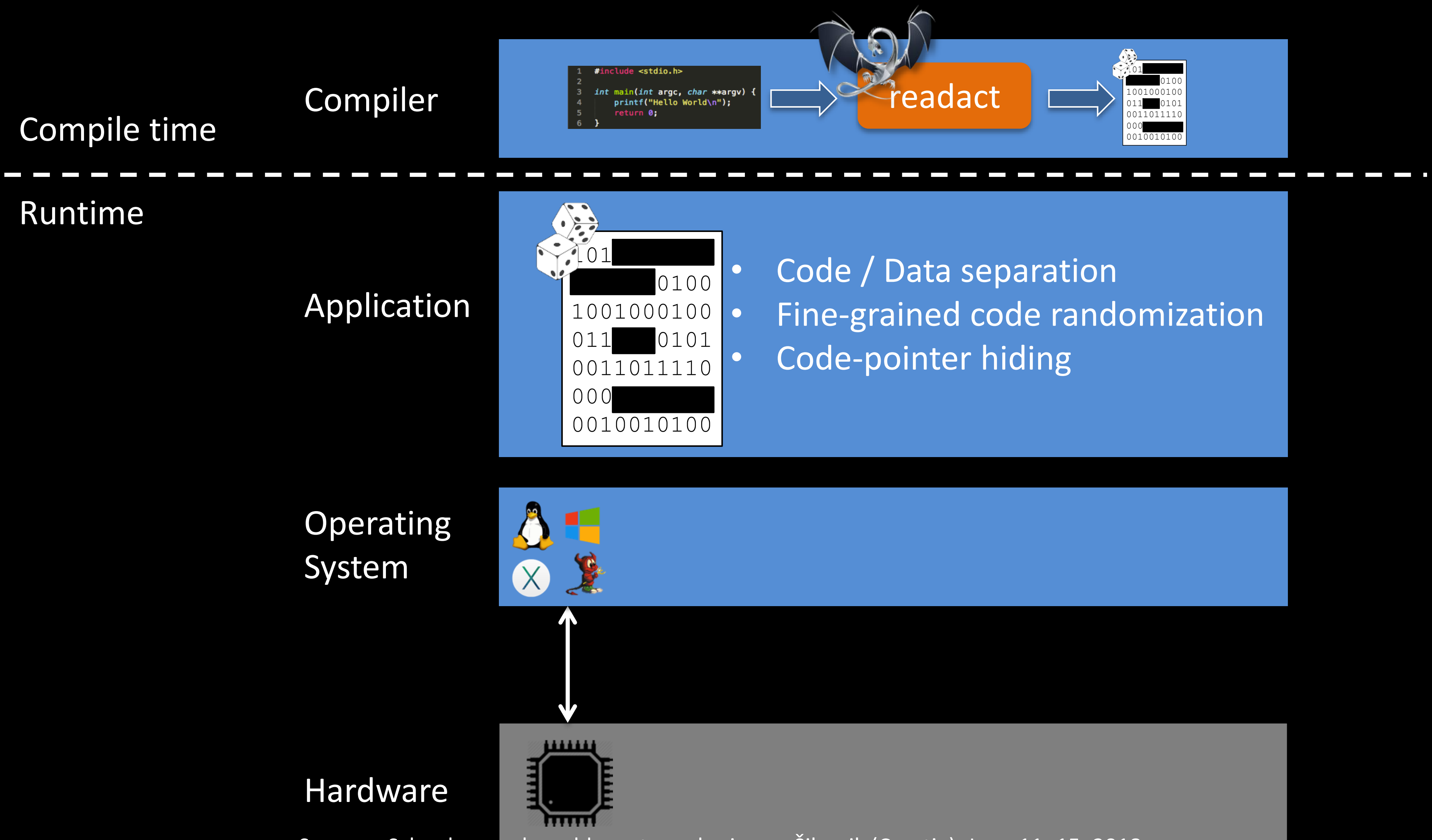
Readactor++: Architecture



```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding

Readactor++: Architecture



Compile time

Runtime

Compiler

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

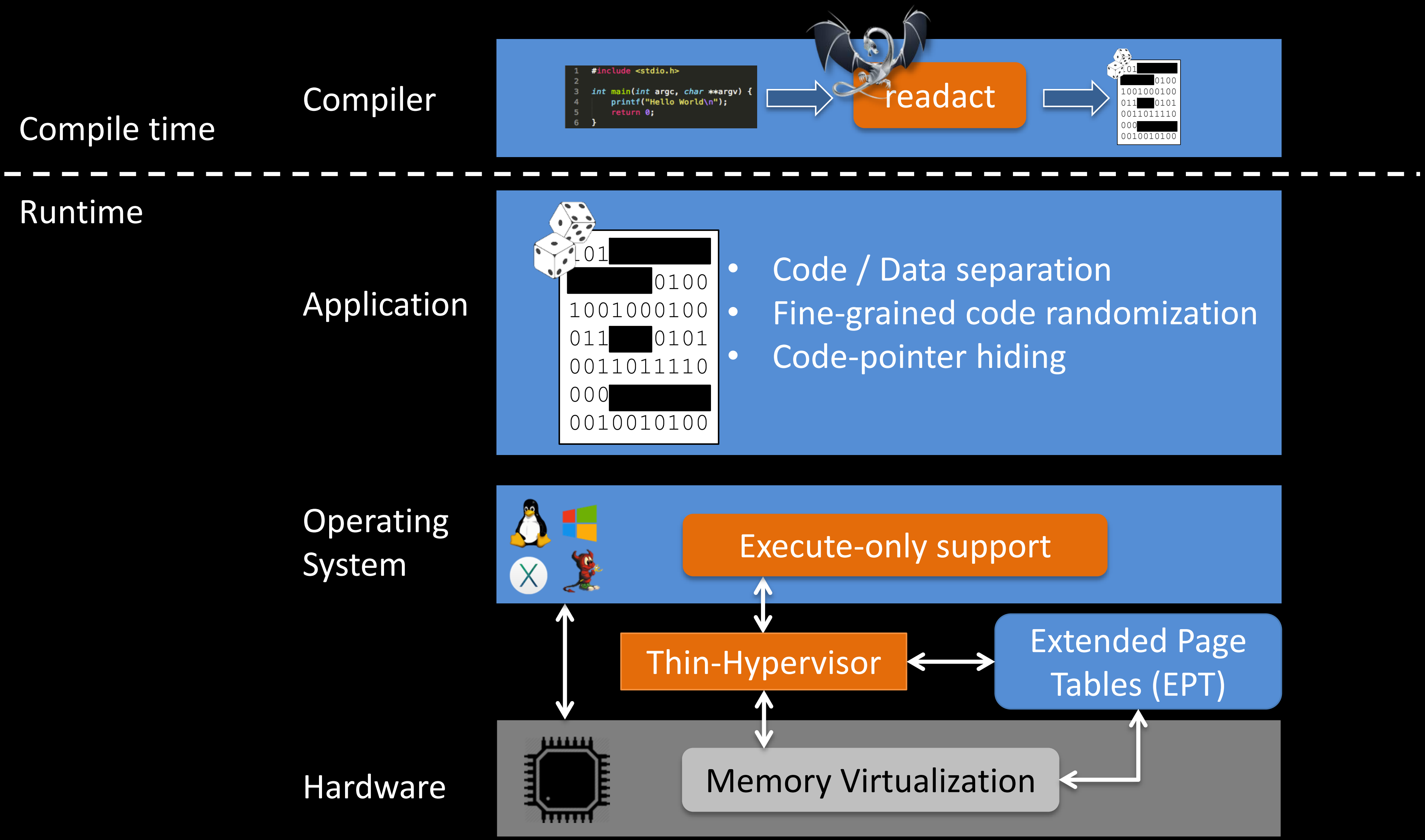
Application

- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding

Operating System

Hardware

Readactor++: Architecture



Compile time

Compiler

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

readact

Runtime

Application

- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding

Operating System

Execute-only support

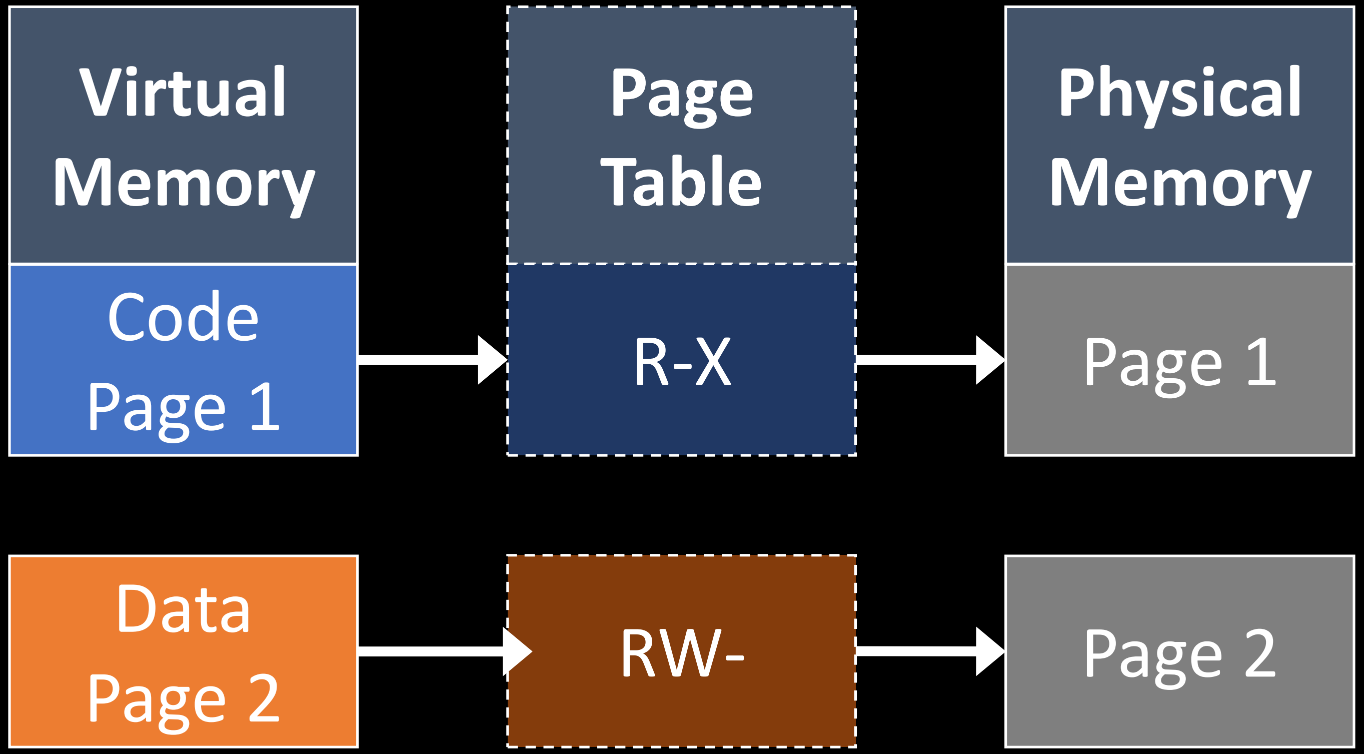
Thin-Hypervisor

Extended Page Tables (EPT)

Hardware

Memory Virtualization

Execute-Only EPT Mapping



Virtual Memory

Page Table

Physical Memory

Code Page 1

R-X

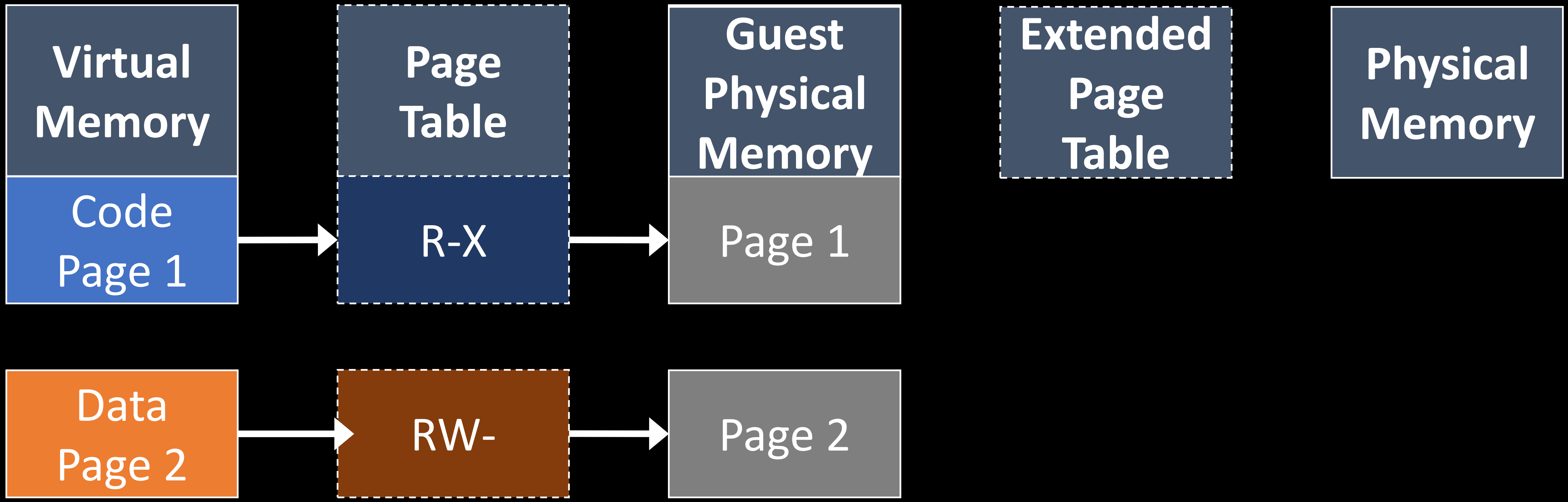
Page 1

Data Page 2

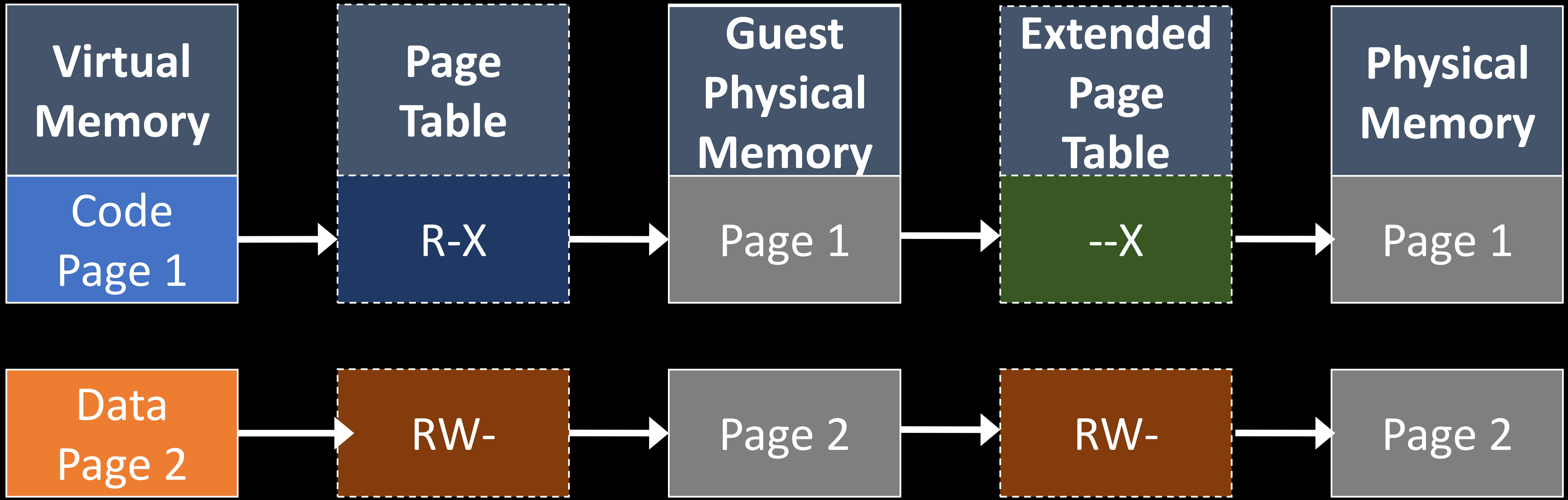
RW-

Page 2

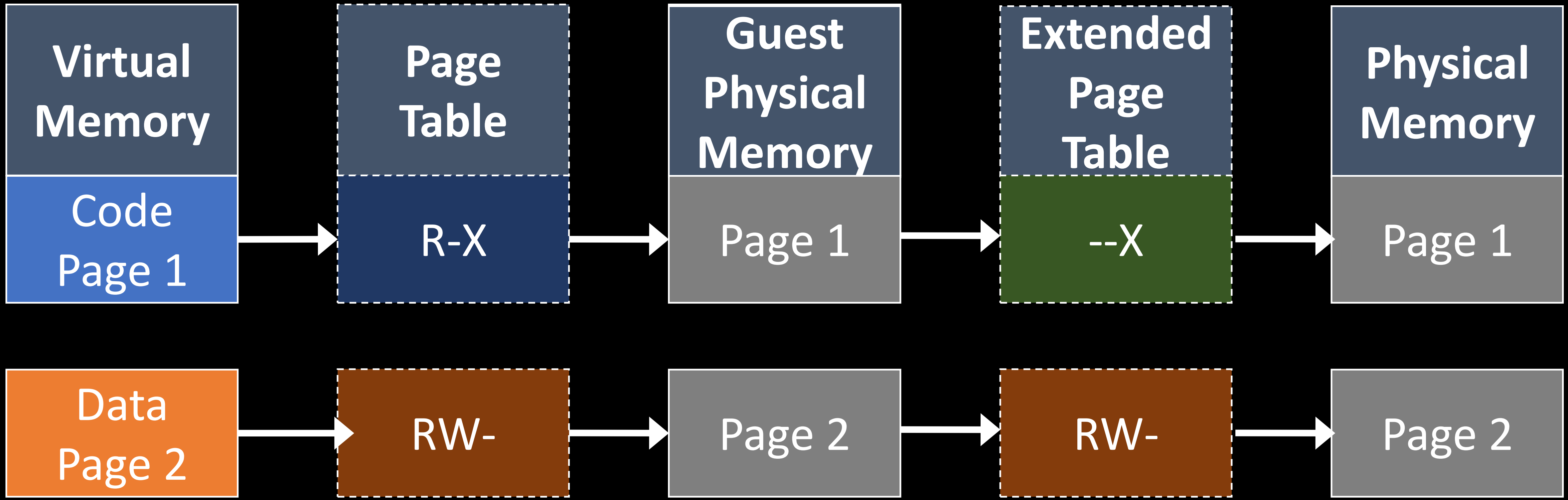
Execute-Only EPT Mapping



Execute-Only EPT Mapping



Execute-Only EPT Mapping



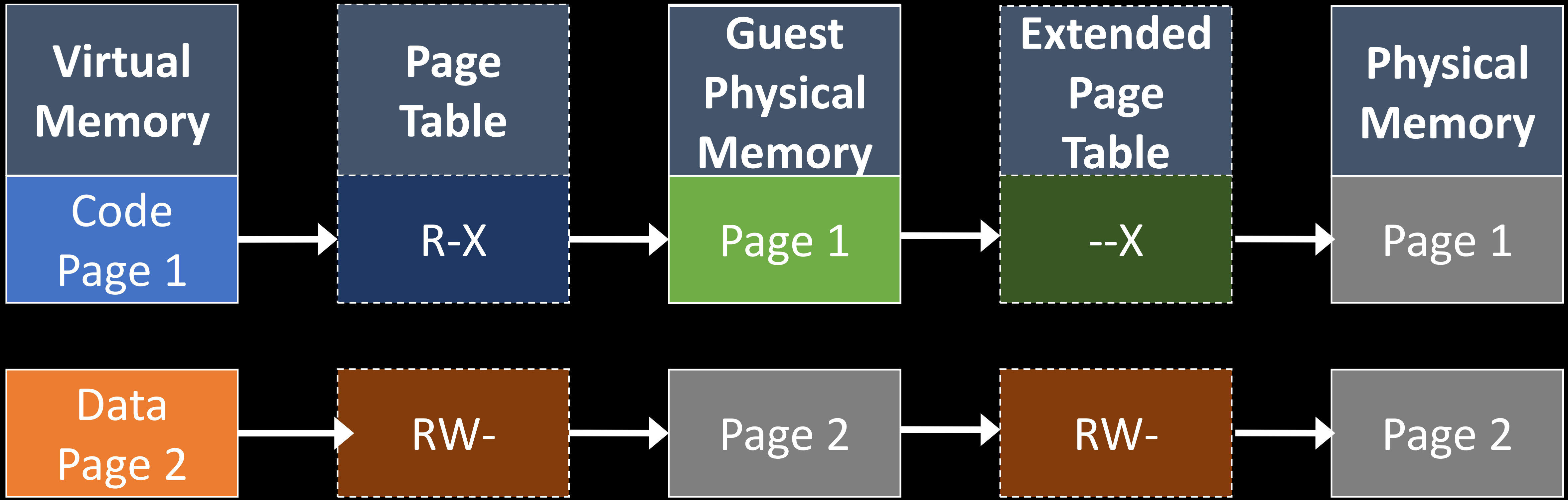
Virtual Memory | Page Table | Guest Physical Memory | Extended Page Table | Physical Memory

Code Page 1 → R-X → Page 1 → --X → Page 1

Data Page 2 → RW- → Page 2 → RW- → Page 2

Effective Permission = Intersection of Page Table and Extended Page Table Permissions

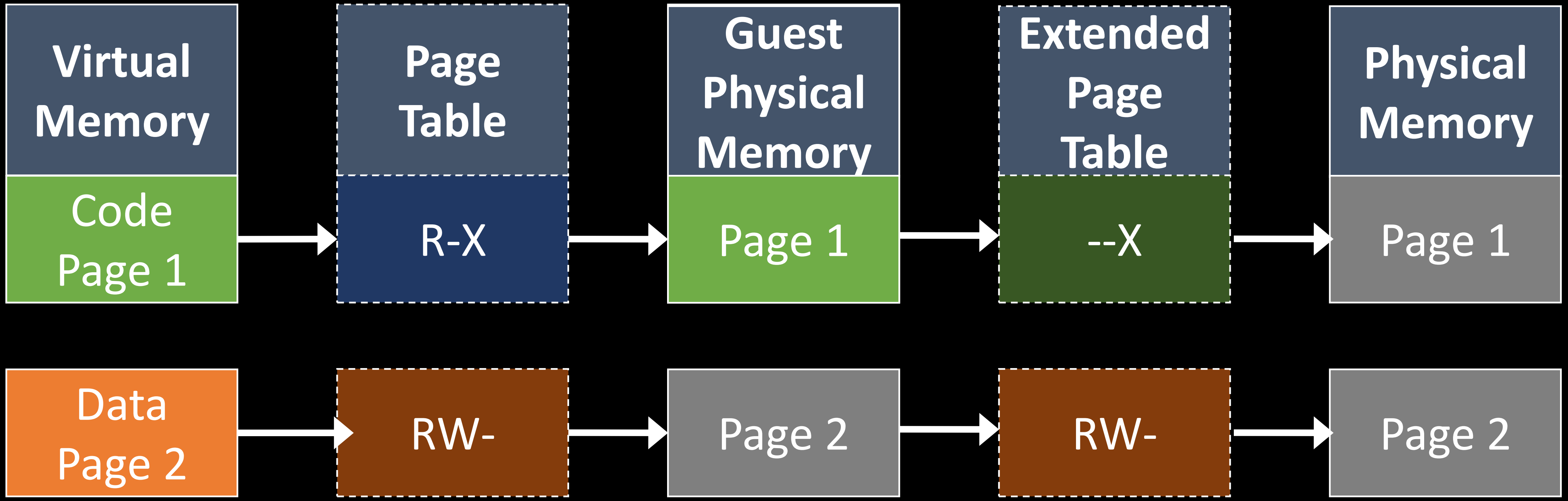
Execute-Only EPT Mapping



Virtual Memory	Page Table	Guest Physical Memory	Extended Page Table	Physical Memory
Code Page 1	R-X	Page 1	--X	Page 1
Data Page 2	RW-	Page 2	RW-	Page 2

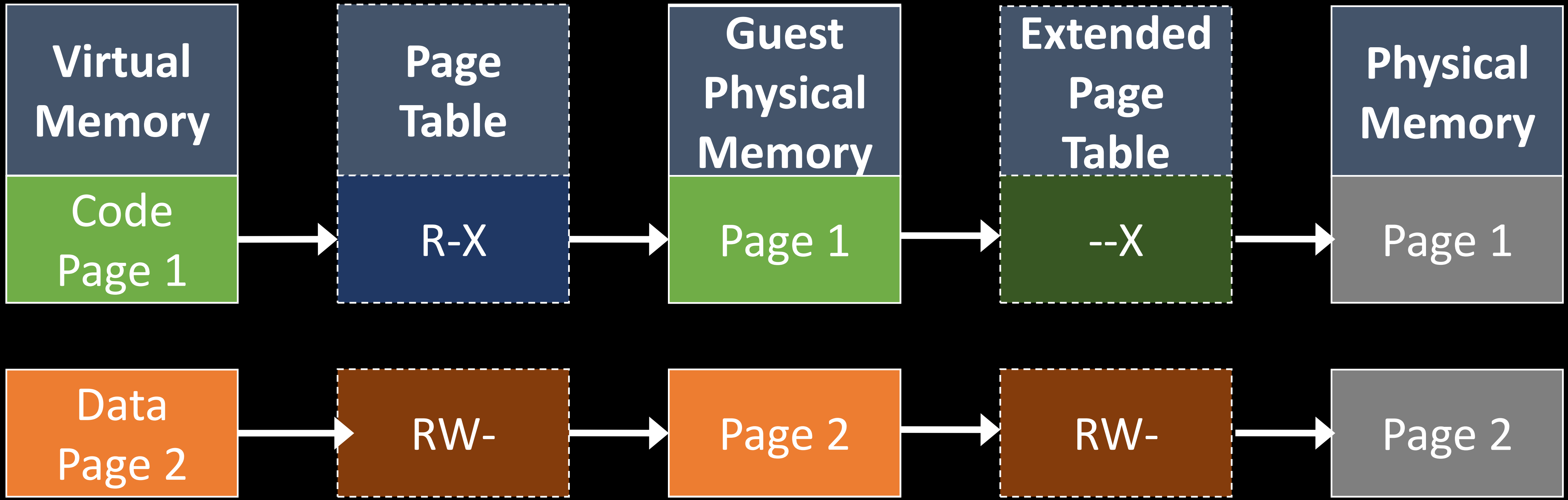
Effective Permission = Intersection of Page Table and Extended Page Table Permissions

Execute-Only EPT Mapping



Effective Permission = Intersection of Page Table and Extended Page Table Permissions

Execute-Only EPT Mapping



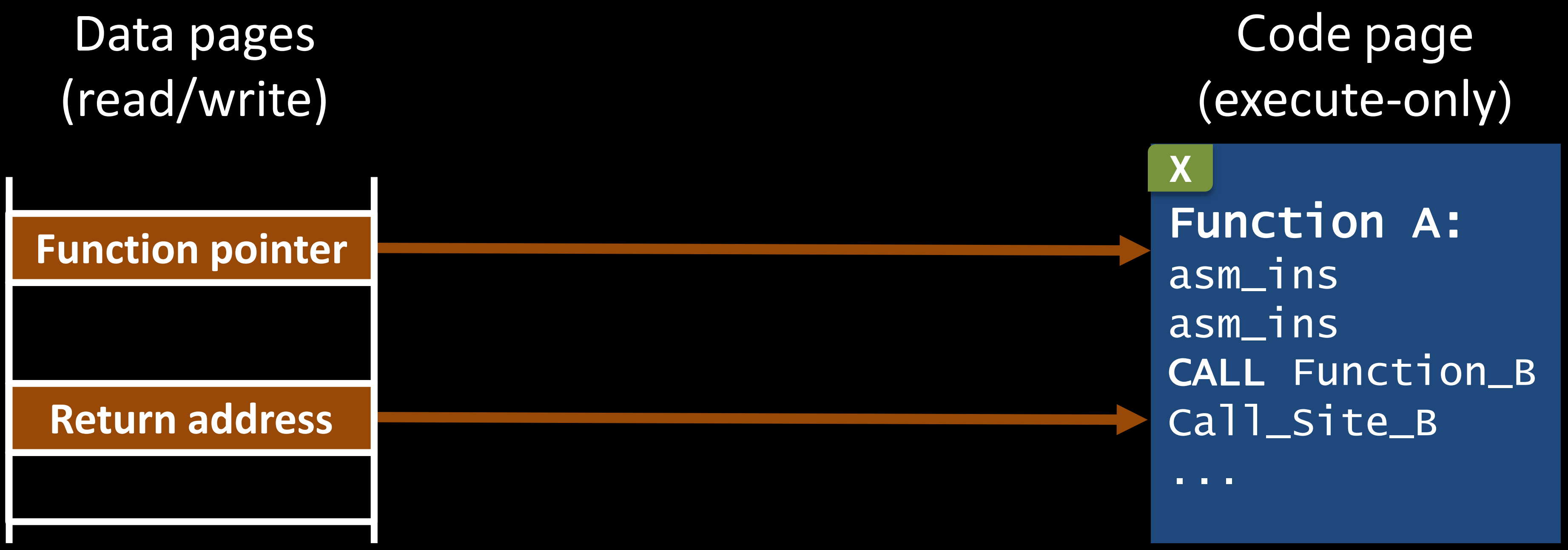
Virtual Memory → Page Table → Guest Physical Memory → Extended Page Table → Physical Memory

Code Page 1 → R-X → Page 1 → --X → Page 1

Data Page 2 → RW- → Page 2 → RW- → Page 2

Effective Permission = Intersection of Page Table and Extended Page Table Permissions

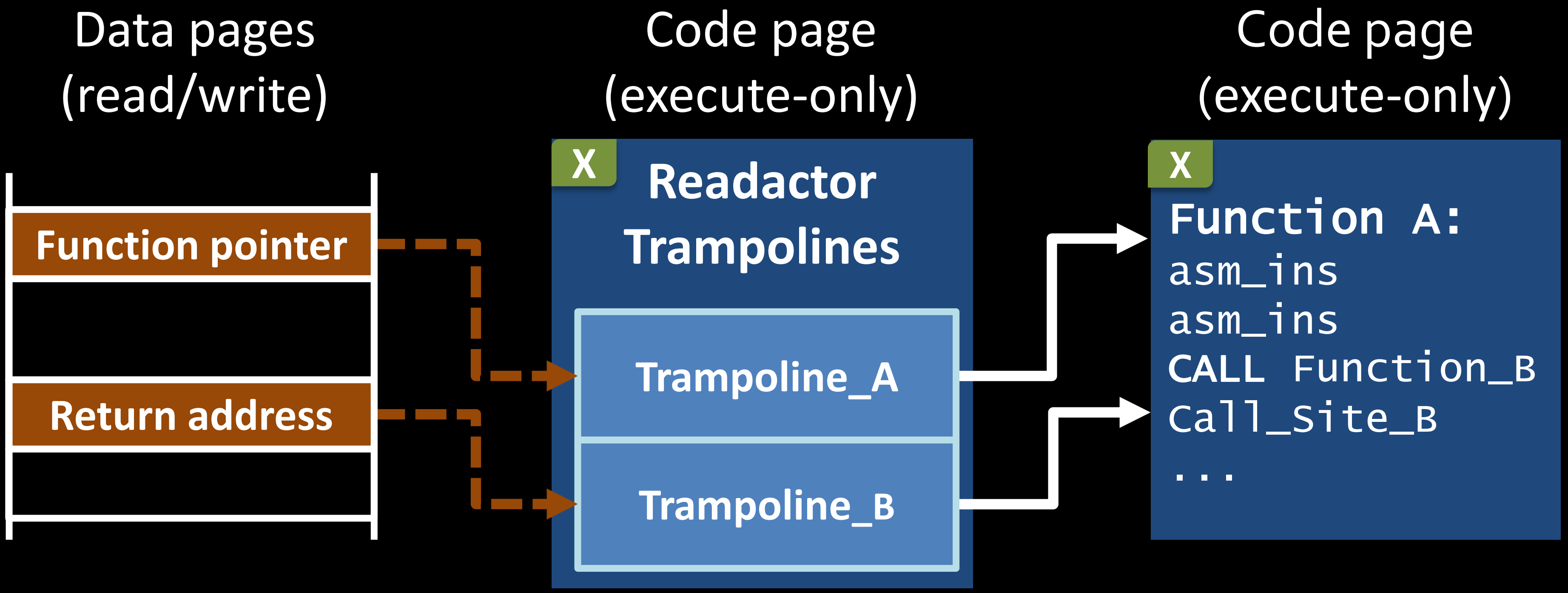
Code-Pointer Hiding



