[AsiaCCS 2016]

Juggling the Gadgets: Binary-level Code Randomization using Instruction Displacement

Hyungjoon Koo and Michalis Polychronakis

Stony Brook University

Memory Corruption: Injection → Reuse

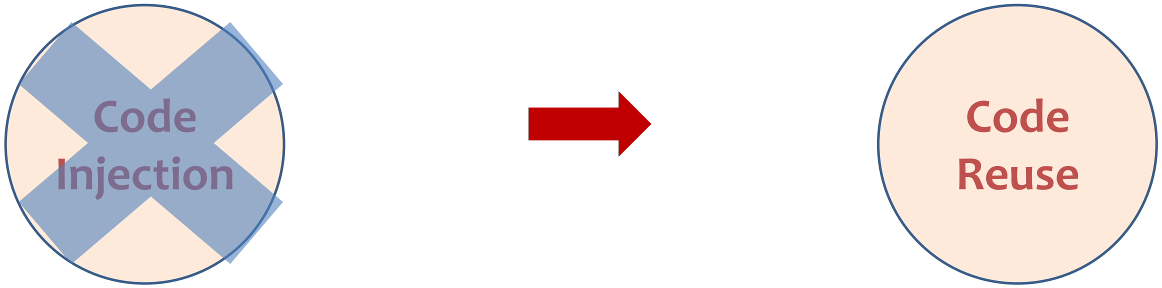
- Attack goal: Divert control flow

Code Injection

Run arbitrary code!

Memory Corruption: Injection → Reuse

- Attack goal: Divert control flow



Code Injection

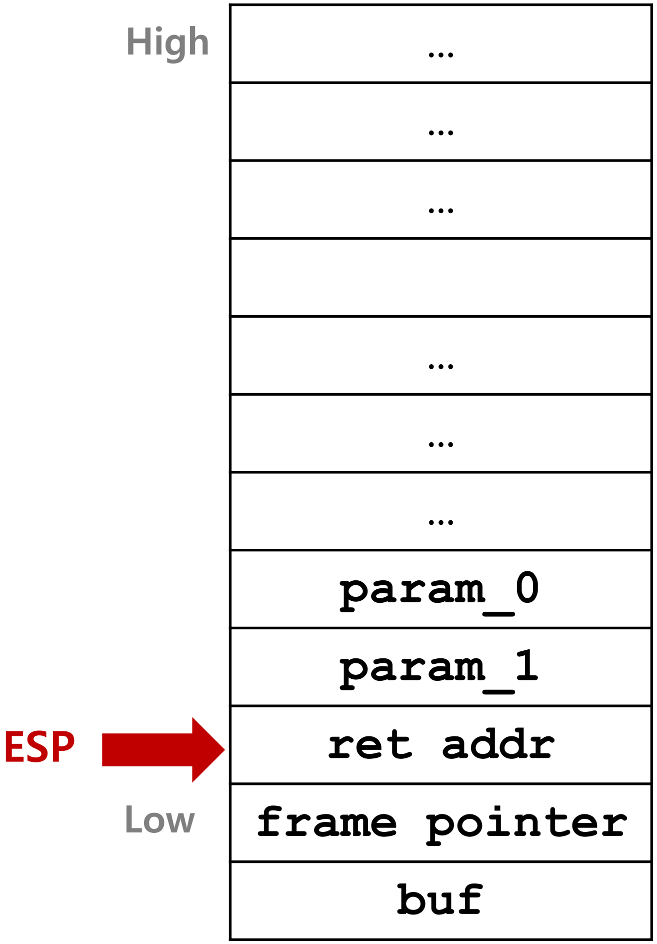
Code Reuse

W⊗X

Run arbitrary code!

Execute existing code!

Memory Corruption: ROP Concept



High

…

…

…

…

…

…

param_0

param_1

ESP

ret addr

Low

frame pointer

buf

[Z]

```
mov [ebx], eax
ret
```

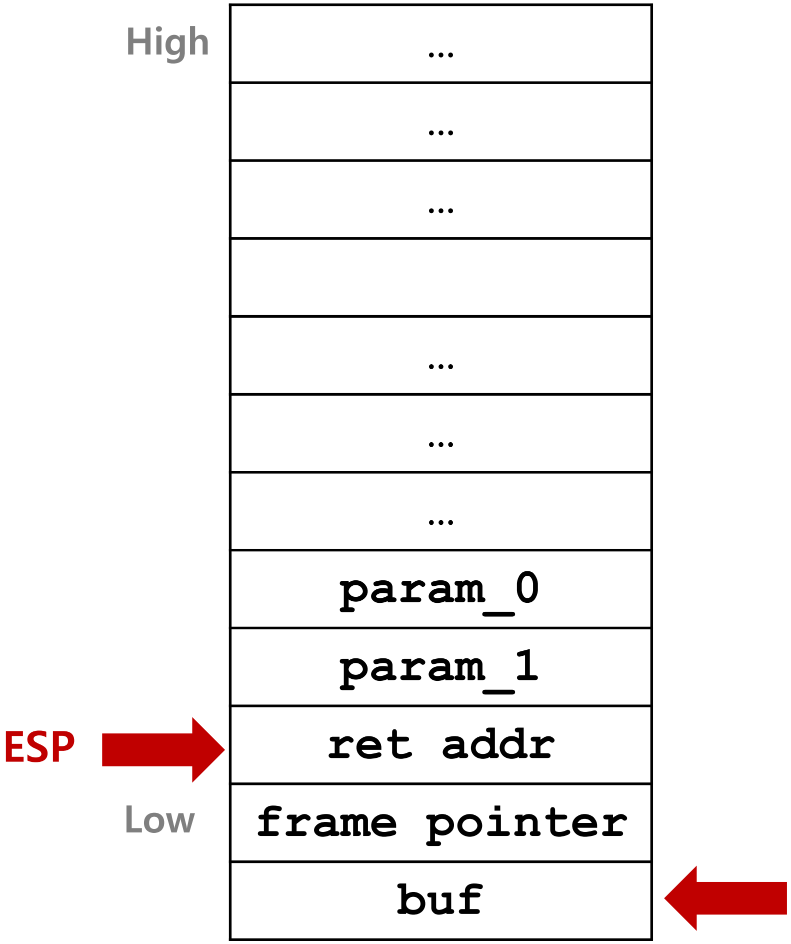
[Y]

```
add eax, ebx
ret
```

[X]

```
pop eax
pop ebx
ret
```

Memory Corruption: ROP Concept


High
...
...
...

...
...
...
param_0
param_1
ret addr
frame pointer
buf
ESP
Low

Exploit Payload

[Z]

```
mov [ebx], eax
ret
```

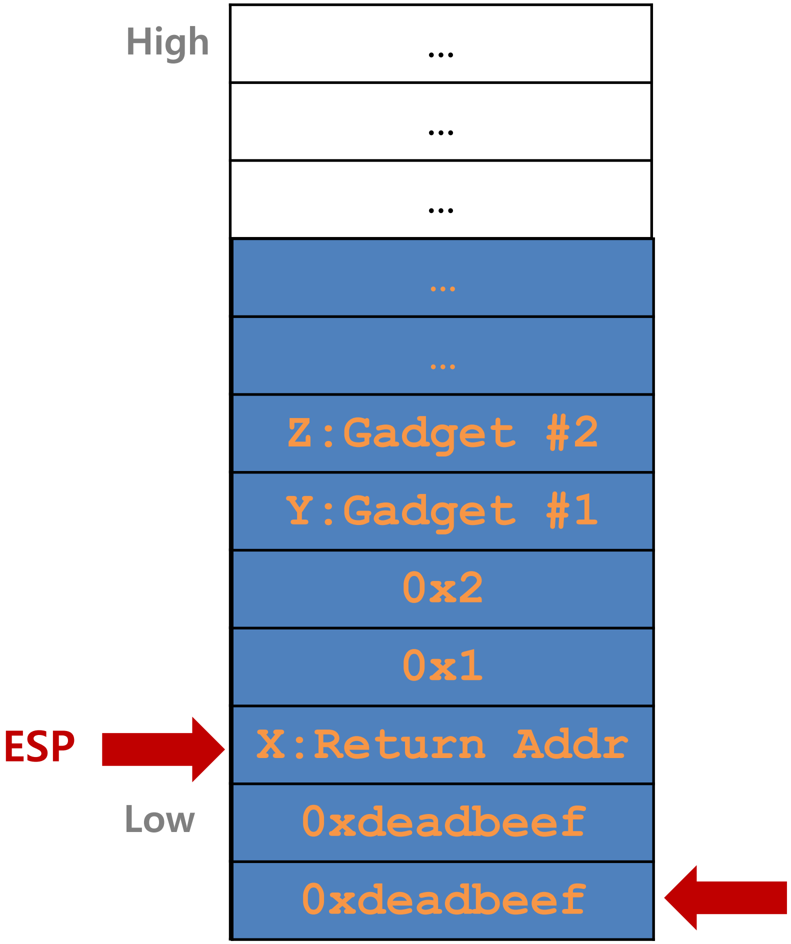
[Y]

```
add eax, ebx
ret
```

[X]

```
pop eax
pop ebx
ret
```

Memory Corruption: ROP Concept



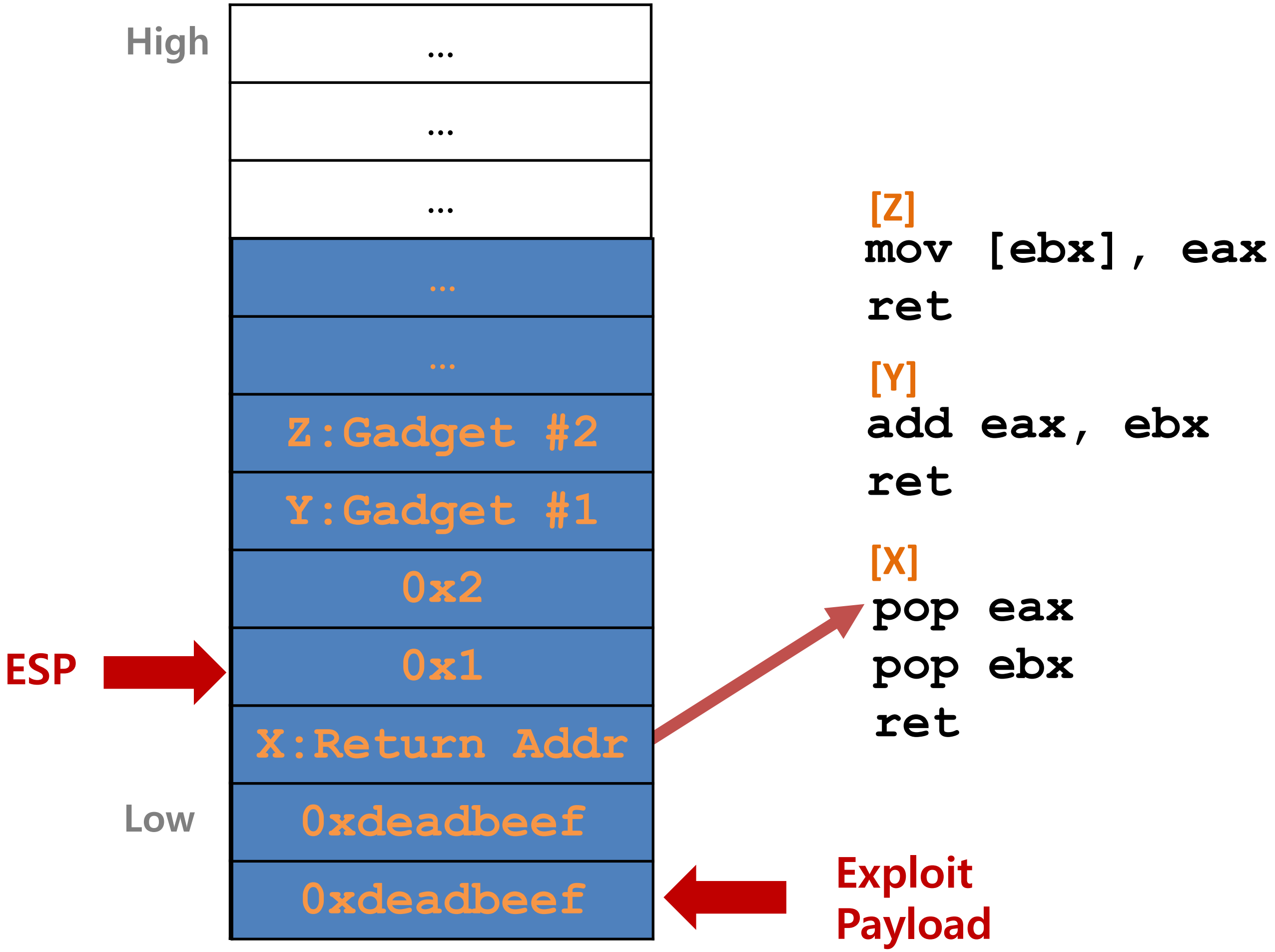
Exploit Payload

```
[Z]
mov [ebx], eax
ret

[Y]
add eax, ebx
ret

[X]
pop eax
pop ebx
ret
```

