

ROP is Still Dangerous: Breaking Modern Defenses

Nicholas Carlini and David Wagner
University of California, Berkeley

Background

Background

Code Injection



Background

Code Injection

Data Execution
Prevention



Background

Code Injection

Data Execution
Prevention

Return Oriented
Programming



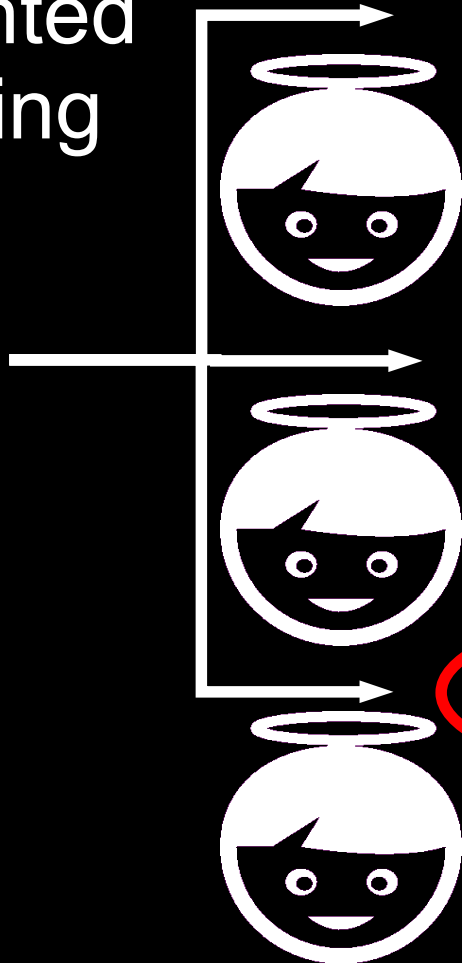
Background

Return Oriented
Programming



Background

Return Oriented
Programming



Address Space
Layout Randomization

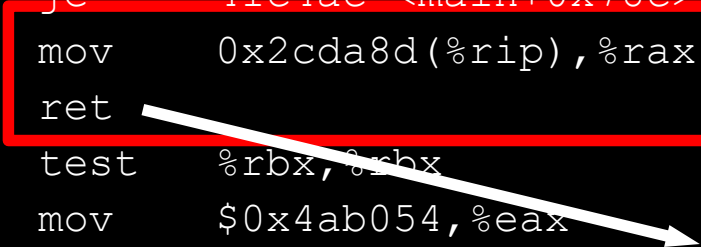
Control Flow
Integrity

kBouncer/ROPecker

Return Oriented Programming

Return Oriented Programming

```
mov    (%rcx),%rbx      mov    %rax,0x2d2945(%rip)  je     41c440 <main+0x720>
test   %rbx,%rbx      mov    0x2cda16(%rip),%rax  xor    %ebp,%ebp
je     41c523 <main+0x803> test   %rax,%rax          mov    $0x4c223a,%ebx
mov    %rbx,%rdi      je     41c112 <main+0x3f2>  add    $0x1,%r14
callq  42ab00         movzbl (%rax),%edx        jmp    41c1a3 <main+0x483>
mov    %rax,0x2cda9d(%rip) callq  41b640 <time@plt>  cmp    (%rbx),%r12b
cmpb   $0x2d,(%rbx)   mov    0xb8(%rsp),%r15d    mov    %ebp,%r13d
je     41c4ac <main+0x78c> cmp    0xc(%rsp),%r15d    jne    41c188 <main+0x468>
mov    0x2cda8d(%rip),%rax mov    %rax,0x2d2670(%rip) mov    %rbx,%rsi
ret    <main+0x78c>   je     41c214 <main+0x4f4> test   %eax,%eax
test   %rbx,%rbx     xchg  %ax,%ax            xchg  %ax,%ax
mov    $0x4ab054,%eax jne    41c188 <main+0x468> jne    41c188 <main+0x468>
cmove  %rax,%rbx     movslq %rdx,%rdx        movslq %ebp,%rax
mov    %rbx,0x2cda6a(%rip) mov    (%rsp),%rdx      ret
test   %rdi,%rdi     movslq %r15d,%rax      cmpl   $0x1,0x4ab3c8(%rax)
je     41c0c2 <main+0x3a2> mov    (%rdx,%rax,8),%r14 je     41c461 <main+0x741>
mov    $0x63b,%edx   ret                    mov    (%rsp),%rcx
mov    $0x4ab01d,%esi cmpb   $0x2d,(%r14)    add    $0x1,%r15d
callq  46cab0 <sh_xfree> jne    41c214 <main+0x4f4> movslq %r15d,%rdx
ret    <main+0x78c>  movzbl 0x1(%r14),%r12d  mov    (%rcx,%rdx,8),%rdx
movl   $0x0,0x18(%rsp) movl   $0x0,0x18(%rsp)  test   %rdx,%rdx
cmp    $0x2d,%r12b  cmp    $0x2d,%r12b    je     41cefd <main+0x11dd>
```