```
Function A:
asm_ins
asm_ins
CALL Function_B
Call_Site_B
...
```

🟫 Readable/Writable 🟦 Execute-Only

Code-Pointer Hiding



🟧 Readable/Writable 🟦 Execute-Only

Leakage Resilient Layout Randomization with no HW Support

LR²:

Leakage-Resilient Layout Randomization for Mobile Devices

The Network and Distributed System Security Symposium (NDSS) 2016

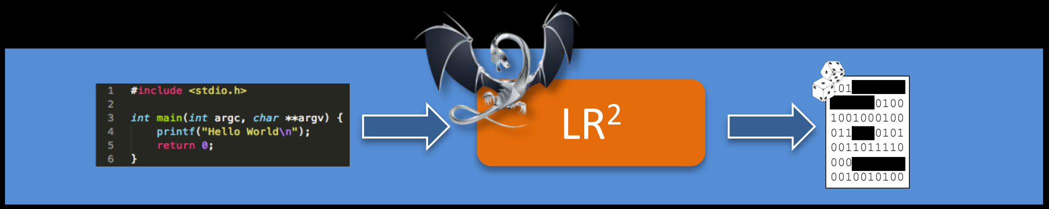
Kjell Braden, Stephen Crane, Lucas Davi, Michael Franz, Per Larsen, Christopher Liebchen, Ahmad-Reza Sadeghi

LR^2: Leakage Resilient Layout Randomization

[Braden et al. NDSS'16]

Compile time

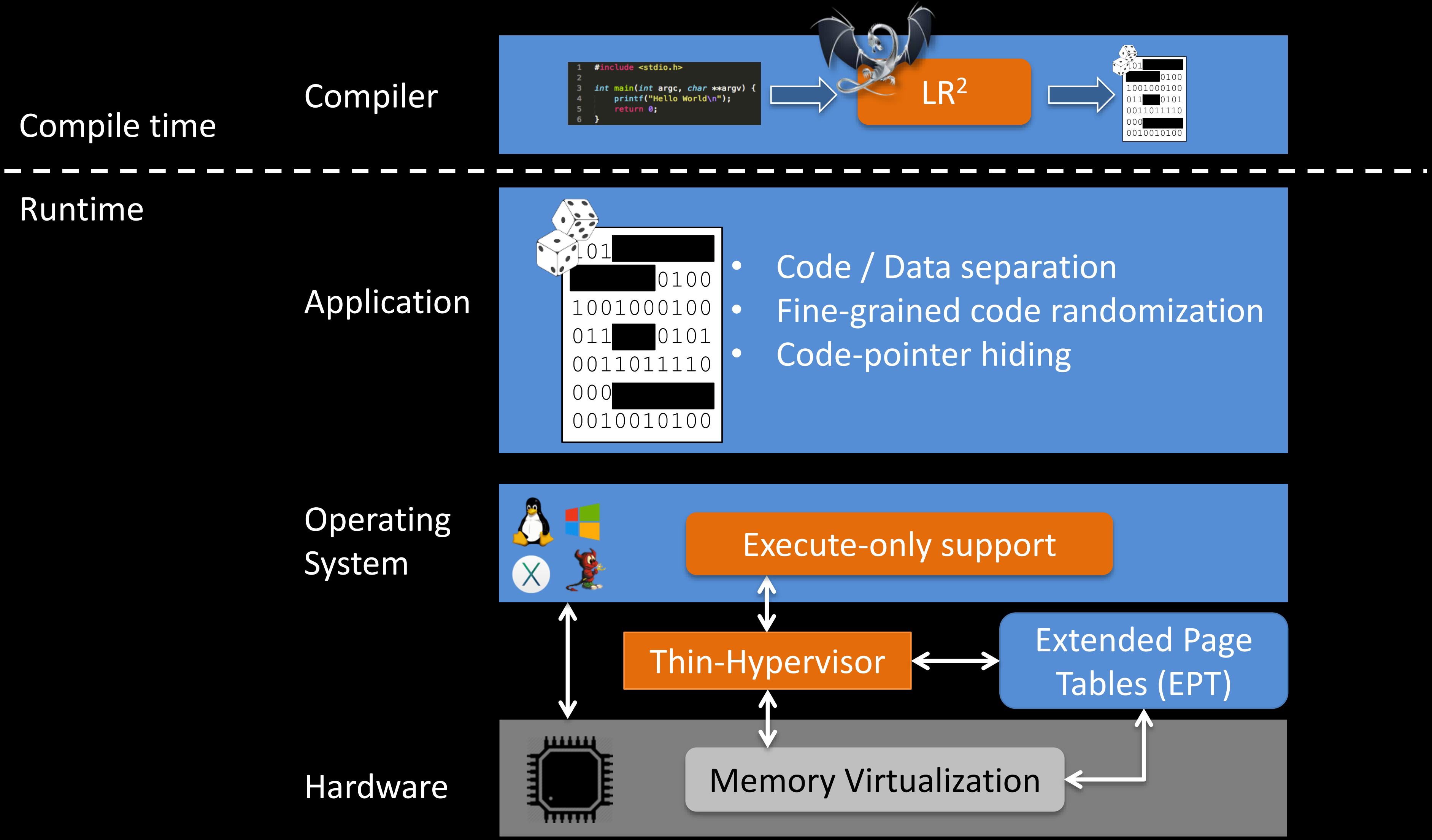
Compiler



```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

LR^2: Leakage Resilient Layout Randomization

[Braden et al. NDSS'16]



Compile time

Compiler

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

LR^2

Runtime

Application

- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding

Operating System

Execute-only support

Thin-Hypervisor

Extended Page Tables (EPT)

Hardware

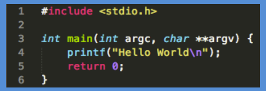
Memory Virtualization

LR^2: Leakage Resilient Layout Randomization

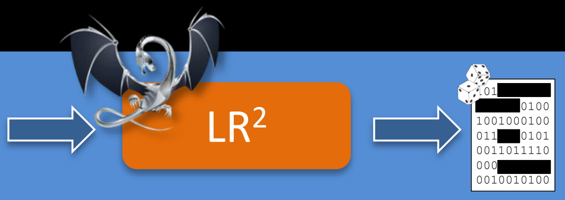
[Braden et al. NDSS'16]

Compile time

Compiler

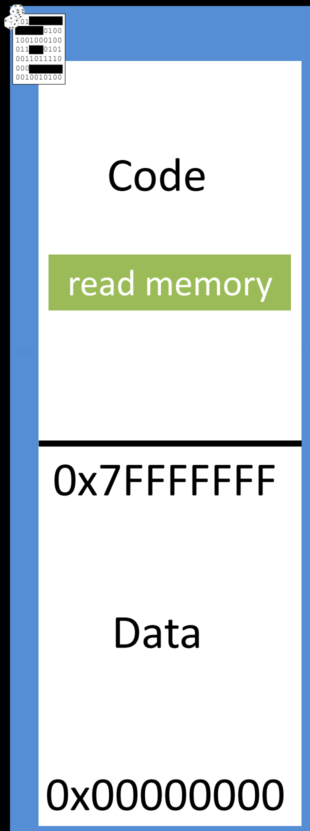


```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```



Runtime

Application



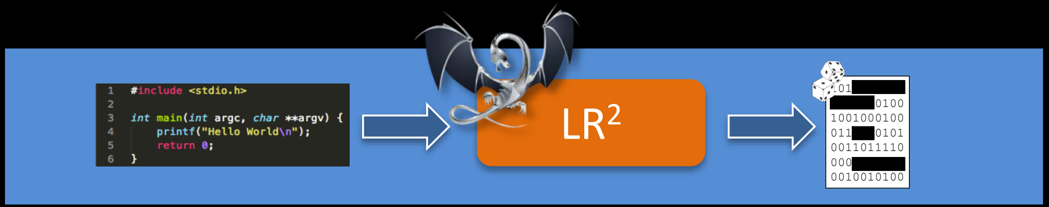
- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding

LR²: Leakage Resilient Layout Randomization

[Braden et al. NDSS'16]

Compile time

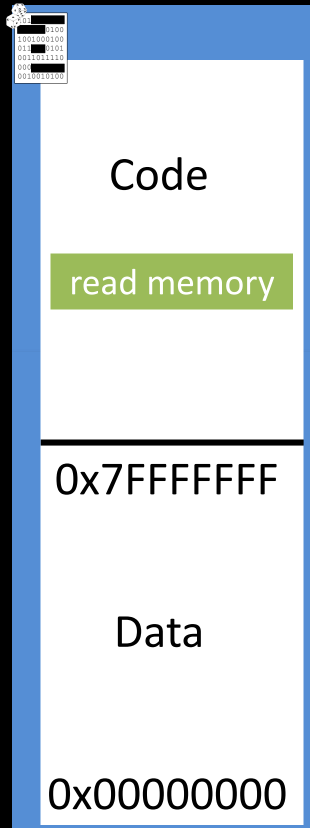
Compiler



```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

Runtime

Application



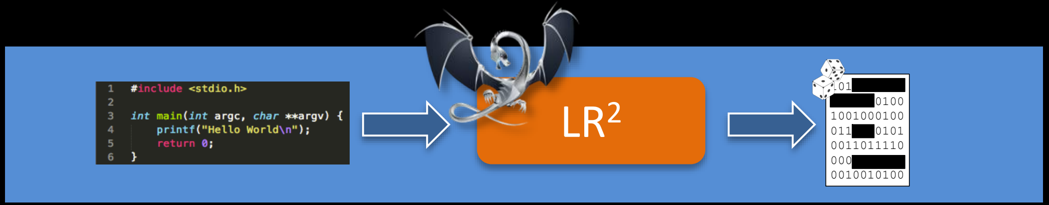
- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding
- Sandboxing Read-instruction (to prevent read access to the code section)

LR^2: Leakage Resilient Layout Randomization

[Braden et al. NDSS'16]

Compile time

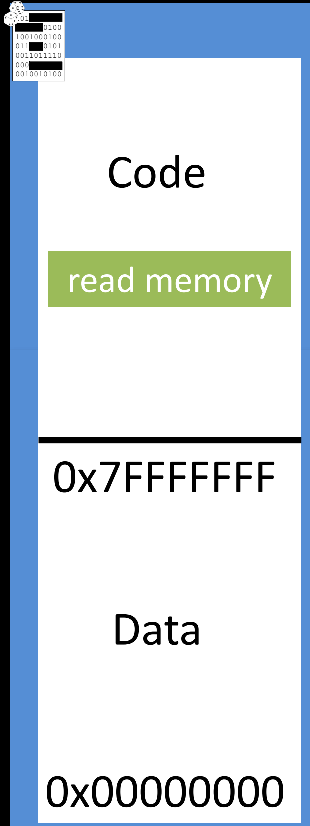
Compiler



```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello World\n");
    return 0;
}
```

Runtime

Application



- Code / Data separation
- Fine-grained code randomization
- Code-pointer hiding
- Sandboxing Read-instruction (to prevent read access to the code section)

```
r1 <- addr                 ; load target addr in r1
r1 = r1 & 0x7FFFFFFF       ; ensures r1 is
                           ; always < 0x8000000
                           ; by removing the
                           ; highest bit
r0 <- [r1]                 ; read memory in r0
```

Hardening Tor Browser against De-anonymization Attacks

Selfrando:

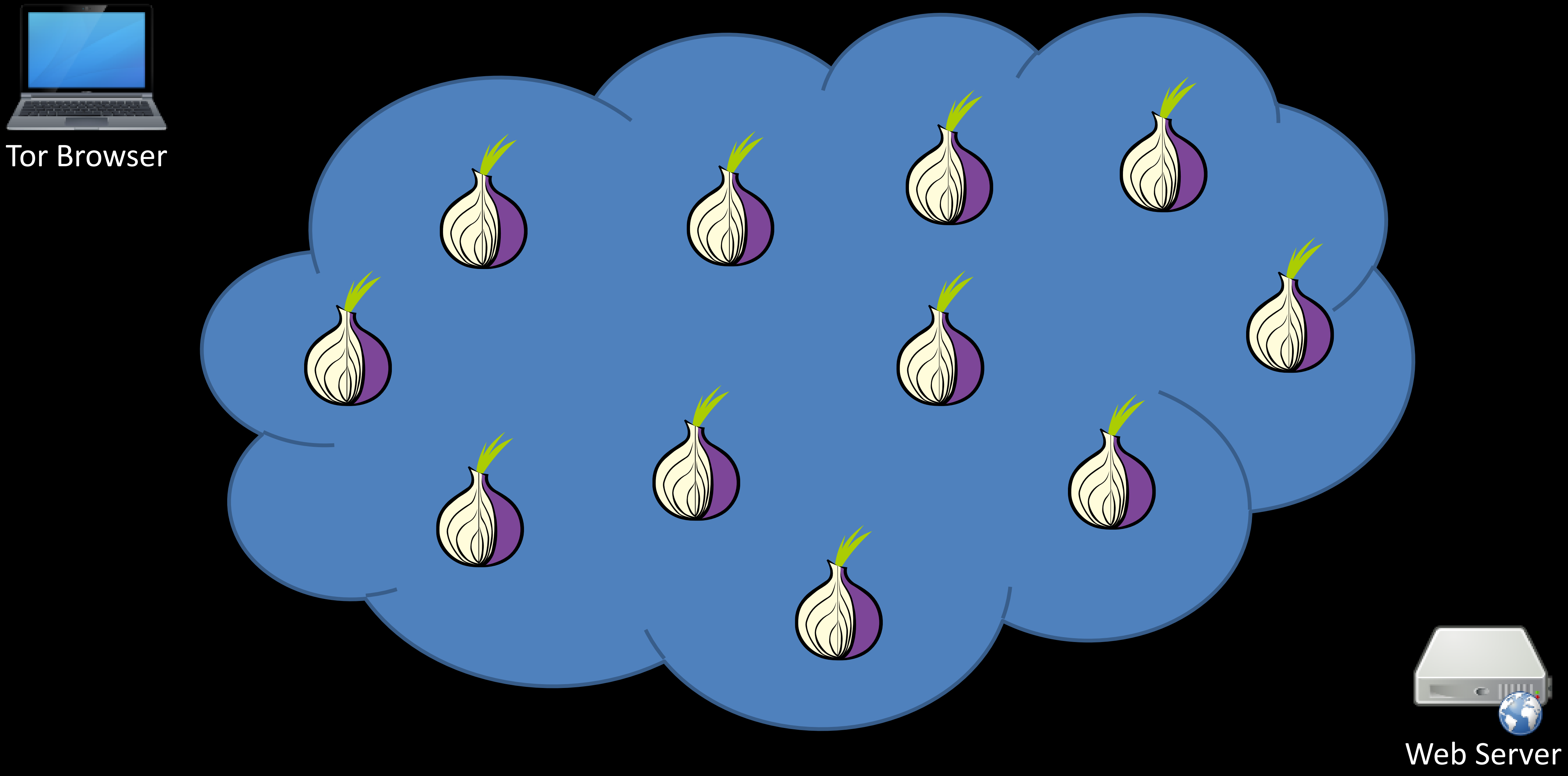
Securing the Tor Browser against De-anonymization Exploits,

Privacy Enhancing Technologies Symposium (PETS) 2016

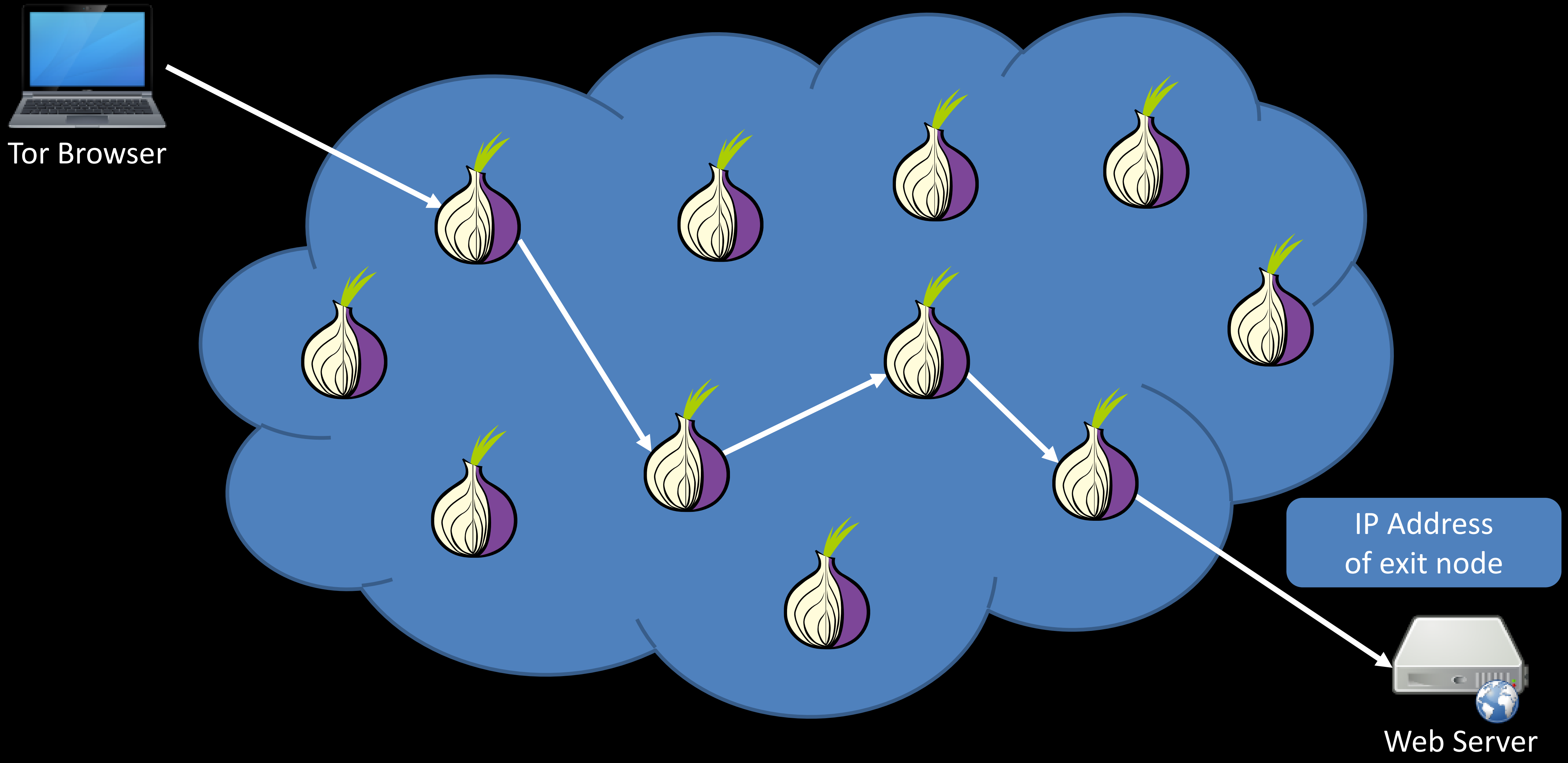
Mauro Conti, Stephen Crane, Tommaso Frassetto, Andrei Homescu, Georg Koppen, Per Larsen, Christopher Liebchen, Mike Perry, Ahmad-Reza Sadeghi

Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

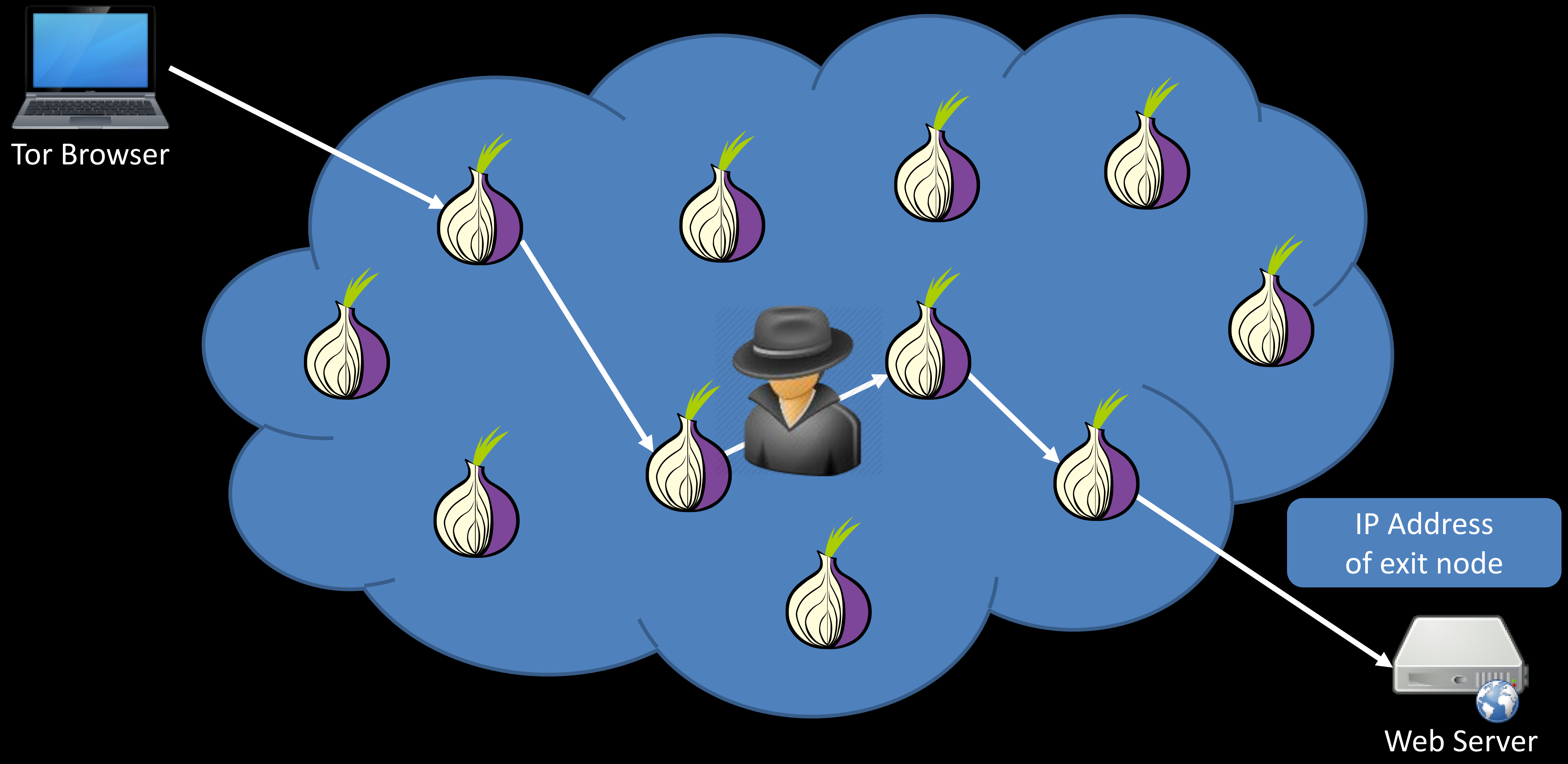
De-anonymization Attacks on Tor Browser (FBI exploit – 2013)



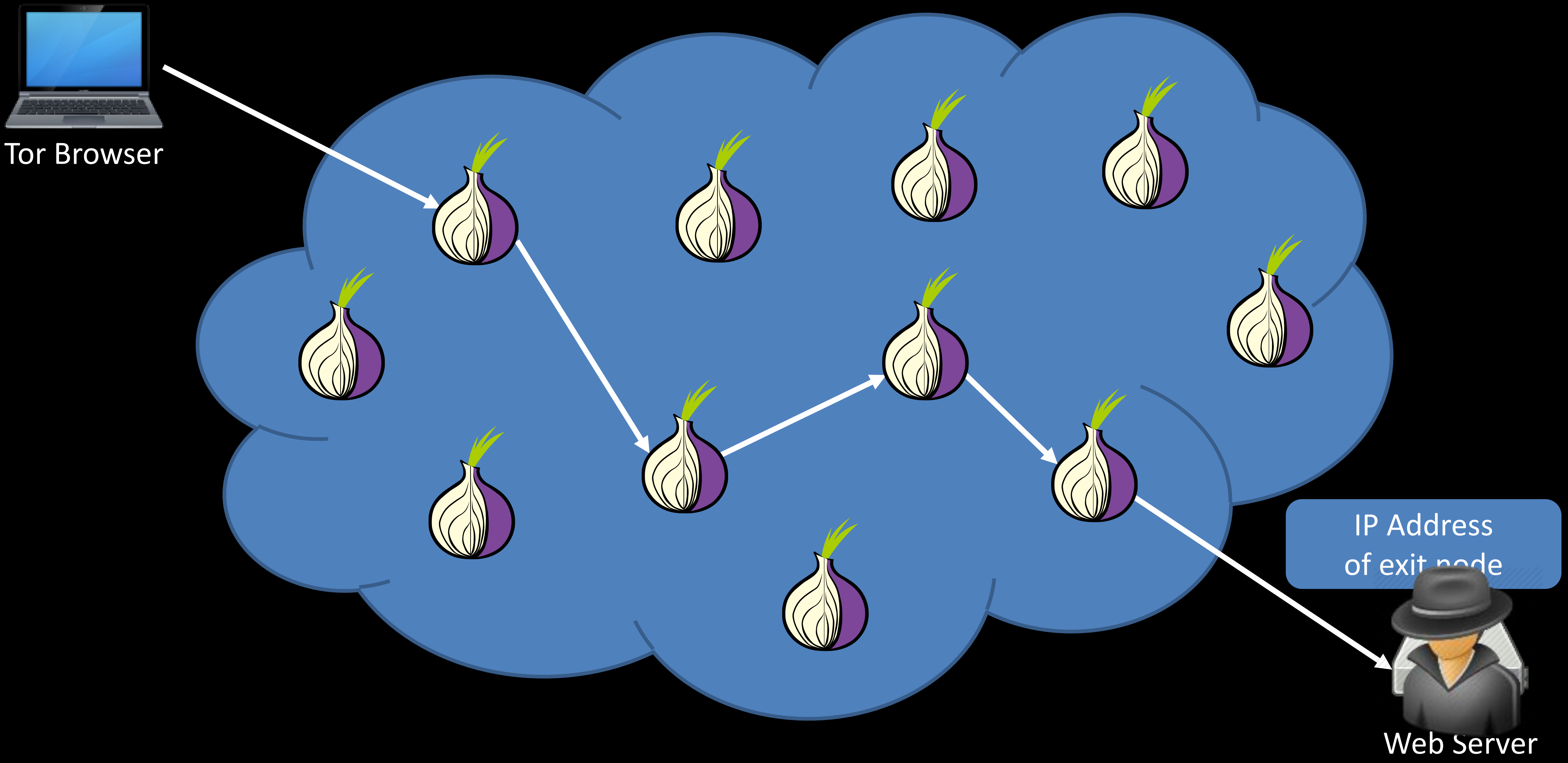
De-anonymization Attacks on Tor Browser (FBI exploit – 2013)



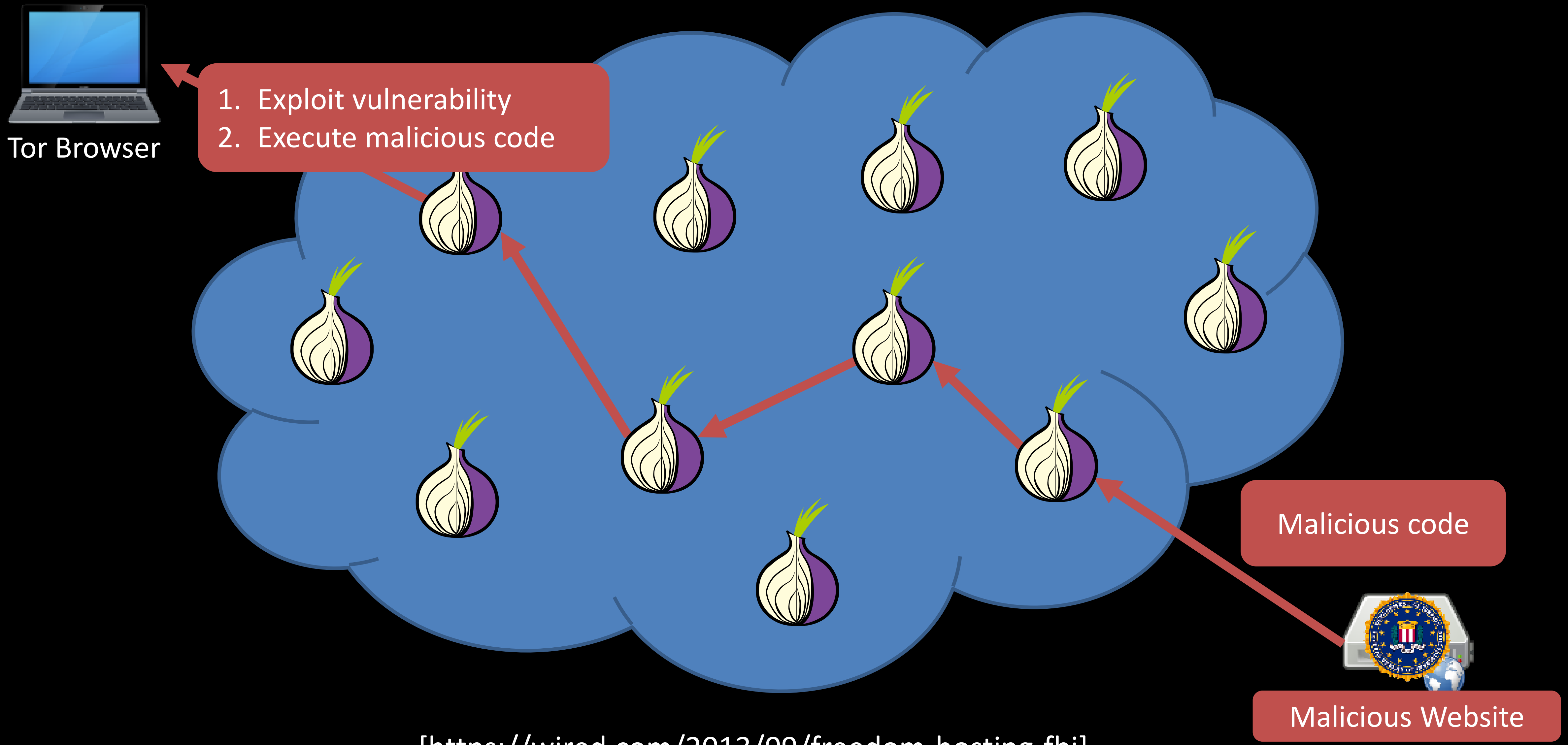
De-anonymization Attacks on Tor Browser (FBI exploit – 2013)



De-anonymization Attacks on Tor Browser (FBI exploit – 2013)

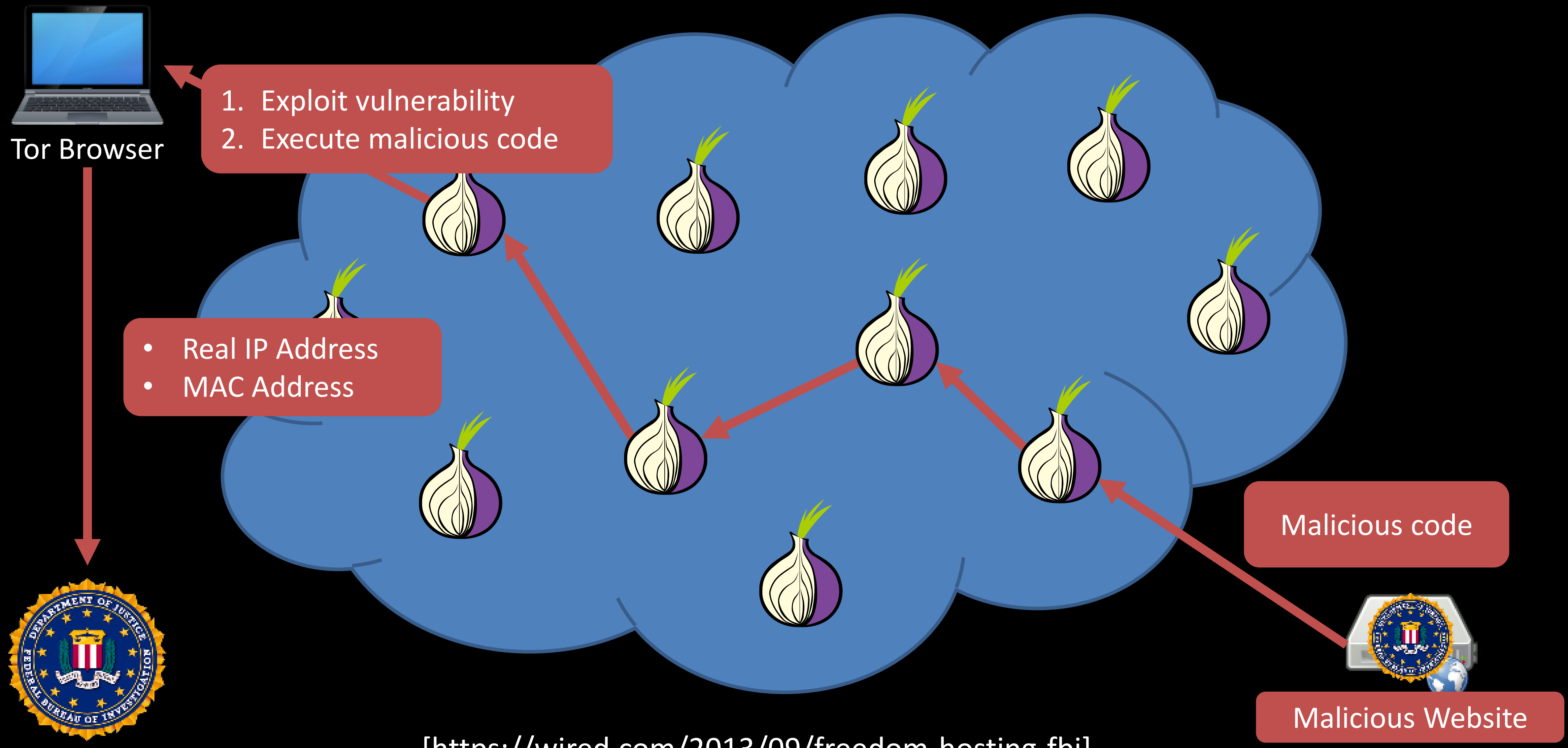


De-anonymization Attacks on Tor Browser (FBI exploit – 2013)



[https://wired.com/2013/09/freedom-hosting-fbi]

De-anonymization Attacks on Tor Browser (FBI exploit – 2013)