Memory Corruption: ROP Concept

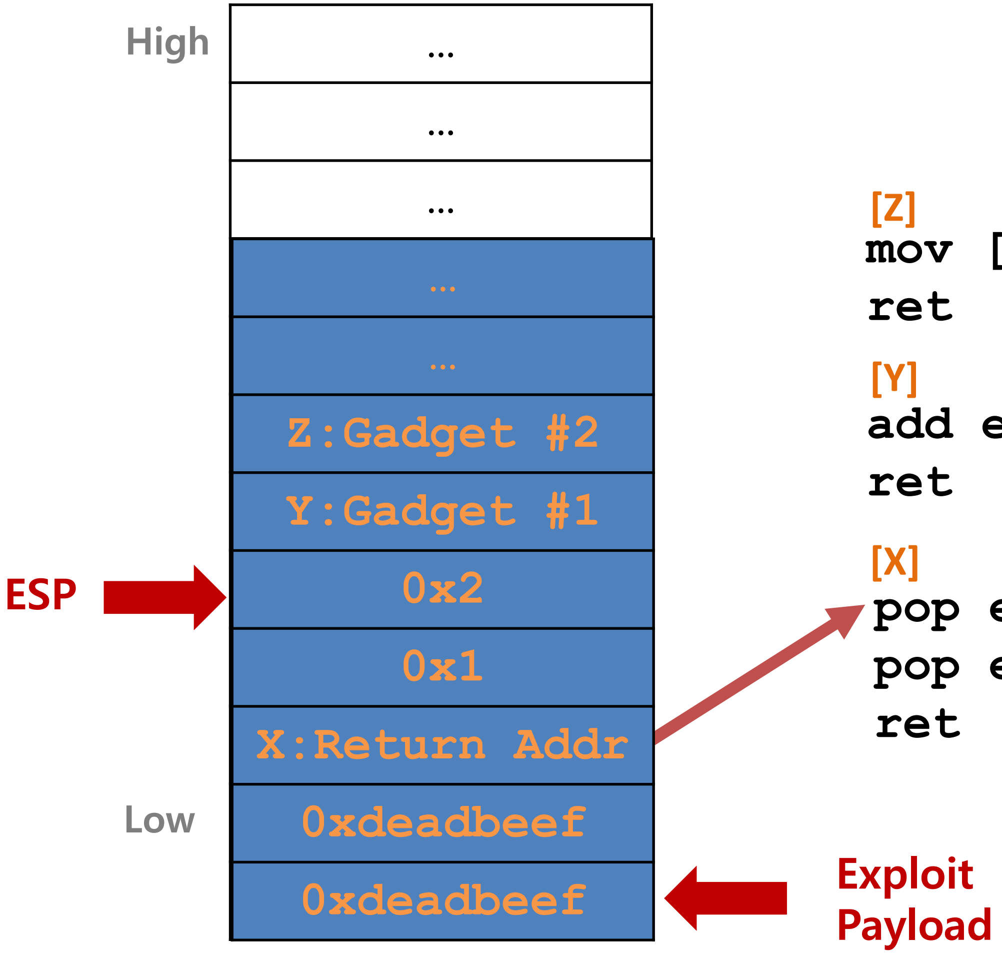


[Z]
```
mov [ebx], eax
ret
```

[Y]
```
add eax, ebx
ret
```

[X]
```
pop eax
pop ebx
ret
```

Memory Corruption: ROP Concept



[Z]

```
mov [ebx], eax
ret
```

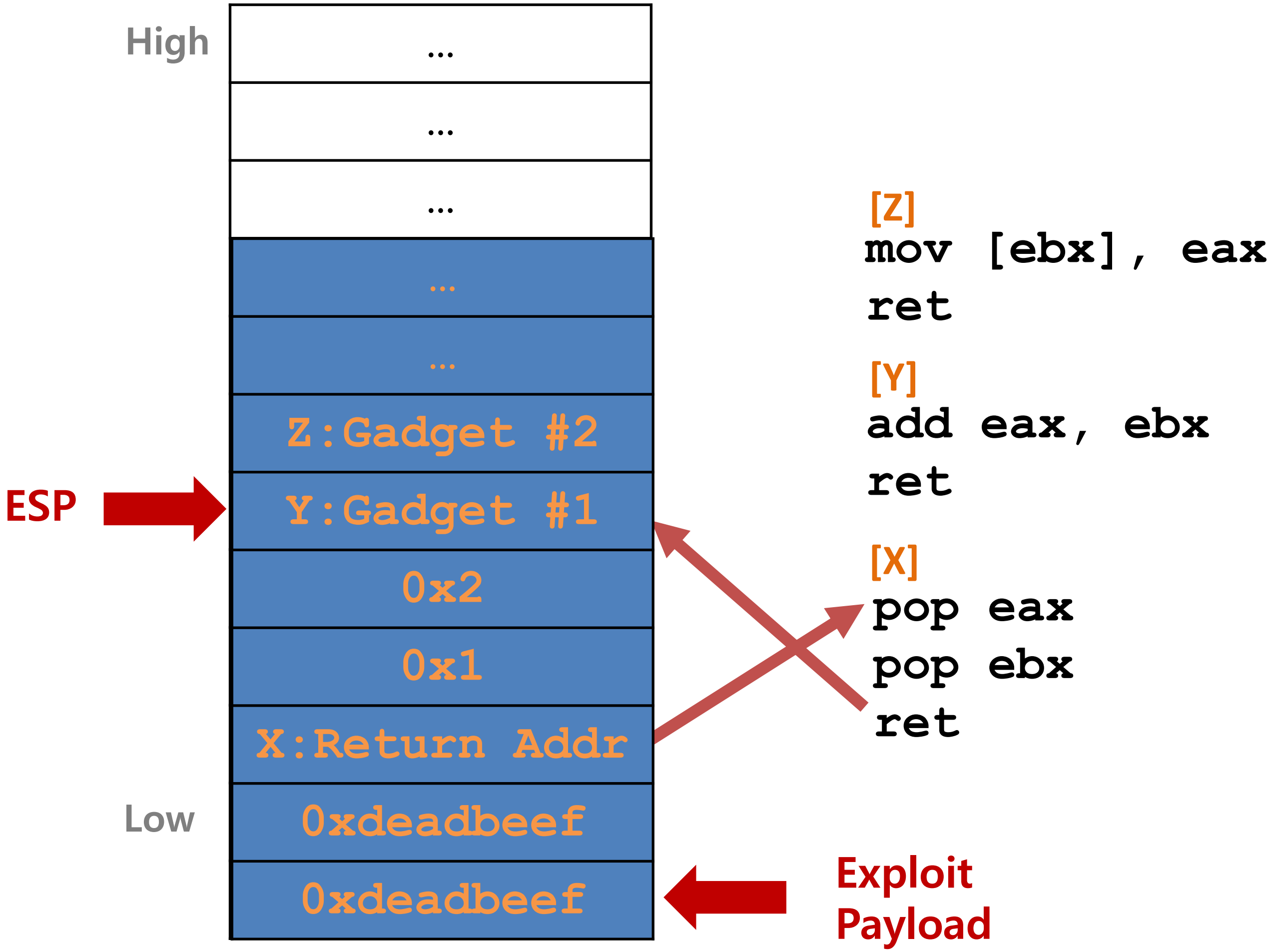
[Y]

```
add eax, ebx
ret
```

[X]

```
pop eax
pop ebx
ret
```

Memory Corruption: ROP Concept



High

...

...

...

...

...

