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
Return Oriented Programming

Address Space Layout Randomization

Control Flow Integrity

kBouncer/ROPecker
Return Oriented Programming
Return Oriented Programming

```
mov    (%rcx),%rbx
test   %rbx,%rbx
je     41c523 <main+0x803>
mov    %rbx,%rdi
callq  42ab00
mov    %rax,0x2cda9d(%rip)
cmpb   $0x2d,(%rbx)
je     41c4ac <main+0x78c>
mov    0x2cda8d(%rip),%rax
ret

test   %rbx,%rbx
mov    $0x4ab054,%eax
cmove  %rax,%rbx
mov    %rbx,0x2cda6a(%rip)
cmpb   $0x2d,(%rbx)
je     41c44c <main+0x720>
xor    %ebp,%ebp
mov    $0x4c223a,%ebx
add    $0x1,%r14
jmp    41c1a3 <main+0x483>
cmp    (%rbx),%r12b
mov    %ebp,%rsi
test   %eax,%eax
xchg   %ax,%ax
jne    41c188 <main+0x468>
movslq %ebp,%rax
ret
cmple  $0x1,0x4ab3c8(%rax)
je     41c461 <main+0x741>
mov    (%rsp),%rcx
add    $0x1,%r15d
movslq %r15d,%rdx
mov    (%rdx,%rax,8),%r14
ret
je     41c214 <main+0x4f4>
cmpb   $0x2d,(%r14)
jne    41c214 <main+0x4f4>
movzbl 0x1(%r14),%r12d
movl    $0x0,0x18(%rsp)
cmp     $0x2d,%r12b
je     41cefd <main+0x11dd>
```
Return Oriented Programming

```assembly
mov    (%rcx),%rbx
test   %rbx,%rbx
je     41c523 <main+0x803>
mov    %rbx,%rdi
callq  42ab00
mov    %rax,0x2cda9d(%rip)
cmpb   $0x2d,(%rbx)
je     41c4ac <main+0x78c>
mov    0x2cda8d(%rip),%rax
ret
mov    %rax,0x2d2945(%rip)
mov    0x2cda16(%rip),%rax
test   %rax,%rax
je     41c112 <main+0x3f2>
movzbl (%rax),%edx
callq  41b640 <time@plt>
mov    %rax,0x2d2670(%rip)
je     41c214 <main+0x4f4>
xchg   %ax,%ax
mov    (%rsp),%rdx
movslq %r15d,%rax
ret
je     41c214 <main+0x4f4>
cmpb   $0x2d,(%r14)
jne    41c214 <main+0x4f4>
movzbl 0x1(%r14),%r12d
movl   $0x0,0x18(%rsp)
cmp     $0x2d,%r12b
je     41cefd <main+0x11dd>
```

Return Oriented Programming

```assembly
mov    (%rcx),%rbx
test   %rbx,%rbx
je     41c523 <main+0x803>
mov    %rbx,%rdi
callq  42ab00
mov    %rax,0x2cda8d(%rip)
cmpb   $0x2d,(%rbx)
je     41c4ac <main+0x78c>
mov    0x2cda8d(%rip),%rax
ret

test   %rbx,%rbx
mov    $0x4ab054,%eax
cmove  %rax,%rbx
mov    %rbx,0x2cda6a(%rip)
cmp    (%rbx),%r12b
ejne   41c188 <main+0x468>
movslq 0x1(%r14),%r12d
mov    %rbx,%rsi
test   %rax,%rax
xchg   %ax,%ax
jne    41c188 <main+0x468>
movslq %rbx,%rdx
mov    (%rdx,%rax,8),%r14
ret

test   %rdi,%rdi
mov    %rax,0x2d2945(%rip)
je     41c440 <main+0x720>
xor    %ebp,%ebp
mov    $0x4c223a,%ebx
add    $0x1,%r14
jmp    41c1a3 <main+0x483>
cmp    (%rbx),%r12b
test   %eax,%eax
xchg   %ax,%ax
jne    41c188 <main+0x468>
mov    %rbx,%rsi
test   %rax,%rax
xchg   %ax,%ax
jne    41c188 <main+0x468>
je     41c214 <main+0x4f4>
mov    0x2cda16(%rip),%rax
test   %rbx,%rbx
mov    %rbx,0x2cda9d(%rip)
cmpb   $0x2d,(%rbx)
jne    41c214 <main+0x4f4>
movzbl (%rax),%edx
mov    %rax,0x2d2670(%rip)
jmp    41c214 <main+0x4f4>
test   %rdi,%rdi
ret

test   %rcx,%rcx
mov    0x2bcd40,%edi
add    $0x1,%edi
jmp    41c1a3 <main+0x483>
cmp    (%rcx),%r12d
test   %rdi,%rdi
ret
```