1. Exploit vulnerability
2. Execute malicious code

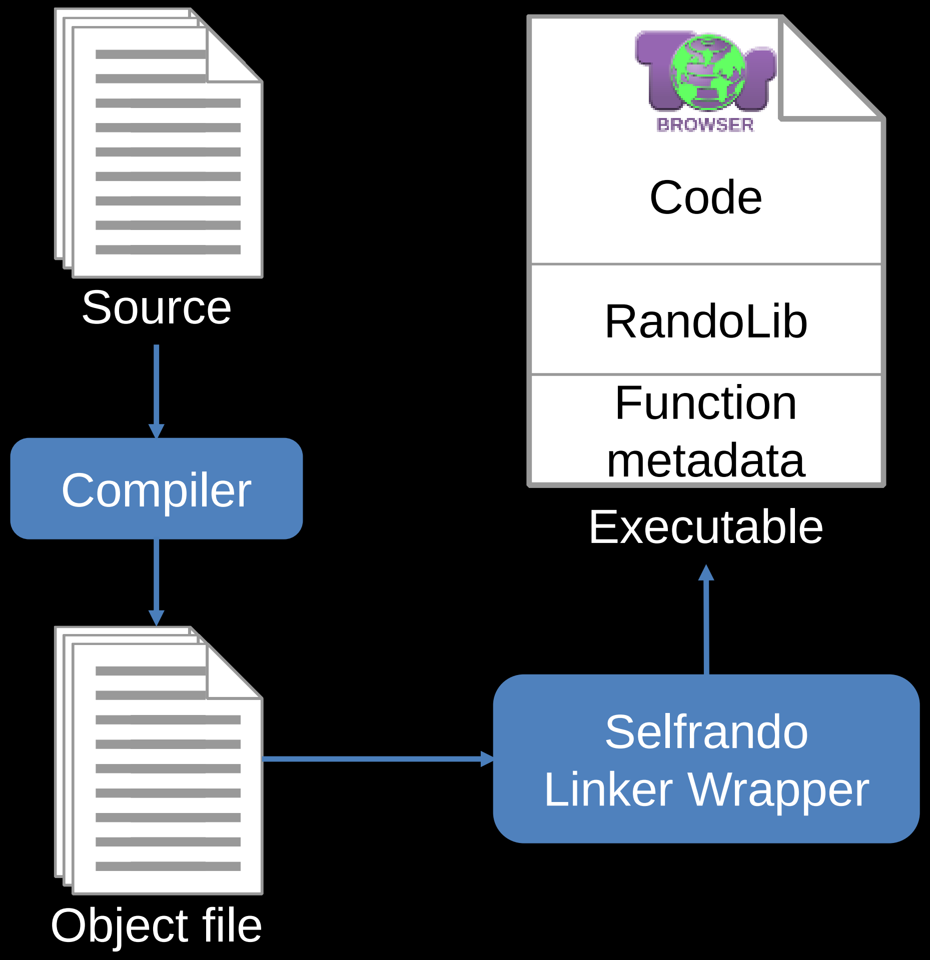
- Real IP Address
- MAC Address

Malicious code

[https://wired.com/2013/09/freedom-hosting-fbi]

Selfrando

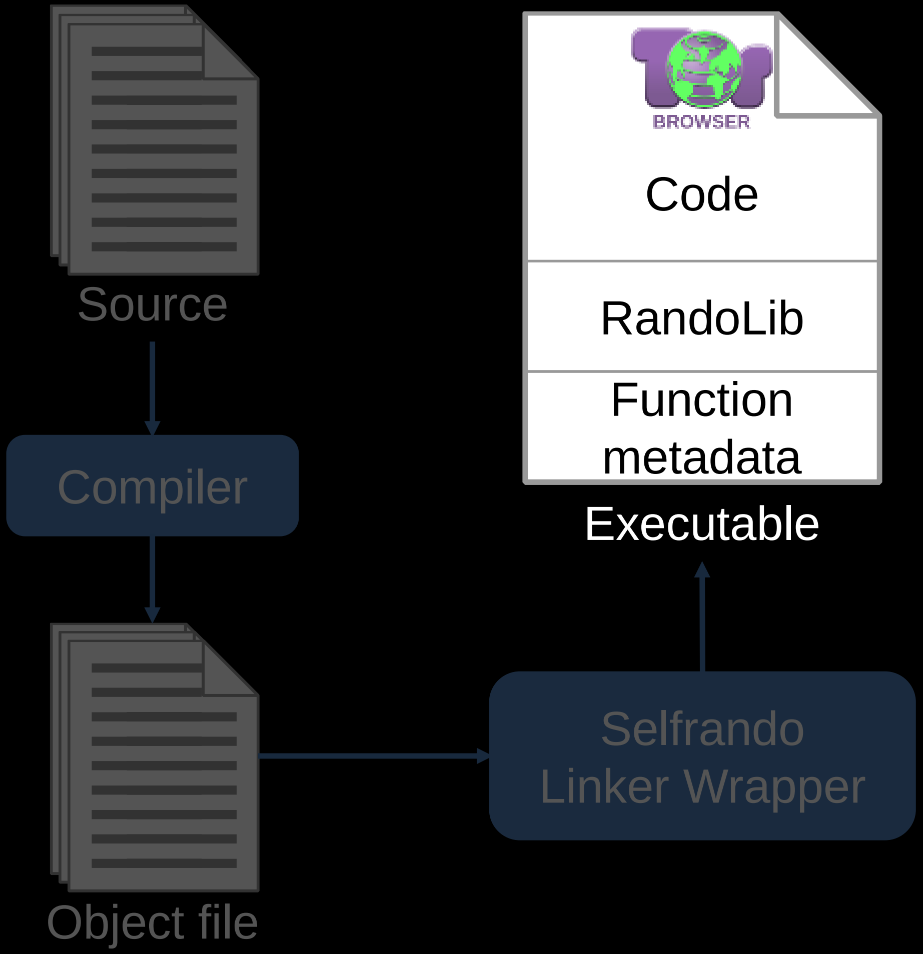
Compile Time



[Selfrando: Frassetto et al., PETS 2016]

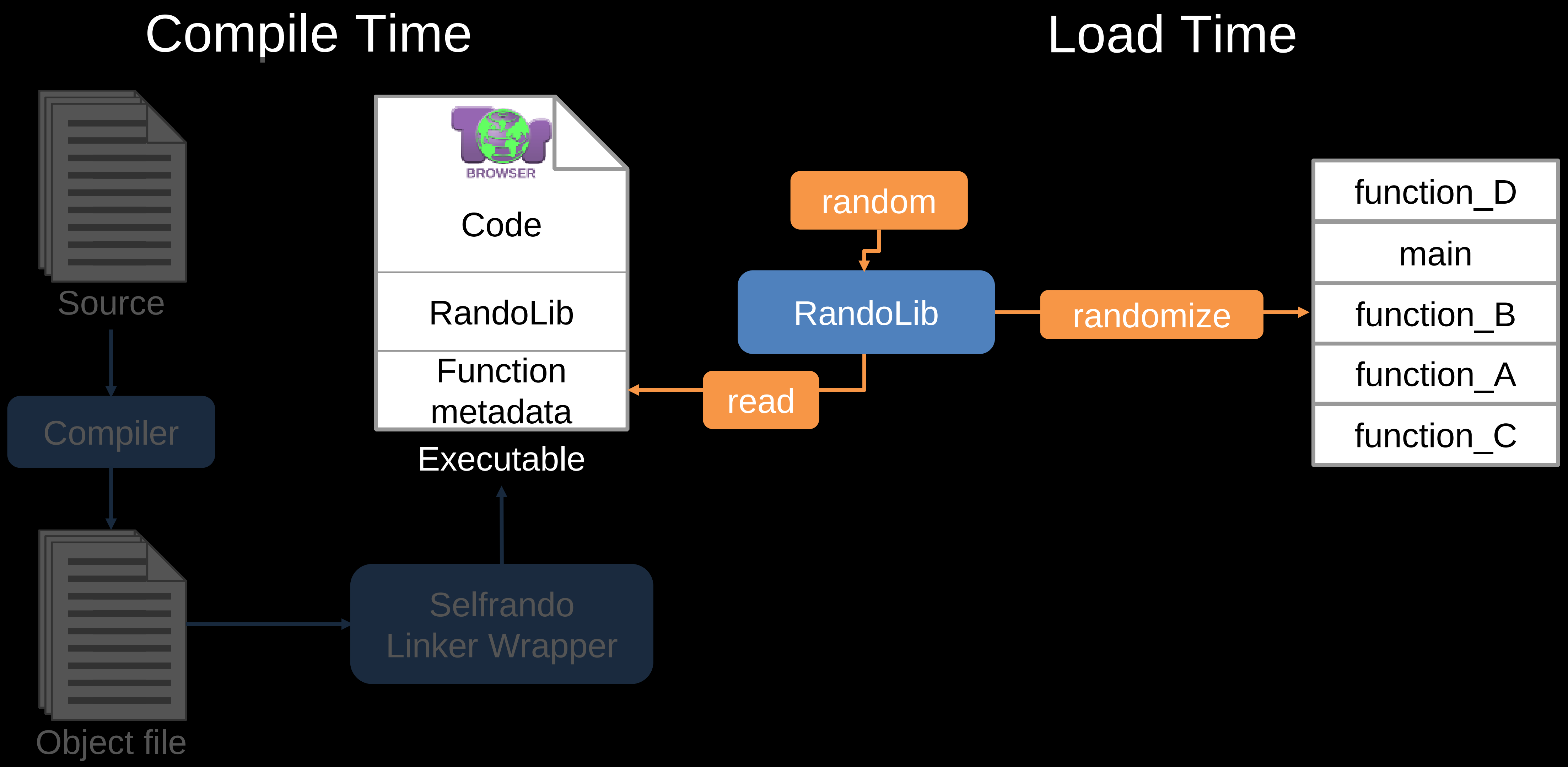
Selfrando

Compile Time



[Selfrando: Frassetto et al., PETS 2016]

Selfrando



[Selfrando: Frassetto et al., PETS 2016]

Selfrando

- Load-time randomization
  - Supports traditional distribution channels
  - Allows traditional code signatures
- Requires no modifications to compiler or system configuration
- Supports AddressSanitizer (requested by Tor Project)

[Selfrando: Frassetto et al., PETS 2016]

Lesson learned:

Moving targets are ineffective if attackers enjoy dancing

Where do we go from here?

- Understanding side-channel attacks
  - Adversary typically requires code execution (e.g., JavaScript, or during local privilege escalation attacks against kernel)
  - Not applicable remotely (unlike memory corruption), yet?

- Randomization is easy
  - to deploy (no code changes)
  - effective if the adversary cannot access advanced attack primitives, as need for JIT-ROP attacks

Current Work

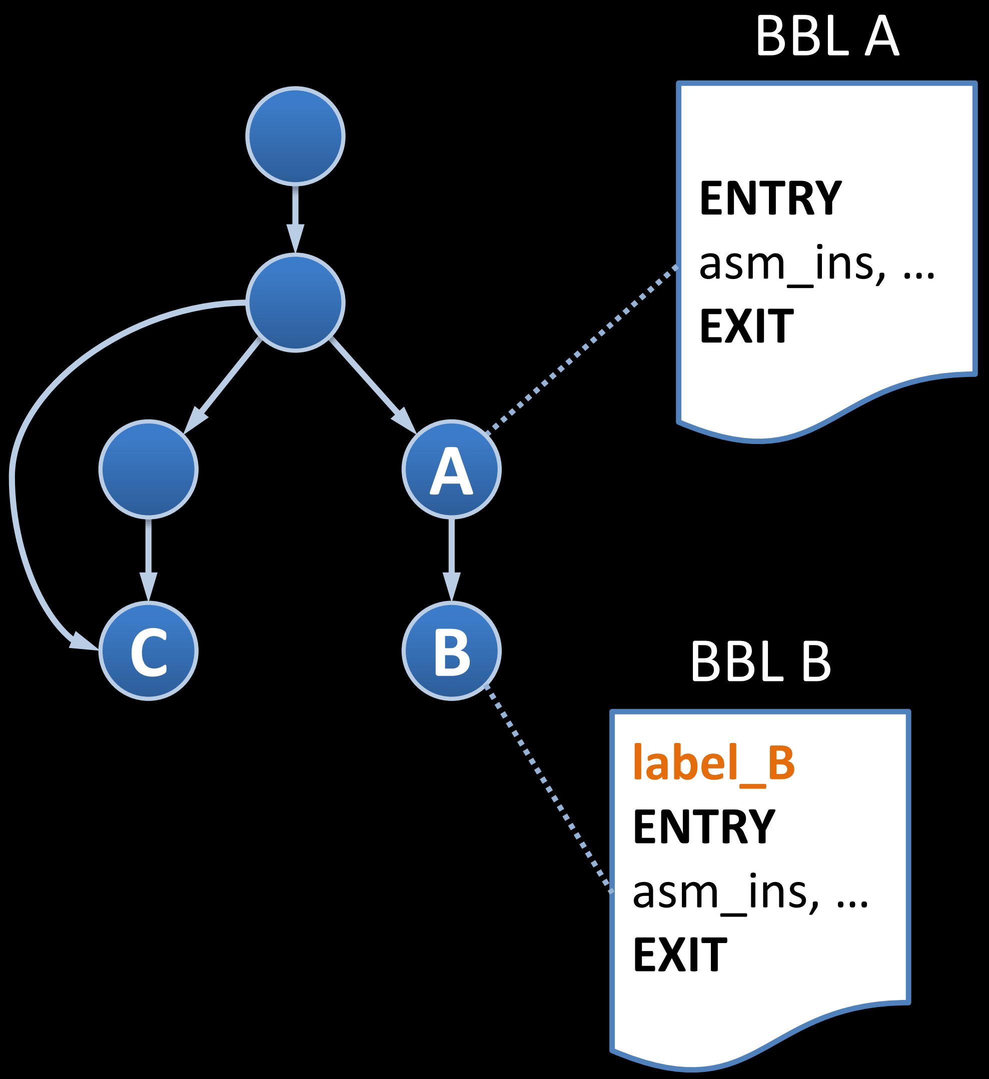
- Side channel resilience for SGX [Brasser et al., Arxiv]
- Page Table Randomization to mitigate Data-only Attacks against Page Table [Davi et al., NDSS'17]
- Side-Channel Resilient Kernel Address Space Layout Randomization [Arias et al., RAID'17]
- Randomization for embedded systems (no JavaScript but entropy problem).

EPISODE II
Control-Flow Integrity (CFI)

Restricting indirect targets
to a pre-defined control-flow graph

Original CFI Label Checking

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

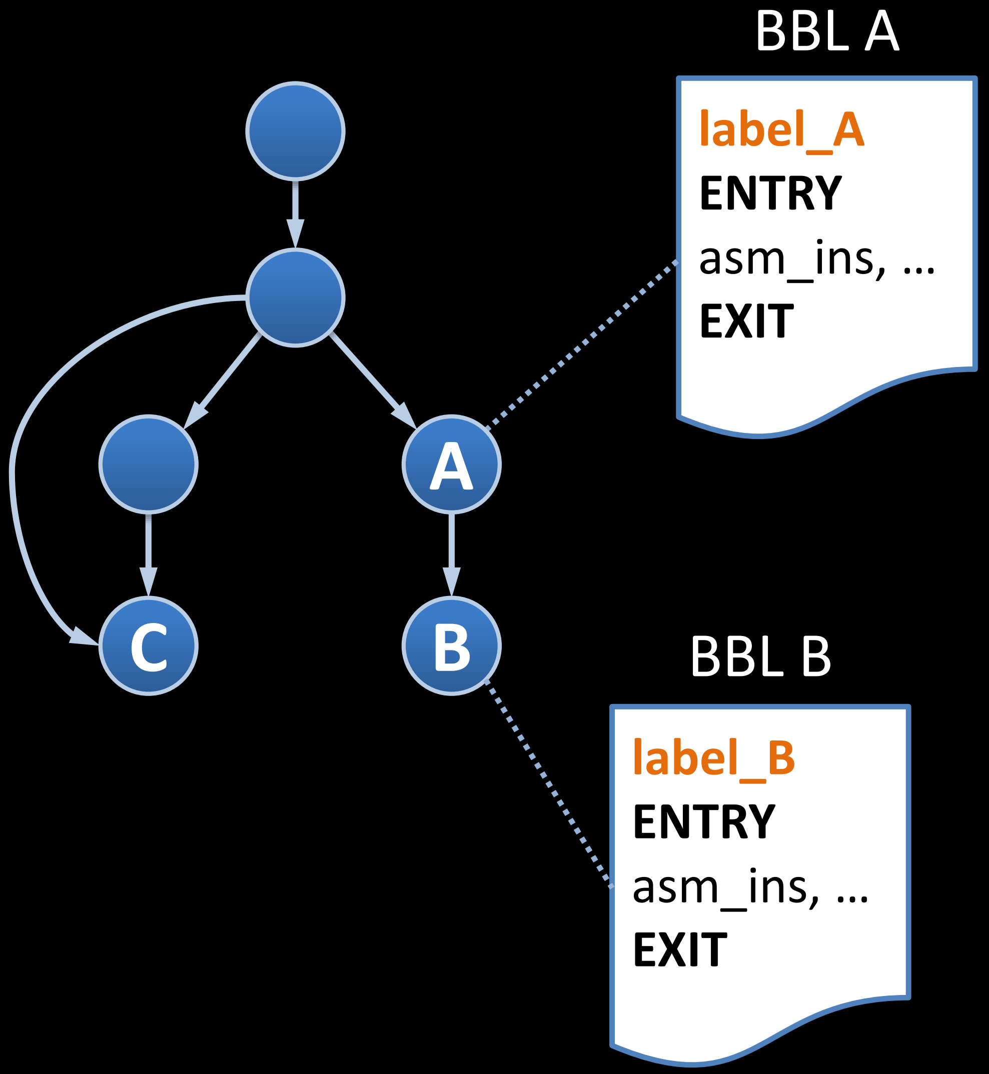


ENTRY
asm_ins, …
EXIT

label_B
ENTRY
asm_ins, …
EXIT

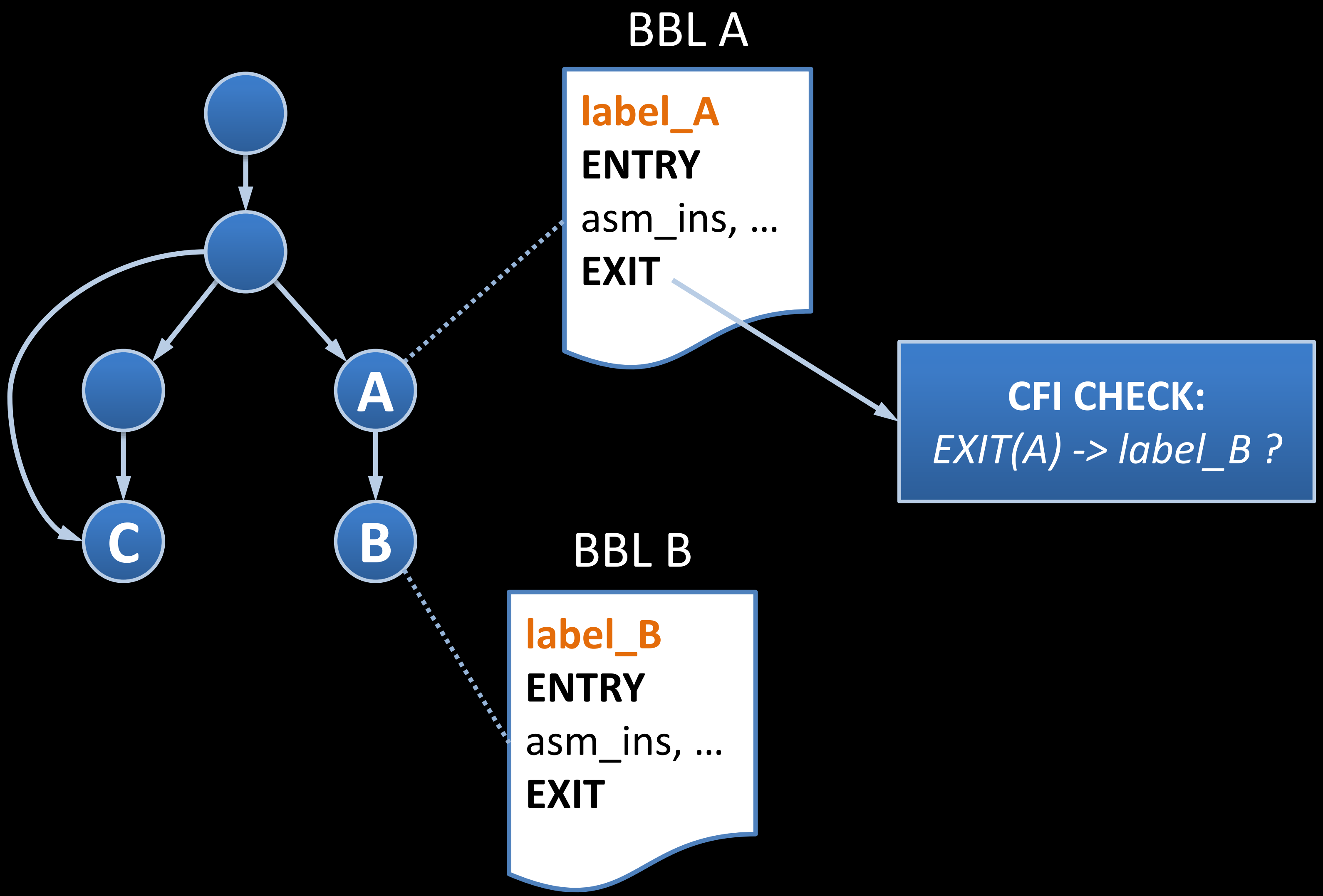
Original CFI Label Checking

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



Original CFI Label Checking

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



BBL A

label_A
ENTRY
asm_ins, …
EXIT

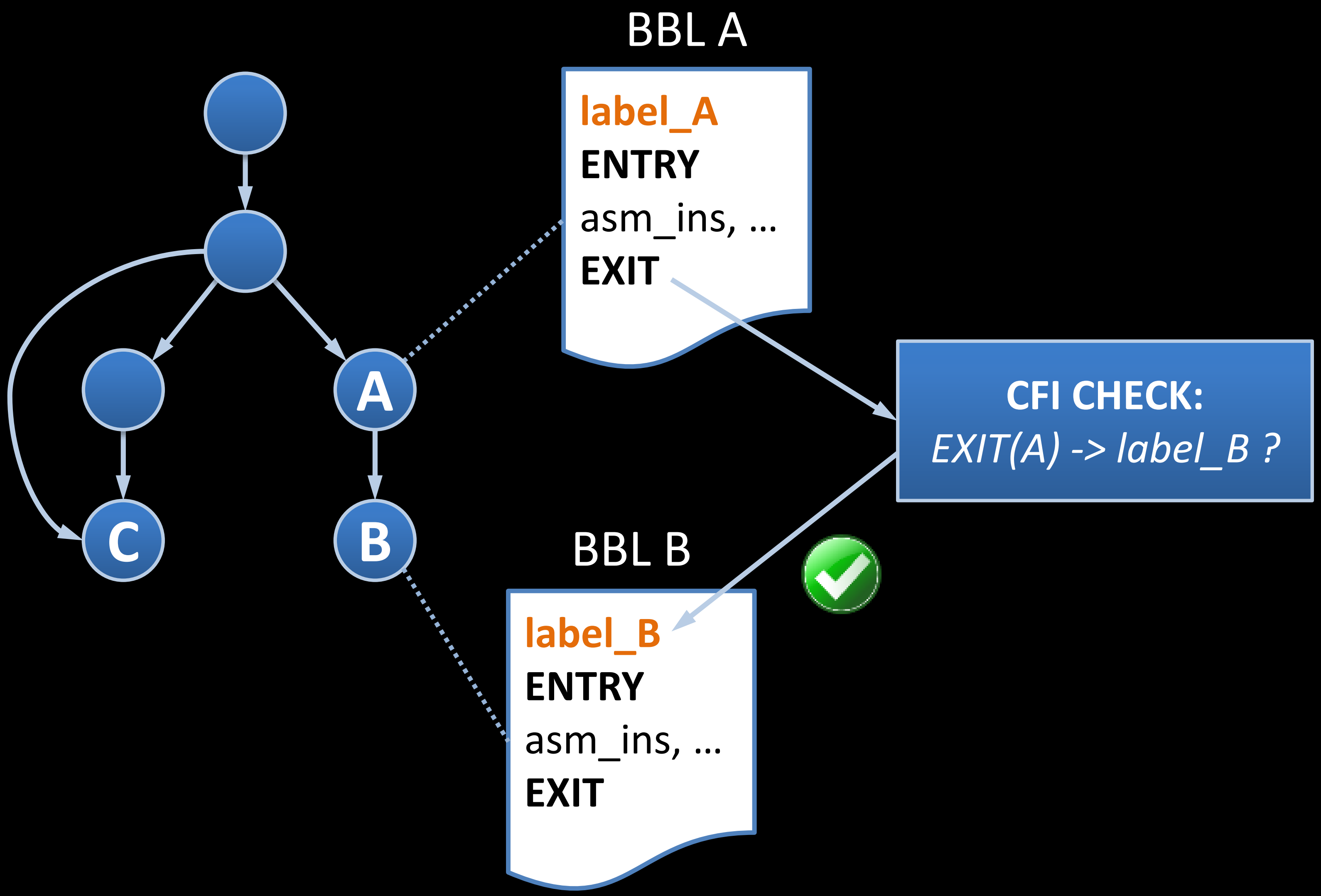
CFI CHECK:
EXIT(A) -> label_B ?

BBL B

label_B
ENTRY
asm_ins, …
EXIT

Original CFI Label Checking

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



BBL A

label_A
ENTRY
asm_ins, …
EXIT

CFI CHECK:
EXIT(A) -> label_B ?

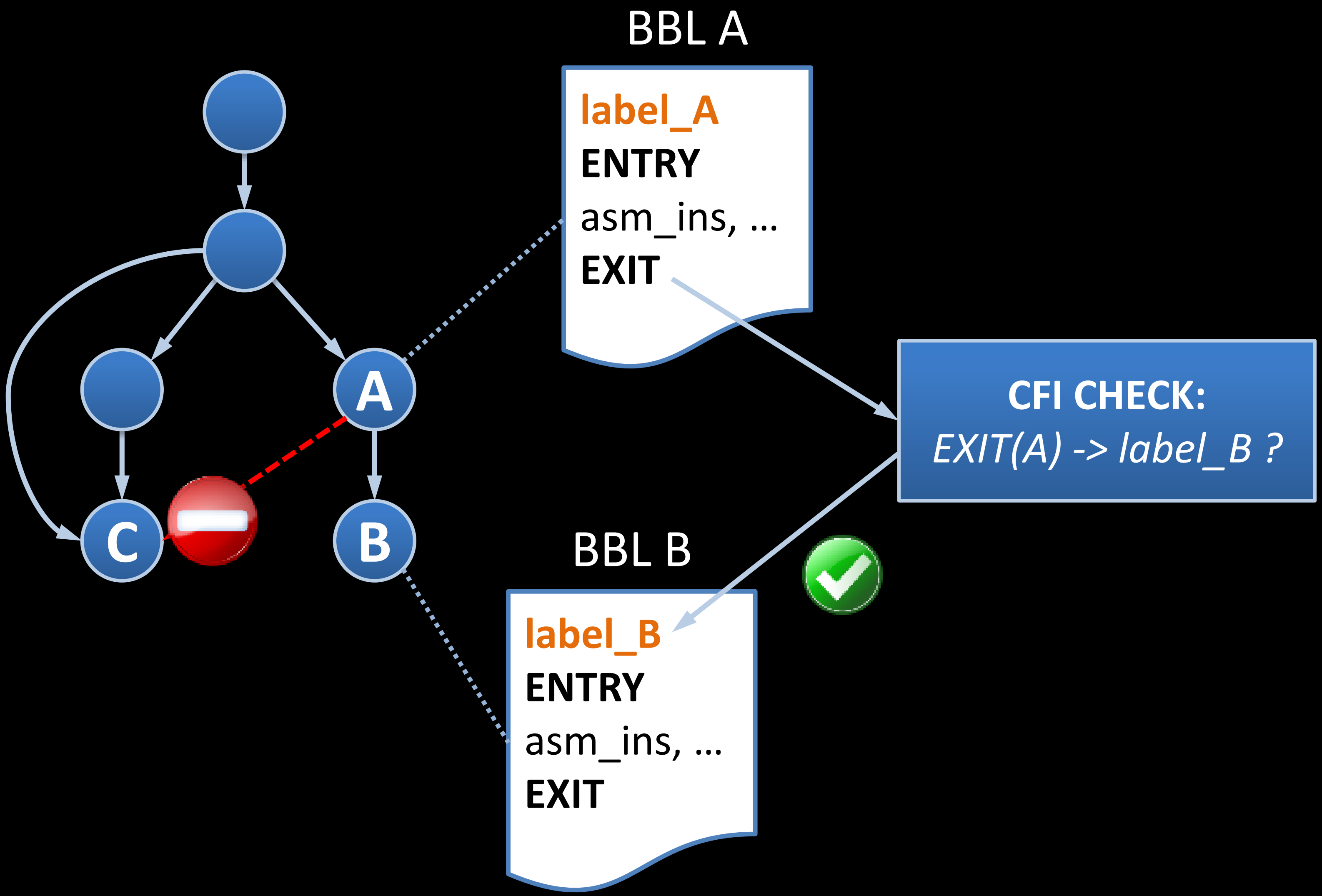
✓

BBL B

label_B
ENTRY
asm_ins, …
EXIT

Original CFI Label Checking

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



BBL A

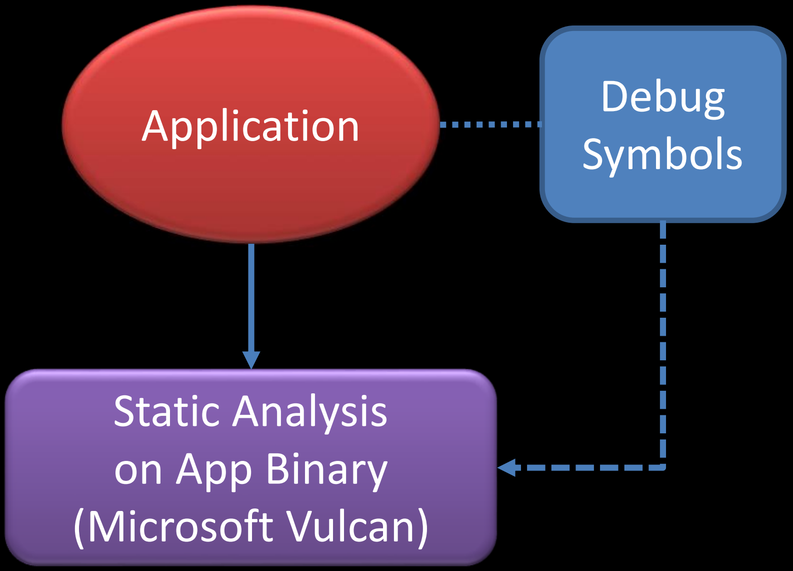
label_A
ENTRY
asm_ins, …
EXIT

CFI CHECK:
EXIT(A) -> label_B ?