Z:Gadget #2

ESP → Y:Gadget #1

0x2

0x1

X:Return Addr

Low

0xdeadbeef

0xdeadbeef ← Exploit Payload

[Z]

```
mov [ebx], eax
ret
```

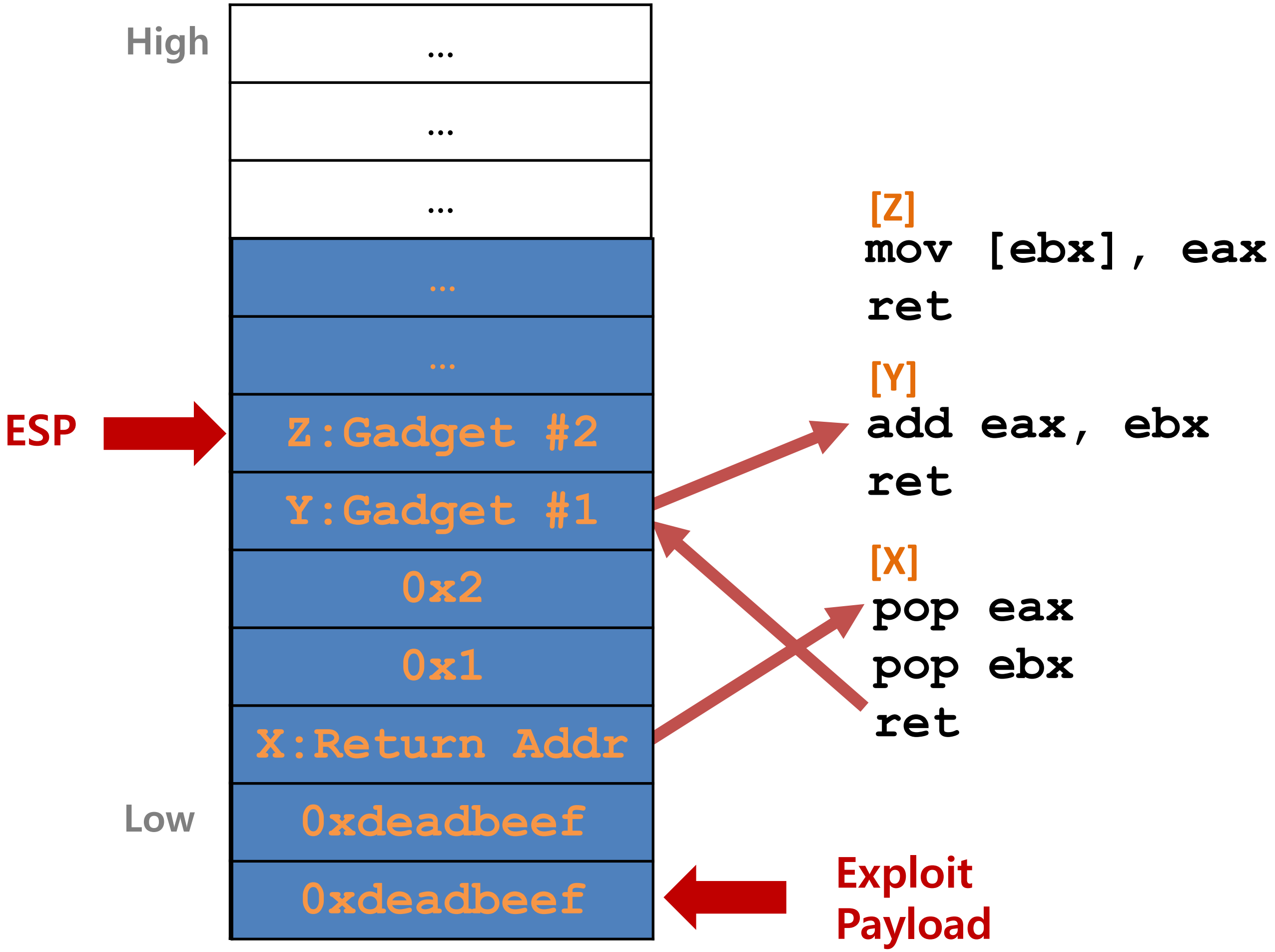
[Y]

```
add eax, ebx
ret
```

[X]

```
pop eax
pop ebx
ret
```

Memory Corruption: ROP Concept

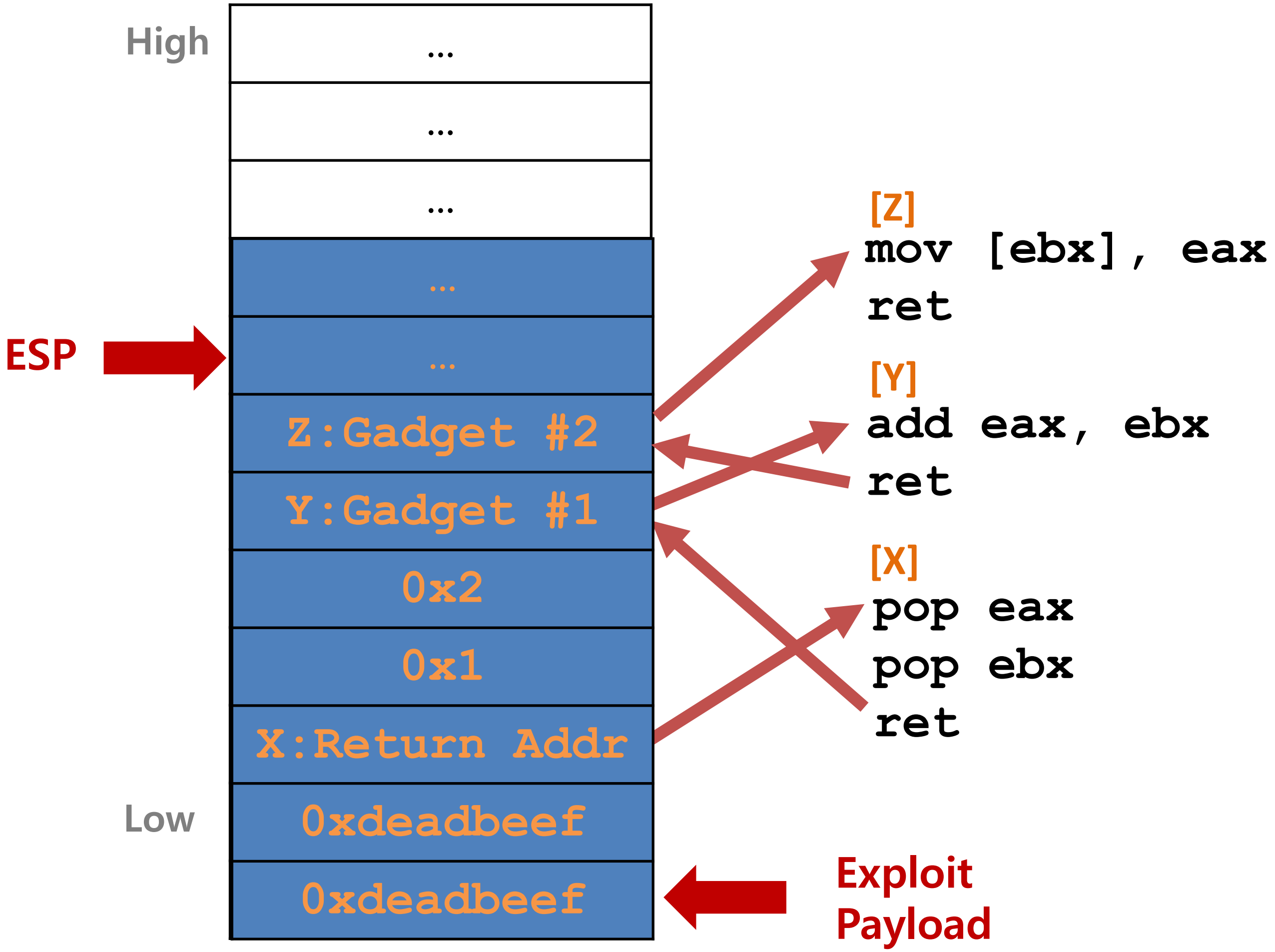


[Z]
```
mov [ebx], eax
ret
```

[Y]
```
add eax, ebx
ret
```

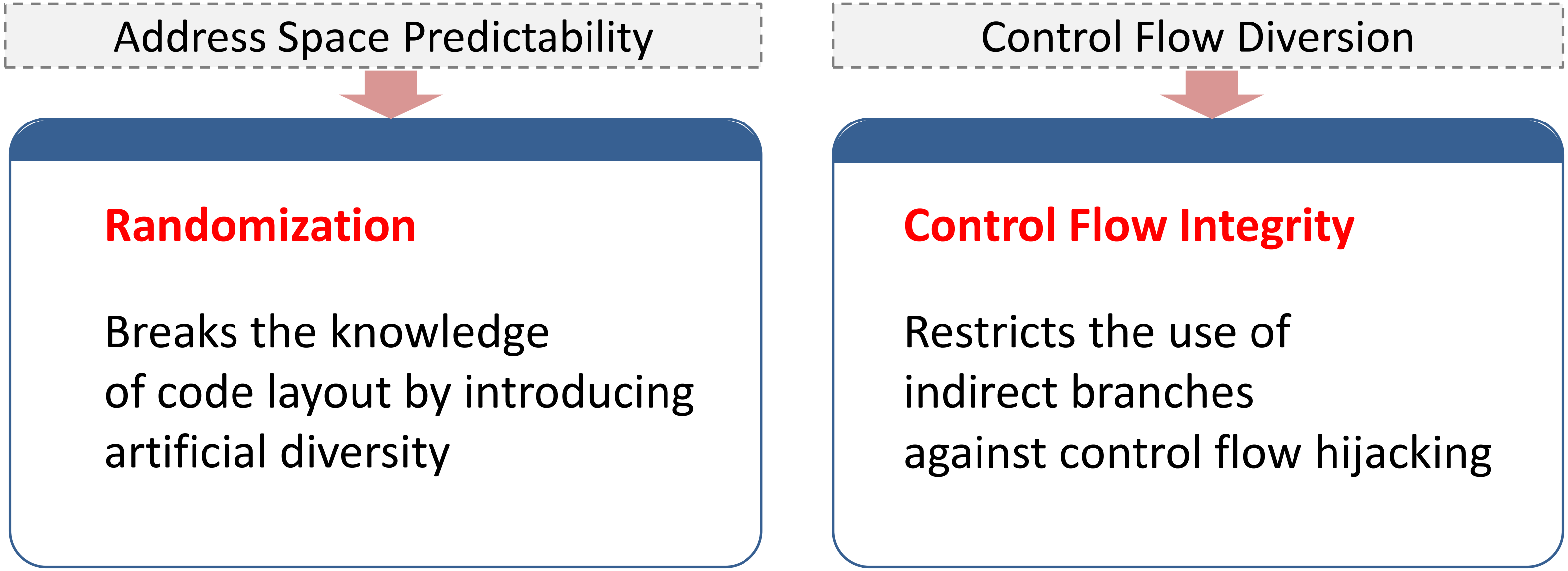
[X]
```
pop eax
pop ebx
ret
```

Memory Corruption: ROP Concept



ROP Defenses

- Two main approaches



Address Space Predictability

Randomization

Breaks the knowledge
of code layout by introducing
artificial diversity

Control Flow Diversion

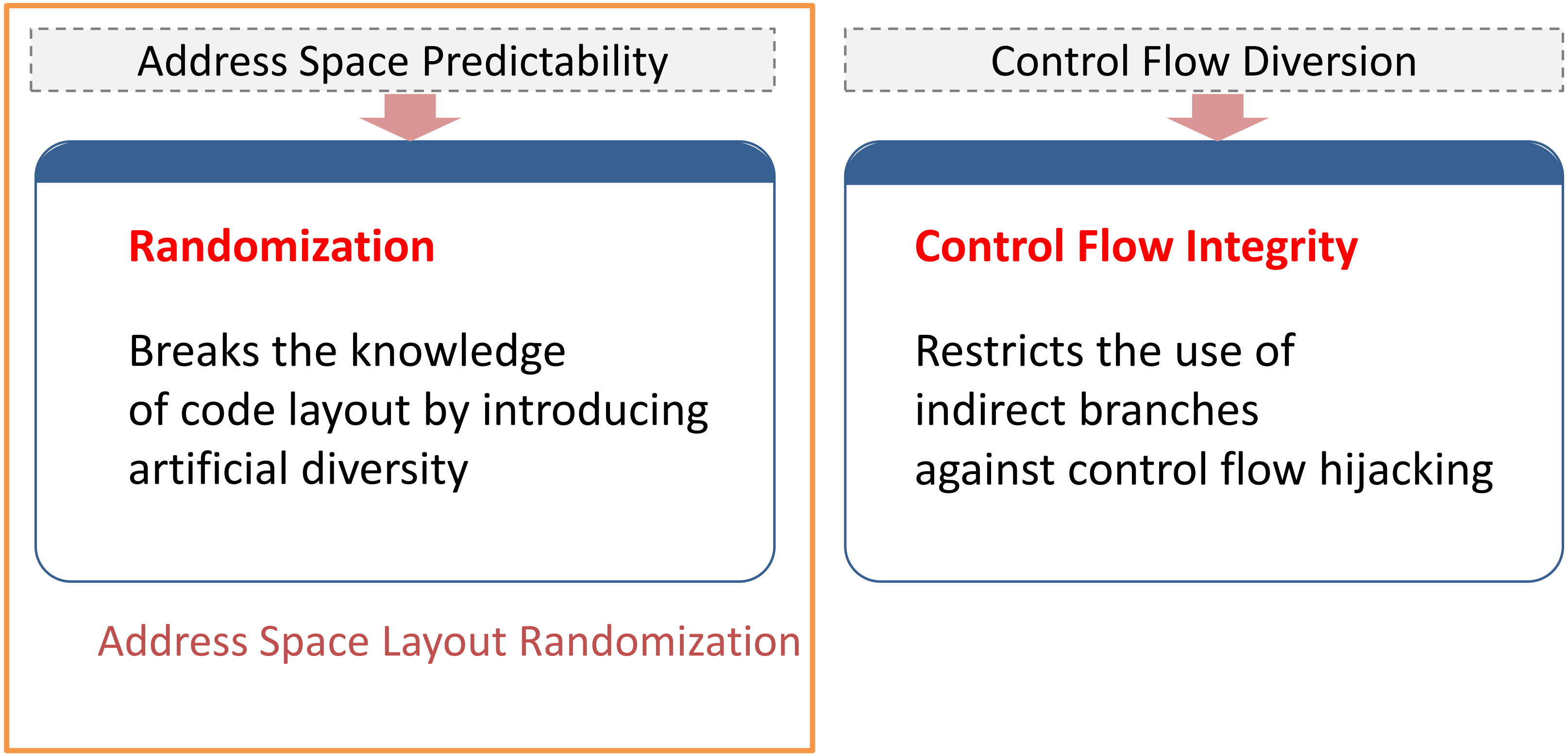
Control Flow Integrity

Restricts the use of
indirect branches
against control flow hijacking

Address Space Layout Randomization

ROP Defenses

- Two main approaches



Address Space Predictability

Randomization

Breaks the knowledge of code layout by introducing artificial diversity

Address Space Layout Randomization

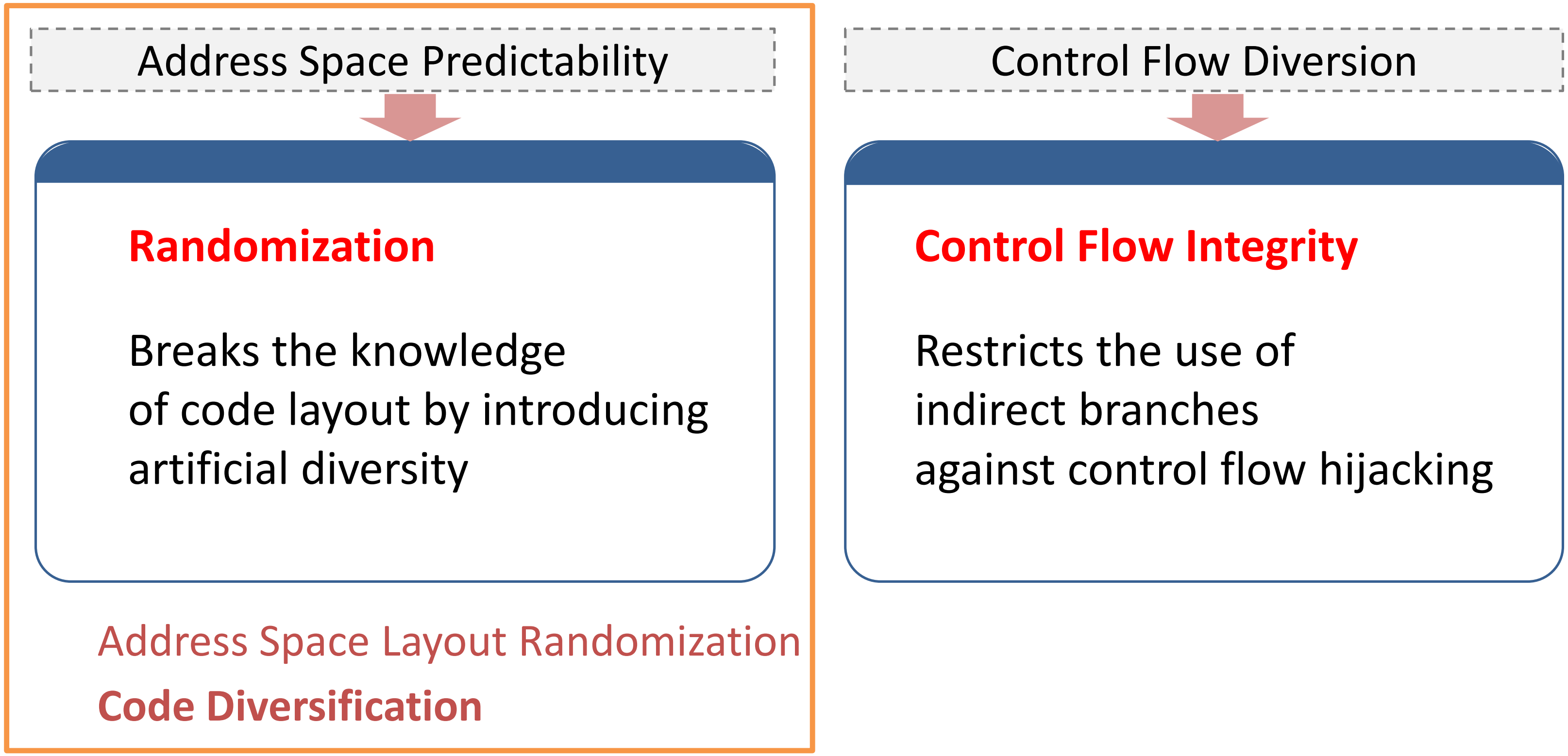
Control Flow Diversion

Control Flow Integrity

Restricts the use of indirect branches against control flow hijacking

ROP Defenses

- Two main approaches



Address Space Predictability

Randomization

Breaks the knowledge of code layout by introducing artificial diversity

Address Space Layout Randomization

Code Diversification

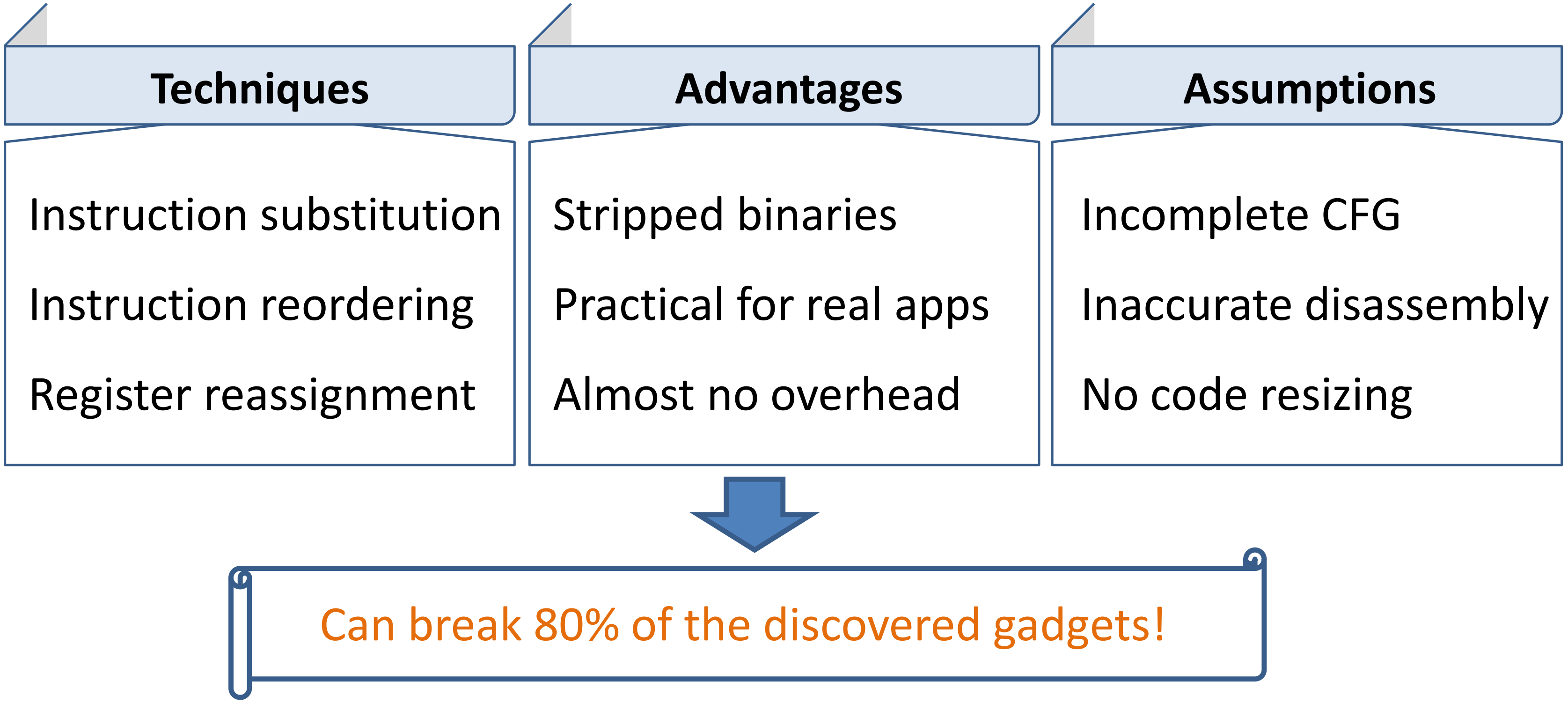
Control Flow Diversion

Control Flow Integrity

Restricts the use of indirect branches against control flow hijacking

Code Transformation

- Previous Work: In-Place Randomization (IPR)



Techniques

Instruction substitution

Instruction reordering

Register reassignment

Advantages

Stripped binaries

Practical for real apps

Almost no overhead

Assumptions

Incomplete CFG

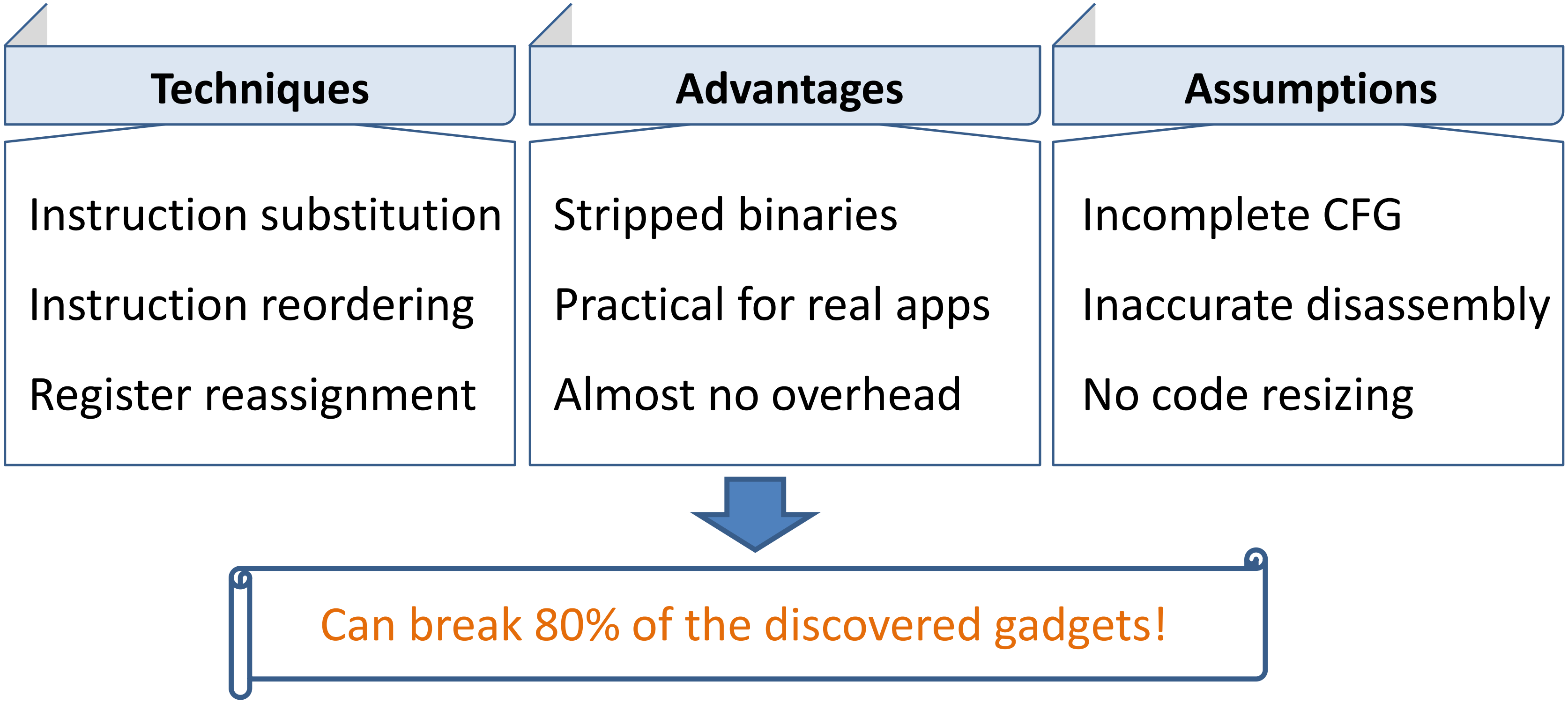
Inaccurate disassembly

No code resizing

Can break 80% of the discovered gadgets!

Code Transformation

- Previous Work: In-Place Randomization (IPR)



Techniques	Advantages	Assumptions
Instruction substitution	Stripped binaries	Incomplete CFG
Instruction reordering	Practical for real apps	Inaccurate disassembly
Register reassignment	Almost no overhead	No code resizing

Can break 80% of the discovered gadgets!

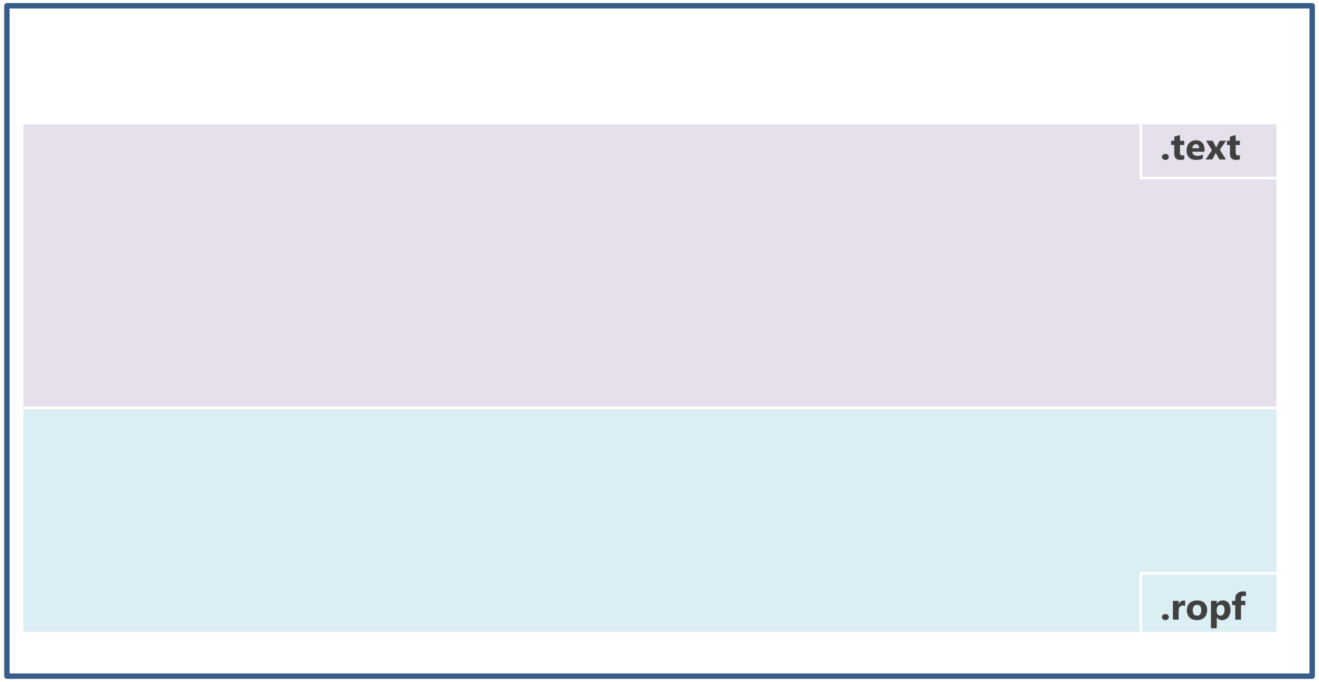
The remaining gadgets (20%) may still be enough for the construction of a functional ROP payload!