Return Oriented Programming

```
mov    (%rcx),%rbx          mov    %rax,0x2d2945(%rip)  je     41c440 <main+0x720>
test   %rbx,%rbx          mov    0x2cda16(%rip),%rax  xor    %ebp,%ebp
je     41c523 <main+0x803>  test   %rax,%rax          mov    $0x4c223a,%ebx
mov    %rbx,%rdi          je     41c112 <main+0x3f2>  add    $0x1,%r14
callq  42ab00             movzbl (%rax),%edx        jmp    41c1a3 <main+0x483>
mov    %rax,0x2cda9d(%rip) callq  41b640 <time@plt>   cmp    (%rbx),%r12b
cmpb   $0x2d,(%rbx)       mov    0xb8(%rsp),%r15d    mov    %ebp,%r13d
je     41c4ac <main+0x78c> cmp    0xc(%rsp),%r15d    jne   41c188 <main+0x468>
mov    0x2cda8d(%rip),%rax mov    %rax,0x2d2670(%rip) mov    %rbx,%rsi
ret                                         je     41c214 <main+0x4f4> test  %eax,%eax
test   %rbx,%rbx          xchg  %ax,%ax            xchg  %ax,%ax
mov    $0x4ab054,%eax      mov    (%rsp),%rdx        jne   41c188 <main+0x468>
cmove  %rax,%rbx          movslq %r15d,%rax        movslq %ebp,%rax
mov    %rbx,0x2cda6a(%rip) mov    (%rdx,%rax,8),%r14 ret
test   %rdi,%rdi          je     41c214 <main+0x4f4> ret
je     41c0c2 <main+0x3a2> cmpb   $0x2d,(%r14)      cmpl  $0x1,0x4ab3c8(%rax)
mov    $0x63b,%edx        jne   41c214 <main+0x4f4> je     41c461 <main+0x741>
mov    $0x4ab01d,%esi     movzbl 0x1(%r14),%r12d   mov    (%rsp),%rcx
callq  46cab0 <sh_xfree>  movl   $0x0,0x18(%rsp)  add    $0x1,%r15d
ret                                         cmp    %rdx,%rdx        movslq %r15d,%rdx
                                         je     41cefd <main+0x11dd>  mov    (%rcx,%rdx,8),%rdx
                                         test  %rdx,%rdx
                                         je     41cefd <main+0x11dd>
```

The diagram illustrates Return Oriented Programming (ROP) by highlighting specific instructions in the assembly code. Two red boxes are drawn around the 'ret' instructions in the first and second columns. A white arrow points from the 'ret' instruction in the first column to the 'ret' instruction in the second column, indicating a jump in control flow. Another white arrow points from the 'ret' instruction in the second column to the 'ret' instruction in the third column, showing a further jump. This demonstrates how a sequence of instructions can be chained together to execute arbitrary code, bypassing the normal return path of a function.