---

This code snippet demonstrates the principles of Return Oriented Programming (ROP), a technique used in reverse engineering and vulnerability exploitation for bypassing control flow integrity mechanisms. The code includes dynamic instruction injection, where the runtime behavior is determined by the order of instructions and the control flow is redirected at runtime.
Return Oriented Programming

```
mov   (%rcx),%rbx
mov   $0x2d,0x4ab3c8(%rax)
test  %rbx,%rbx
test  %rax,%rax
je    41c461 <main+0x741>
je    41c214 <main+0x4f4>
xchg  %ax,%ax
jne   41c188 <main+0x468>
je    41c214 <main+0x4f4>
cmp    (%rbx),%r12b
xchg   %ax,%ax
jne   41c188 <main+0x468>
movslq %ebp,%rax
ret
movslq %r15d,%rax
mov   (%rdx,%rax,8),%r14
ret
je    41c214 <main+0x4f4>
cmpb   $0x2d,(%r14)
jne    41c214 <main+0x4f4>
movzbl (0x18(%rsp)),%r12d
movl   $0x0,0x18(%rsp)
cmp    $0x2d,%r12b
je     41cefd <main+0x11dd>
je     41c440 <main+0x720>
xor    %ebp,%ebp
mov    $0x4c223a,%ebx
add    $0x1,%r14
jmp    41c1a3 <main+0x483>
41c188 <main+0x468>
 cmp    (%rbx),%r12b
 test   %eax,%eax
 xchg   %ax,%ax
 jne    41c188 <main+0x468>
mov    %rcx,0x2d2945(%rip)
 mov    %rax,0x2d2670(%rip)
test   %rax,%rax
je     41c112 <main+0x3f2>
mov    0x2cda9d(%rip),%rax
cmpb   $0x2d,(%rbx)
je     41c4ac <main+0x78c>
mov    %rbx,0x2cda8d(%rip),%rax
ret
mov    %rbx,%rdi
callq  42ab00
cmpb   $0x2d,%rbx
je     41c523 <main+0x803>
mov    %rbx,%rdi
callq  42ab00
mov    %rbx,0x2cda9d(%rip)
cmpb   $0x2d,0x4ab054(%rax)
cmove  %rax,%rbx
mov    %rbx,0x2cda6a(%rip)
test  %rdi,%rdi
je     41c0c2 <main+0x3a2>
mov    %rbx,0x2cda0d(%rip)
test  %r13d,%rdx
je     41c188 <main+0x468>
mov    %rbx,%rsi
test   %eax,%eax
xchg   %ax,%ax
jne    41c188 <main+0x468>
movslq %ebp,%rax
ret
mov   0x2cd2945(%rip),%rax
mov   0x2cd2670(%rip),%rax
test   %rbx,%rbx
je     41c4ac <main+0x78c>
mov    %rbx,%rdi
callq  42ab00
mov    %rax,0x2cda9d(%rip)
ret
mov   %rax,0x2cda16(%rip),%rax
mov   %rbx,0x4ab01d(%rax)
callq  46cab0 <sh_xfree>
ret
```
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.  
kBouncer

Time

Normal Execution  Syscall
kBouncer

Time

Normal Execution  Syscall

Visible History
(Last Branch Record)
kBouncer

Time

Normal Execution  Syscall  ROP Attack  Syscall
kBouncer

Time

Normal Execution  Syscall  ROP Attack  Syscall

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], 0xfdf
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], 0xf
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], 0xf
d
mov edx, 0x768
d
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

```assembly
and [rax], 0x0fd
mov edx, 0x768
mov esi, 0x4ab632
mov rdi, rbx
call 0x2b2130

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret
```
Normal Execution

and [rax], 0xffd
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

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
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

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret
Normal Execution

and [rax], 0x7fd
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

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret
## Normal Execution

<table>
<thead>
<tr>
<th>Operation</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>and [rax], 0xfd</code></td>
<td>mov edx, 0x768</td>
</tr>
<tr>
<td><code>mov edx, 0x768</code></td>
<td>mov esi, 0x4ab632</td>
</tr>
<tr>
<td><code>mov esi, 0x4ab632</code></td>
<td>mov rdi, rbx</td>
</tr>
<tr>
<td><code>mov rdi, rbx</code></td>
<td>call 0x2b2130</td>
</tr>
<tr>
<td><code>call 0x2b2130</code></td>
<td>test rbp, rbp</td>
</tr>
<tr>
<td><code>test rbp, rbp</code></td>
<td>cmov [rbp], 0x0</td>
</tr>
<tr>
<td><code>cmov [rbp], 0x0</code></td>
<td>add rsp, 0x8</td>
</tr>
<tr>
<td><code>add rsp, 0x8</code></td>
<td>pop rbx</td>
</tr>
<tr>
<td><code>pop rbx</code></td>
<td>pop rbp</td>
</tr>
<tr>
<td><code>pop rbp</code></td>
<td>ret</td>
</tr>
</tbody>
</table>

0x2b2130:
- `push rbx`  
- `mov ebx, eax`  
- `add ebx, ebx`  
- `add ebx, eax`  
- `pop rbx`  
- `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

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
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

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret
Normal Execution

and [rax], 0xfd
mov edx, 0x768
mov esi, 0x4ab632
mov rdi, rbx
call 0x2b2130

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret

test rbp, rbp
cmov [rbp], 0x0
add rsp, 0x8
pop rbx
pop rbp
ret

pop rbx
pop rbp
ret
Normal Execution

and [rax], 0xfdf
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

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret
Call-Preceded Return

```
and [rax], 0xfdf
mov edx, 0x768
mov esi, 0x4ab632
mov rdi, rbx
call 0x2b2130

test rbp, rbp
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
add ebx, ebx
add ebx, eax
```

```
0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
add ebx, eax
pop rbx
ret
```
Non-Call-Preceded Return

and       [rax], 0xf
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

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret
Non-Call-Preceded Return

and [rax], 0xf0d
mov edx, 0x768
mov esi, 0x4ab632
mov rdi, rbx
call 0x2b2130

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx
ret

add rsp, 0x8
pop rbx
pop rbp
ret
Non-Call-Preceded Return

and [rax], 0xfd
mov edx, 0x768
mov esi, 0x4ab632
mov rdi, rbx
call 0x2b2130

0x2b2130:
push rbx
mov ebx, eax
add ebx, ebx
add ebx, eax
pop rbx

ret
Defense (1):
All return instructions target Call-Preceded addresses.
Defense (1):
Restrict returns to only 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 \texttt{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

- 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 Large
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
  - Call-Preceded? ✔
  - No long chain? ✔

- Flush History
- Issue Syscall

Visible History
So kBouncer is broken
So kBouncer is broken

any limited history defense
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

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.