Our Work

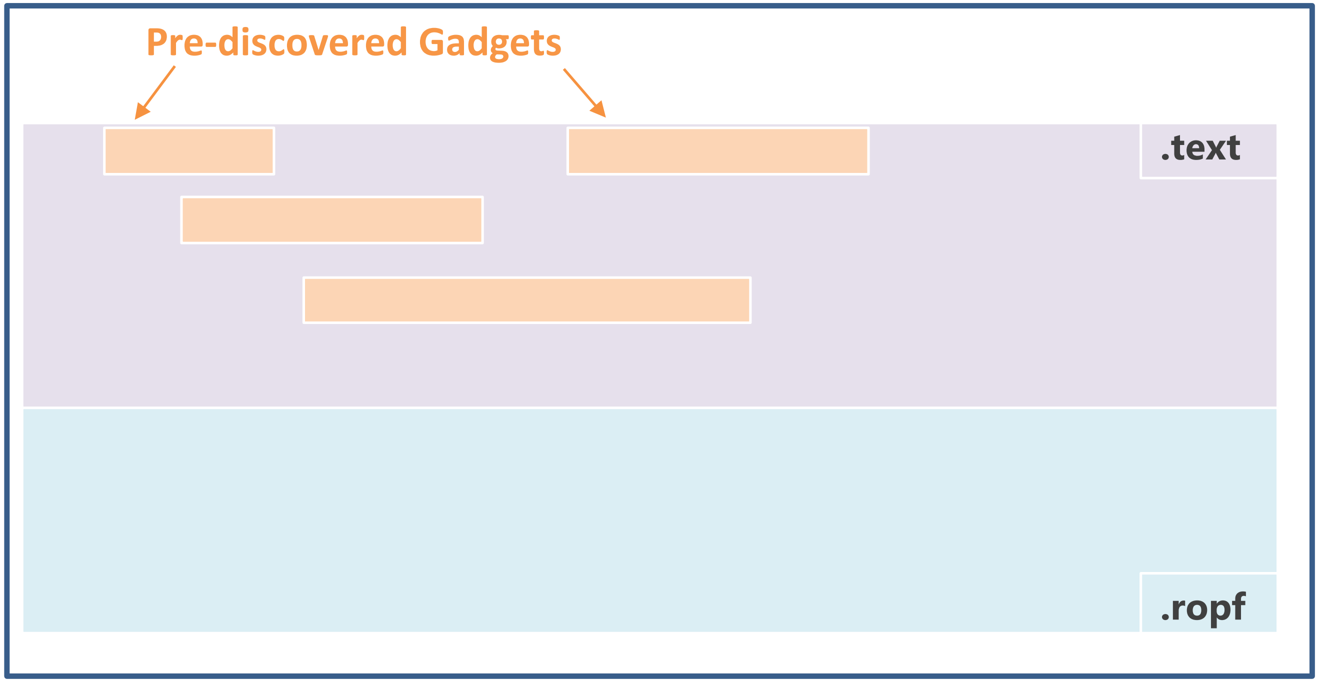
- Idea: breaking gadgets by displacing them
- Goal: maximize the gadget coverage on top of IPR
- **Highly practical: can be applied on stripped binaries**

- Assume an adversary has the power of ROP:
  - ✓ Functional payload with initial hijacking and memory disclosure
  - ✓ Existing protections (DEP/ASLR) are enabled
  - ✓ **Attacker does not have access to the randomized binary**

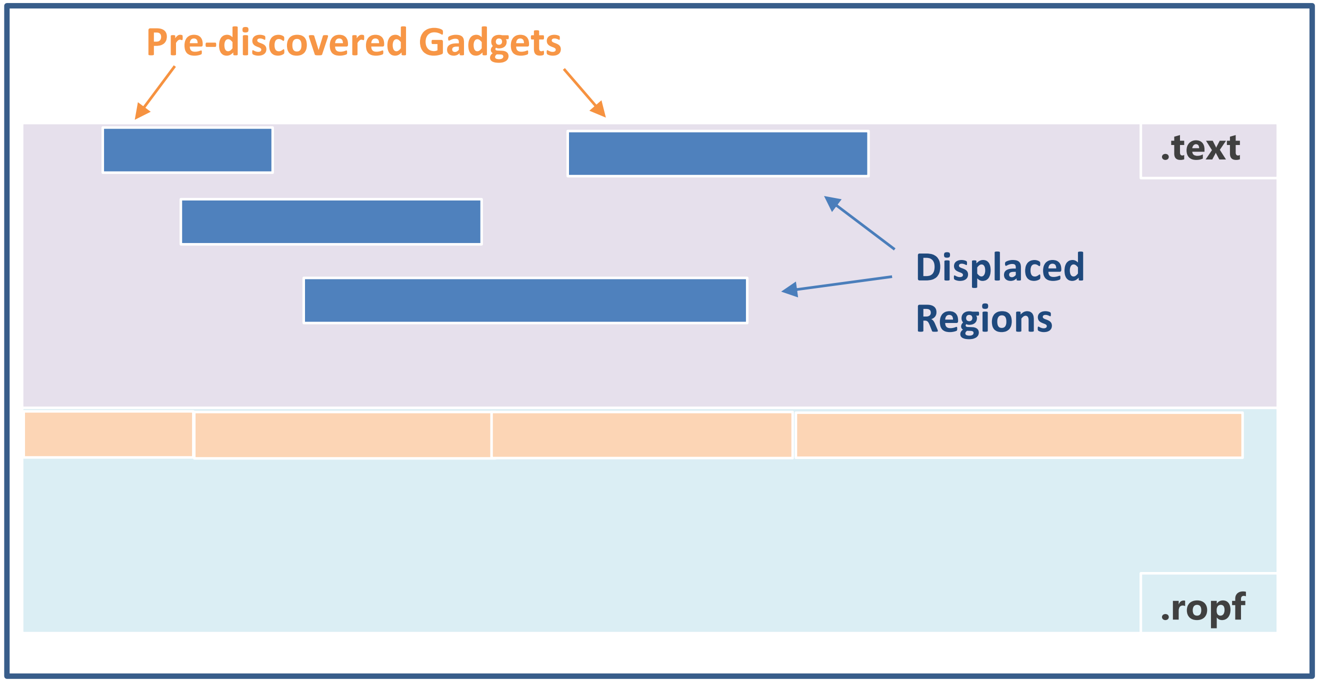
High Level View of Gadget Displacement



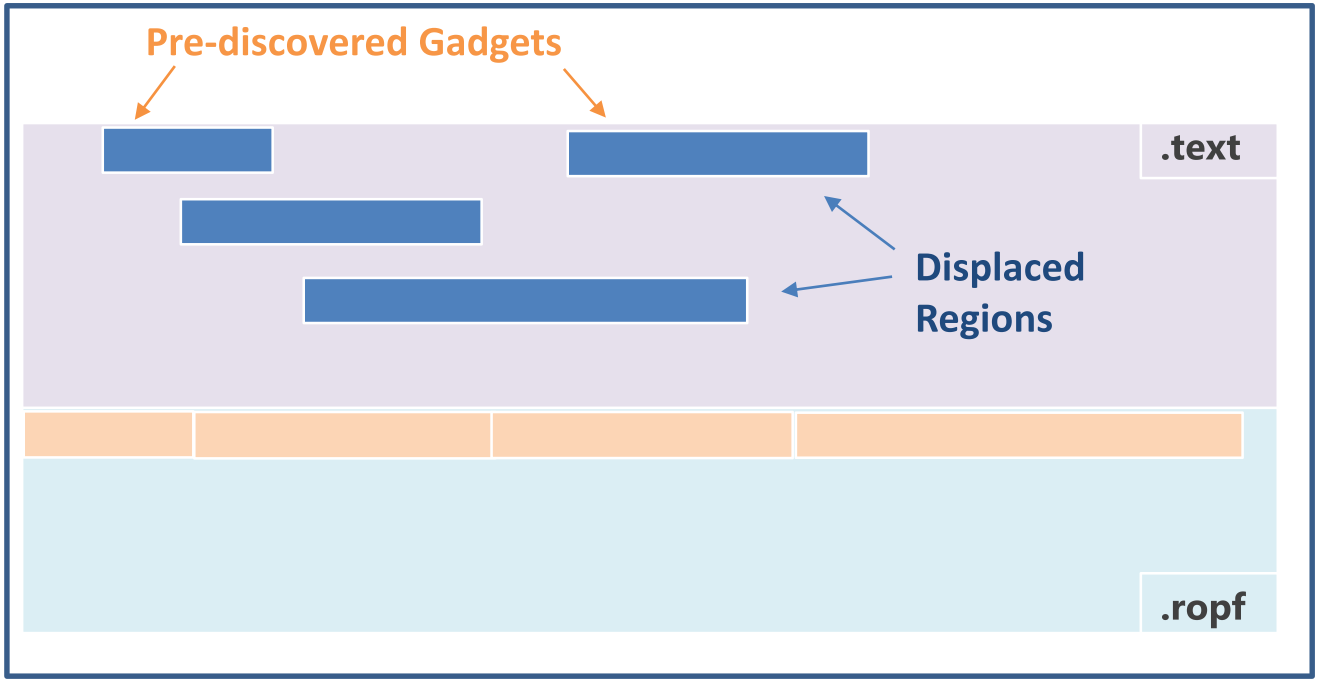
High Level View of Gadget Displacement



High Level View of Gadget Displacement



High Level View of Gadget Displacement



Basic Block (BBK)

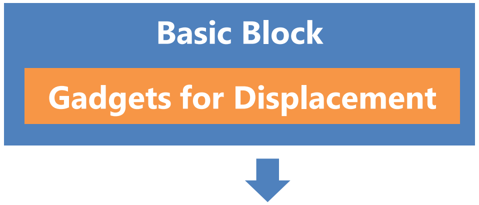
Need *jmp* instructions

Displaced regions (>=5B)

```
jmp [rel-addr]
int 3
```

Intended vs. Unintended Gadgets

Intended vs. Unintended Gadgets

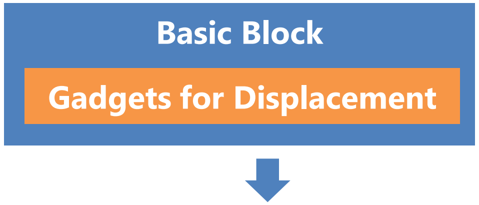


Basic Block

Gadgets for Displacement

```
.text:070082D6 E8 D2 FF FF FF          call     sub_70082AD
.text:070082DB C7 06 88 09 01 07       mov      dword ptr [esi], offset 7010988
.text:070082E1 8B C6                   mov      eax, esi
.text:070082E3 5E                      pop      esi
.text:070082E4 C3                      retn
```

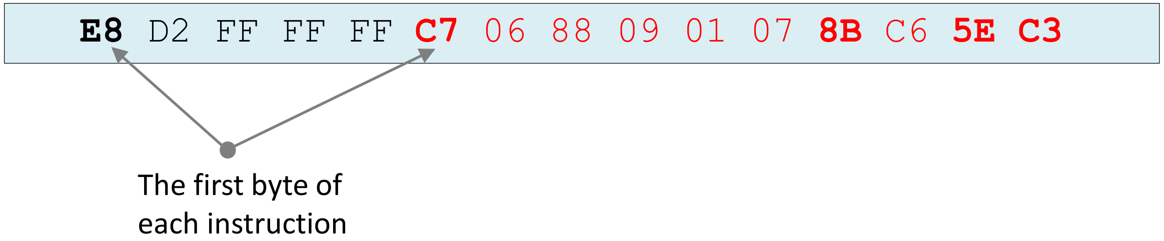
Intended vs. Unintended Gadgets



Basic Block

Gadgets for Displacement

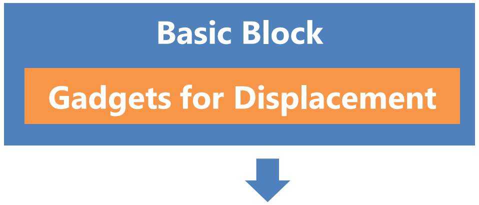
```
.text:070082D6 E8 D2 FF FF FF       call    sub_70082AD
.text:070082DB C7 06 88 09 01 07    mov     dword ptr [esi], offset 7010988
.text:070082E1 8B C6                mov     eax, esi
.text:070082E3 5E                   pop     esi
.text:070082E4 C3                   retn
```



```
E8 D2 FF FF FF C7 06 88 09 01 07 8B C6 5E C3
```

The first byte of each instruction

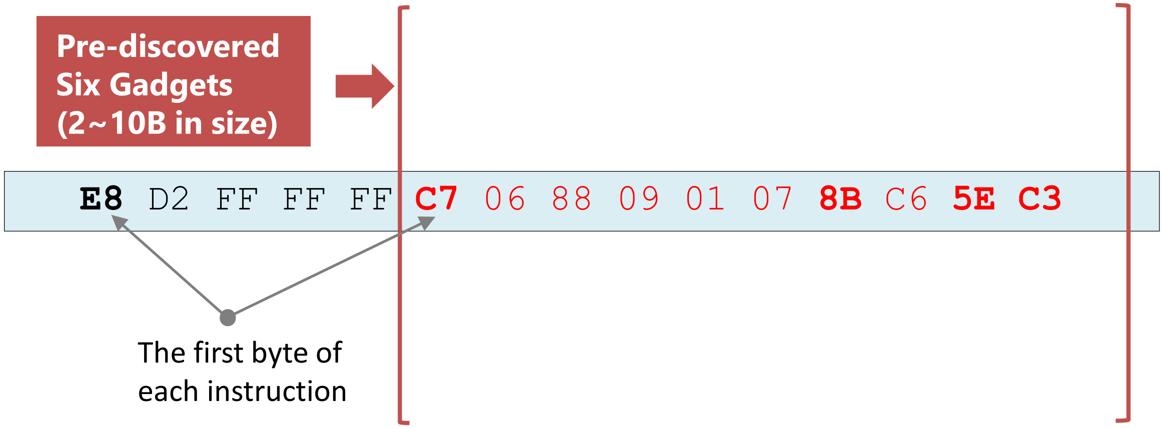
Intended vs. Unintended Gadgets



Basic Block

Gadgets for Displacement

```
.text:070082D6 E8 D2 FF FF FF          call     sub_70082AD
.text:070082DB C7 06 88 09 01 07       mov      dword ptr [esi], offset 7010988
.text:070082E1 8B C6                   mov      eax, esi
.text:070082E3 5E                      pop      esi
.text:070082E4 C3                      retn
```



Pre-discovered Six Gadgets (2~10B in size)

```
E8 D2 FF FF FF C7 06 88 09 01 07 8B C6 5E C3
```

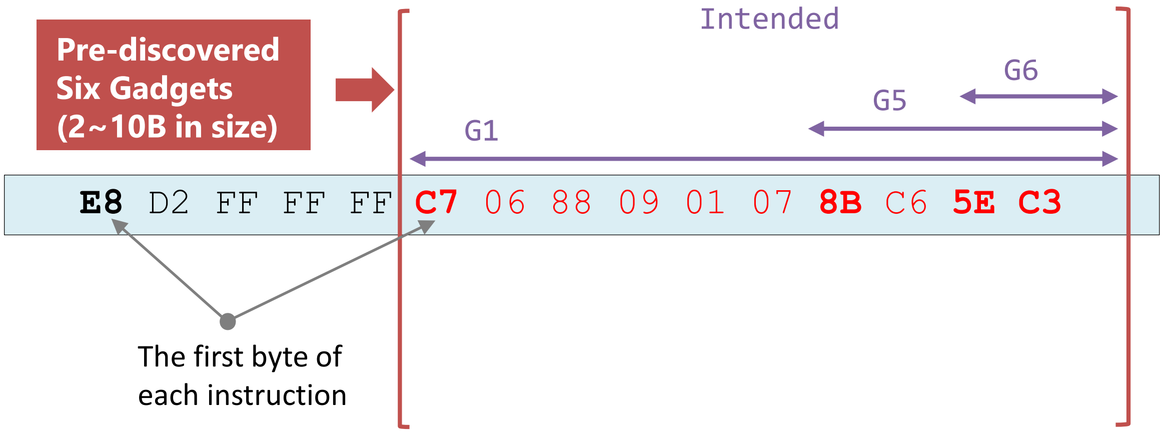
The first byte of each instruction

Intended vs. Unintended Gadgets

Basic Block

Gadgets for Displacement

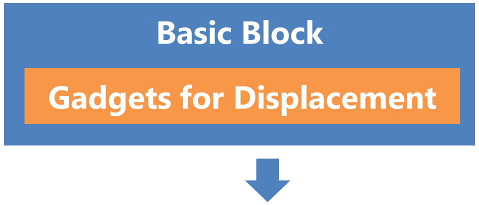
```
.text:070082D6 E8 D2 FF FF FF           call     sub_70082AD
.text:070082DB C7 06 88 09 01 07        mov      dword ptr [esi], offset 7010988
.text:070082E1 8B C6                    mov      eax, esi
.text:070082E3 5E                       pop      esi
.text:070082E4 C3                       retn
```