Return Oriented Programming

```
mov    (%rcx),%rbx          mov    %rax,0x2d2945(%rip)   je     41c440 <main+0x720>
test   %rbx,%rbx           mov    0x2cda16(%rip),%rax  xor    %ebp,%ebp
je     41c523 <main+0x803>   test   %rax,%rax           mov    $0x4c223a,%ebx
mov    %rbx,%rdi           je     41c112 <main+0x3f2>   add    $0x1,%r14
callq  42ab00              movzbl (%rax),%edx         jmp    41c1a3 <main+0x483>
mov    %rax,0x2cda9d(%rip) callq  41b640              cmp    (%rbx),%r12b
cmpb   $0x2d,(%rbx)        mov    0xb8(%rip),%r13     %ebp,%r13d
je     41c4ac <main+0x78c>  cmp    0xc(%rsp),%r13     41c188 <main+0x468>
mov    0x2cda8d(%rip),%rax  mov    %rax,0x2d2670(%rip) test   %eax,%eax
ret                                         xchg   %ax,%ax
test   %rbx,%rbx          xchg   %ax,%ax           jne   41c188 <main+0x468>
mov    $0x4ab054,%eax      mov    (%rsp),%rdx        movslq %ebp,%rax
cmove  %rax,%rbx          movslq %r15d,%rax         ret
mov    %rbx,0x2cda6a(%rip) mov    (%rdx,%rax,8),%r14
test   %rdi,%rdi          ret
je     41c0c2 <main+0x3a2>  je     41c214 <main+0x4f4>  cmpl   $0x1,0x4ab3c8(%rax)
mov    $0x63b,%edx        cmpb   $0x2d,(%r14)       je     41c461 <main+0x741>
mov    $0x4ab01d,%esi     jne   41c214 <main+0x4f4>  mov    (%rsp),%rcx
callq  46cab0 <sh_xfree>   movzbl 0x1(%r14),%r12d    add    $0x1,%r15d
ret                               movslq %r15d,%rdx
movl   $0x0,0x18(%rsp)    mov    (%rcx,%rdx,8),%rdx
cmp    $0x2d,%r12b        test   %rdx,%rdx
je     41cefd <main+0x11dd>  je     41cefd <main+0x11dd>
```

Gadget

kBouncer

If we could inspect the past execution ...
... maybe we could detect ROP attacks

Transparent ROP exploit mitigation using indirect branch tracing.
Vasilis Pappas, Michalis Polychronakis, and Angelos D Keromytis.
USENIX Security, 2013.

kBouncer

Time



Normal Execution

Syscall

kBouncer

Time



Visible History
(Last Branch Record)

kBouncer

Time



kBouncer

Time



Visible History
(Last Branch Record)

kBouncer Observation (1):

kBouncer Observation (1):
ROP attacks issue returns to
non-Call-Preceded addresses.

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768
mov    esi, 0x4ab632
mov    rdi, rbx
call   0x2b2130
test   rbp, rbp
cmov   [rbp], 0x0
add    rsp, 0x8
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768
mov    esi, 0x4ab632
mov    rdi, rbx
call  0x2b2130
test   rbp, rbp
cmov   [rbp], 0x0
add    rsp, 0x8
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768
mov    esi, 0x4ab632
mov    rdi, rbx
call  0x2b2130
test   rbp, rbp
cmov  [rbp], 0x0
add   rsp, 0x8
pop   rbx
pop   rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768
mov    esi, 0x4ab632
mov    rdi, rbx
call  0x2b2130
test   rbp, rbp
cmov   [rbp], 0x0
add    rsp, 0x8
pop    rbx
pop    rbp
ret
```


Normal Execution


```
and    [rax], 0xfd
mov    edx, 0x768
mov    esi, 0x4ab632
mov    rdi, rbx
call   0x2b2130
test   rbp, rbp
cmov   [rbp], 0x0
add    rsp, 0x8
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768
mov    esi, 0x4ab632
mov    rdi, rbx
call   0x2b2130
test   rbp, rbp
cmov   [rbp], 0x0
add    rsp, 0x8
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130           add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0        pop  rbx
add    rsp, 0x8          ret
pop    rbx
pop    rbp
ret
```



Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130           add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0        pop  rbx
add    rsp, 0x8          ret
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push rbx
mov    rdi, rbx           mov ebx, eax
call   0x2b2130           add ebx, ebx
test   rbp, rbp           add ebx, eax
cmov   [rbp], 0x0         pop  rbx
add    rsp, 0x8           ret
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130          add  ebx,  ebx
test   rbp, rbp         add  ebx,  eax
cmov   [rbp], 0x0       pop   rbx
add    rsp, 0x8         ret
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130          add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0        pop  rbx
add    rsp, 0x8          ret
pop    rbx
pop    rbp
ret
```