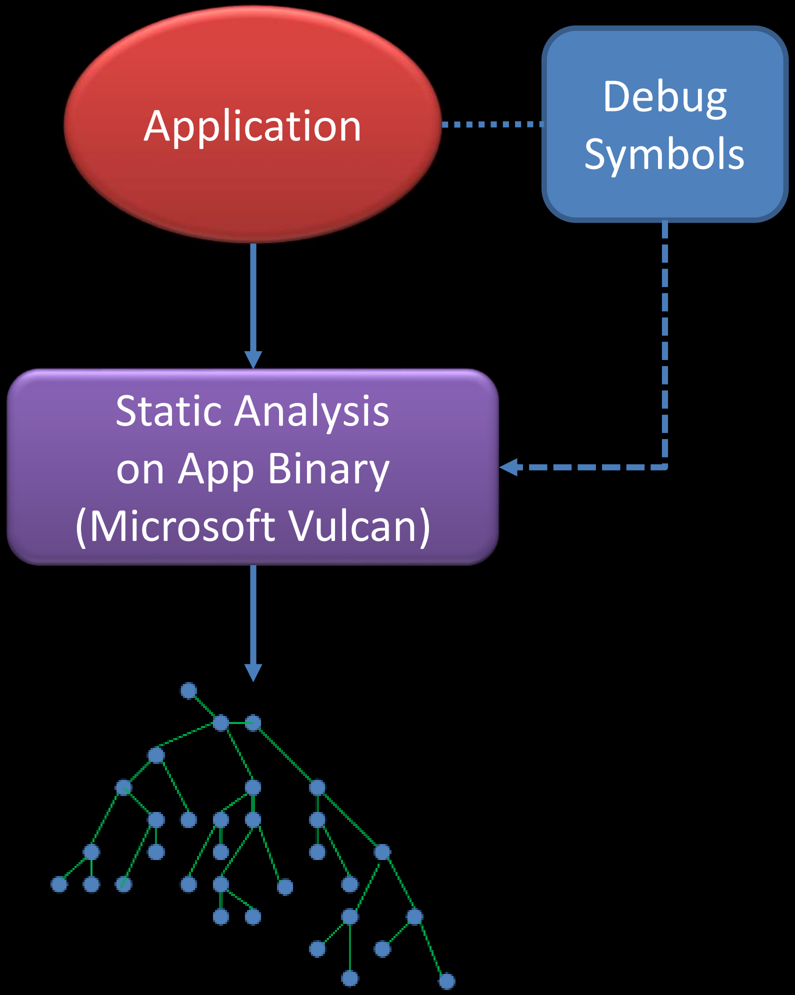
BBL B

label_B
ENTRY
asm_ins, …
EXIT

CFI Instrumentation Workflow



CFI Instrumentation Workflow



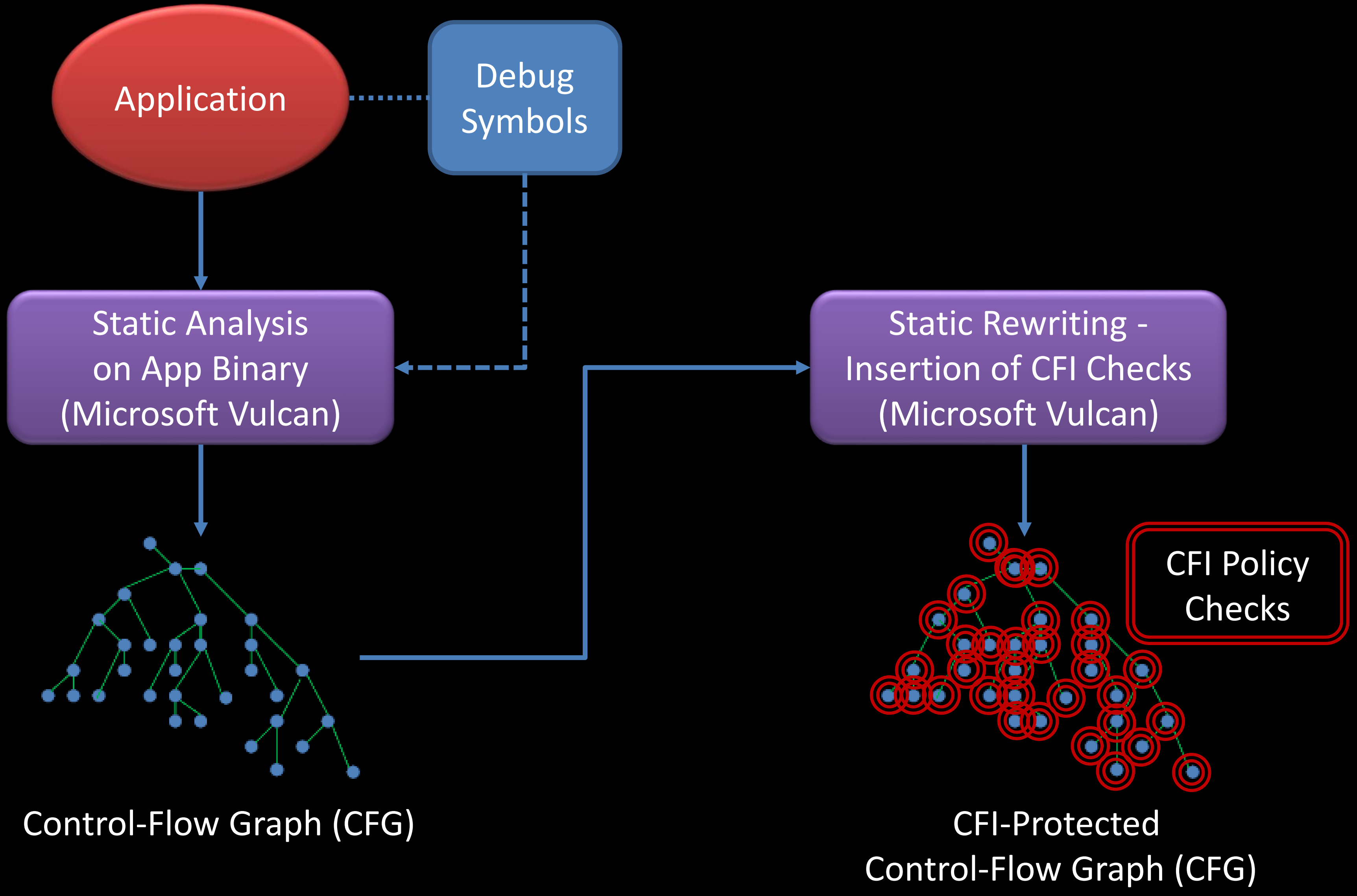
Application

Debug Symbols

Static Analysis on App Binary (Microsoft Vulcan)

Control-Flow Graph (CFG)

CFI Instrumentation Workflow


Application
Debug Symbols
Static Analysis on App Binary (Microsoft Vulcan)
Static Rewriting - Insertion of CFI Checks (Microsoft Vulcan)
CFI Policy Checks
Control-Flow Graph (CFG)
CFI-Protected Control-Flow Graph (CFG)

Which Instructions to Protect?

Returns

- **Purpose**: Return to calling function
- **CFI Relevance**: Return address located on stack

Indirect Jumps

- **Purpose**: switch tables, dispatch to library functions
- **CFI Relevance**: Target address taken from either processor register or memory

Indirect Calls

- **Purpose**: call through function pointer, virtual table calls
- **CFI Relevance**: Target address taken from either processor register or memory

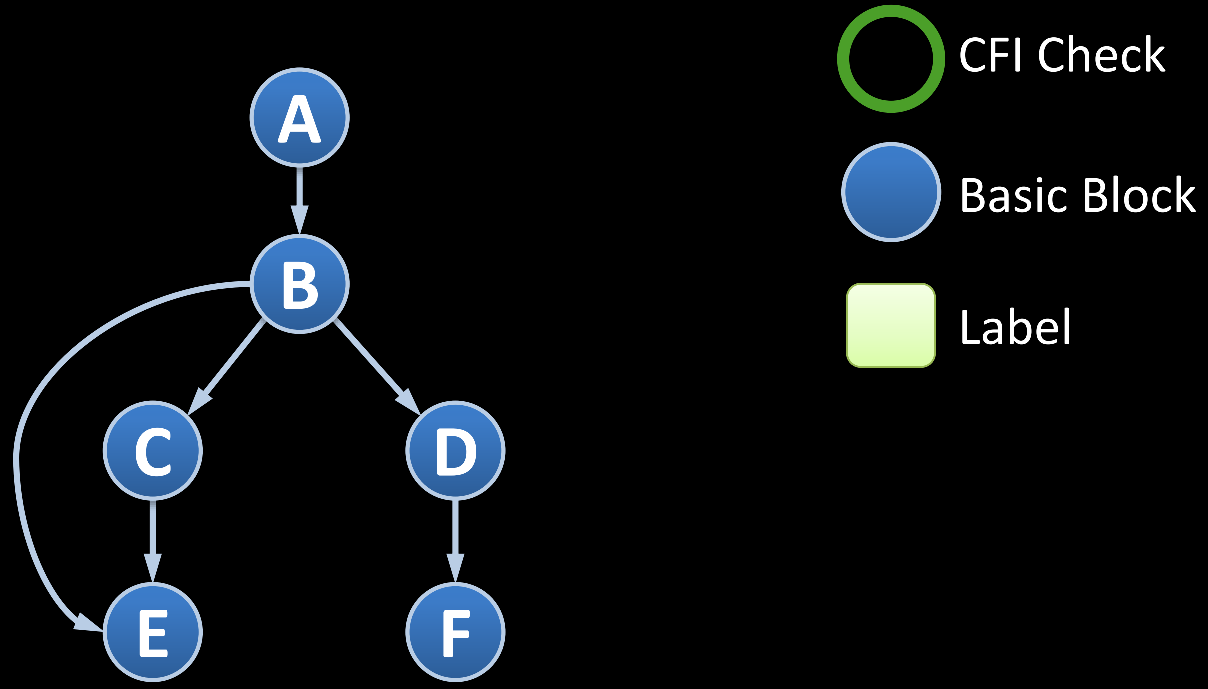
Challenges

Performance

Control-Flow Graph Analysis and Coverage

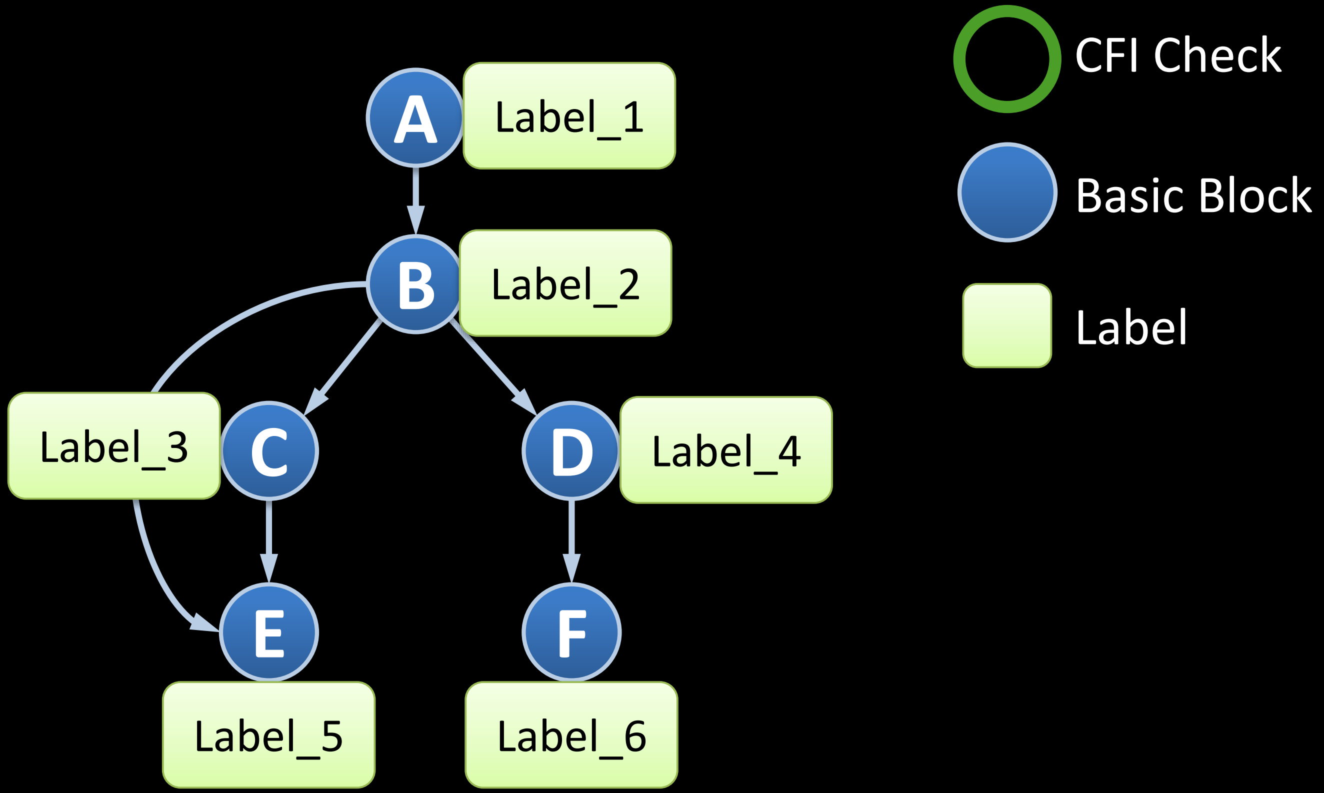
Label Granularity: Trade-Offs (1/2)

- Many CFI checks are required if unique labels are assigned per node



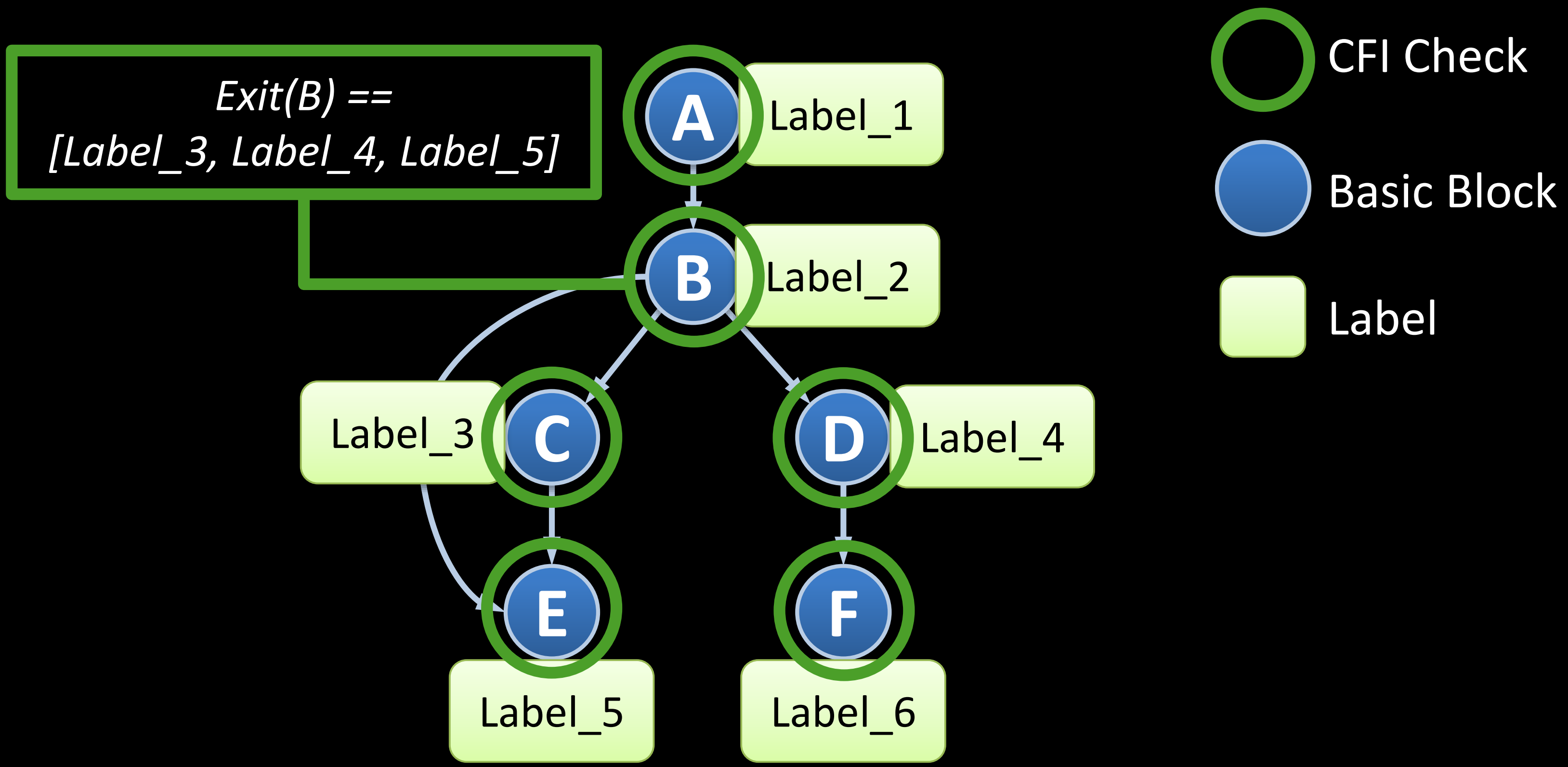
Label Granularity: Trade-Offs (1/2)

- Many CFI checks are required if unique labels are assigned per node



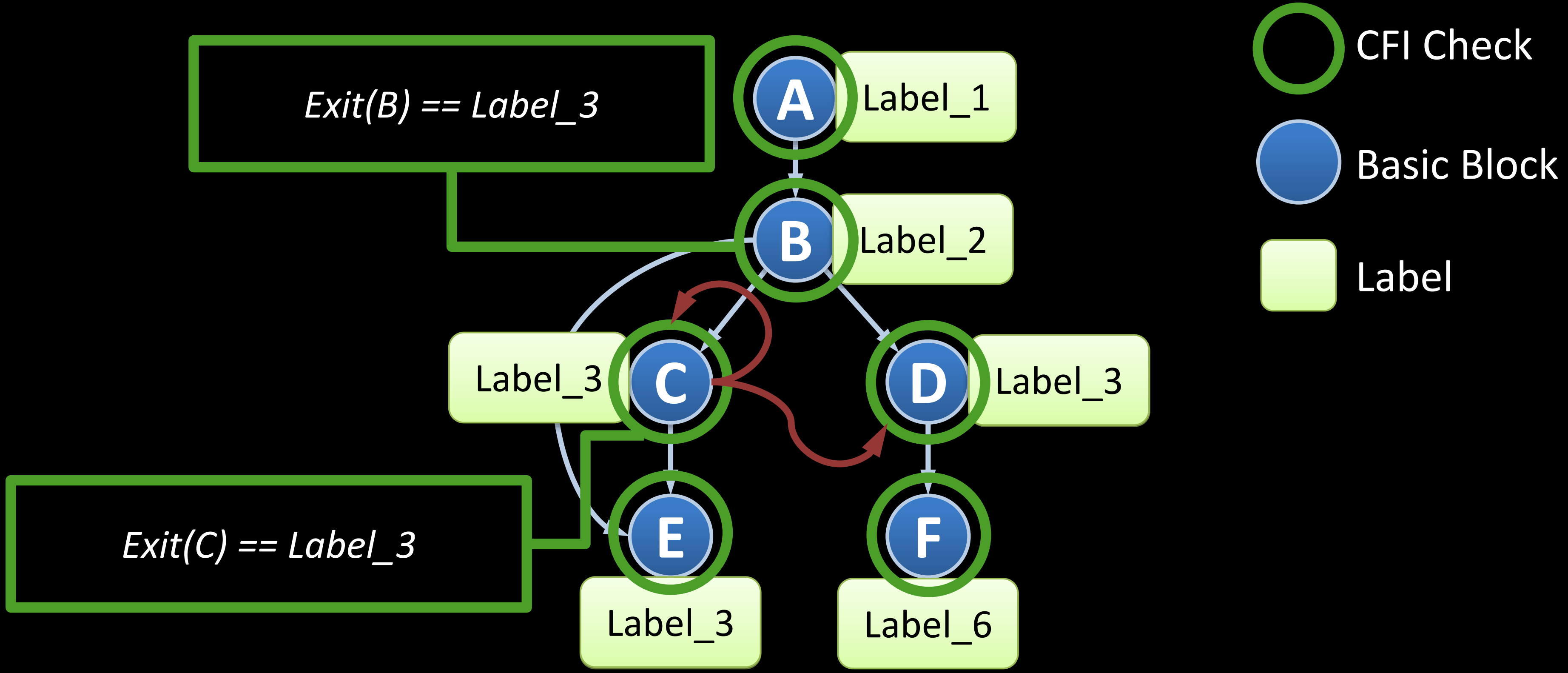
Label Granularity: Trade-Offs (1/2)

- Many CFI checks are required if unique labels are assigned per node



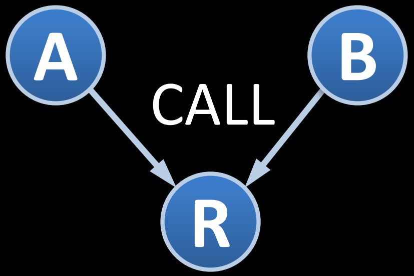
Label Granularity: Trade-Offs (2/2)

- Optimization step: Merge labels to allow single CFI check
- However, this allows for unintended control-flow paths



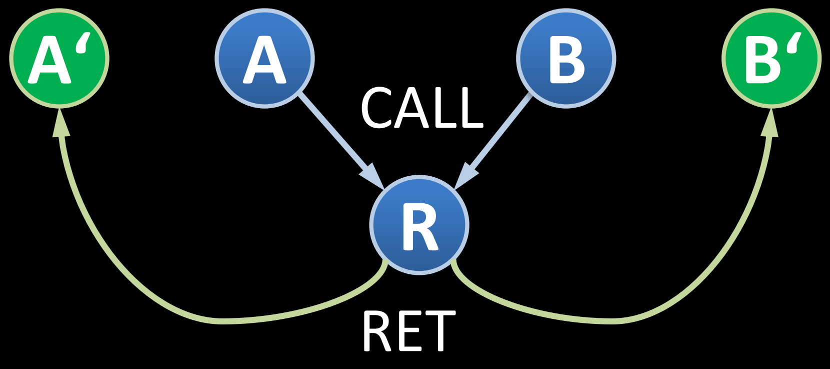
Label Problem for Returns

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



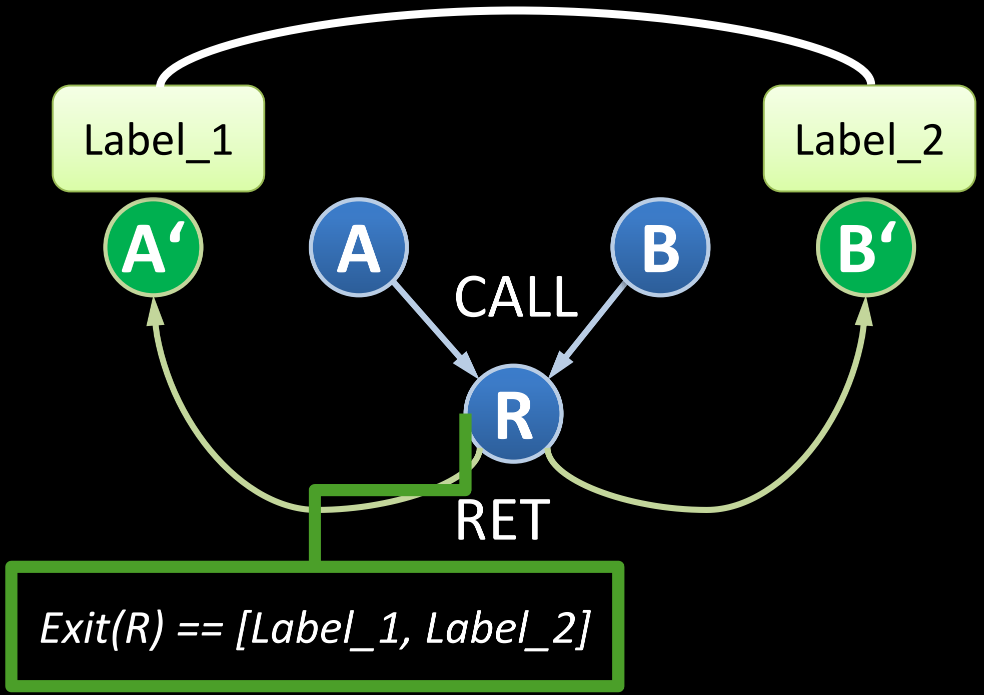
Label Problem for Returns

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



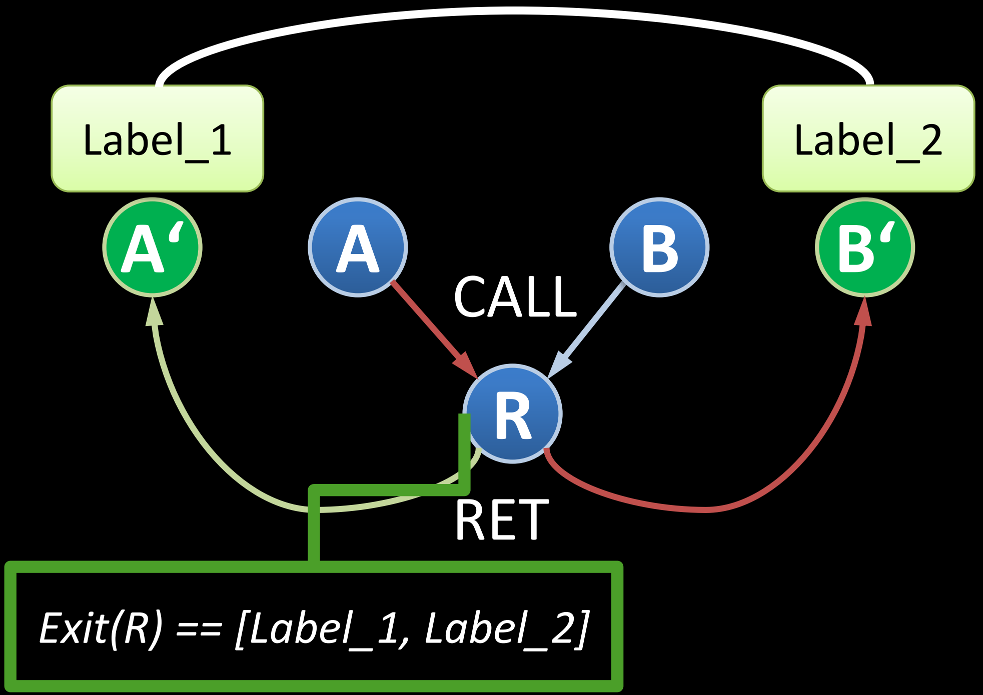
Label Problem for Returns

- 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

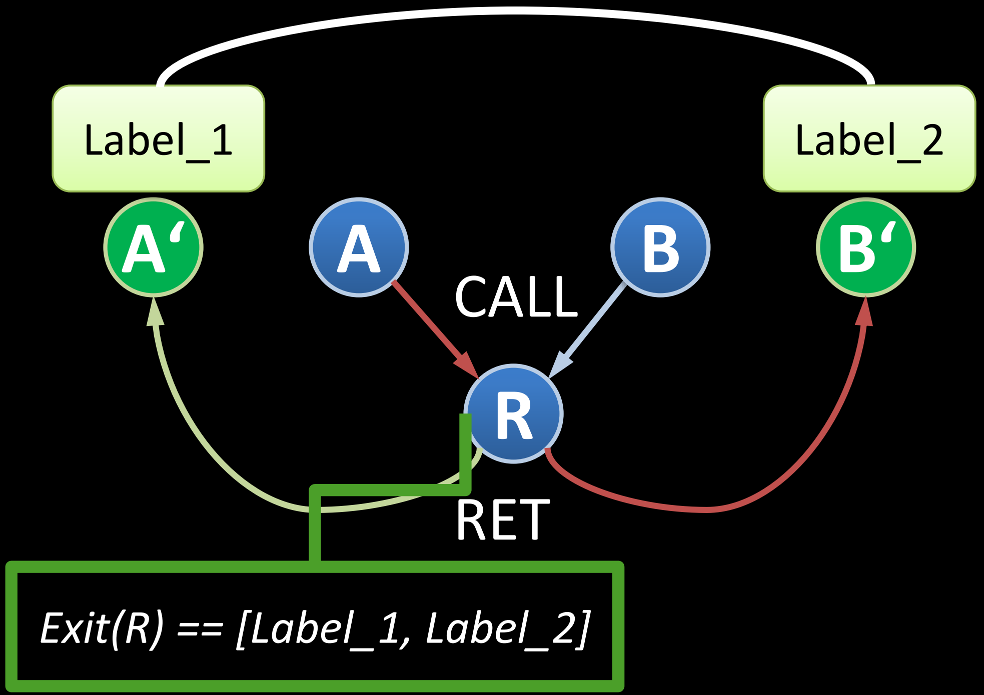


Exit(R) == [Label_1, Label_2]

Label Problem for Returns

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

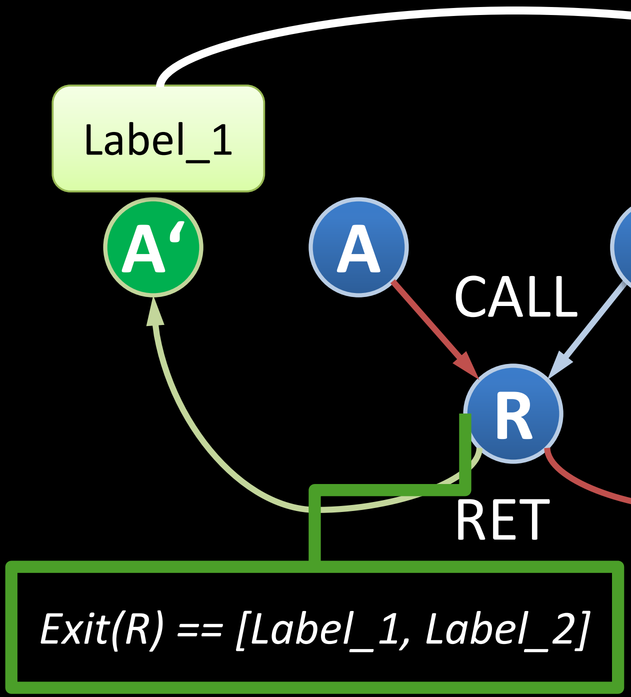
- Shadow stack allows for fine-grained return address protection but incurs higher overhead



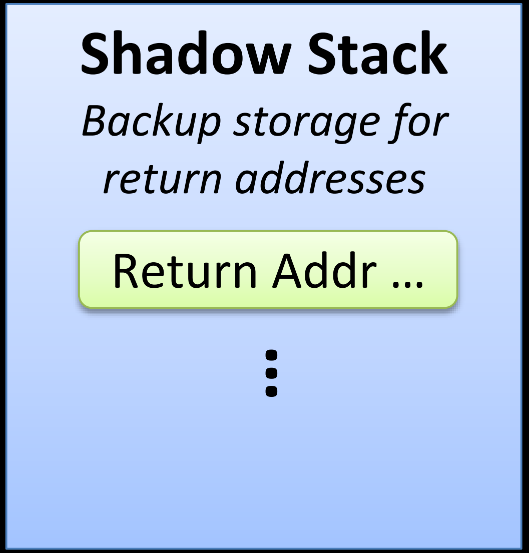
Exit(R) == [Label_1, Label_2]

Label Problem for Returns

- Static CFI label checking leads to coarse-grained protection for returns
- Shadow stack allows for fine-grained return address protection but incurs higher overhead



Exit(R) == [Label_1, Label_2]


Shadow Stack

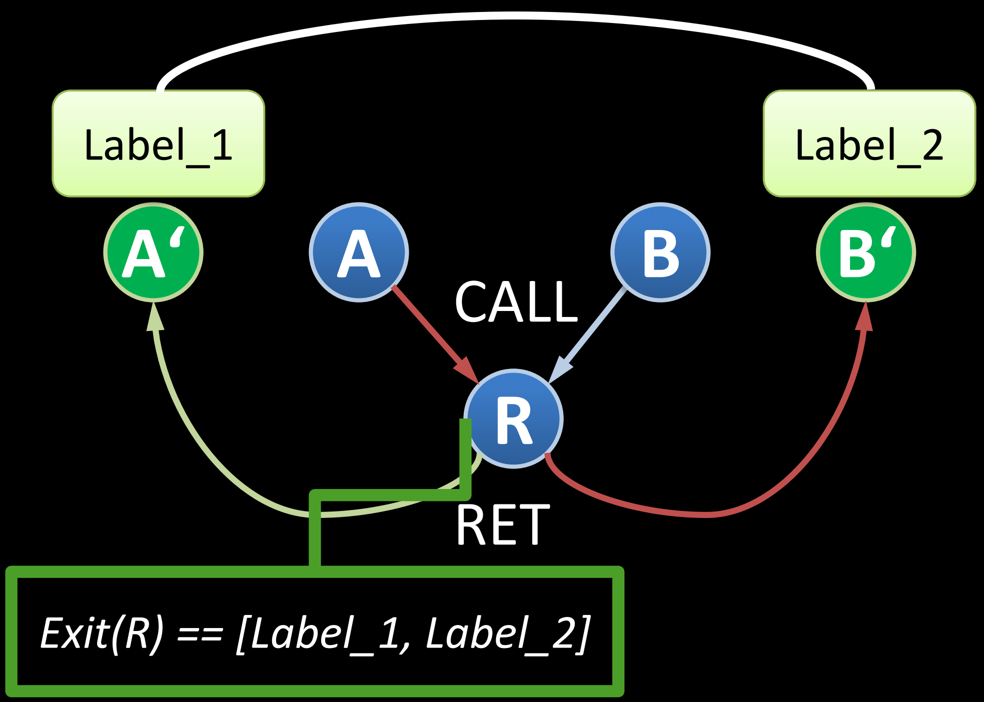
Backup storage for return addresses

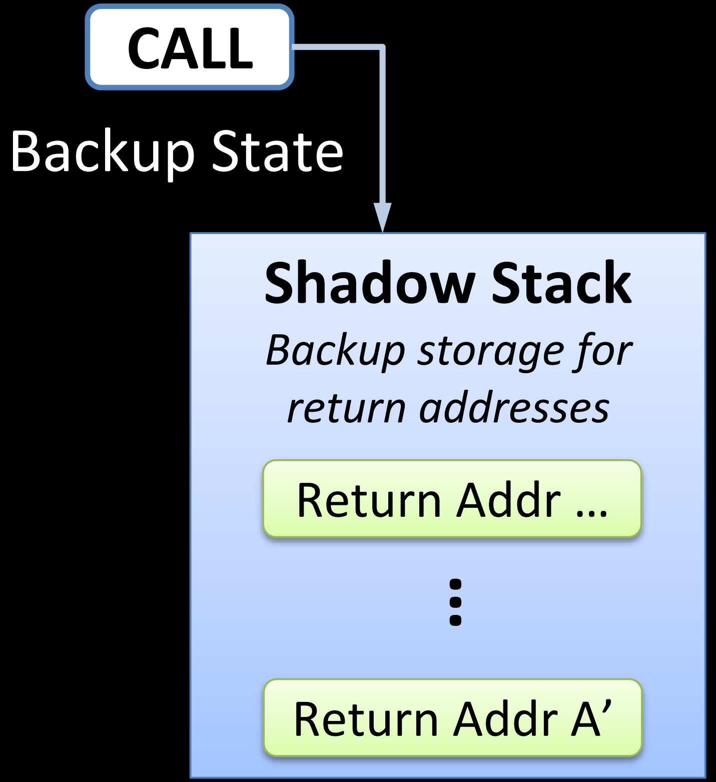
Return Addr …

⋮

Label Problem for Returns

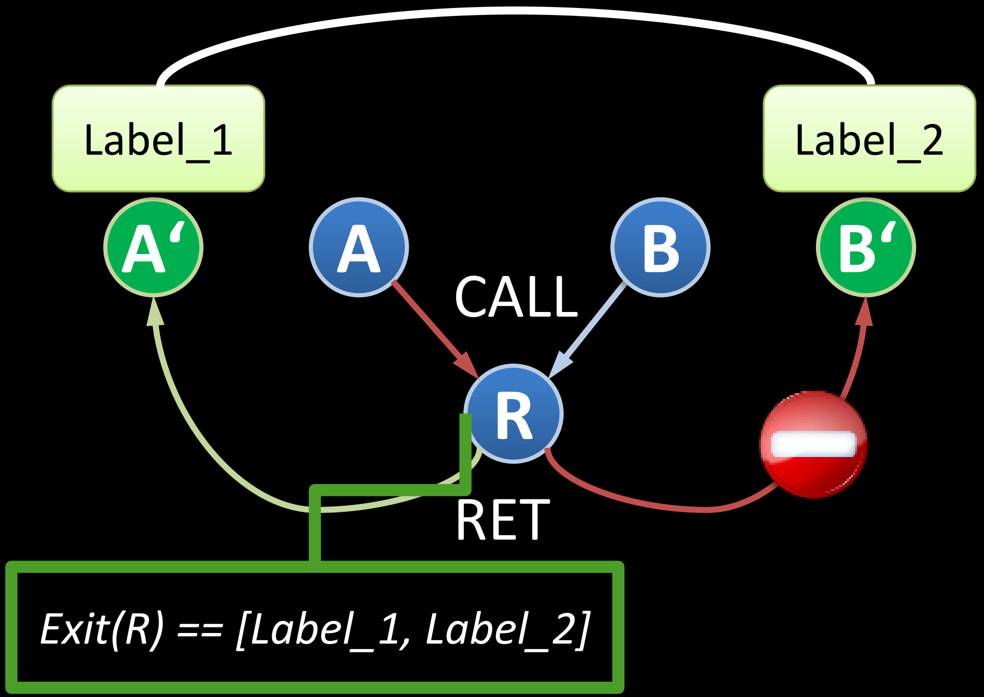
- Static CFI label checking leads to coarse-grained protection for returns
- Shadow stack allows for fine-grained return address protection but incurs higher overhead