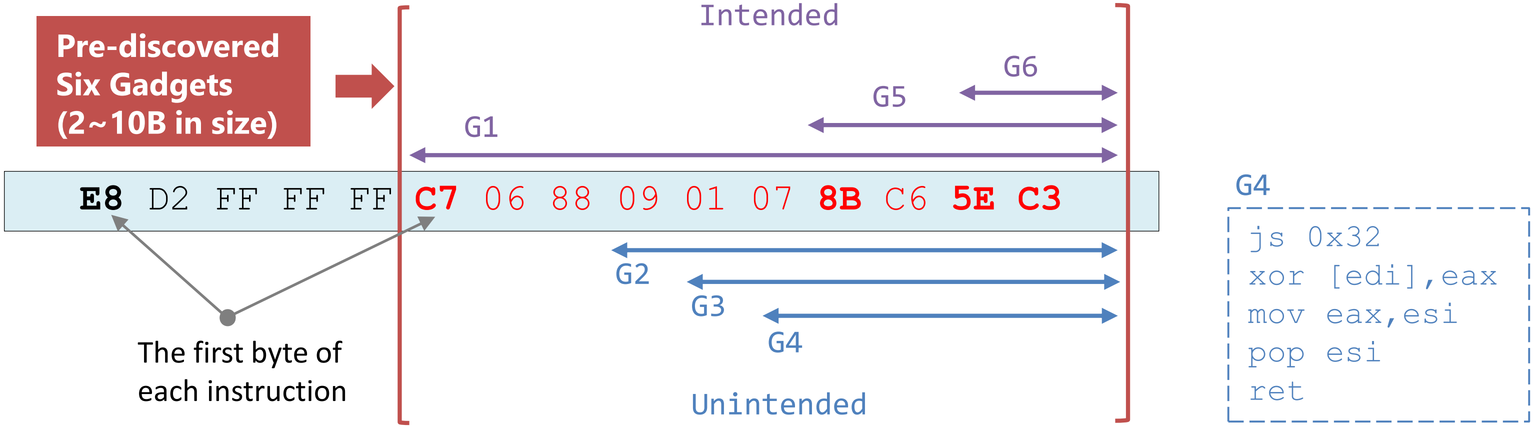
Pre-discovered Six Gadgets (2~10B in size)

The first byte of each instruction

Intended vs. Unintended Gadgets



```
.text:070082D6 E8 D2 FF FF FF        call     sub_70082AD
.text:070082DB C7 06 88 09 01 07     mov      dword ptr [esi], offset 7010988
.text:070082E1 8B C6                 mov      eax, esi
.text:070082E3 5E                    pop      esi
.text:070082E4 C3                    retn
```


Pre-discovered Six Gadgets (2~10B in size)
Intended
G6
G5
G1
E8 D2 FF FF FF C7 06 88 09 01 07 8B C6 5E C3
G2
G3
G4
The first byte of each instruction
Unintended
G4

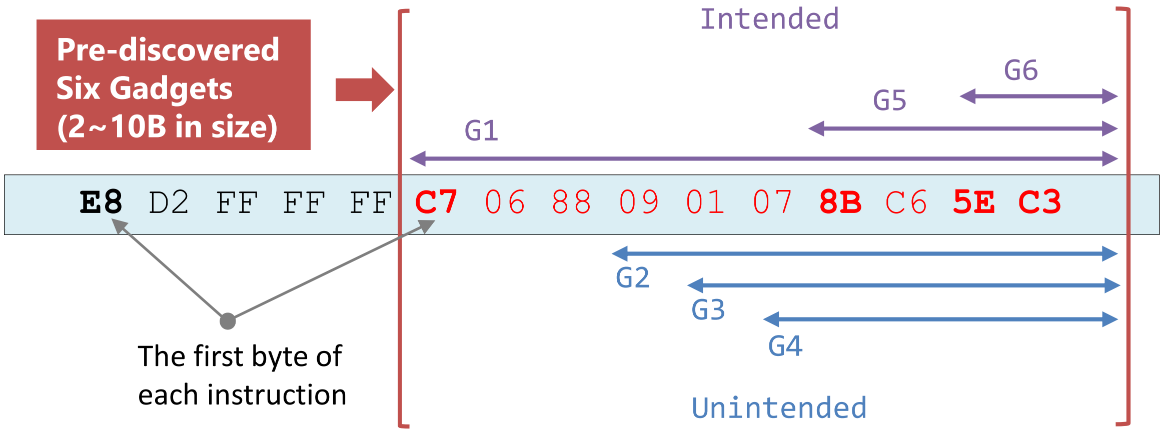
```
js 0x32
xor [edi],eax
mov eax,esi
pop esi
ret
```

Intended vs. Unintended Gadgets

Basic Block

Gadgets for Displacement

```
.text:070082D6 E8 D2 FF FF FF          call     sub_70082AD
.text:070082DB C7 06 88 09 01 07       mov      dword ptr [esi], offset 7010988
.text:070082E1 8B C6                   mov      eax, esi
.text:070082E3 5E                      pop      esi
.text:070082E4 C3                      retn
```


Pre-discovered Six Gadgets (2~10B in size)
Intended
G6
G5
G1
E8 D2 FF FF FF C7 06 88 09 01 07 8B C6 5E C3
G2
G3
G4
The first byte of each instruction
Unintended

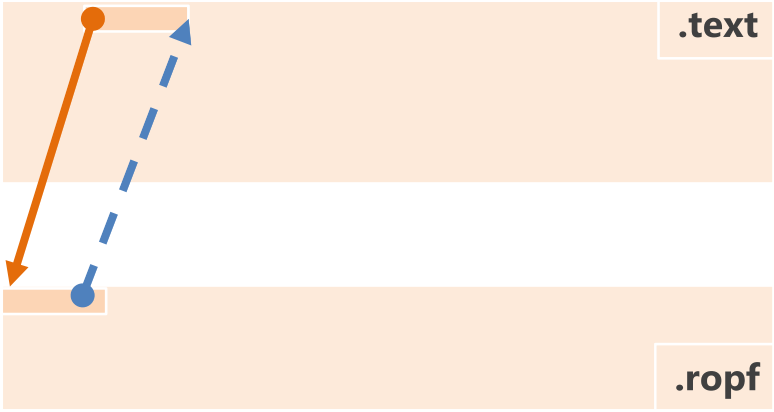
G4

```
js 0x32
xor [edi],eax
mov eax,esi
pop esi
ret
```

✓ Nested in nature

Requirements for Displacement

❖ Maintain the original code semantics



✓ 5-byte long space to insert *jmp* instruction

✓ Recalculate code references
- *branches* and *calls* with relative addresses

✓ Update all relocation entries

Displacement Strategy

- ✓ Paired jump instructions for every displacement?
- ✓ Keep the number of displaced regions low
- ✓ For unintended gadgets
- ✓ For intended gadgets
- ✓ Avoid generating the same binary

Displacement Strategy

- ✓ Paired jump instructions for every displacement?

  No (Needless for unconditional “*JMP*” and “*RET*”)

- ✓ Keep the number of displaced regions low

- ✓ For unintended gadgets

- ✓ For intended gadgets

- ✓ Avoid generating the same binary

Displacement Strategy

✓ Paired jump instructions for every displacement?

No (Needless for unconditional “*JMP*” and “*RET*”)

✓ Keep the number of displaced regions low

Select the largest gadget to break the *nested* ones

✓ For unintended gadgets

✓ For intended gadgets

✓ Avoid generating the same binary

Displacement Strategy

✓ Paired jump instructions for every displacement?
No (Needless for unconditional “*JMP*” and “*RET*”)

✓ Keep the number of displaced regions low
Select the largest gadget to break the *nested* ones

✓ For unintended gadgets
Find the starting byte of the first intended instruction of the gadget

✓ For intended gadgets

✓ Avoid generating the same binary

Displacement Strategy

- ✓ Paired jump instructions for every displacement?

  No (Needless for unconditional “*JMP*” and “*RET*”)

- ✓ Keep the number of displaced regions low

  Select the largest gadget to break the *nested* ones

- ✓ For unintended gadgets

  Find the starting byte of the first intended instruction of the gadget

- ✓ For intended gadgets

  Find the instruction all the way back in the same BBK

- ✓ Avoid generating the same binary

Displacement Strategy

✓ Paired jump instructions for every displacement?

No (Needless for unconditional "*JMP*" and "*RET*")

✓ Keep the number of displaced regions low

Select the largest gadget to break the *nested* ones

✓ For unintended gadgets

Find the starting byte of the first intended instruction of the gadget

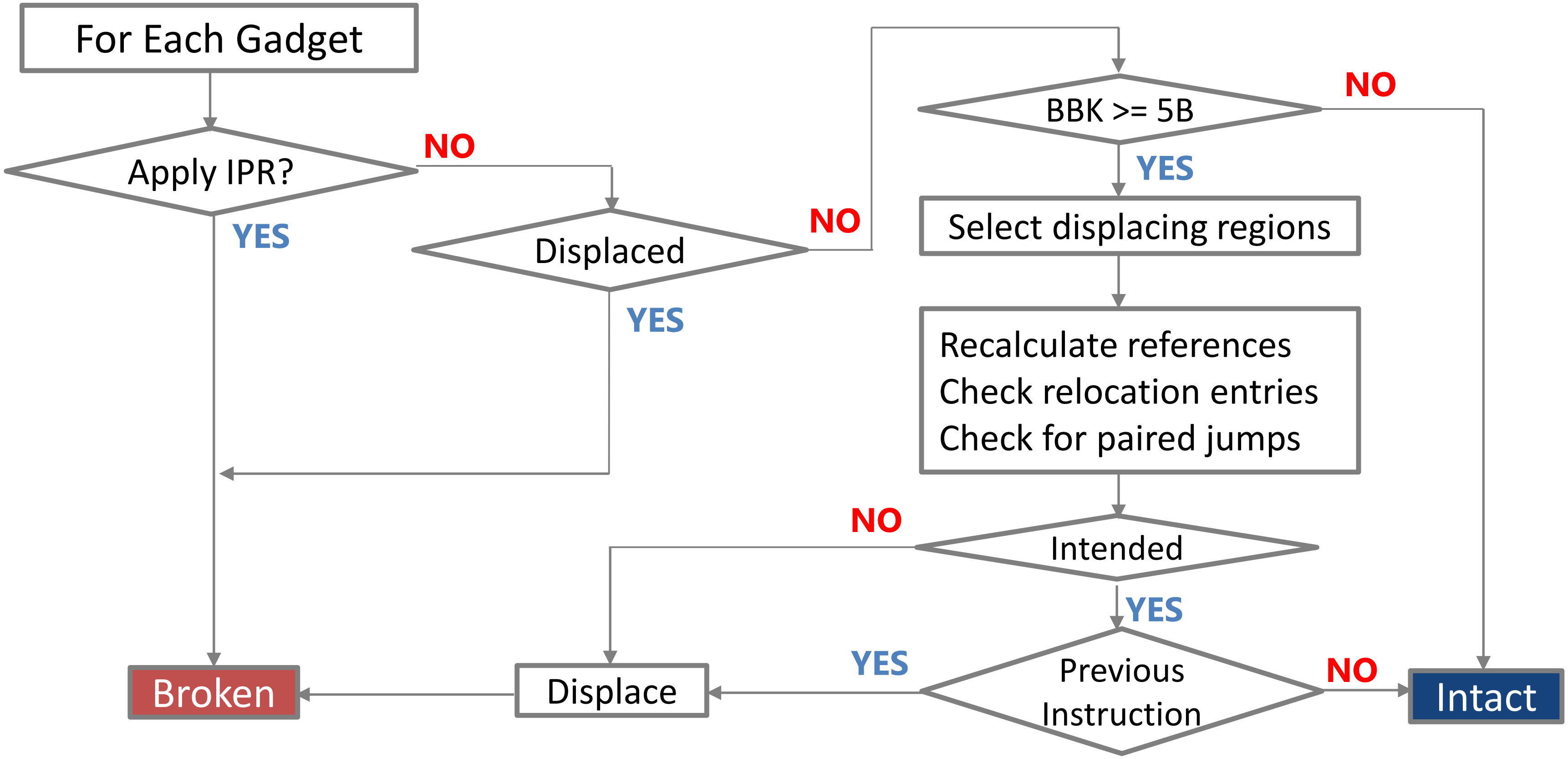
✓ For intended gadgets

Find the instruction all the way back in the same BBK

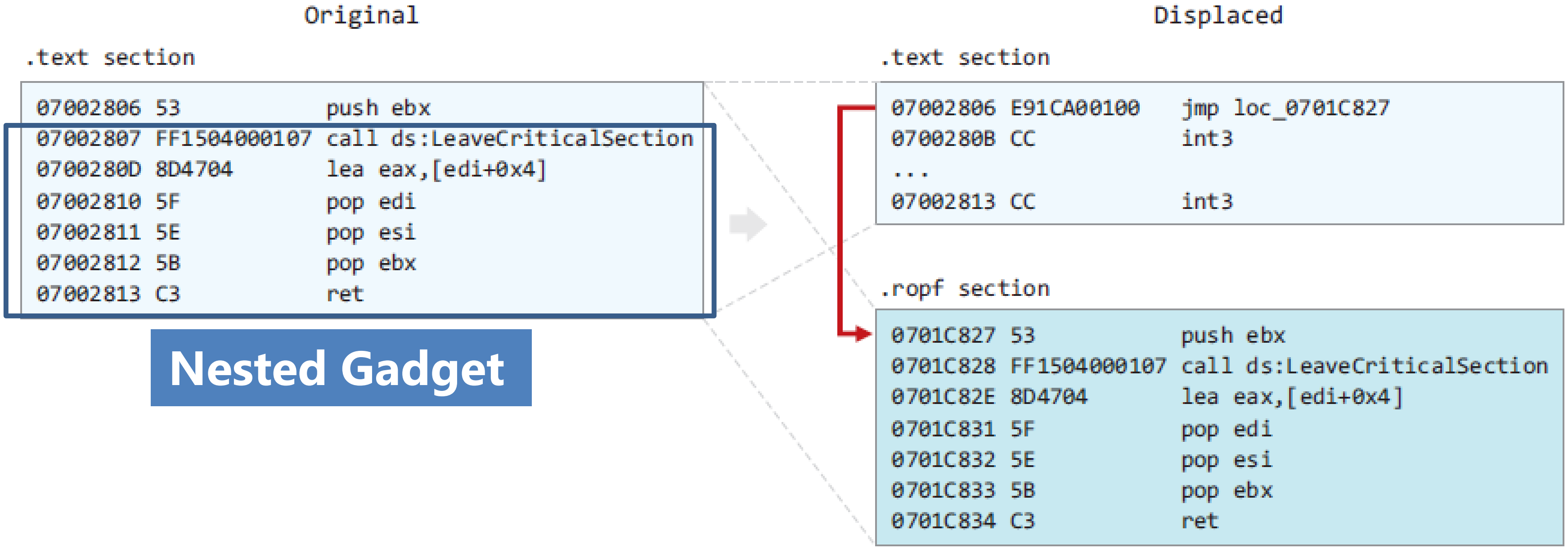
✓ Avoid generating the same binary

Randomized placement of the displaced instructions

Displacement Algorithm



Displacement Example



Original

.text section

```
07002806 53          push ebx
07002807 FF1504000107 call ds:LeaveCriticalSection
0700280D 8D4704      lea eax,[edi+0x4]
07002810 5F          pop edi
07002811 5E          pop esi
07002812 5B          pop ebx
07002813 C3          ret
```