Normal Execution

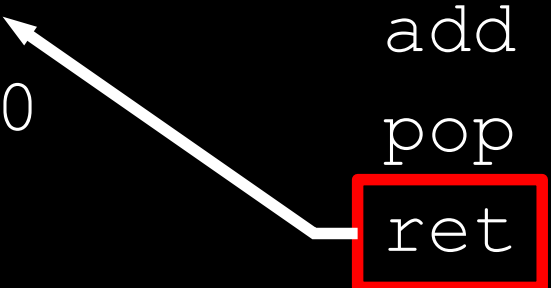
```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130          add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0        pop  rbx
add    rsp, 0x8          ret
pop    rbx
pop    rbp
ret
```


Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130           add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0         pop  rbx
add    rsp, 0x8           ret
pop    rbx
pop    rbp
ret
```

Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130          add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0        pop  rbx
add    rsp, 0x8          ret
pop    rbx
pop    rbp
ret
```

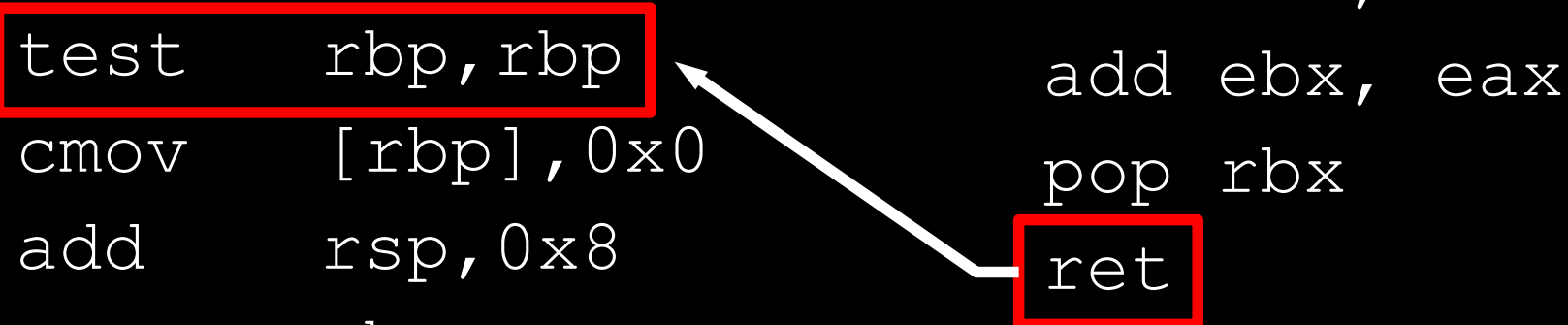


Normal Execution

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130          add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0        pop  rbx
add    rsp, 0x8          ret
pop    rbx
pop    rbp
ret
```

Call-Preceded Return

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130           add  ebx,  ebx
test   rbp, rbp           add  ebx,  eax
cmov   [rbp], 0x0         pop  rbx
add    rsp, 0x8           ret
pop    rbx
pop    rbp
ret
```



Non-Call-Preceded Return


```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push  rbx
mov    rdi, rbx           mov  ebx,  eax
call   0x2b2130           add  ebx,  ebx
test   rbp, rbp          add  ebx,  eax
cmov   [rbp], 0x0         pop  rbx
add    rsp, 0x8           ret
pop    rbx
pop    rbp
ret
```

Non-Call-Preceded Return

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push rbx
mov    rdi, rbx           mov ebx, eax
call   0x2b2130           add ebx, ebx
test   rbp, rbp          add ebx, eax
cmov   [rbp], 0x0         pop  rbx
add    rsp, 0x8          ret
pop    rbx
pop    rbp
ret
```

Non-Call-Preceded Return