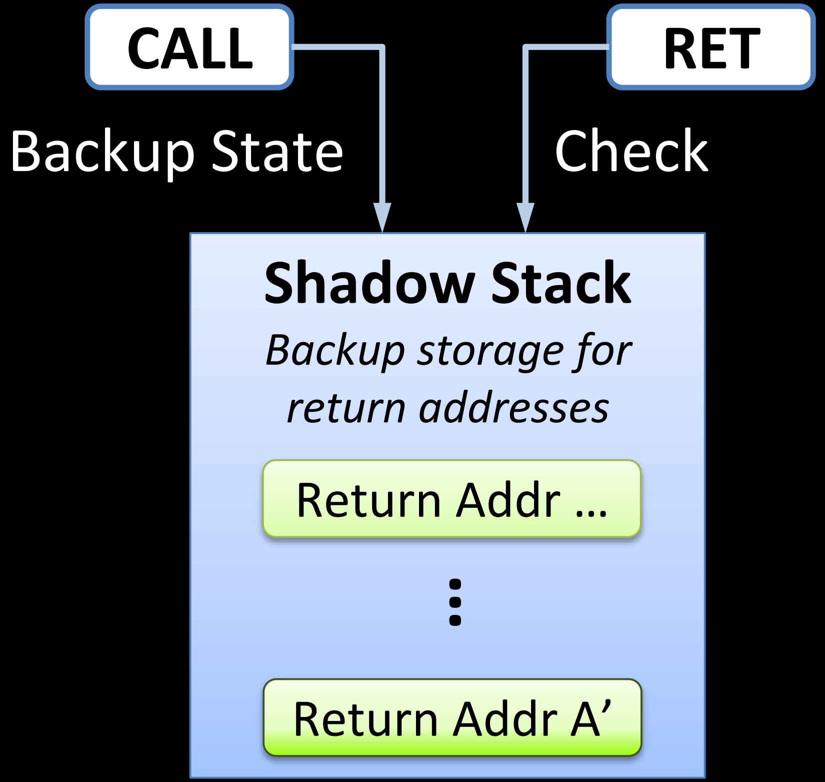




Label Problem for Returns

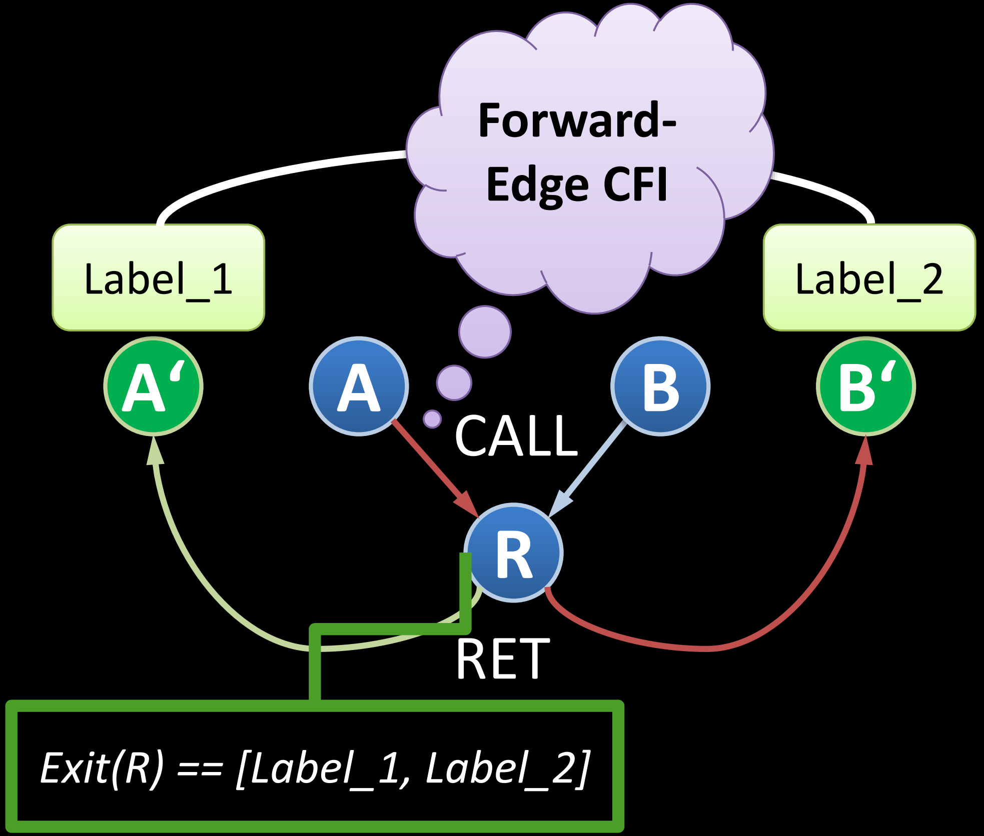
- Static CFI label checking leads to coarse-grained protection for returns
- Shadow stack allows for fine-grained return address protection but incurs higher overhead





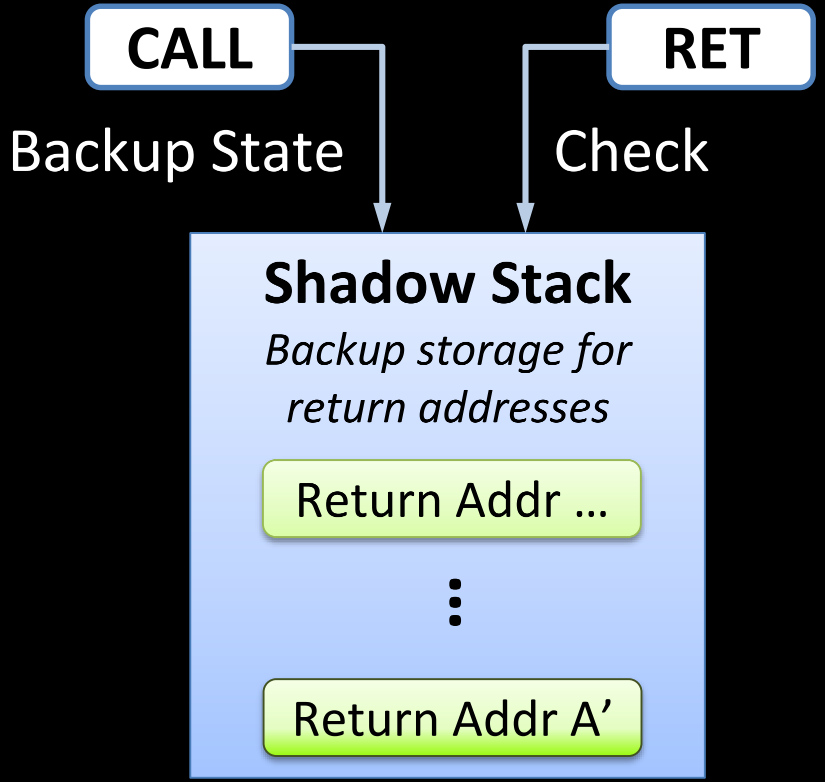
Label Problem for Returns

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



Exit(R) == [Label_1, Label_2]

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



Shadow Stack

Backup storage for return addresses

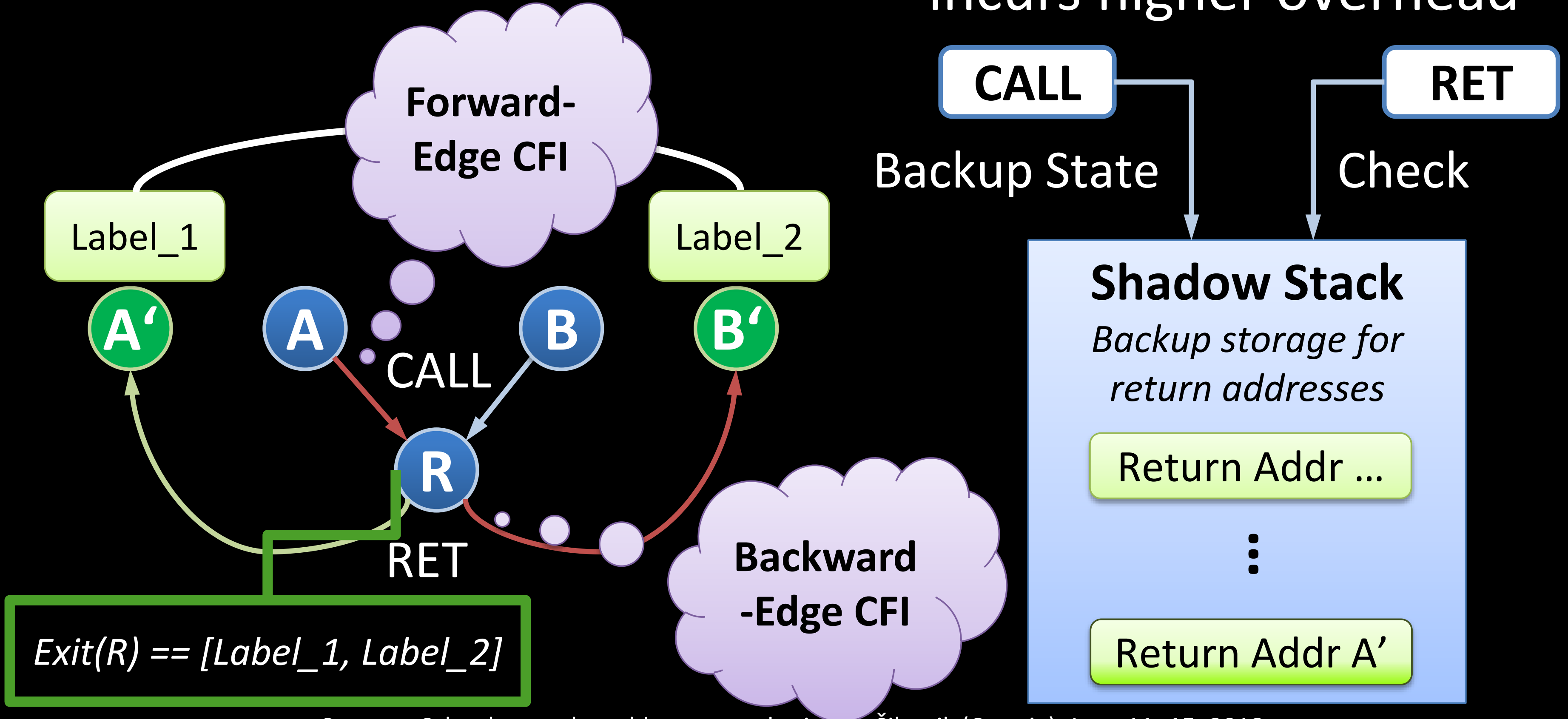
Return Addr …

⋮

Return Addr A’

Label Problem for Returns

- Static CFI label checking leads to coarse-grained protection for returns
- Shadow stack allows for fine-grained return address protection but incurs higher overhead



Forward- vs. Backward-Edge

- Some CFI schemes consider only forward-edge CFI
  - Google's VTV and IFCC [Tice et al., USENIX Sec 2015]
  - SAFEDISPATCH [Jang et al., NDSS 2014]
  - And many more: TVIP, VTint, vfguard
- Assumption: Backward-edge CFI through stack protection
- Problems of stack protections:
  - Stack Canaries: memory disclosure of canary
  - ASLR (base address randomization of stack): memory disclosure of base address
  - Variable reordering (memory disclosure)

StackDefiler

Protecting Stack is Hard!

Losing Control:
On the Effectiveness of Control-Flow Integrity under Stack Attacks
ACM CCS 2015
Christopher Liebchen, Marco Negro, Per Larsen, Lucas Davi, Ahmad-Reza Sadeghi, Stephen Crane, Mohaned Qunaibit, Michael Franz, Mauro Conti

	Randomization	CFI – Returns	CFI – Ind. Calls	CFI – Ind. Jumps	Code Ptr. Integrity	Code Ptr. Hiding	Implementation
CFI [Abadi et al, CCS 2005]		Shadow Stack	CFG	CFG			Binary
DFI [Costa et al, OSDI 2006]					●		Compiler
WIT [Akritidis et al, IEEE S&P 2008]					●		Compiler
MoCFI [with Davi et al, NDSS 2012]		Shadow Stack	CFG-relaxed	CFG			Binary
ORP [Pappas et al, IEEE S&P 2012]	●						Binary
ILR [Hiser et al, IEEE S&P 2012]	●						Binary
STIR [Wartell et al, CCS 2013]	●						Binary
Xifer [with Davi et al, AsiaCCS 2013]	●						Binary
CCFIR [Zhang et al, IEEE S&P 2013]	●	Call Site	CFG-relaxed	CFG-relaxed			Binary
binCFI [Zhang et al, USENIX Sec 2013]		Call Site	CFG-relaxed	CFG-relaxed			Binary
kBouncer [Pappas et al, USENIX Sec 2013]		Call Site	Sequence Length	Sequence Length			Kernel/HW
ROPecker [Zheng et al, NDSS 2013]		Sequence Length	Sequence Length	Sequence Length			Kernel/HW
Oxymoron [Backes et al, USENIX Sec 2014]	●					●	Kernel
XnR [Backes et al, CCS 2014]	●					●	Kernel
Forward-Edge CFI [Tice et al, USENIX Sec 2014]			vtable/IFCC				Compiler
SAFEDISPATCH [Jang et al., NDSS 2014]			vtable				Compiler
CPI [Kuznetsov et al., OSDI 2014]					●		Compiler
Microsoft EMET / ROPGuard		Call Site					Kernel
HW-SW Co-Design [with Davi et al, DAC 2014]		Active Call Site	Function Entry	Sequence Length			Compiler/HW
Isomeron [Davi et al, NDSS 2015]	●					Returns	Binary
Readactor [Crane et al, IEEE S&P 2015]	●					●	Compiler
Opaque-CFI [Larsen et al, NDSS 2015]	●	Range Checks	Range Checks	Range Checks			Binary

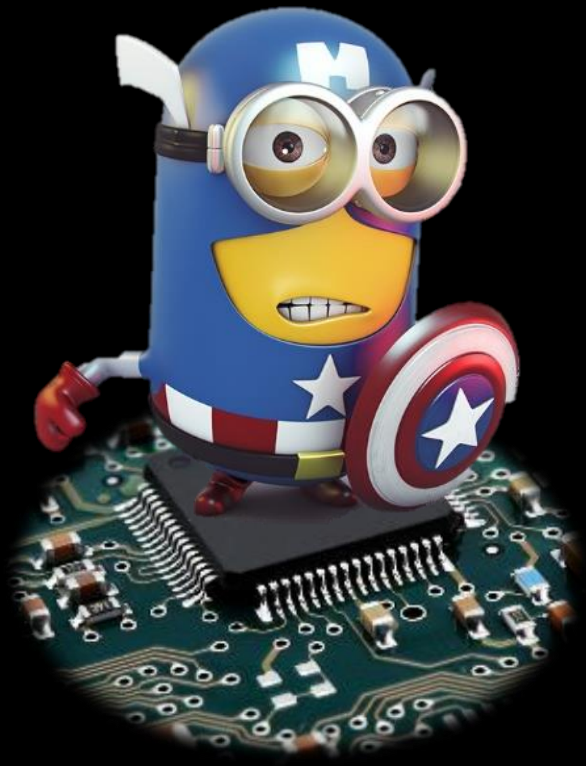
Bypassing (Coarse-grained) CFI

Stitching the Gadgets
USENIX Security 2014
Lucas Davi, Daniel Lehmann,
Ahmad-Reza Sadeghi, Fabian Monrose

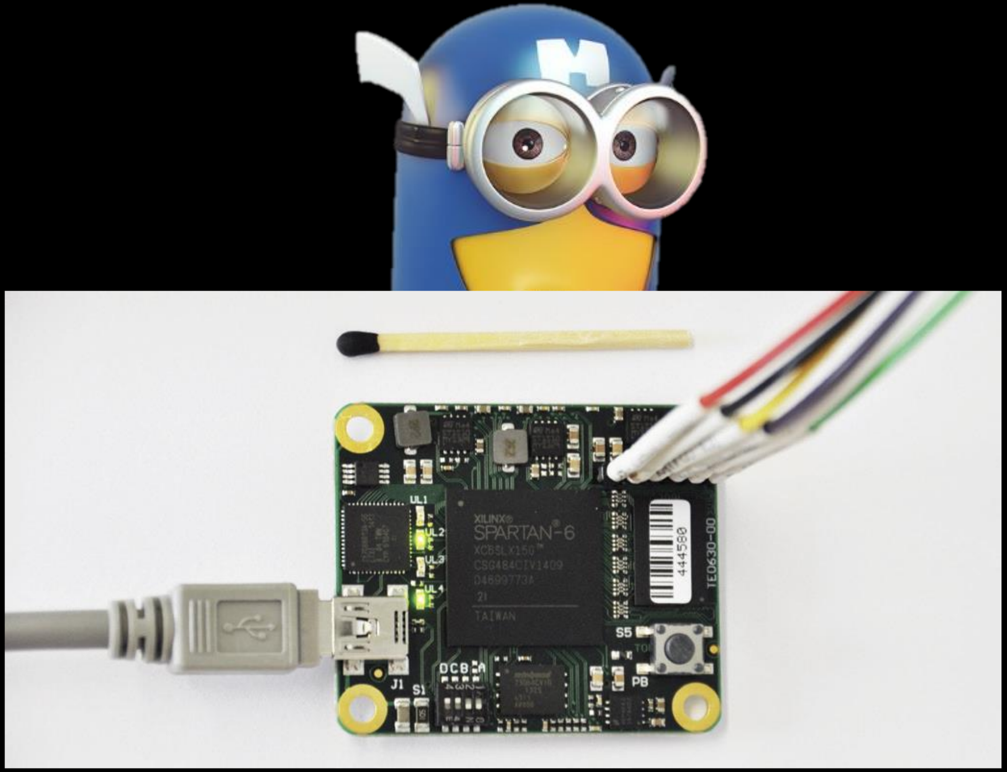
COOP
IEEE S&P 2015
Felix Schuster, Thomas Tendyck,
Christopher Liebchen, Lucas Davi,
Ahmad-Reza Sadeghi, Thorsten Holz

Guarding the Guard

Hardware CFI



HAFIX and HAFIX++



HAFIX:

Hardware-Assisted Flow Integrity Extension

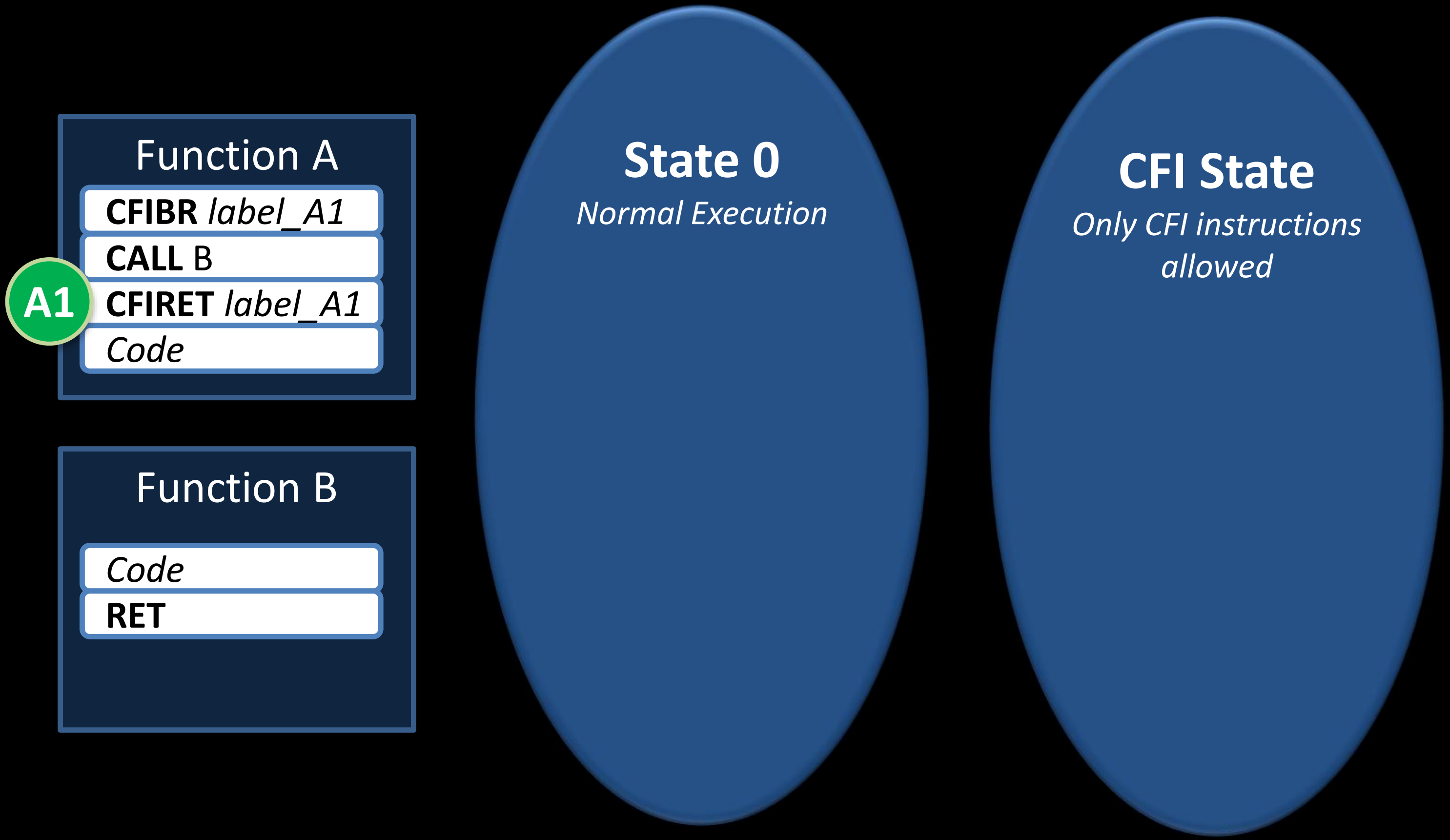
Design Automation Conference (DAC 2015), ***Best Paper Award***

Orlando Arias, Lucas Davi, Matthias Hanreich, Yier Jin, Patrick Koeberl, Debayan Paul, Ahmad-Reza Sadeghi, Dean Sullivan

Objectives

Backward-Edge and Forward-Edge CFI	Stateful, CFI policy agnostic
No burden on developer	No code annotations/changes
Security	Hardware protection On-Chip Memory for CFI Data No unintended sequences
High performance	< 3% overhead
Enabling technology	All applications can use CFI features Support of Multitasking
Compatibility to legacy code	CFI and non-CFI code on same platform