Nested Gadget

Displaced

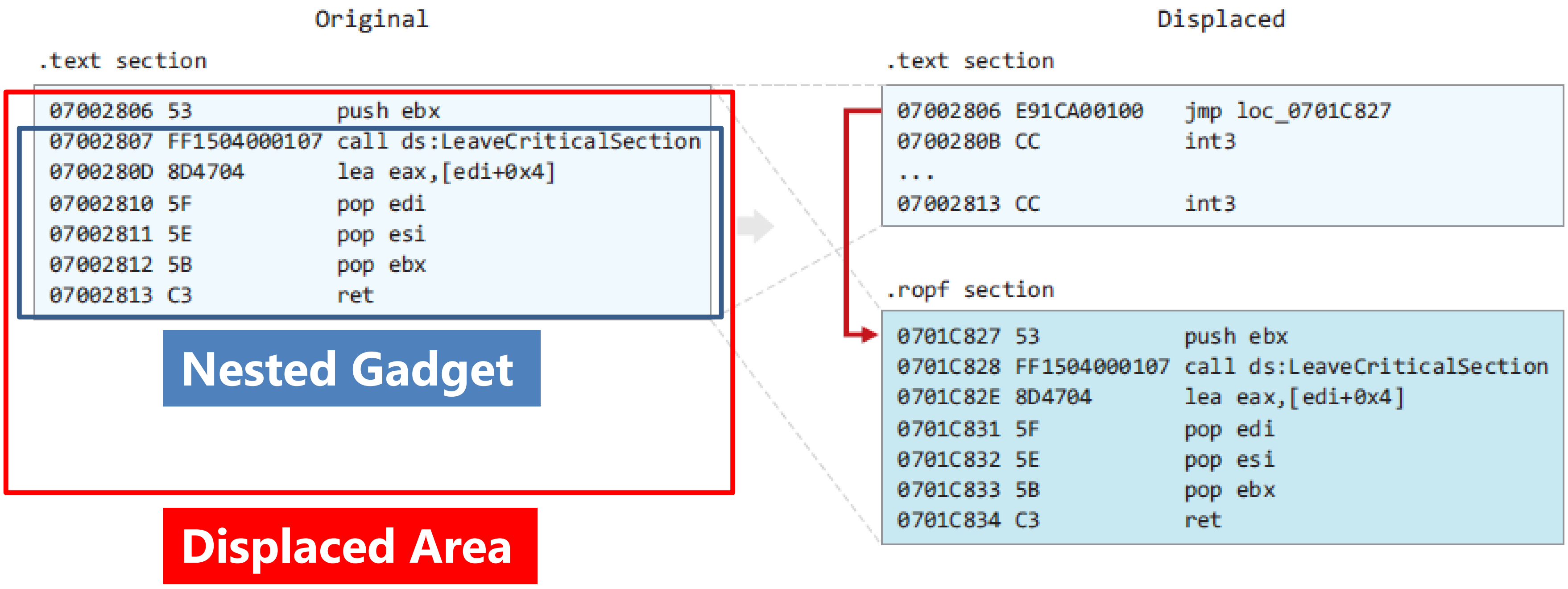
.text section

```
07002806 E91CA00100   jmp loc_0701C827
0700280B CC           int3
...
07002813 CC           int3
```

.ropf section

```
0701C827 53           push ebx
0701C828 FF1504000107 call ds:LeaveCriticalSection
0701C82E 8D4704       lea eax,[edi+0x4]
0701C831 5F           pop edi
0701C832 5E           pop esi
0701C833 5B           pop ebx
0701C834 C3           ret
```

Displacement Example



Original

```
.text section
07002806 53           push ebx
07002807 FF1504000107 call ds:LeaveCriticalSection
0700280D 8D4704       lea eax,[edi+0x4]
07002810 5F           pop edi
07002811 5E           pop esi
07002812 5B           pop ebx
07002813 C3           ret
```

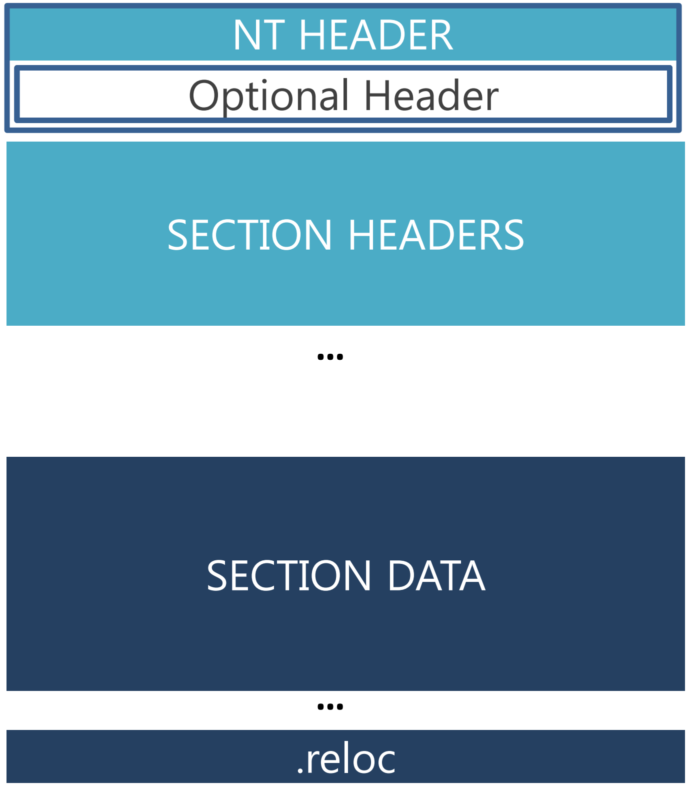
Displaced

```
.text section
07002806 E91CA00100   jmp loc_0701C827
0700280B CC           int3
...
07002813 CC           int3

.ropf section
0701C827 53           push ebx
0701C828 FF1504000107 call ds:LeaveCriticalSection
0701C82E 8D4704       lea eax,[edi+0x4]
0701C831 5F           pop edi
0701C832 5E           pop esi
0701C833 5B           pop ebx
0701C834 C3           ret
```

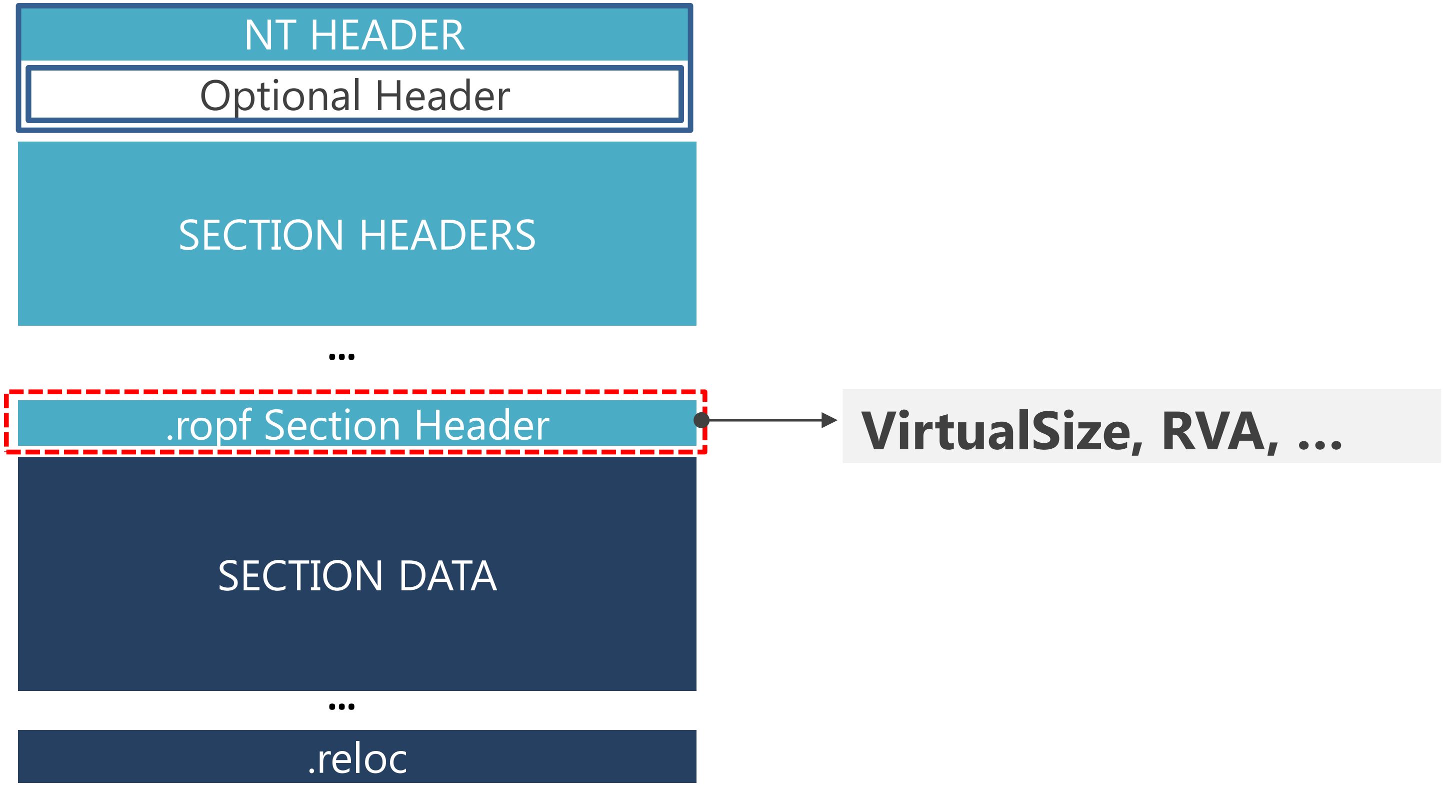
Binary Instrumentation (1/2)

- PE adjustment: headers and sections



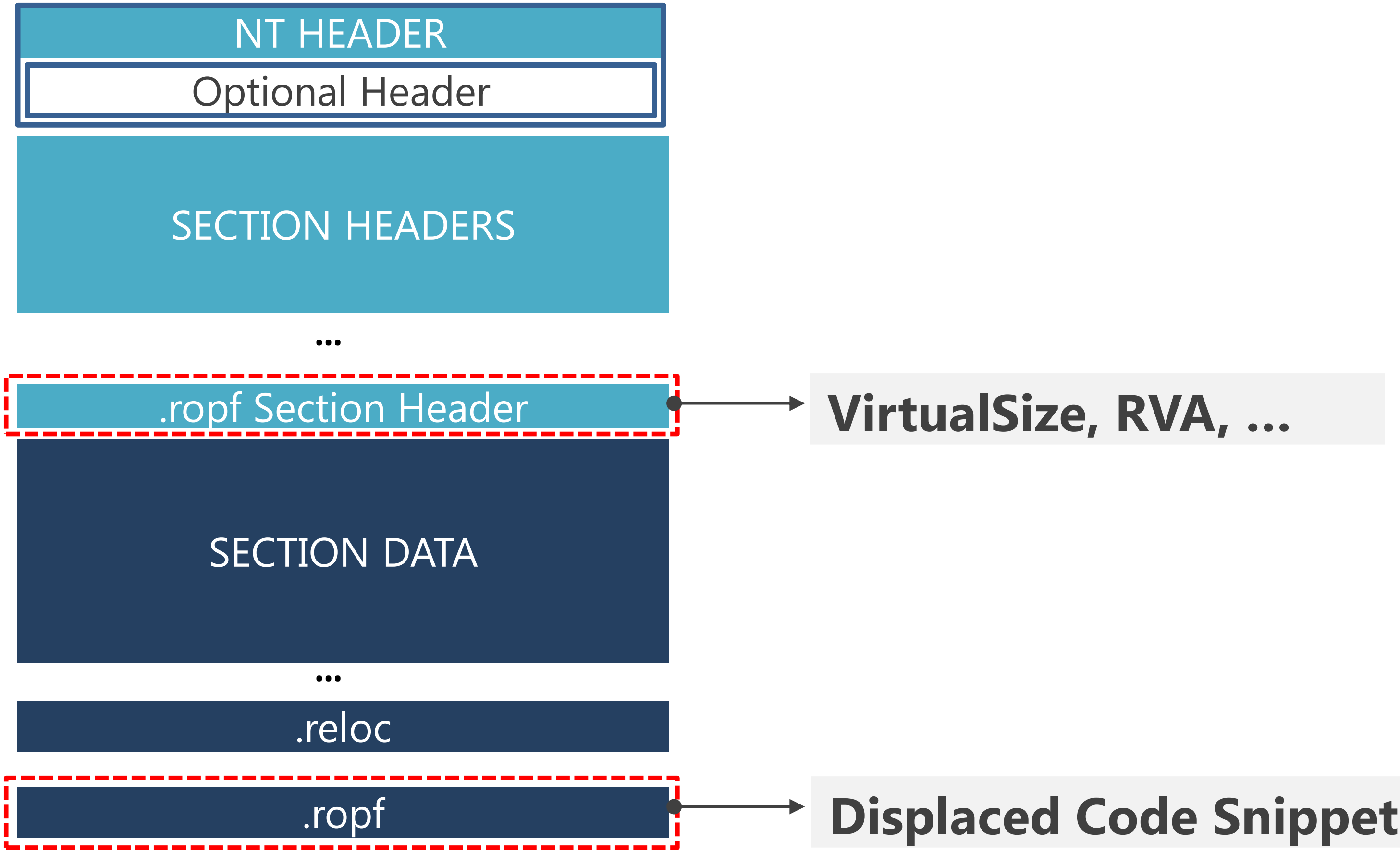
Binary Instrumentation (1/2)

- PE adjustment: headers and sections



Binary Instrumentation (1/2)

- PE adjustment: headers and sections

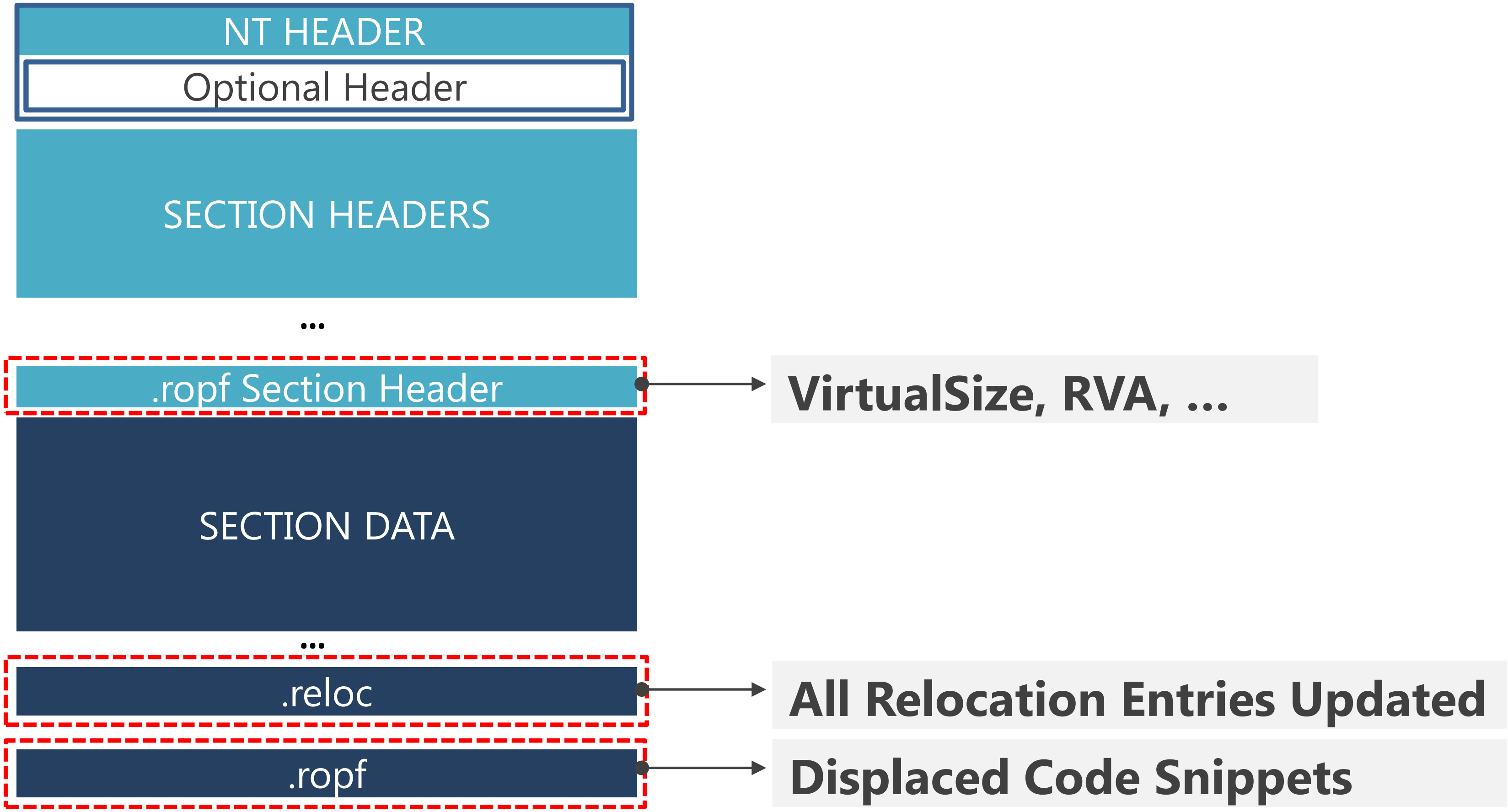


VirtualSize, RVA, …

Displaced Code Snippets

Binary Instrumentation (1/2)