```
and    [rax], 0xfd
mov    edx, 0x768          0x2b2130:
mov    esi, 0x4ab632      push rbx
mov    rdi, rbx           mov ebx, eax
call   0x2b2130           add ebx, ebx
test   rbp, rbp           add ebx, eax
cmov   [rbp], 0x0         pop rbx
add    rsp, 0x8           ret
pop    rbx
pop    rbp
ret
```



Defense (1):

All return instructions target
Call-Preceded addresses.

kBouncer Observation (2):

kBouncer Observation (2):
ROP attacks are built of long
sequences of short gadgets.

“gadget”: sequence of <20 instructions, ending in `ret`

“long sequence”: 8 gadgets occurring sequentially

Defense (2):
Do not allow long sequences
of short gadgets.

Detecting Attacks

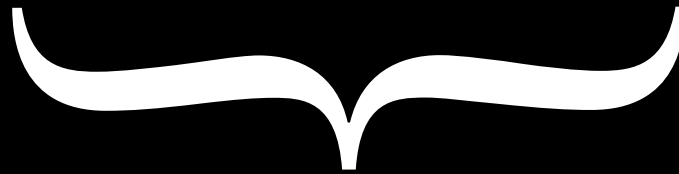
ROP Attack

Issue
Syscall

Detecting Attacks

ROP Attack

Issue
Syscall

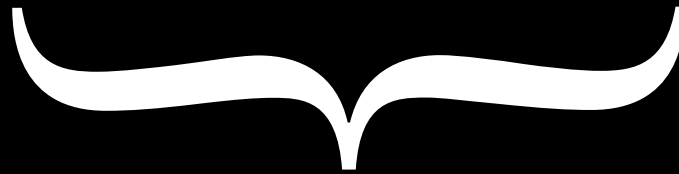


Visible History

Detecting Attacks

ROP Attack

Issue
Syscall



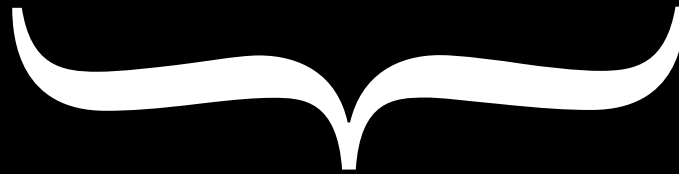
Visible History

- Call-Preceded?
- No long chain?

Detecting Attacks

ROP Attack

Issue
Syscall



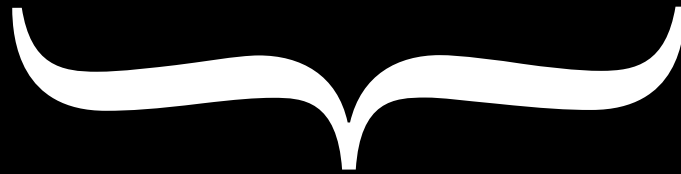
Visible History

- Call-Preceded? X
- No long chain?

Detecting Attacks

ROP Attack

Issue
Syscall



Visible History

- Call-Preceded? X
- No long chain? X

kBouncer is exciting

But does it work?

Breaking kBouncer with History Flushing

Breaking kBouncer with History Flushing

Goal: issue a single system call

Large NOP Gadget

- It must be Call-Preceded
- It must be long (>20 instructions)
- It must act as an effective no-op