Function Return Policy



Function A

CFIBR *label_A1*

CALL B

A1 **CFIRET** *label_A1*

Code

Function B

Code

RET

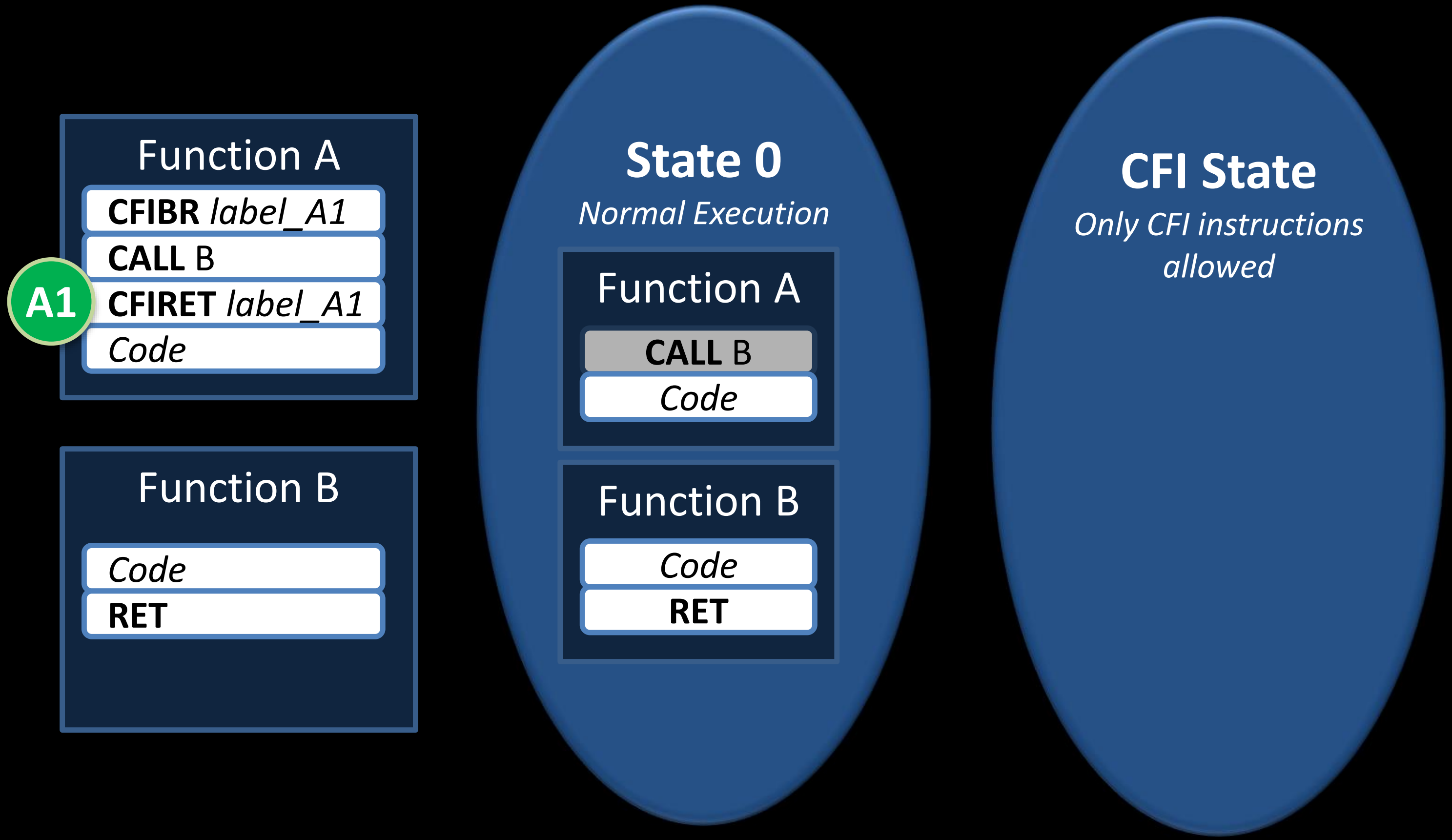
State 0

Normal Execution

CFI State

Only CFI instructions allowed

Function Return Policy



Function A

CFIBR *label_A1*

CALL B

A1

CFIRET *label_A1*

Code

Function B

Code

RET

State 0

Normal Execution

Function A

CALL B

Code

Function B

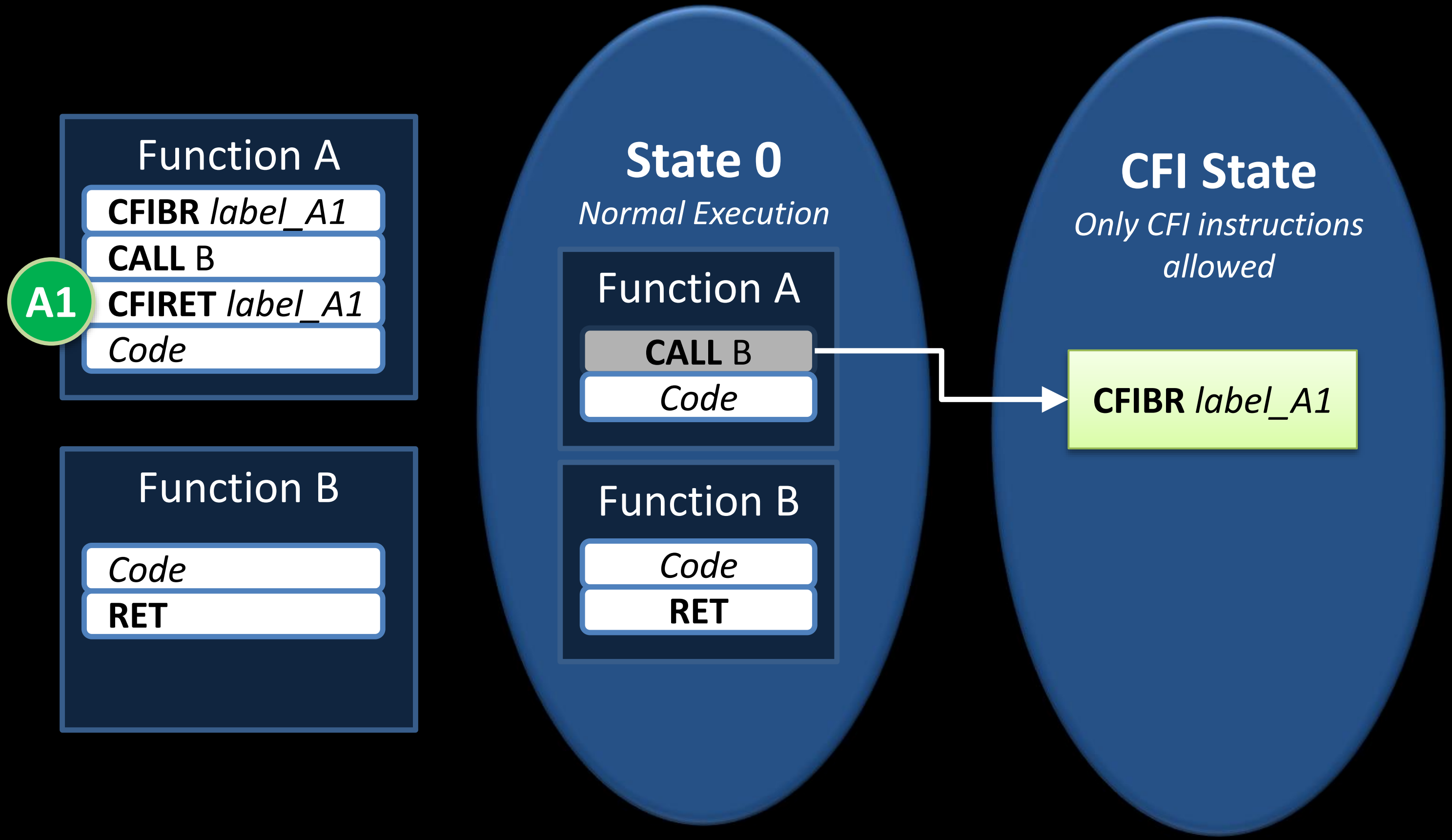
Code

RET

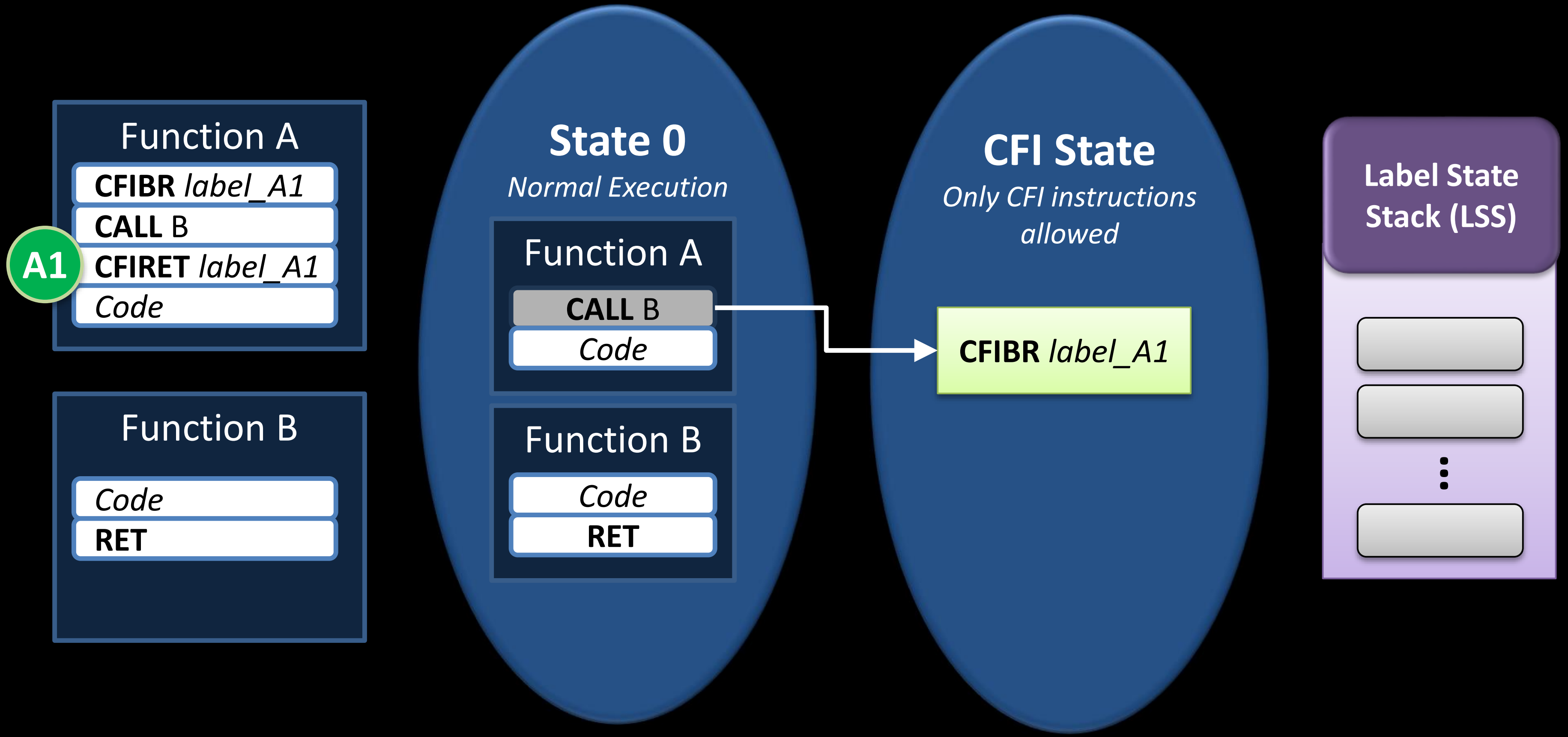
CFI State

Only CFI instructions allowed

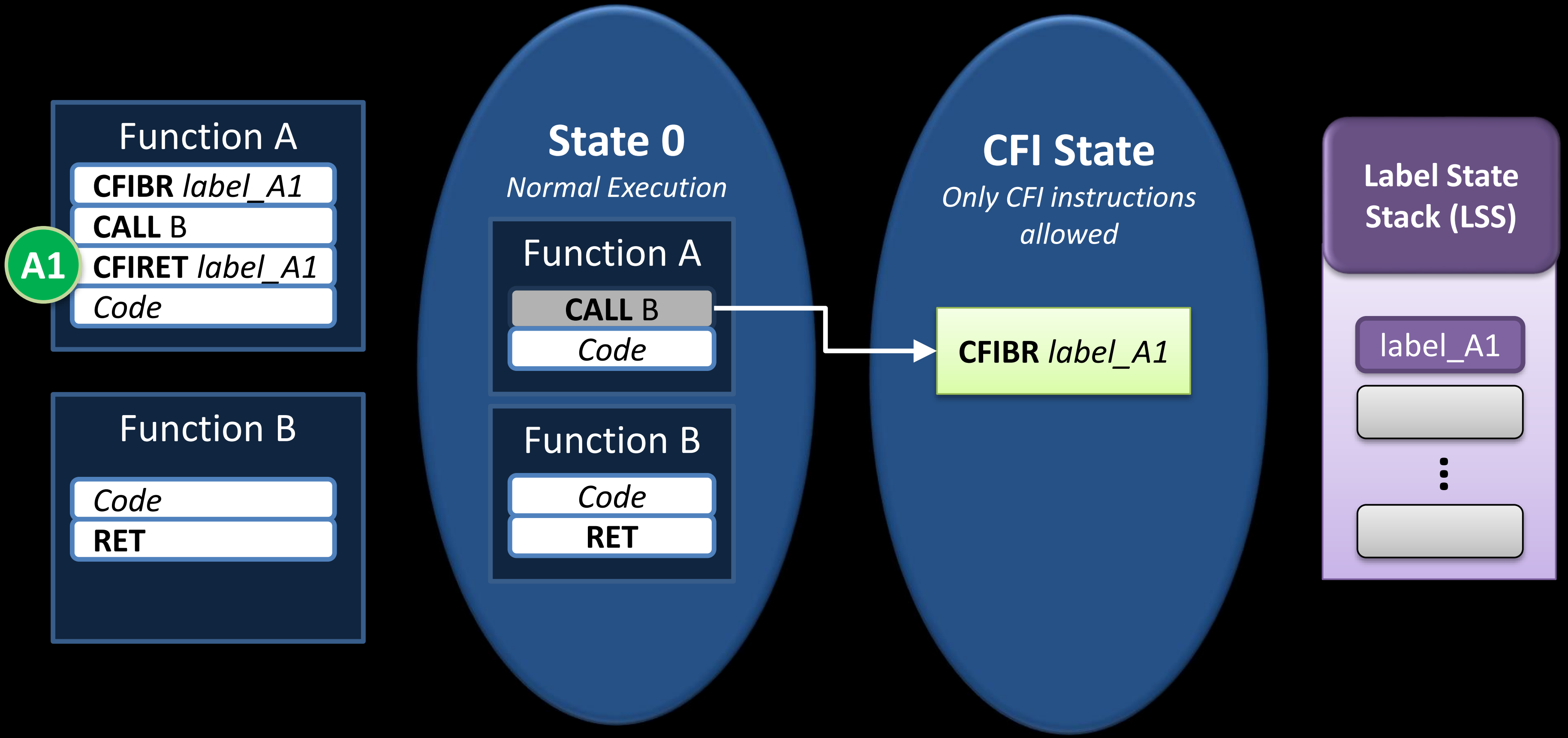
Function Return Policy



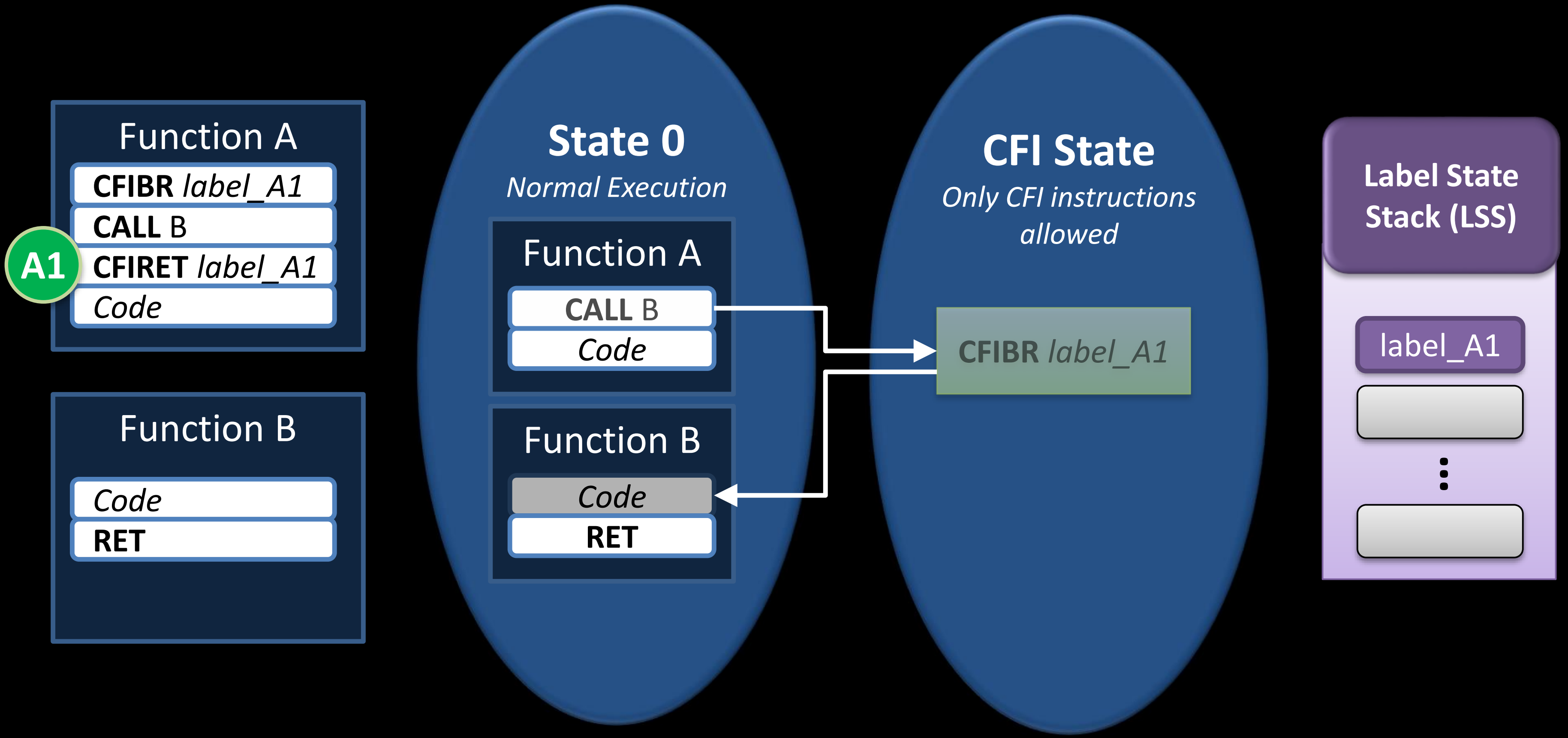
Function Return Policy



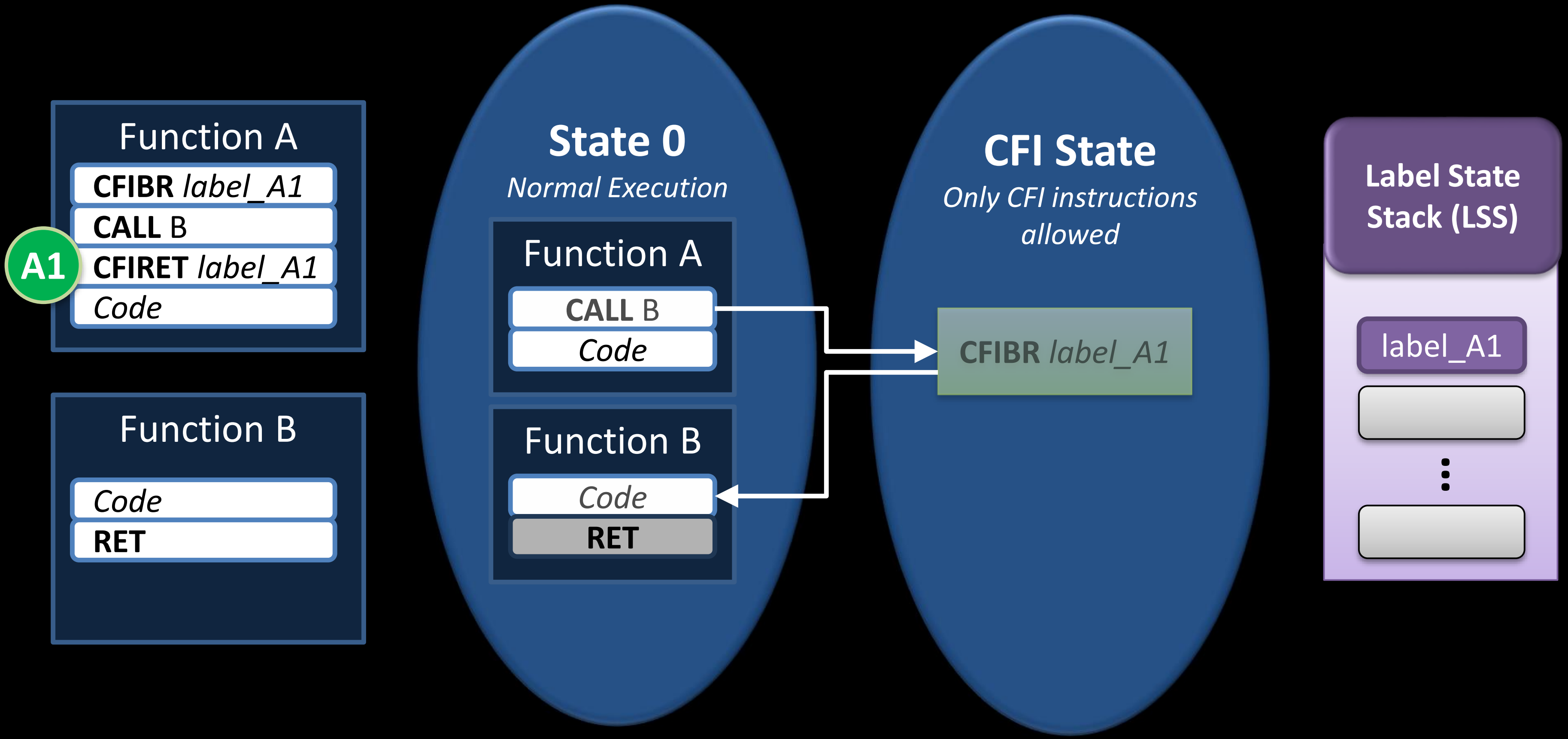
Function Return Policy



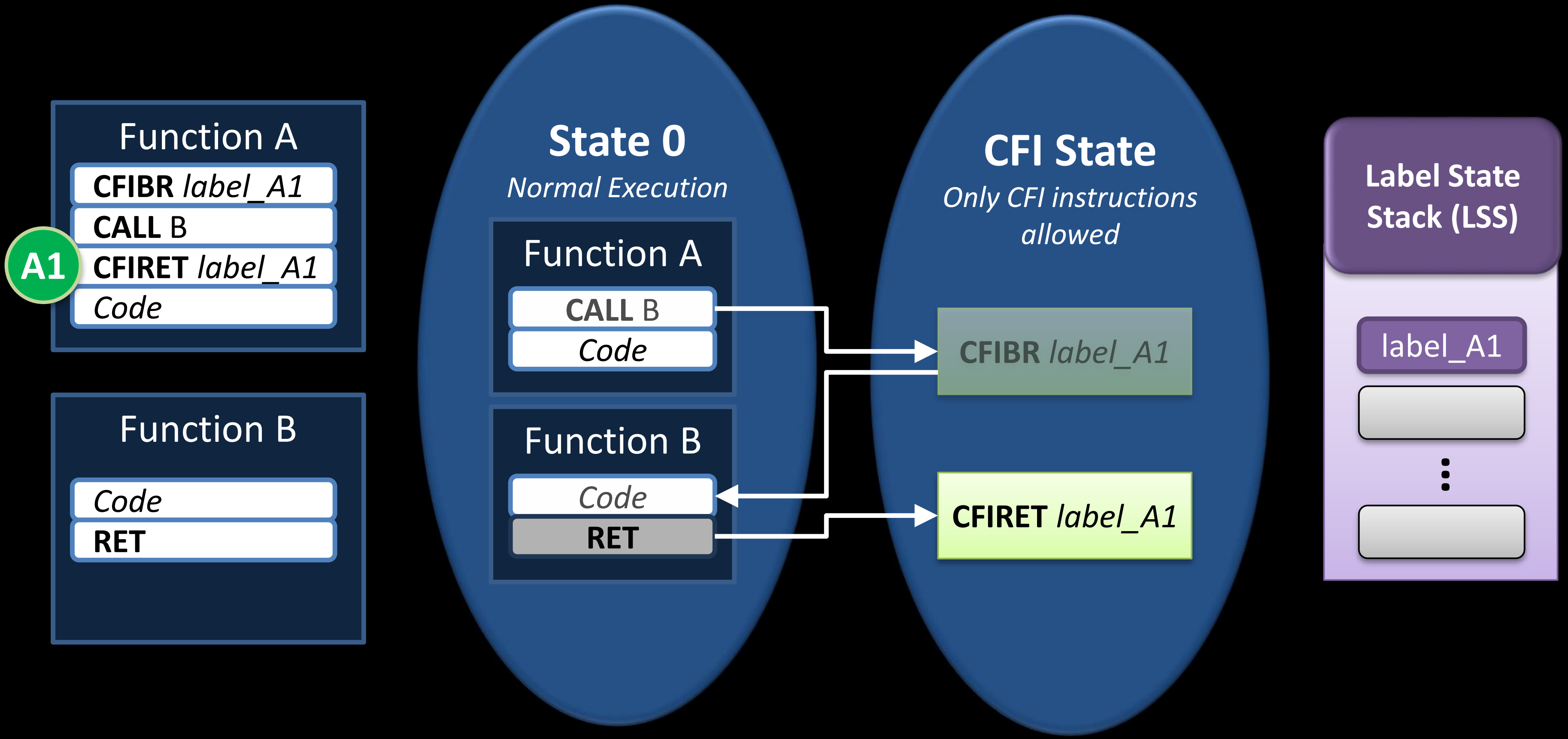
Function Return Policy



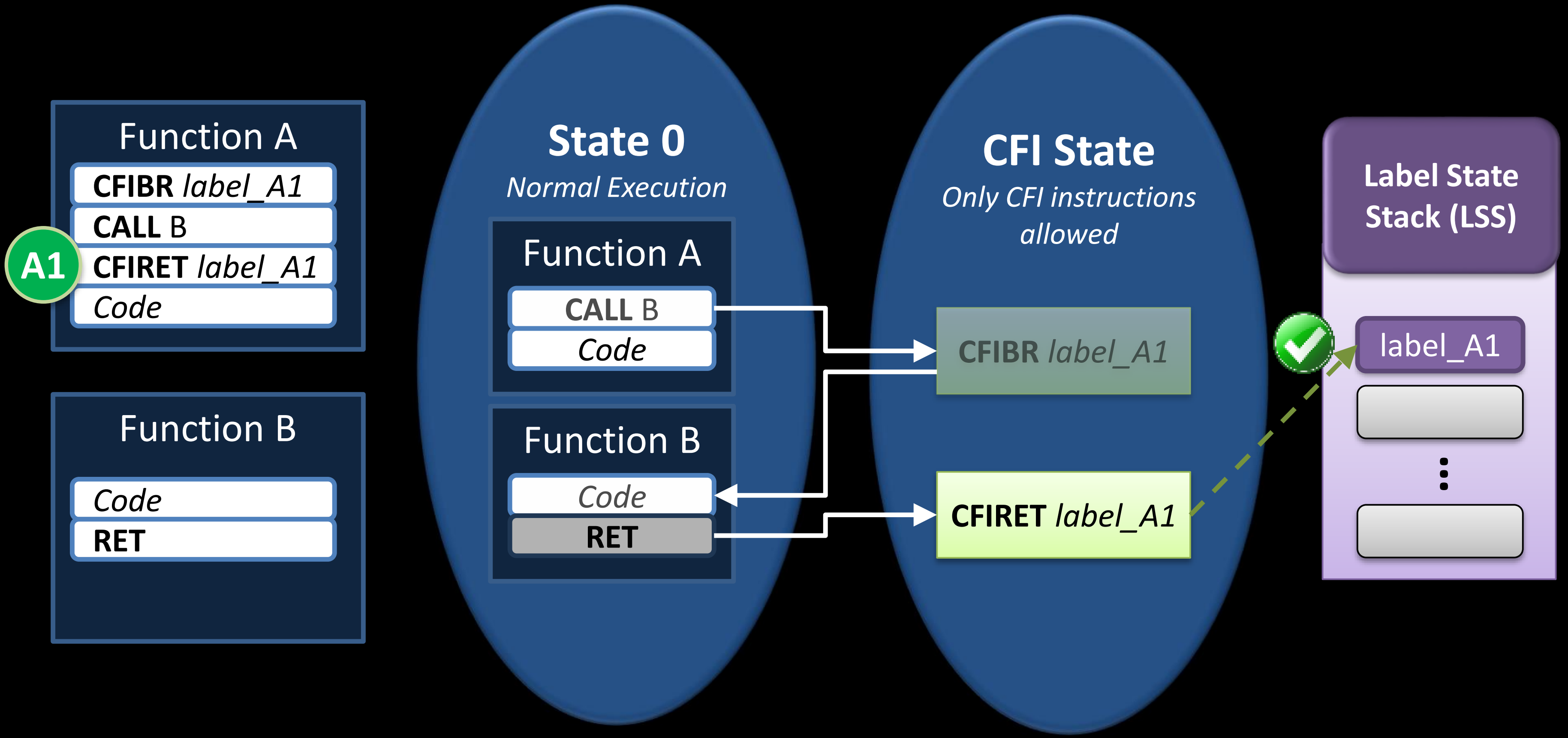
Function Return Policy



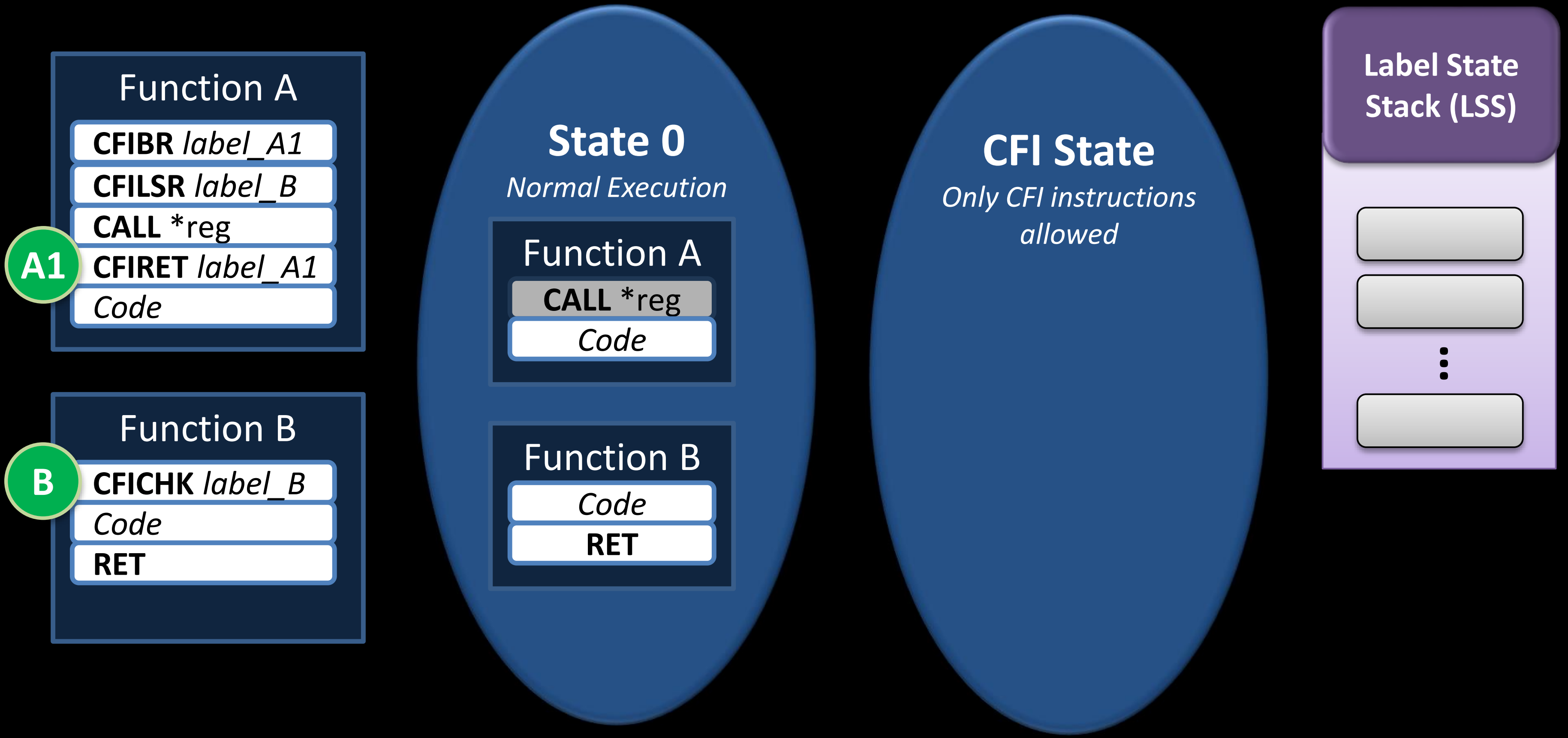
Function Return Policy



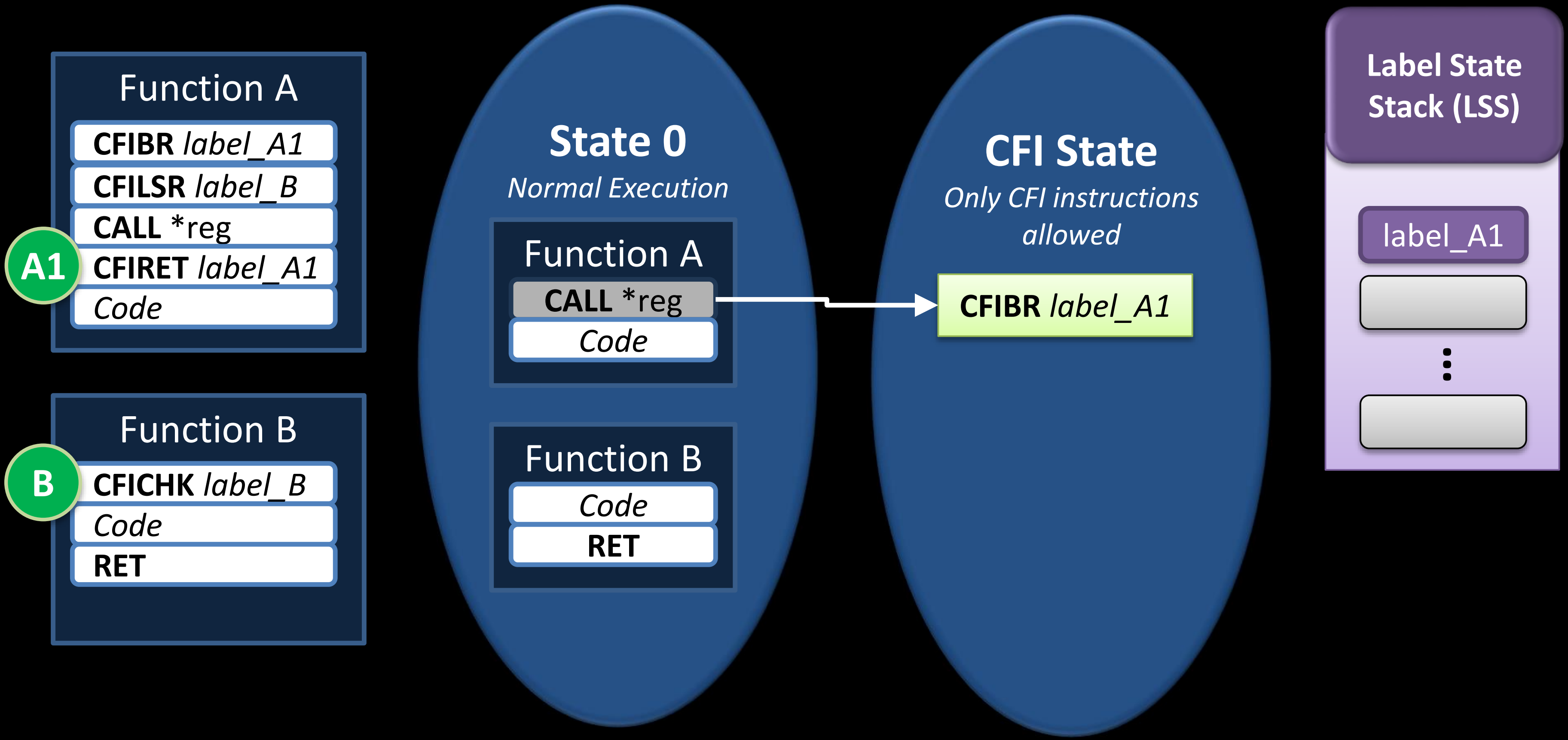
Function Return Policy



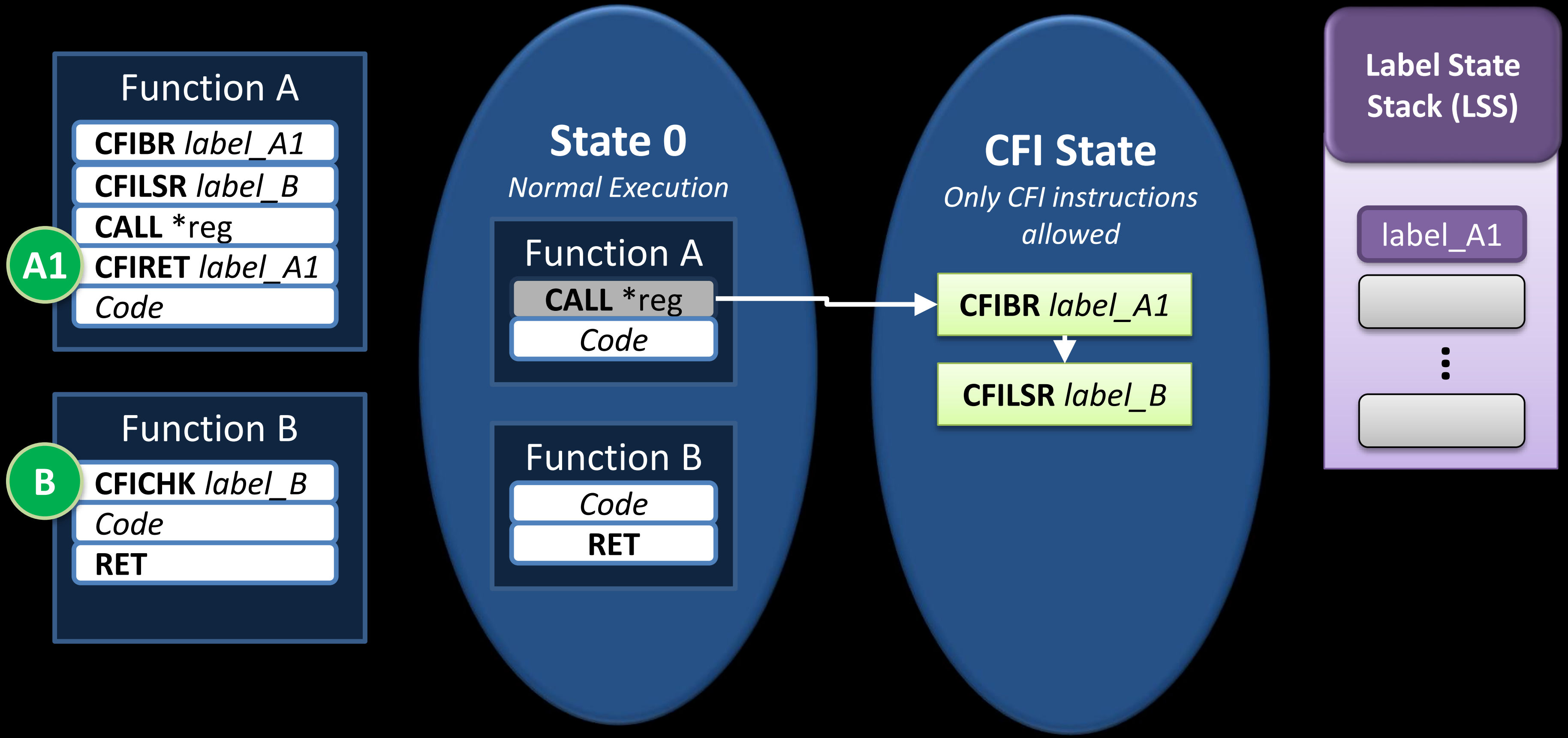
Indirect Call Policy



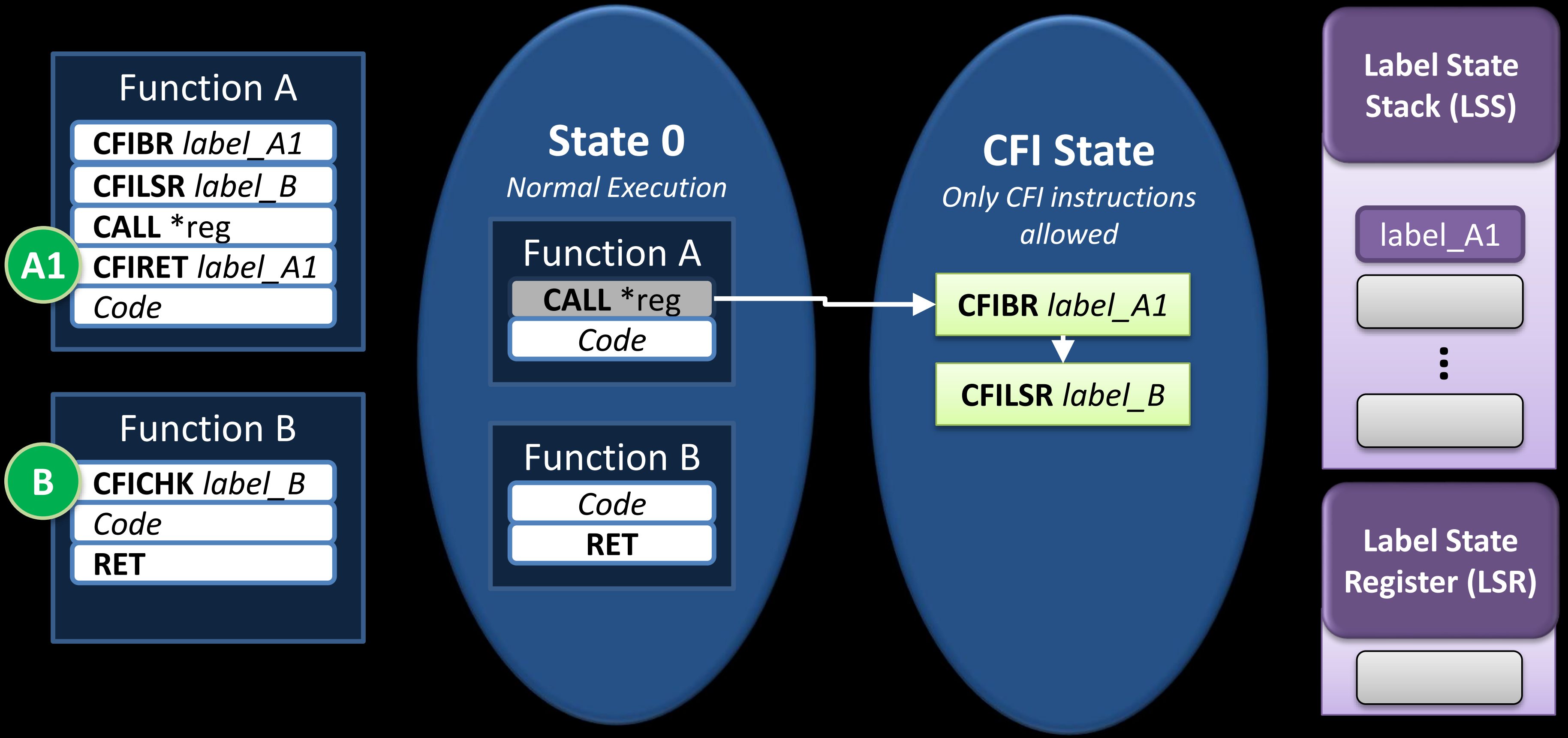
Indirect Call Policy



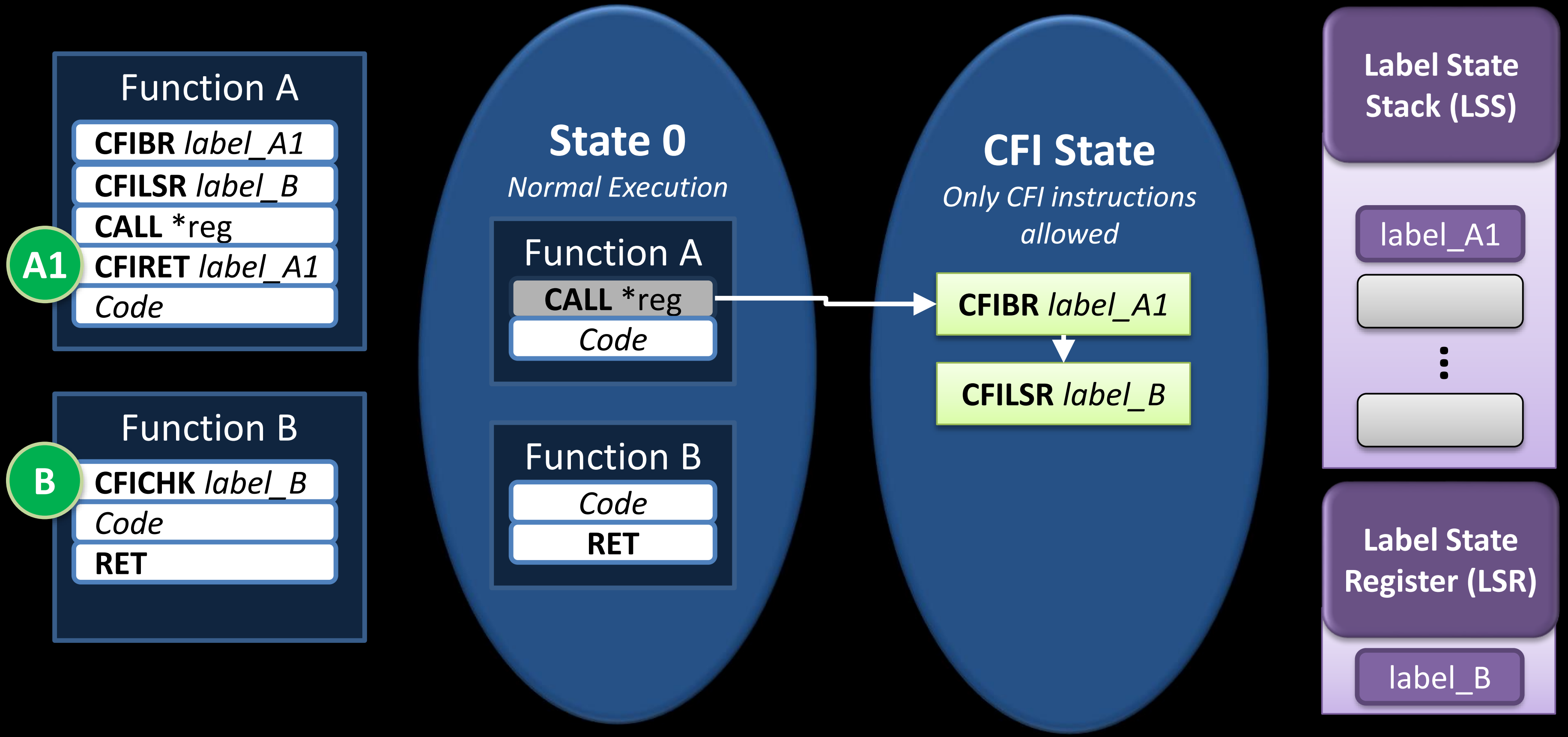
Indirect Call Policy



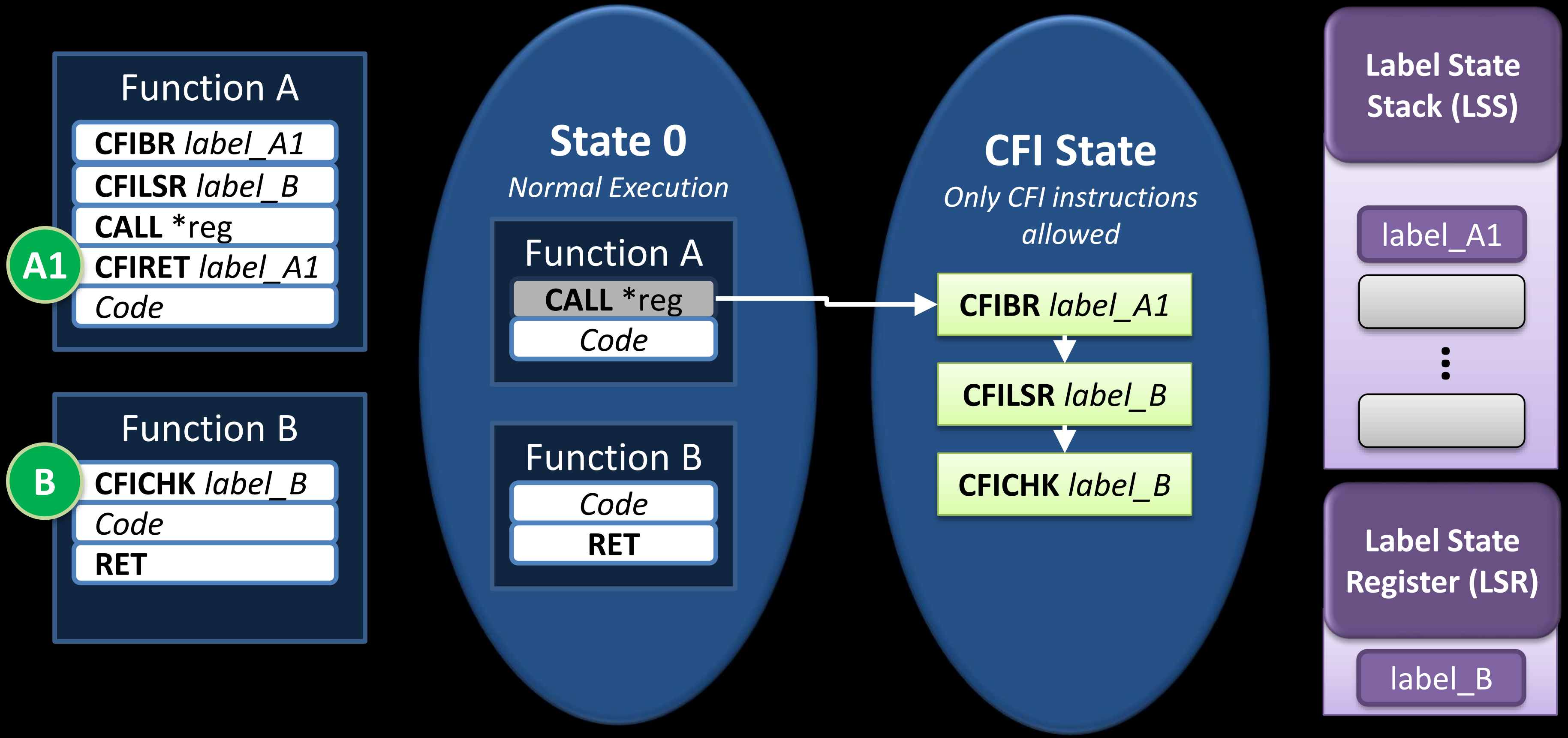
Indirect Call Policy



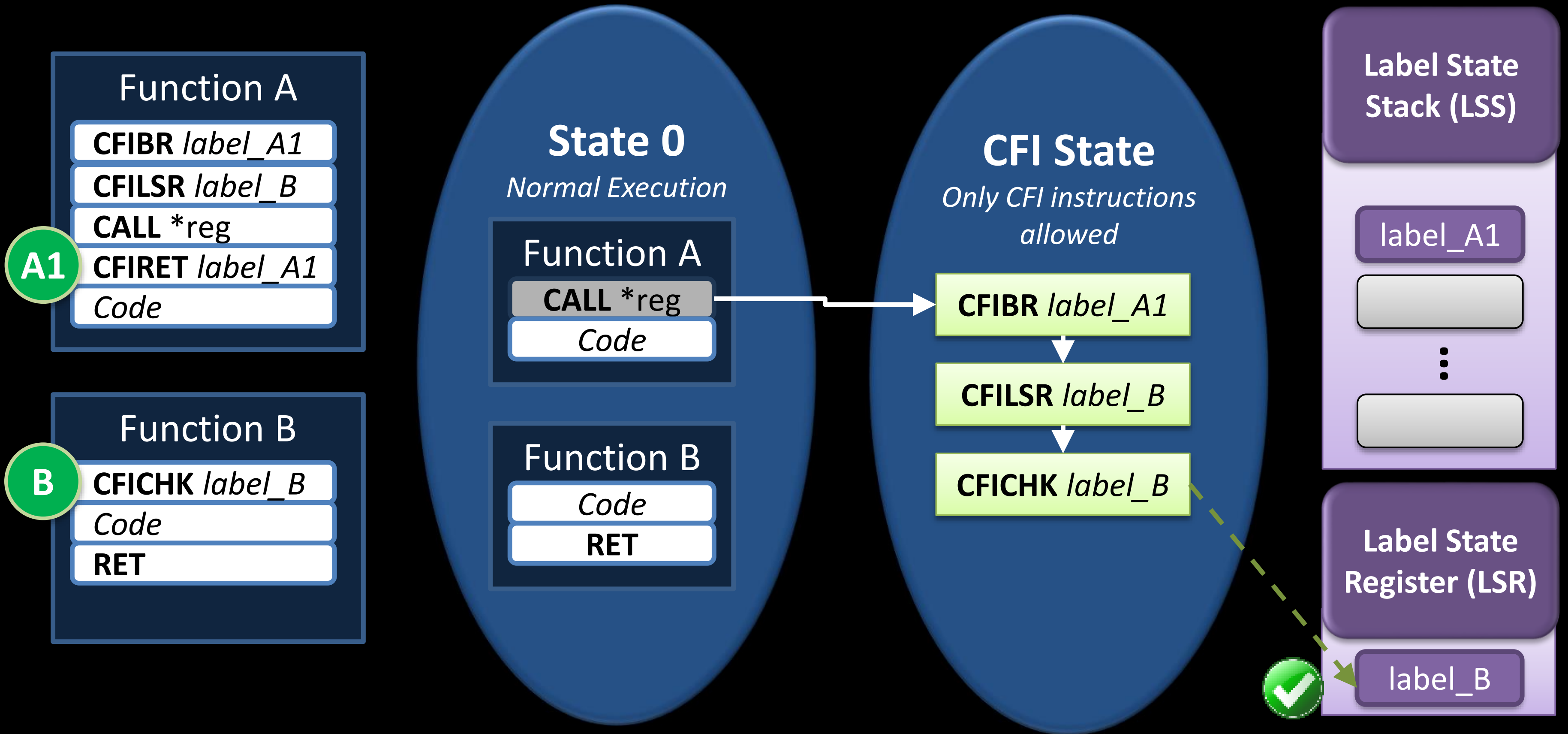
Indirect Call Policy