- PE adjustment: headers and sections



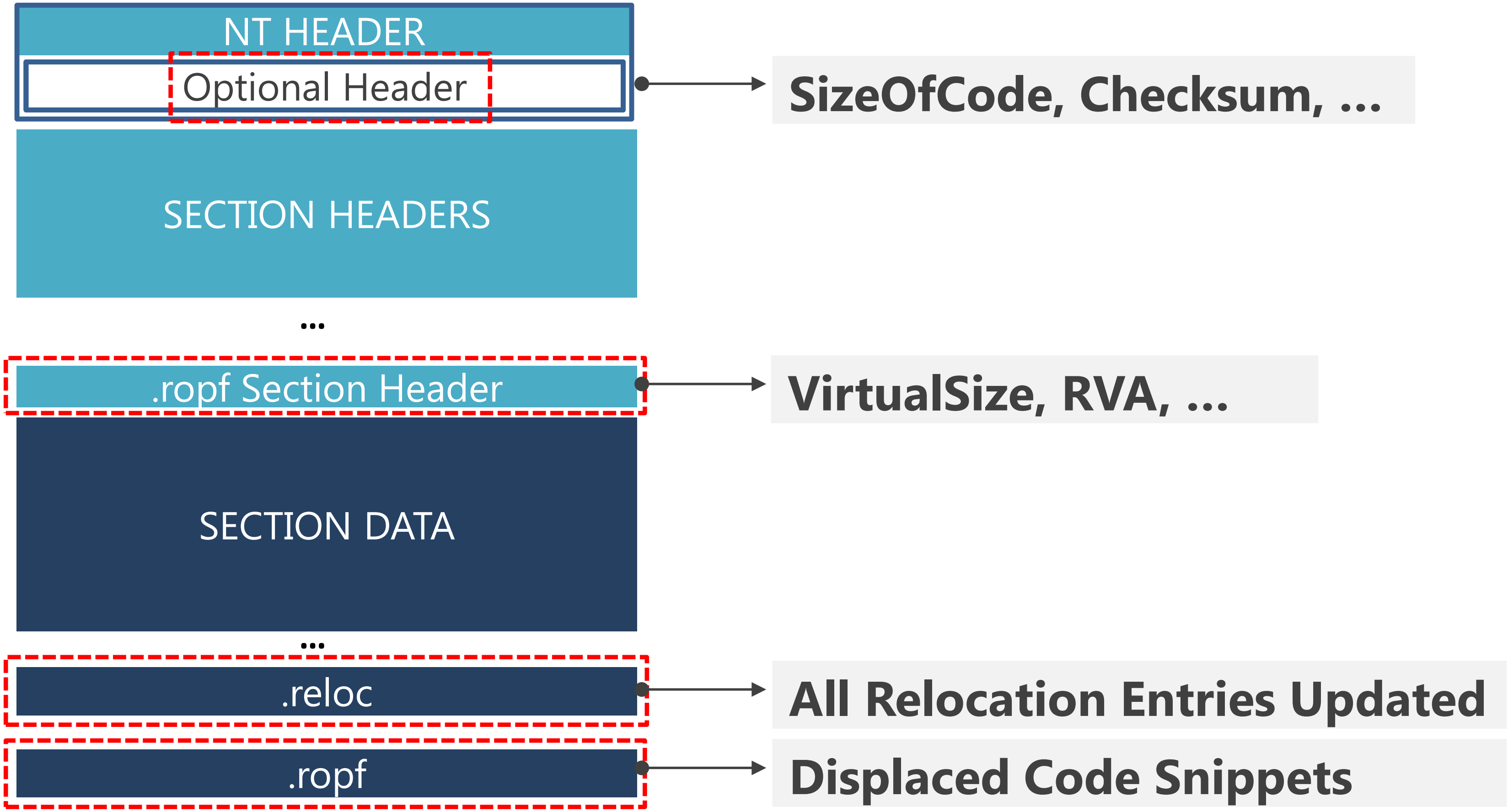
VirtualSize, RVA, …

All Relocation Entries Updated

Displaced Code Snippets

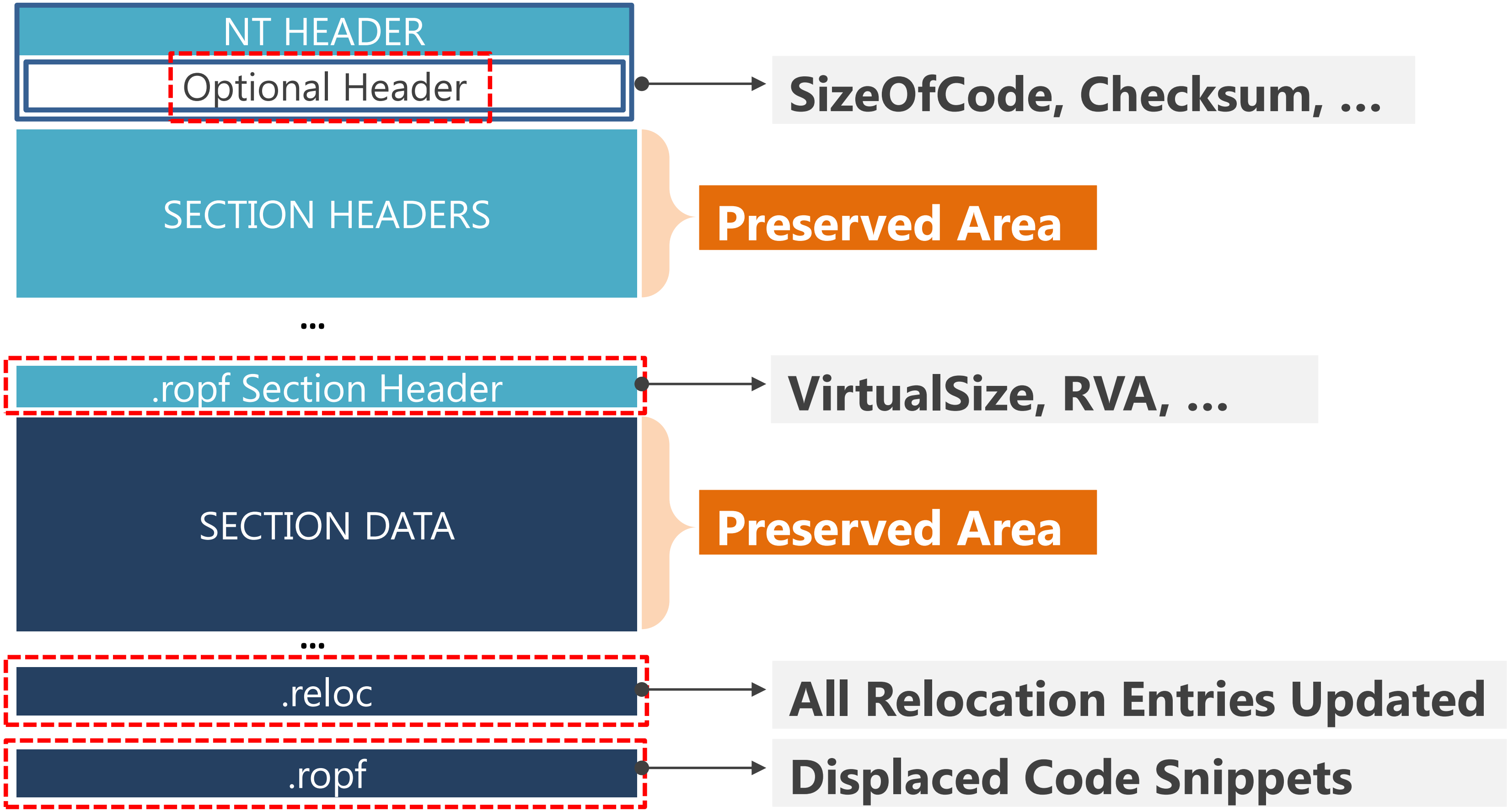
Binary Instrumentation (1/2)

- PE adjustment: headers and sections



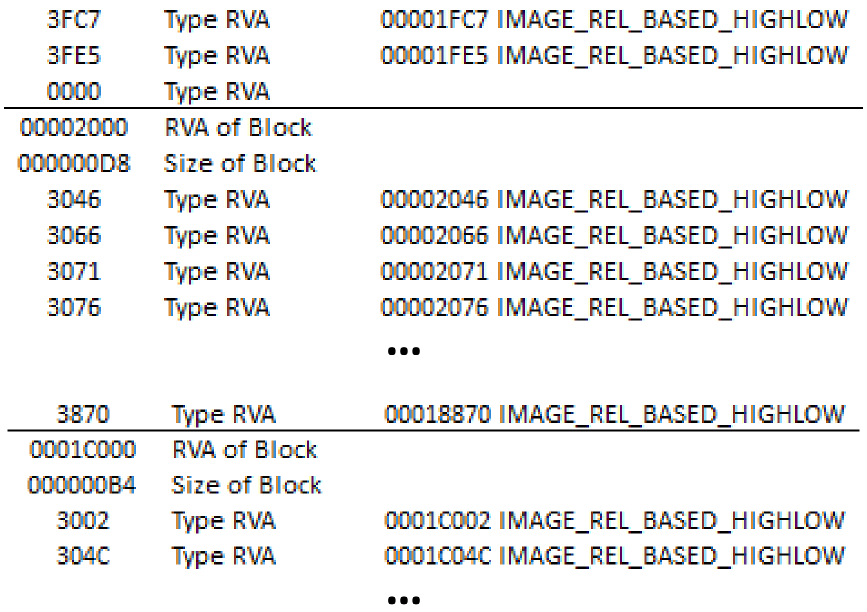
Binary Instrumentation (1/2)

- PE adjustment: headers and sections



Binary Instrumentation (2/2)

- Rebuild the relocation table



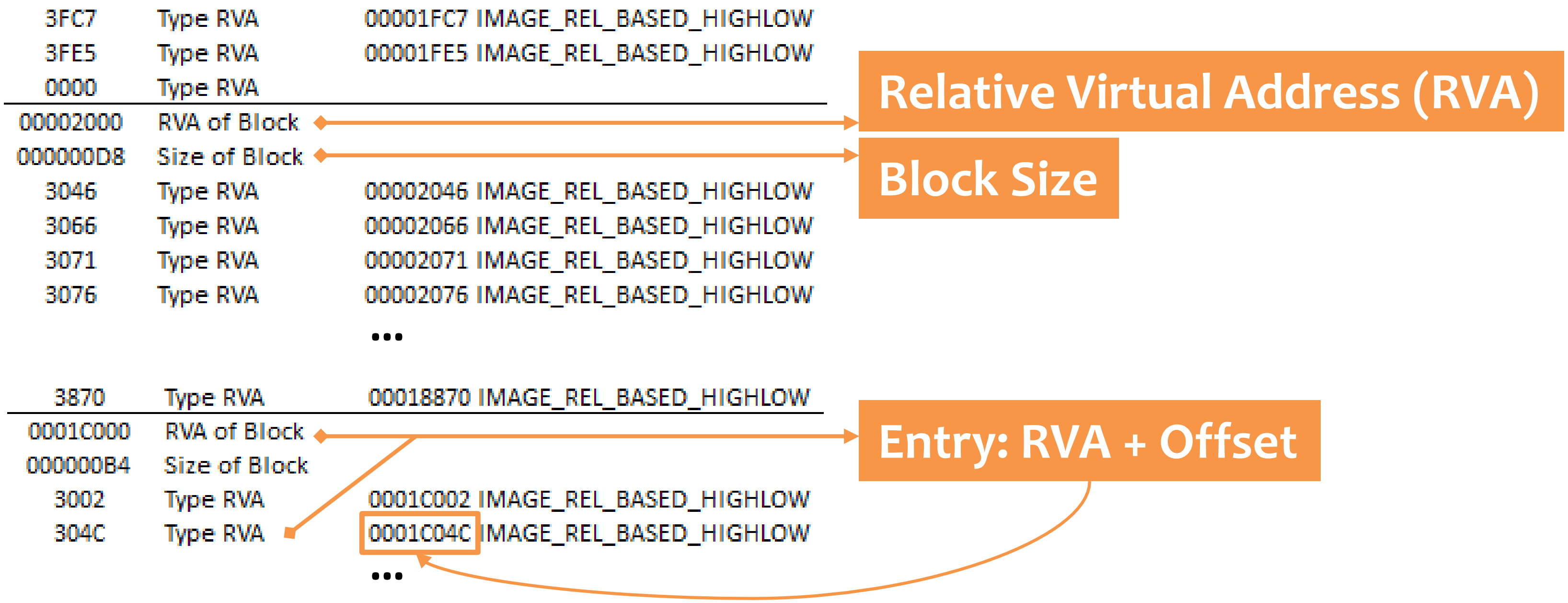
```
    3FC7      Type RVA        00001FC7 IMAGE_REL_BASED_HIGHLOW
    3FE5      Type RVA        00001FE5 IMAGE_REL_BASED_HIGHLOW
    0000      Type RVA
------------------------------------------------------------------
  00002000    RVA of Block
  000000D8    Size of Block
    3046      Type RVA        00002046 IMAGE_REL_BASED_HIGHLOW
    3066      Type RVA        00002066 IMAGE_REL_BASED_HIGHLOW
    3071      Type RVA        00002071 IMAGE_REL_BASED_HIGHLOW
    3076      Type RVA        00002076 IMAGE_REL_BASED_HIGHLOW
                              …

    3870      Type RVA        00018870 IMAGE_REL_BASED_HIGHLOW
------------------------------------------------------------------
  0001C000    RVA of Block
  000000B4    Size of Block
    3002      Type RVA        0001C002 IMAGE_REL_BASED_HIGHLOW
    304C      Type RVA        0001C04C IMAGE_REL_BASED_HIGHLOW
                              …
```

- ✓ Multiple relocation blocks
- ✓ Total number of all entries should be identical

Binary Instrumentation (2/2)

- Rebuild the relocation table



```
3FC7       Type RVA        00001FC7 IMAGE_REL_BASED_HIGHLOW
3FE5       Type RVA        00001FE5 IMAGE_REL_BASED_HIGHLOW
0000       Type RVA
00002000   RVA of Block
000000D8   Size of Block
3046       Type RVA        00002046 IMAGE_REL_BASED_HIGHLOW
3066       Type RVA        00002066 IMAGE_REL_BASED_HIGHLOW
3071       Type RVA        00002071 IMAGE_REL_BASED_HIGHLOW
3076       Type RVA        00002076 IMAGE_REL_BASED_HIGHLOW
                           ...
3870       Type RVA        00018870 IMAGE_REL_BASED_HIGHLOW
0001C000   RVA of Block
000000B4   Size of Block
3002       Type RVA        0001C002 IMAGE_REL_BASED_HIGHLOW
304C       Type RVA        0001C04C IMAGE_REL_BASED_HIGHLOW
                           ...
```

Relative Virtual Address (RVA)

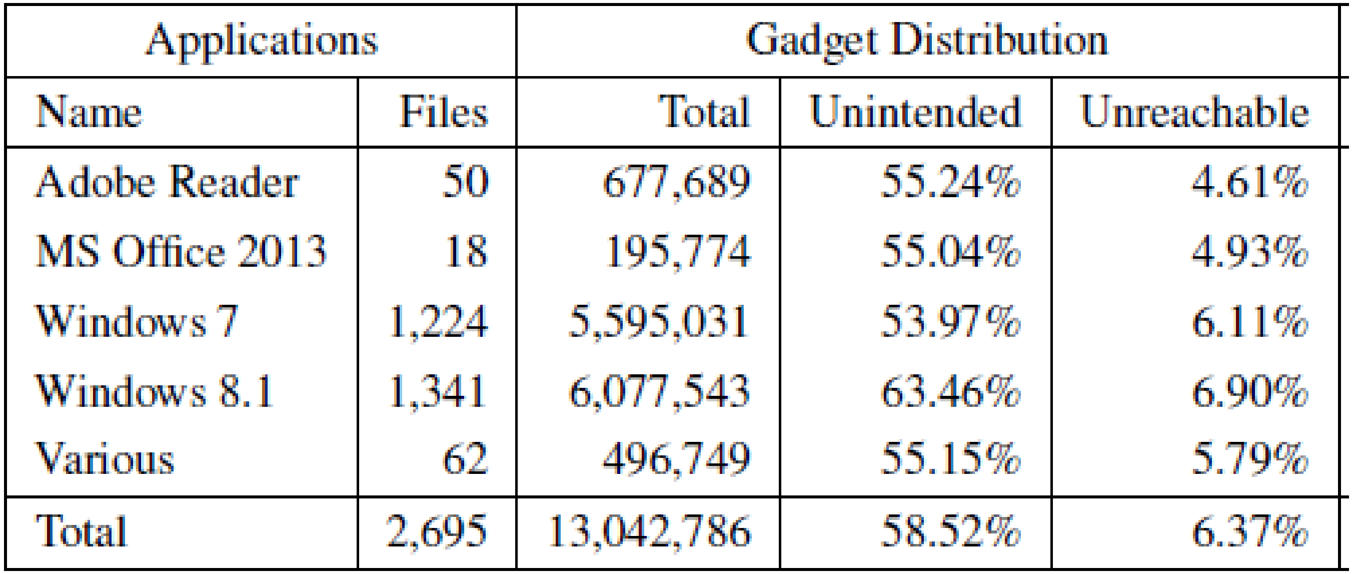
Block Size

Entry: RVA + Offset

- ✓ Multiple relocation blocks
- ✓ Total number of all entries should be identical