```
add    [esp+17Ch],ebx
mov    ebx,[esp+17Ch]
sub    ebx,ebp
jmp    A
...
A: add  [esp+64h],ebx
jmp    B
...
B: mov  esi,[esp+1C0h]
lea    eax,[esi*8-4]
sub    eax,[esp+64]
and    eax,7h
mov    edi,[esp+64]
lea    eax,[edi+eax+4]
shr    eax,3
cmp    eax,esi
jbe    C
...
C: mov  eax,[esp+1C0h]
add    esp,19Ch
pop    ebx
pop    esi
pop    edi
pop    ebp
ret
```

History Flushing

Traditional ROP Attack

La	La	La	La	La	Large
N	N	N	N	N	NOP

History Flushing

Traditional ROP Attack

Flush
History

Issue
Syscall



Visible History

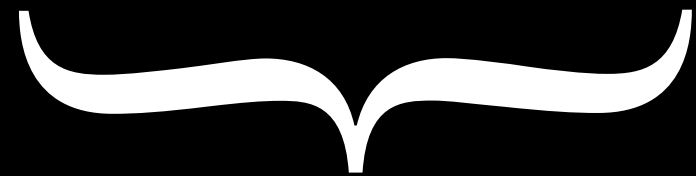
- Call-Preceded?
- No long chain?

History Flushing

Traditional ROP Attack

Flush
History

Issue
Syscall



Visible History

- Call-Preceded?
- No long chain?

History Flushing

Traditional ROP Attack

Flush
History

Issue
Syscall



Visible History

- Call-Preceded? ✓
- No long chain? ✓

So kBouncer is broken

any limited history defense

So ~~kBouncer~~ is broken

Can we fix it?

Introducing kBouncer++

LBR with infinite entries

Introducing kBouncer++

Defense runs continuously

Introducing kBouncer++

Traditional ROP Attack



Visible History

- Call-Preceded?
- No long chain?

Introducing kBouncer++

Traditional ROP Attack



Visible History

- Call-Preceded?
- No long chain?

Introducing kBouncer++

Traditional ROP Attack



Visible History

- Call-Preceded?
- No long chain?

Does this work?

Breaking kBouncer++

Call-Preceded Detector Insufficient

- kBouncer: call-preceded ROP is *not* possible
- Our work: call-preceded ROP is possible
- 10 of 10 binaries of size 70k have sufficient text to mount a call-preceded ROP attack

Defeating kBouncer++

Call-Preceded ROP Attack



Visible History

- Call-Preceded?
- No long chain?

Defeating kBouncer++

Call-Preceded ROP Attack



Visible History

- Call-Preceded? ✓
- No long chain?

Defeating kBouncer++

Call-Preceded ROP Attack



Visible History

- Call-Preceded? ✓
- No long chain? X

Large No-Op Gadgets

Defeating kBouncer++

Call-Preceded ROP Attack



Visible History

- Call-Preceded? ✓
- No long chain?

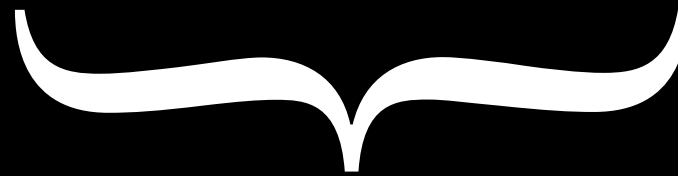
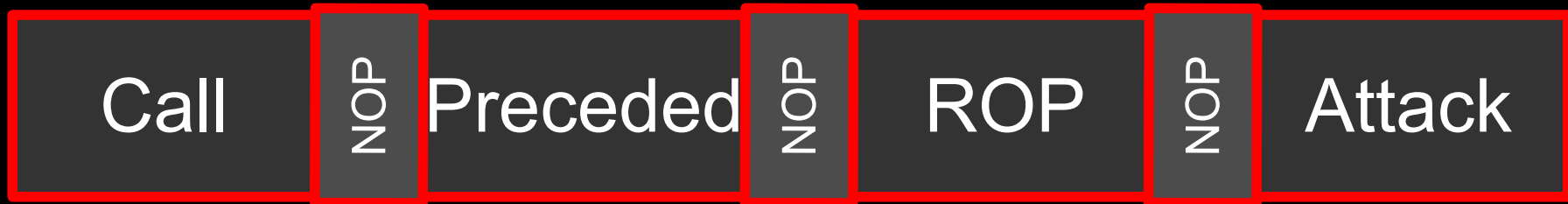
Defeating kBouncer++



Visible History

- Call-Preceded? ✓
- No long chain?

Defeating kBouncer++



Visible History

- Call-Preceded? ✓
- No long chain? ✓

Even with unlimited history,
ROP attacks are possible

ROPecker is also broken

ROPecker: A generic and practical approach for defending against rop attacks.
Yueqiang Cheng, Zongwei Zhou, Miao Yu, Xuhua Ding, and Robert H Deng.
NDSS, 2014.

Results

Modified four real-world exploits so they won't be detected by kBouncer

Results

Modified four real-world exploits so they won't be detected by kBouncer

Adobe Reader 9

Adobe Flash 11

Mplayer Lite

Internet Explorer 8

Related Work

- [Goktas, S&P14] discussed the existence of call-preceded ROP and use it to break many existing CFI defenses
- [Davi, Usenix14] and [Goktas, Usenix14] both independently and concurrently discovered very similar attacks on kBouncer & ROPecker

Implication for Defenses

Implication for Defenses

Do not rely on limited history

Implication for Defenses

Call-Preceded ROP is possible

Implication for Defenses

CFI needs to return to its roots

Implication for Defenses

Classifying code as “gadget”
vs. “non-gadget” is not easy

Defenses should focus on
fundamental differences
between normal execution
and ROP attacks.