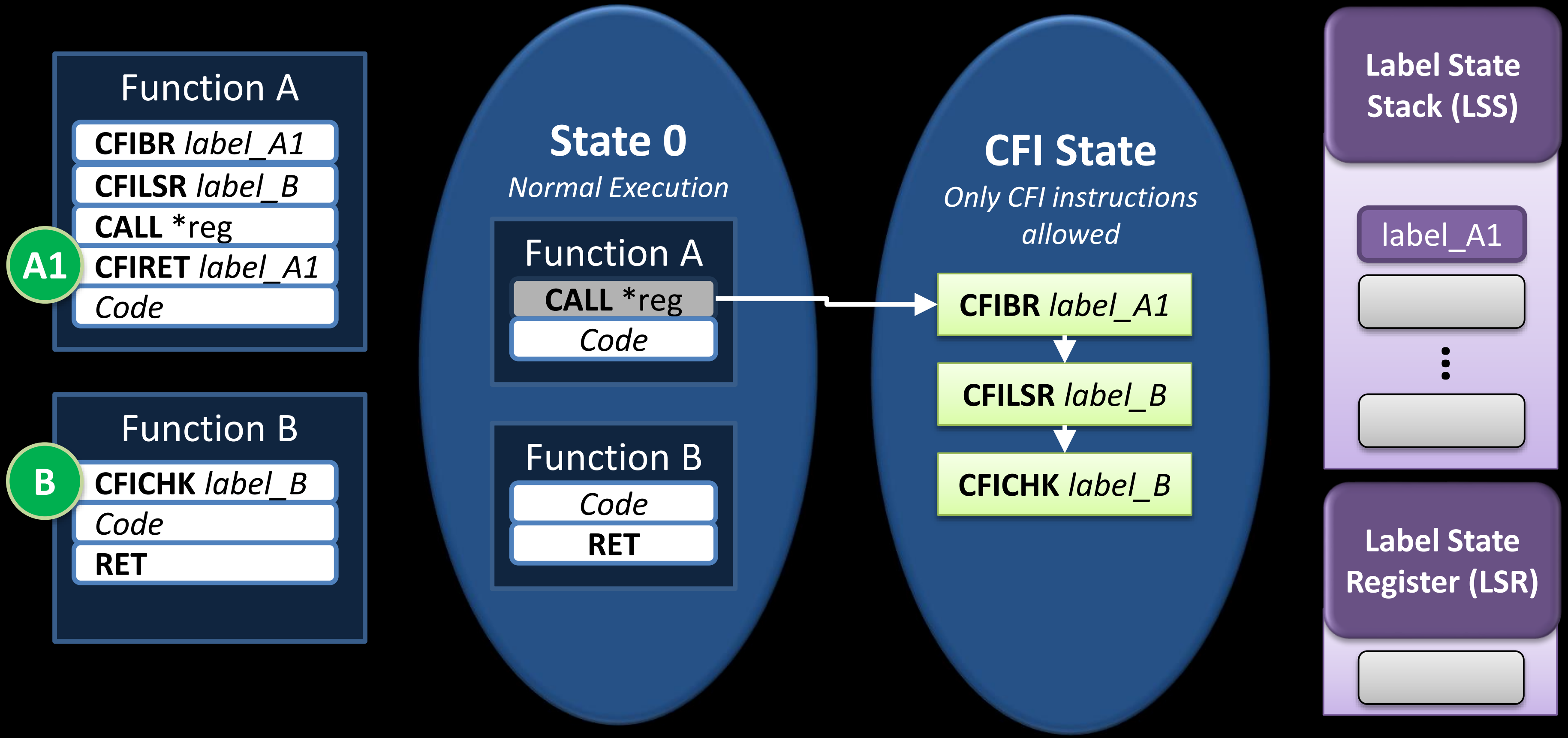
Indirect Call Policy



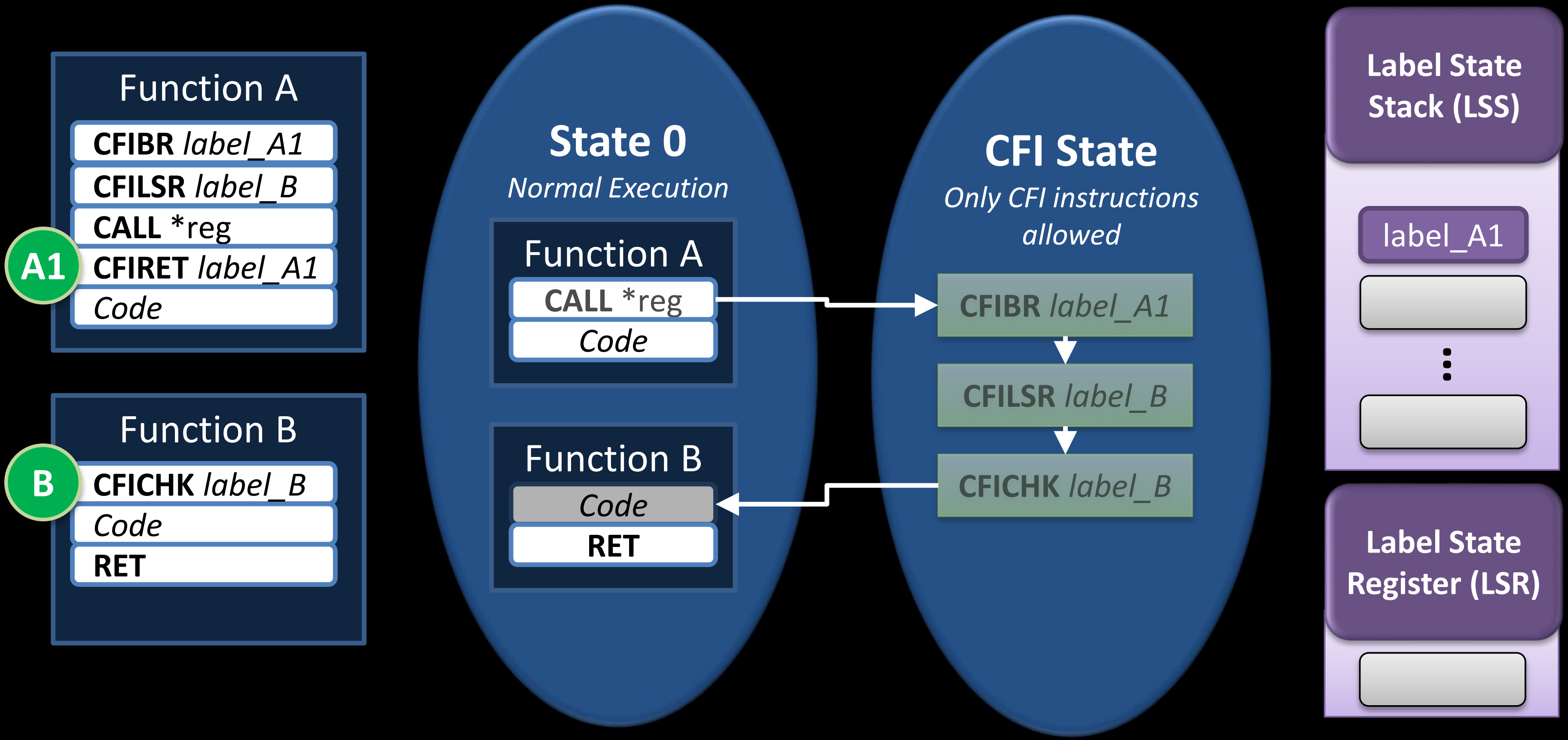
Indirect Call Policy



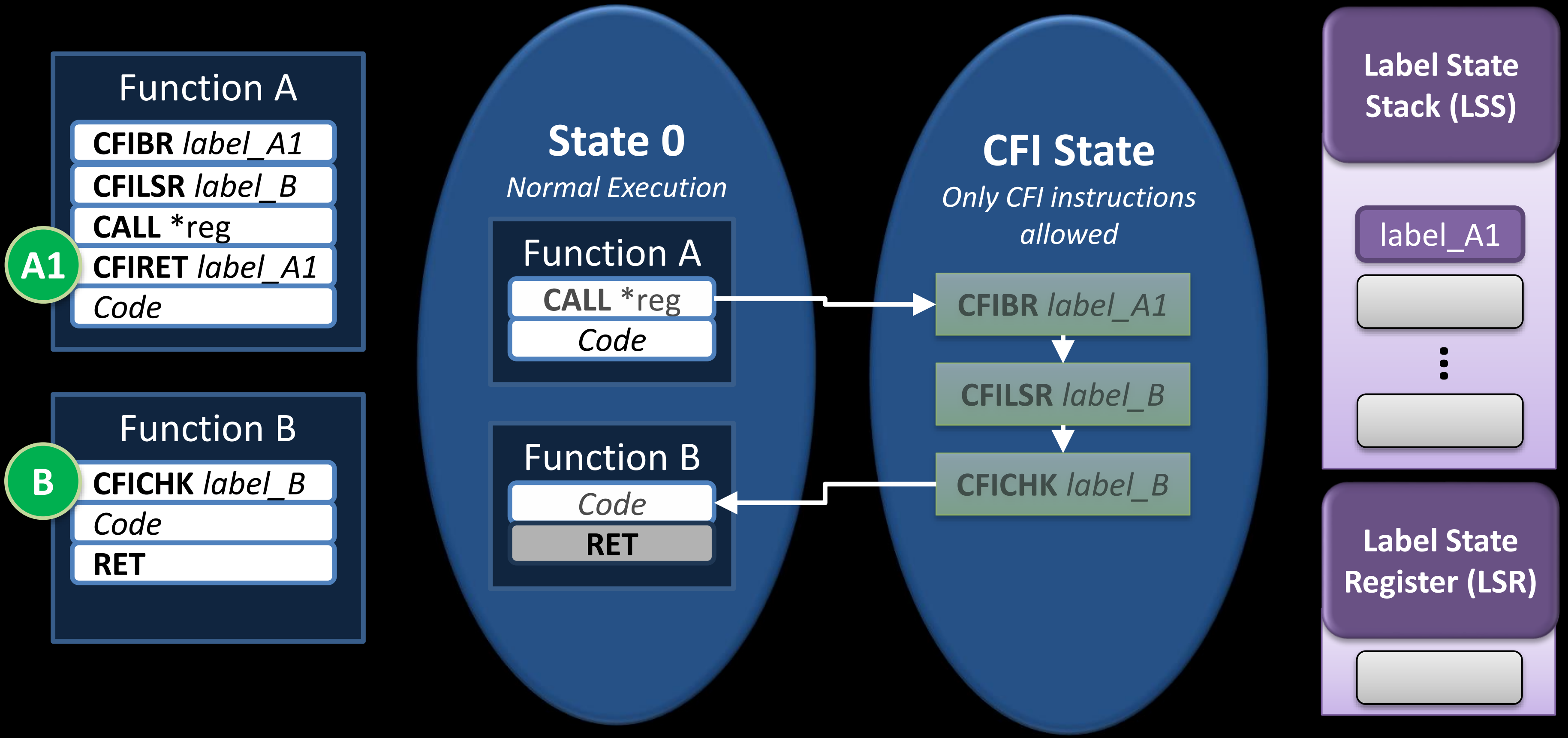
Indirect Call Policy



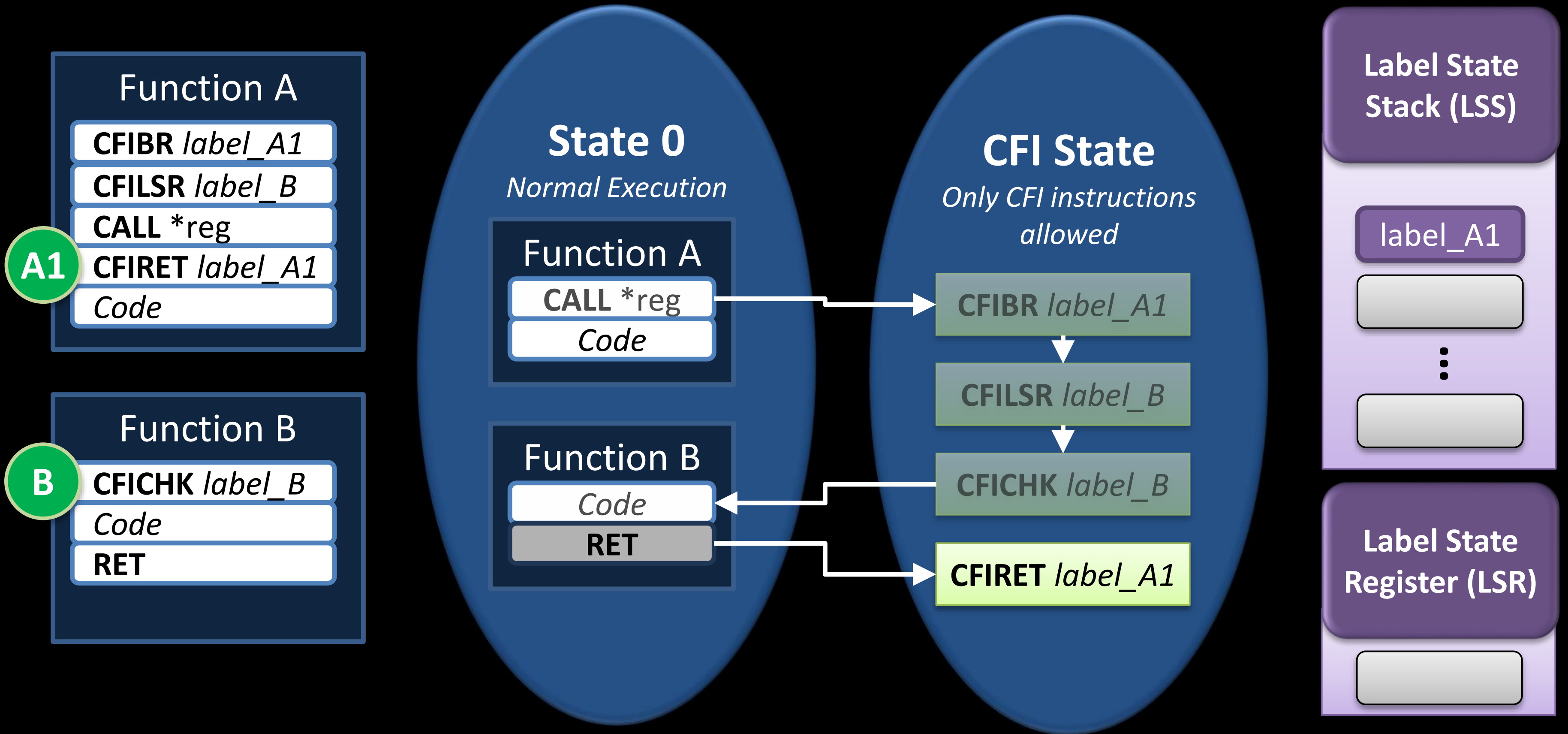
Indirect Call Policy



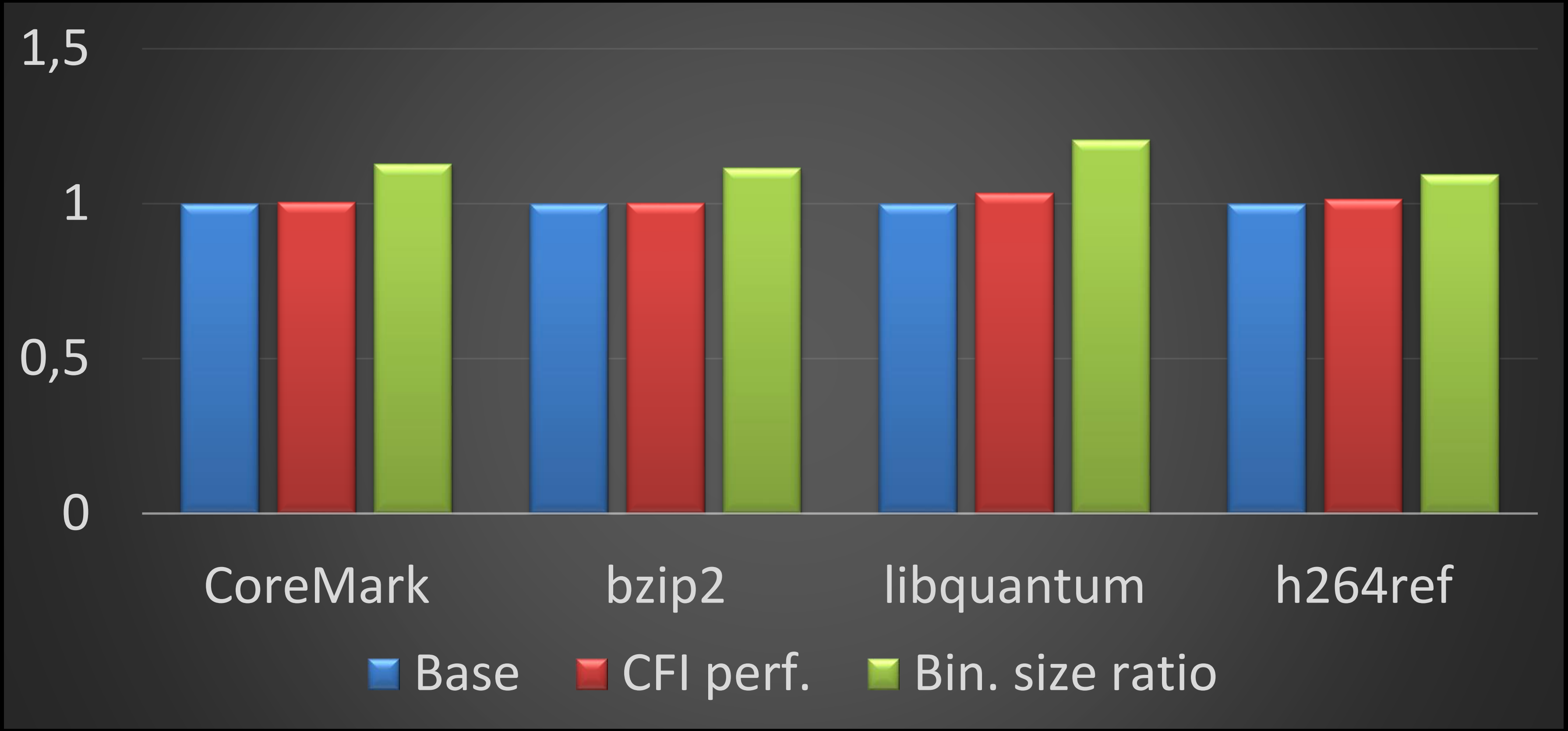
Indirect Call Policy



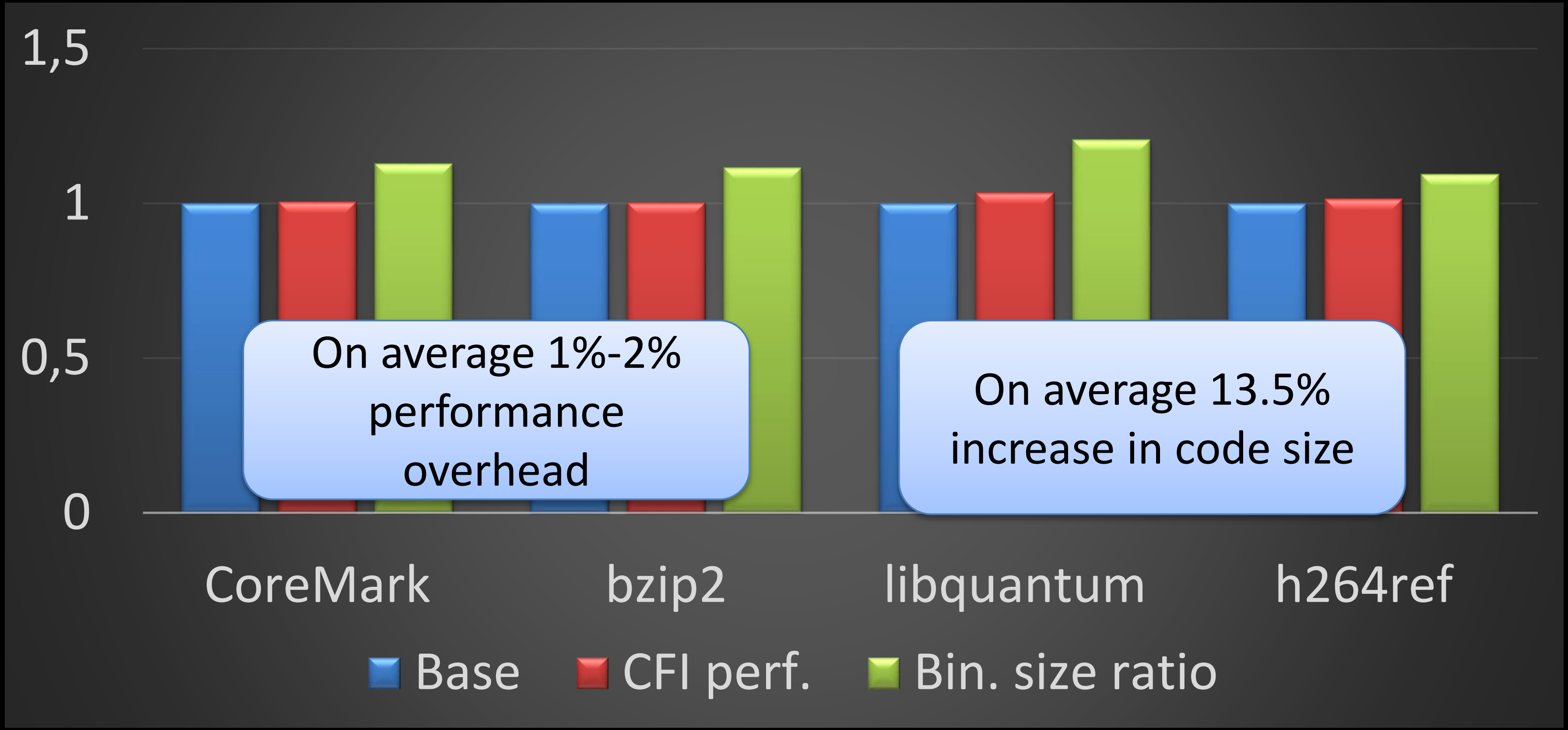
Indirect Call Policy



Evaluation


1,5
1
0,5
0
CoreMark
bzip2
libquantum
h264ref
Base
CFI perf.
Bin. size ratio

Evaluation



On average 1%-2% performance overhead

On average 13.5% increase in code size

HAFIX++ ISA Extensions

cfibr	Issued at call site → setup Backward (BW) Edge
cfiret	Issue at return site → check BW Edge
cfiprc	Issued at call site → setup call target
cfiprj	Issued at jump site → setup jump target
cfichk	Issued at call/jmp target → check Forward (FW) Edge

- Fine-grained FW edge control-flow policy
  - Separation of call/jump
  - Unique label per target
- Fine-grained BW edge control-flow policy
  - Return to only most recently issued return label

HAFIX++ ISA Extensions

Backward-Edge Code Reuse attacks.

cfibr lbl/cfiret lbl instruction pair A return only allowed to target a cfiret if it is the most recent in execution path history, i.e., it is a valid State (checking the label at the top of the LSS against cfiret lbl at the return target)

Forward-Edge Code Reuse Attacks.