Evaluation – Dataset

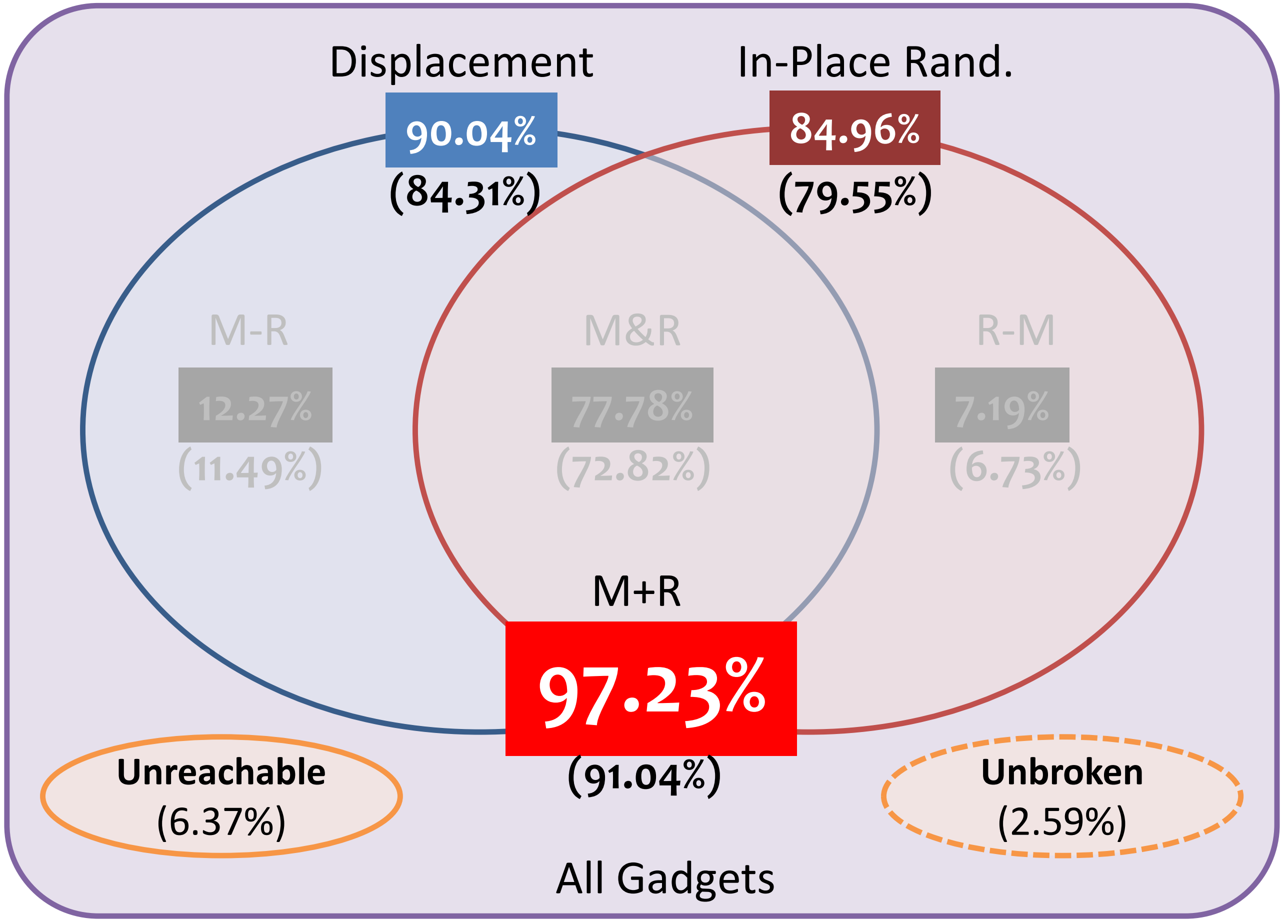
- 2,695 samples from Windows 7, 8.1 and benign apps



Applications		Gadget Distribution		
Name	Files	Total	Unintended	Unreachable
Adobe Reader	50	677,689	55.24%	4.61%
MS Office 2013	18	195,774	55.04%	4.93%
Windows 7	1,224	5,595,031	53.97%	6.11%
Windows 8.1	1,341	6,077,543	63.46%	6.90%
Various	62	496,749	55.15%	5.79%
Total	2,695	13,042,786	58.52%	6.37%

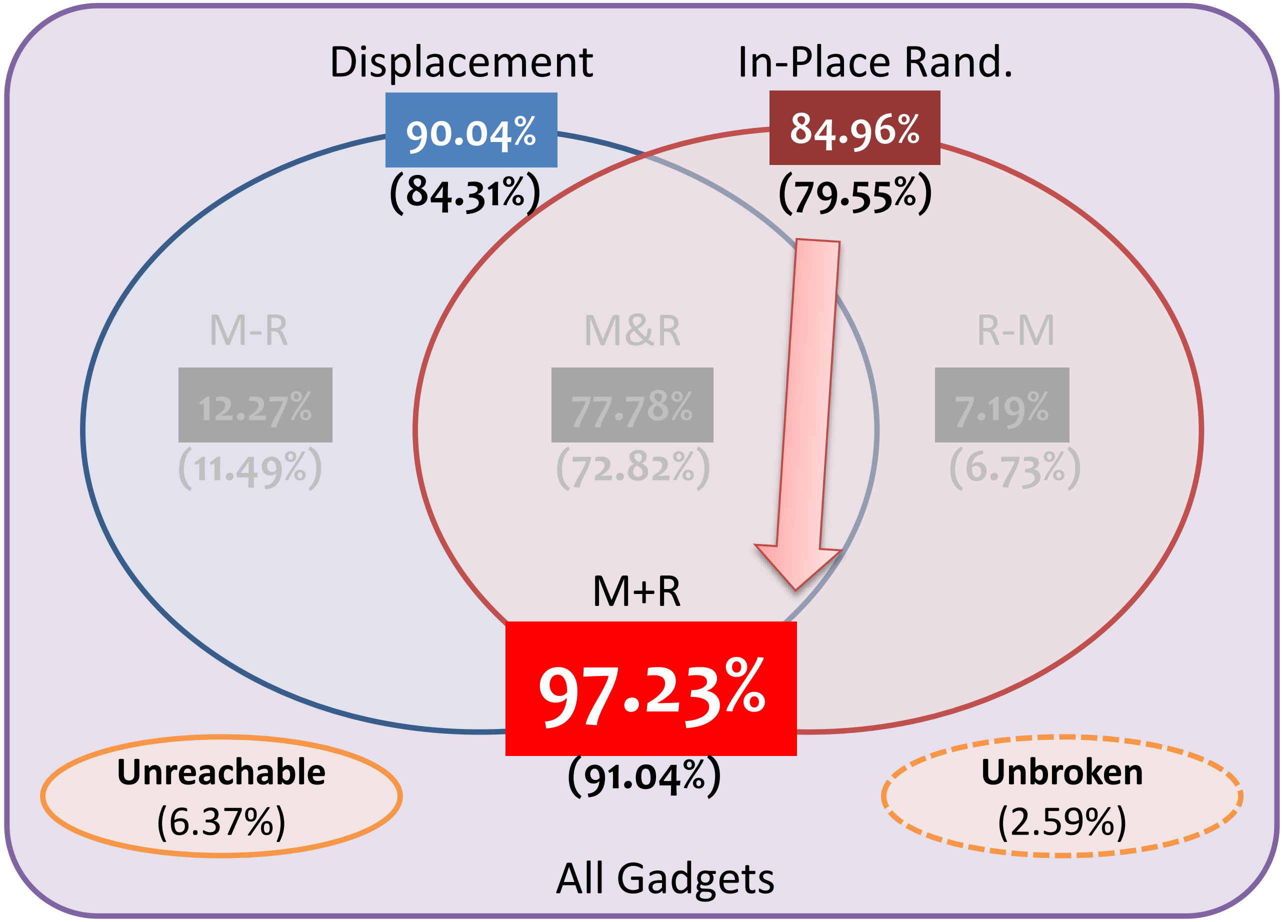
Evaluation – Gadget Coverage (1/2)

- Broken gadgets by displacement and IPR



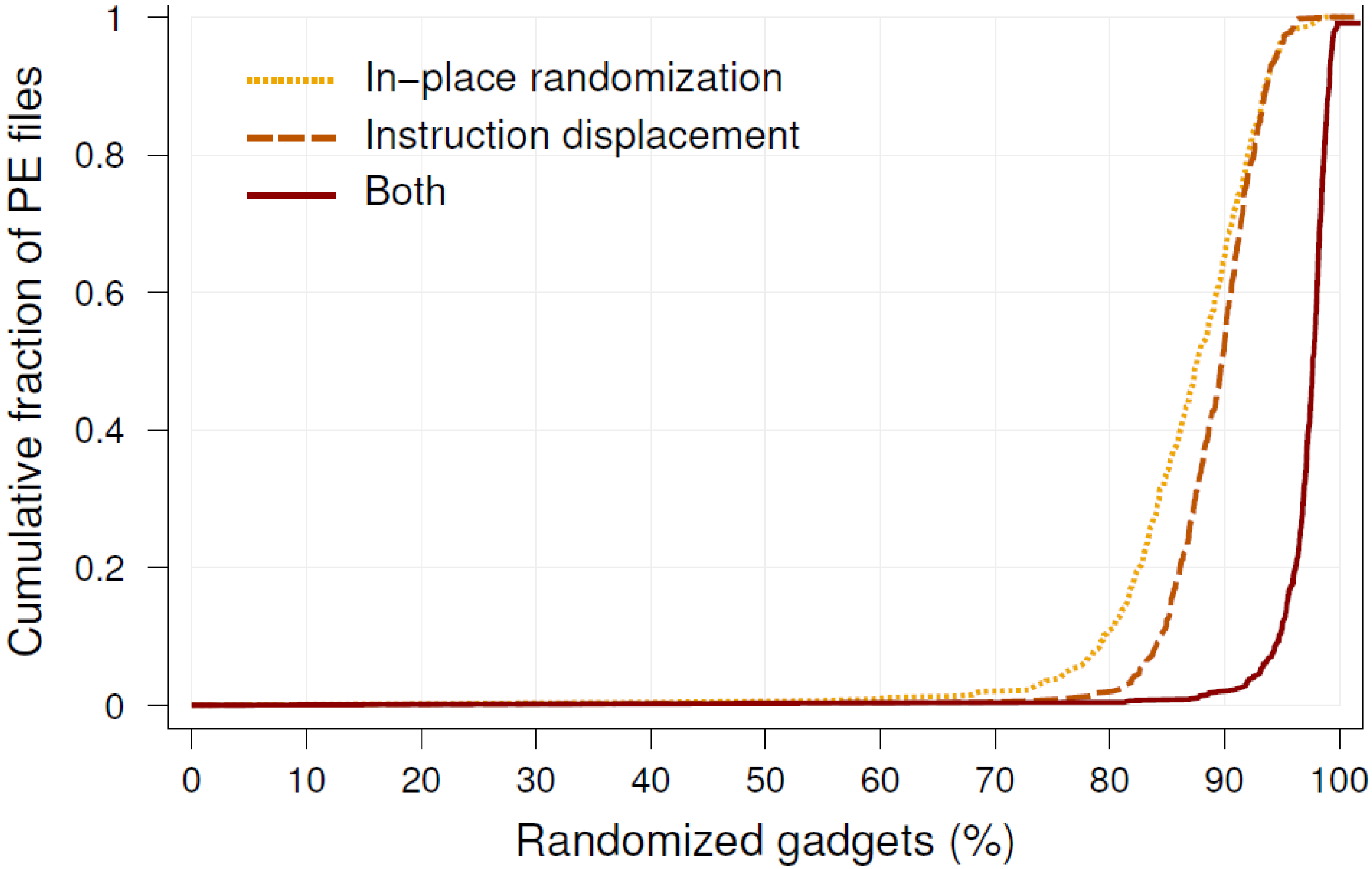
Evaluation – Gadget Coverage (1/2)

- Broken gadgets by displacement and IPR

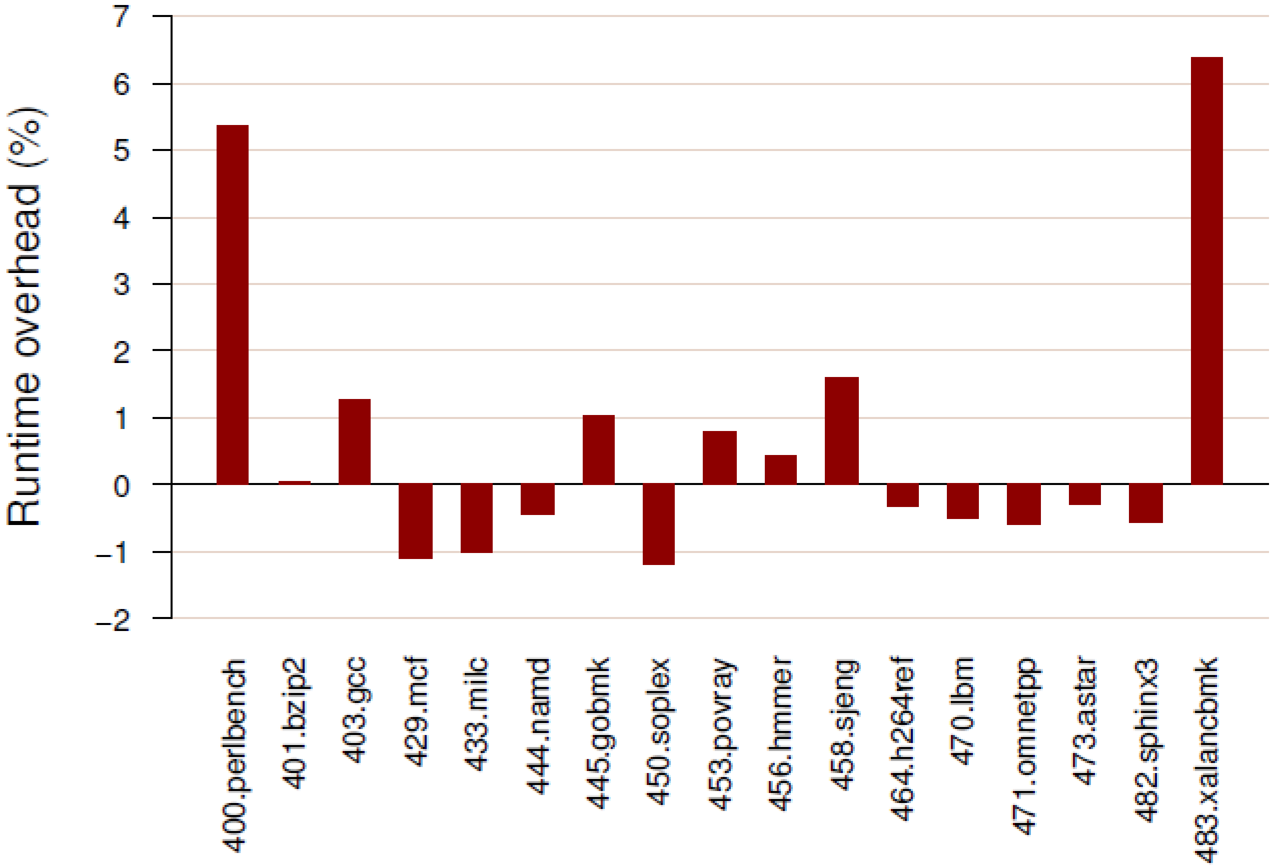


Evaluation – Gadget Coverage (2/2)

- Cumulative distribution of randomized gadgets



Evaluation – Runtime Overhead



- ✓ *SPEC2006:* 0.36% average overhead
- ✓ Statistical *t-test* shows no significant difference for negative overheads

Limitations

- ✓Number of gadgets that can be displaced still depends on the coverage of disassembly and CFG extraction
- ✓Gadget displacement needs at least 5 bytes
- ✓Cannot defend against JIT-ROP
- ✓Cannot break entry-point gadgets (less than 1%)

Wrap-up

- ✓Presented a novel approach: gadget displacement
- ✓Broken gadget coverage: 85% → 97%
- ✓Practical: no source code or debug symbols requirement
- ✓Negligible overhead: 0.36%

```
Code available:
  https://github.com/kevinkoo001/ropf
```