A return instruction is only allowed to target a cfiret instruction if it is the most recent in the execution path history, i.e., it is a valid state. This is determined by checking the label at the top of the LSS against cfiret lbl at the return target. Only cfiret instructions may be targeted by

Full-Function Code-Reuse Attacks.

We prevent CFB attacks since they require redirection to any call-preceded slot in a stateless CFI protection system. We offer precise, stateful CFI so that only the most recently executed forward-edge transition may be returned to. As described above, this is ensured with a unique cfibr lbl/cfiret lbl instruction pair. A return instruction is only allowed to target a call-preceded slot if it is the most recent in the execution path history.

Control-Flow Bending.

Issued at jump site → setup jump target

cfichk

Issued at call/jmp target → check Forward (FW) Edge

Requirements

Backward-Edge and Forward-Edge CFI

High performance

No burden on developer to instrument the code

Stateful CFI with protected state

Enabling technology

Compatibility to Legacy Code

Hardware-Based Solutions

	BE-Support	FE-Support	Shared library & Multitasking	Granularity	Overhead
XFI Budiu et al, ASID 2006	⊖	✓	✓	Coarse	3.75%
HAFIX Davi et al., DAC 2015	✓	⊖	⊖	Coarse	2%
LandHere http://langalois.com	⊖	⊖	✓	Coarse	N/A
HCFI Christoulakis et al., CODASPY 2016	✓	✓	⊖	Fine	1%
Intel CET Intel Tech Review	✓	✓	✓	Coarse	N/A
HAFIX++ Sullivan et al., DAC 2016	✓	✓	✓	Fine	1.75%

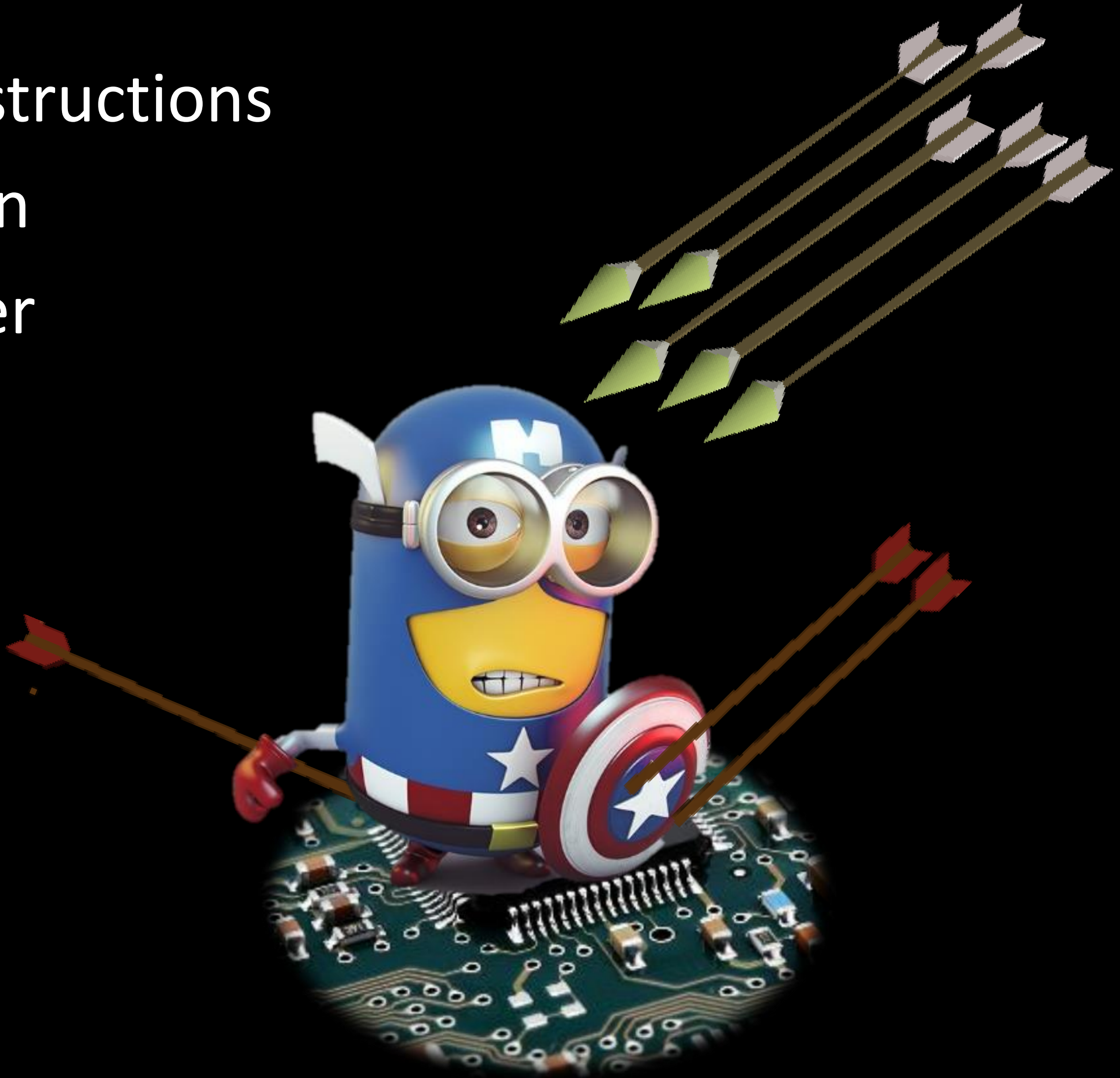
Hardware-Based Solutions

	BE-Support	FE-Support	Shared library & Multitasking	Granularity	Overhead
XFI Budiu et al, ASID 2006	⊖	✓	✓	Coarse	3.75%
HAFIX Davi et al., DAC 2015	✓	⊖	⊖	Coarse	2%
LandHere http://langalois.com	⊖	⊖	✓		N/A
HCFI Christoulakis et al., CODASPY 2016	✓	✓	⊖	Fine	1%
Intel CET Intel Tech Review	✓	✓	✓	Coarse	N/A
HAFIX++ Sullivan et al., DAC 2016	✓	✓	✓	Fine	1.75%

Architectural dependent optimizations

Problems of Existing HW-CFI Schemes

- Offline training phase
- Single CFI label register: Lots of CFI instructions
- Large storage of branching information
- CFI checks must follow execution order



Hardware CFI [Budiu et al, ASID 2006],
kBouncer [Pappas et al., USENIX Sec. 2013],
CFIMon [Xia et al., DSN 2012],
Reconfigurable DTPM [Das et al., VLSI 2014],
Branch Regulation [Kayaalp et al., ISCA 2012]

CYSEC
Cybersecurity
TU Darmstadt

IMIX: Hardware-Enforced In-Process Memory Isolation

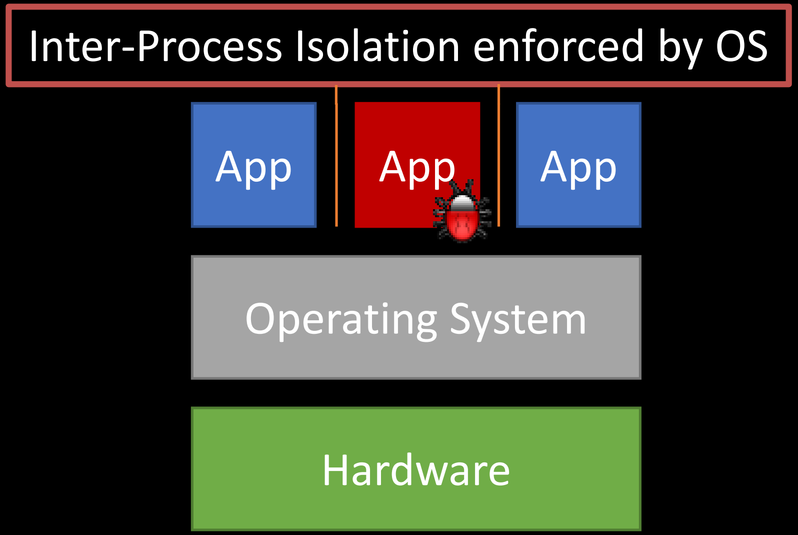
IMIX: In-Process Memory Isolation EXtension

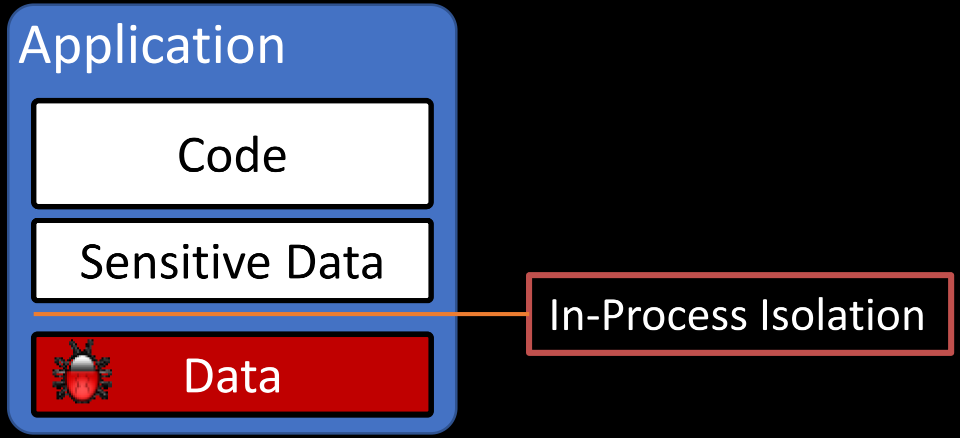
27th USENIX Security Symposium 2018

Tommaso Frassetto, Patrick Jauernig, Christopher Liebchen, Ahmad-Reza Sadeghi

Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

Inter- & In-Process Isolation

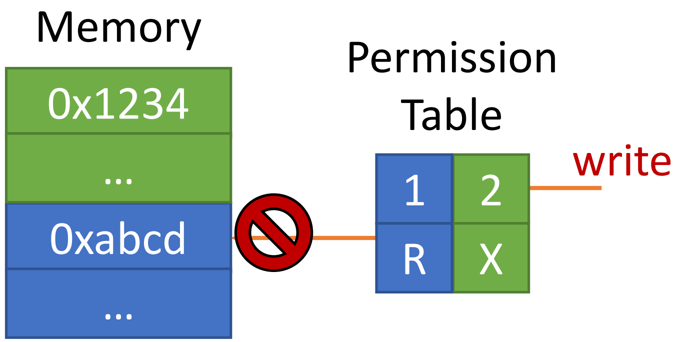




In-Process Isolation

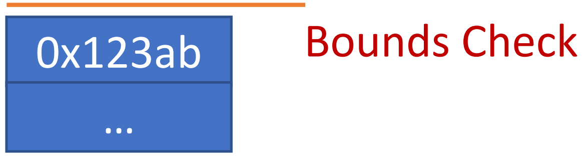
Memory Protection Keys

Intel MPK/PKU

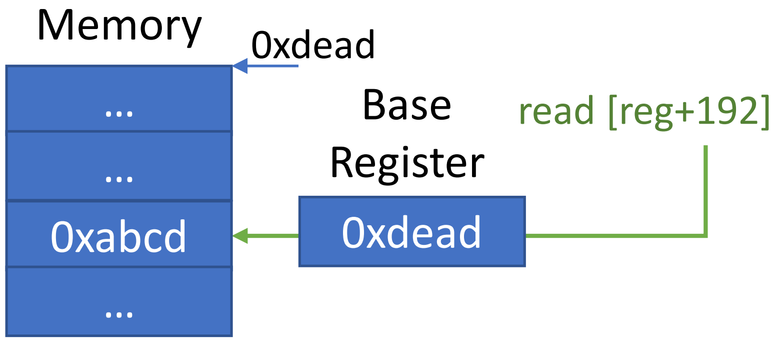


Hardware Bounds Checking

e.g. Intel MPX



Randomization

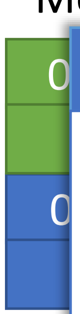


In-Process Isolation

Memory Protection Keys

Intel MPK/PKU

Memory



Problems

- High performance overhead for frequent switches

Hardware Bounds Checking

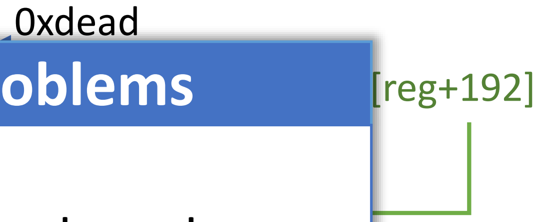
e.g. Intel MPX

Problems

- Excessive instrumentation
- High performance overhead

Randomization

Memory

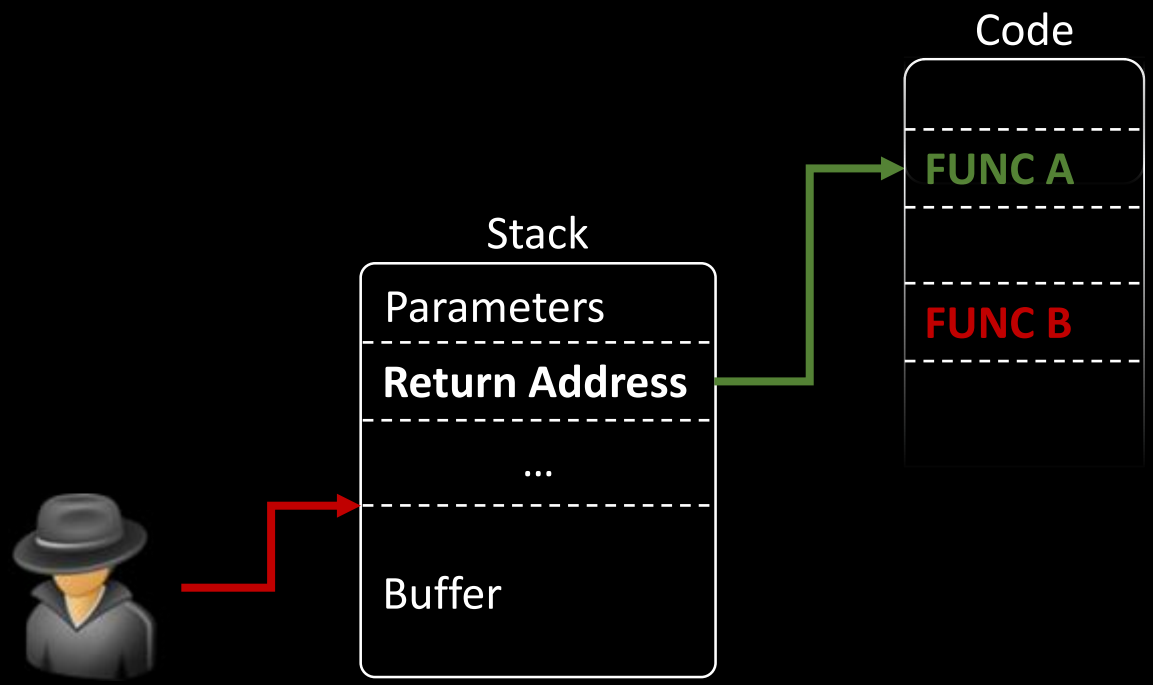


Problems

- Entropy-based: single information leak breaks isolation

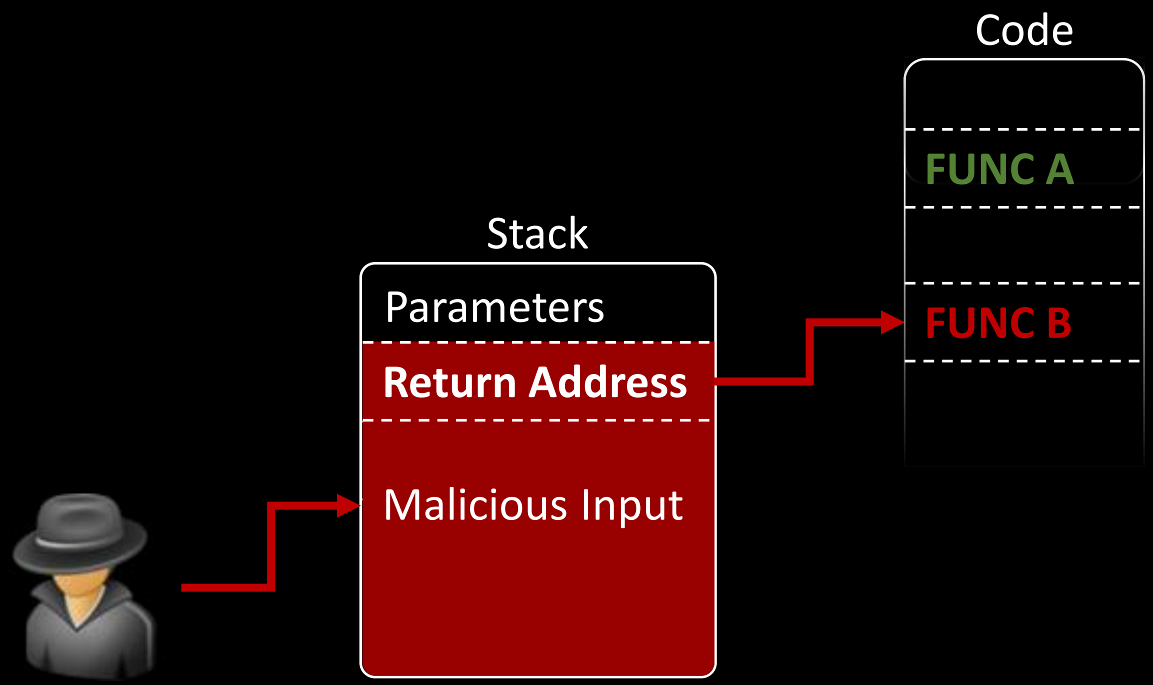
Memory Corruption Attacks

- Malicious input to alter stack/heap memory



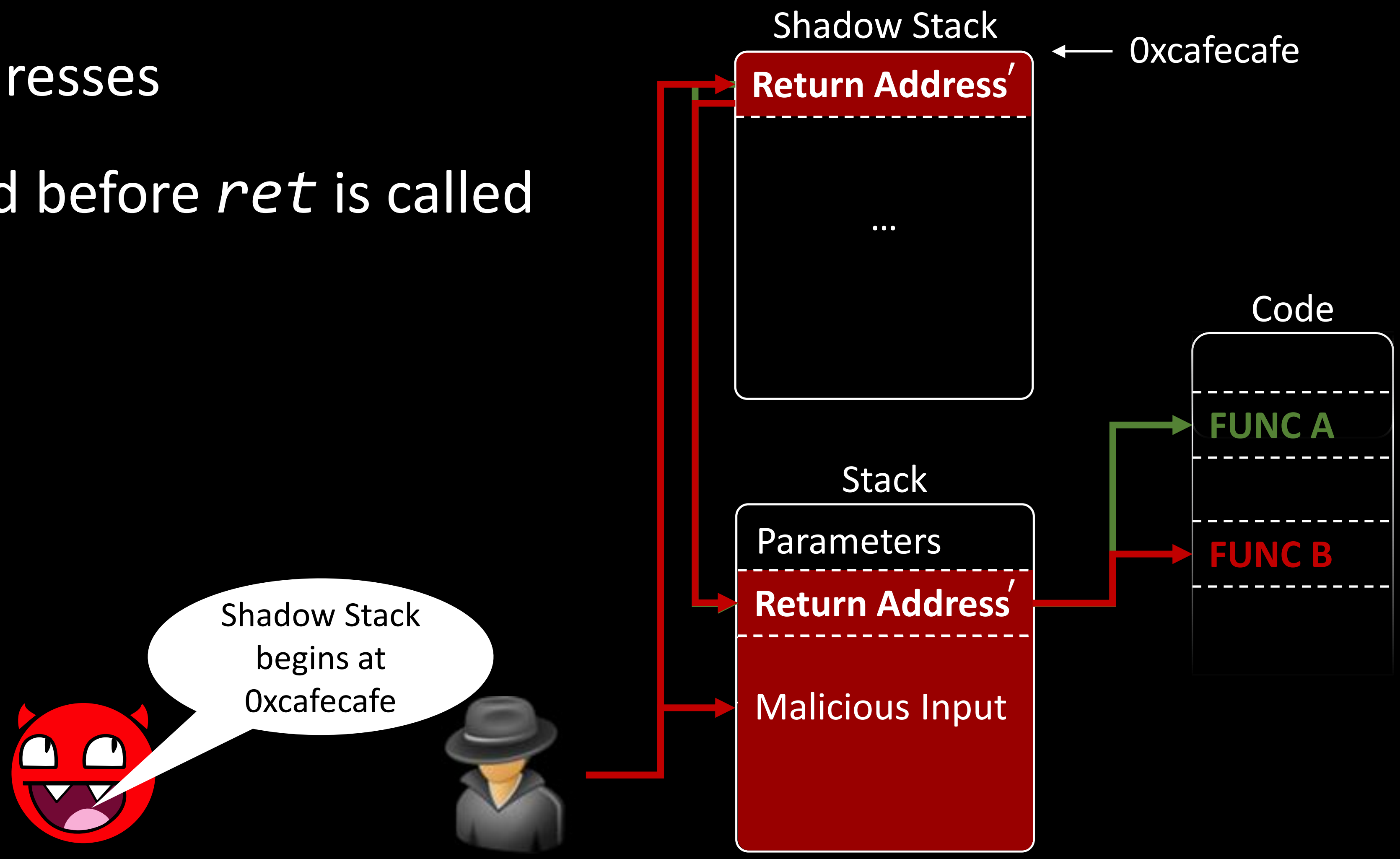
Memory Corruption Attacks

- Malicious input to alter stack/heap memory



Shadow Stack

- Backup return addresses
- Address is restored before *ret* is called

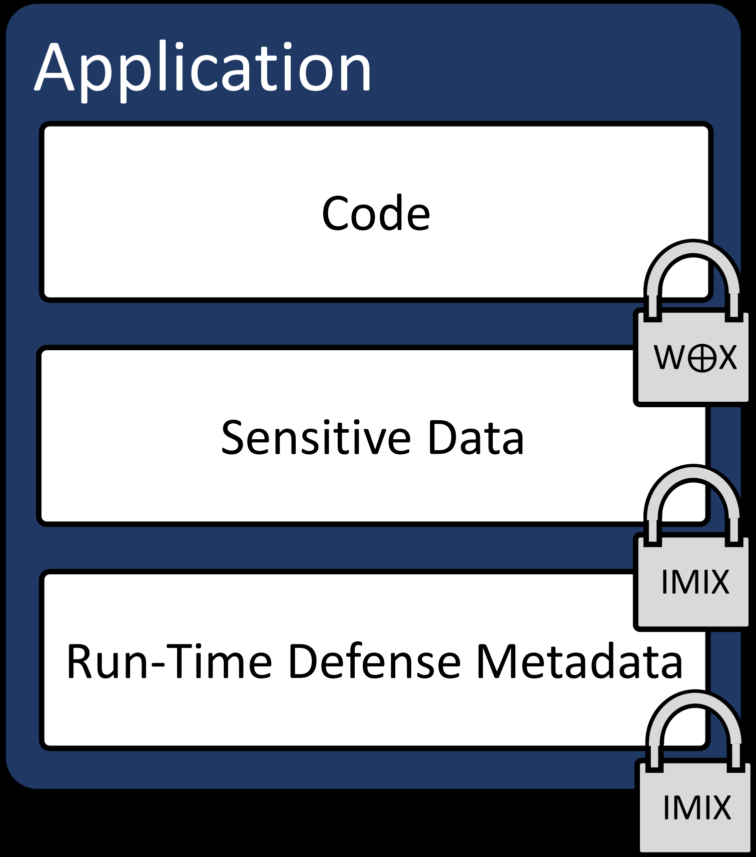


IMIX

[Frassetto et al., IMIX: In-Process Memory Isolation EXtension. USENIX Sec. 2018]

- Hardware-enforced in-process memory isolation
- Isolation primitive for mitigations at page granularity
- Two separate memory realms
  - smov to load/store sensitive data
  - mov for regular memory
- Limitation: Code-Reuse of smov

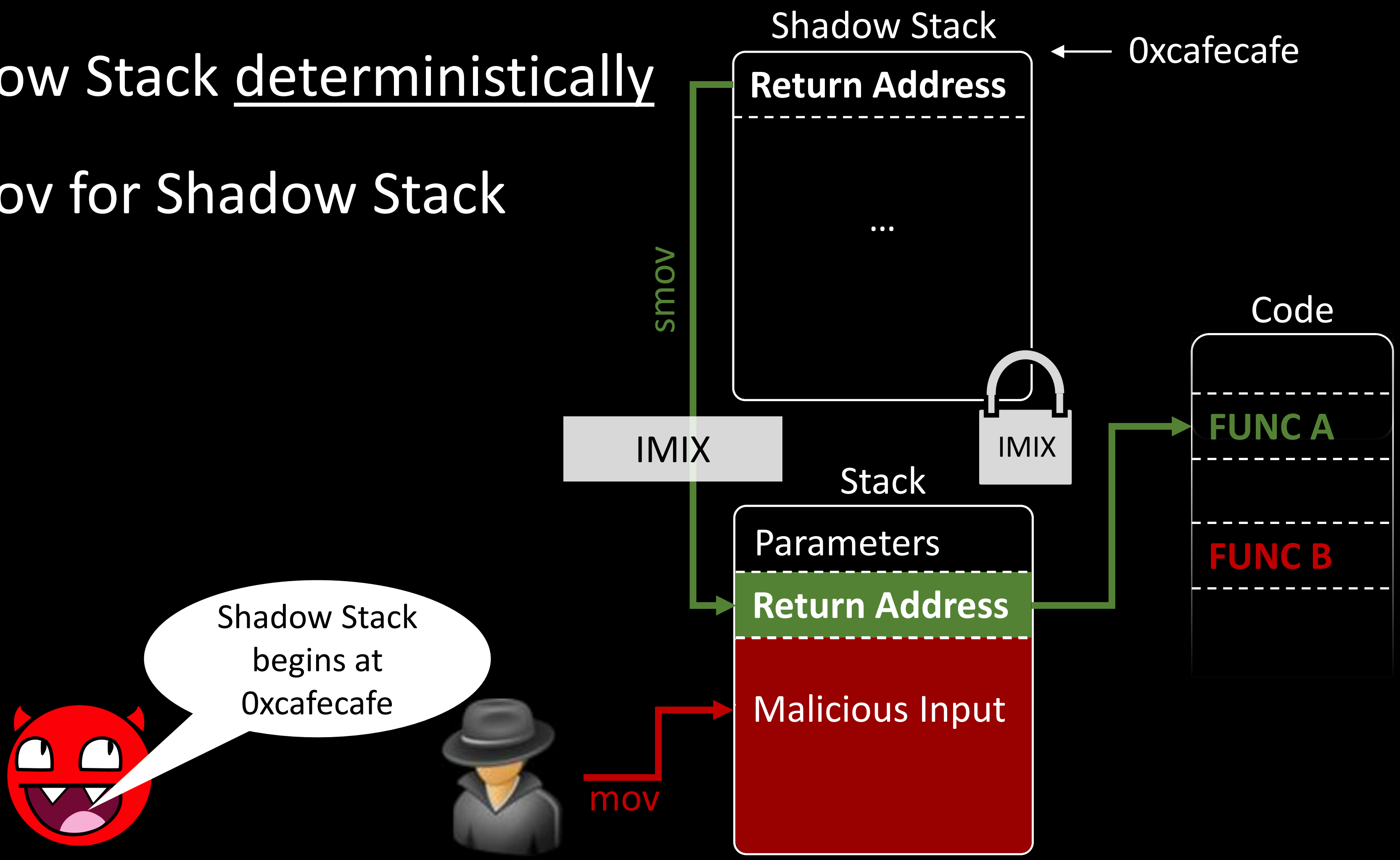
→Use smov to protect CFI/CPI



IMIX in Action: Shadow Stack Revisited

[Frassetto et al., IMIX: In-Process Memory Isolation EXtension. USENIX Sec. 2018]

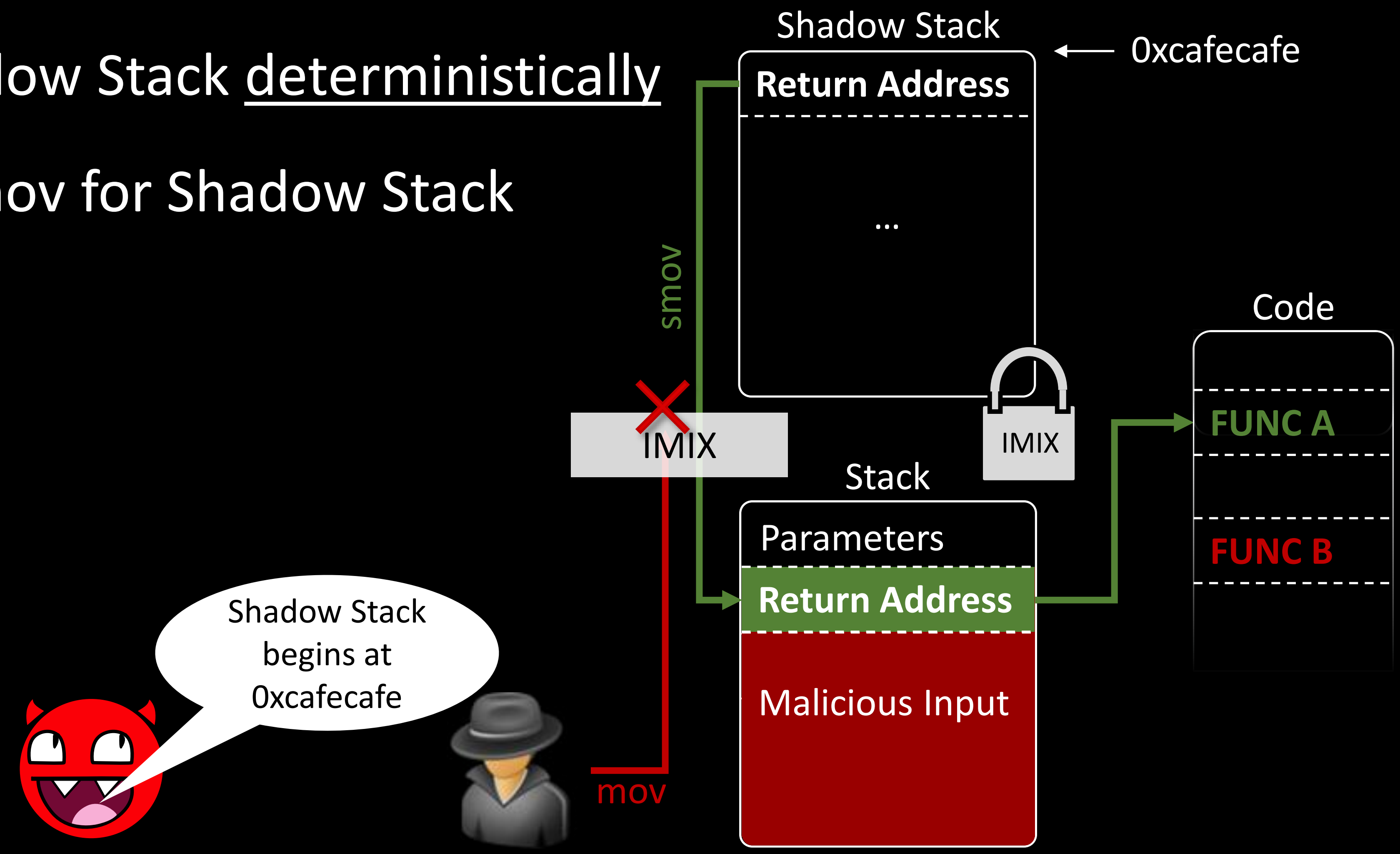
- IMIX isolates Shadow Stack <u>deterministically</u>
- Exclusively use smov for Shadow Stack



IMIX in Action: Shadow Stack Revisited

[Frassetto et al., IMIX: In-Process Memory Isolation EXtension. USENIX Sec. 2018]

- IMIX isolates Shadow Stack deterministically
- Exclusively use smov for Shadow Stack



Thank you!

Ahmad-Reza Sadeghi

www.trust.cased.de