Control Flow Bending: On the Effectiveness of Control Flow Integrity

Nicholas Carlini$^1$, Antonio Barresi$^2$, Mathias Payer$^3$
David Wagner$^1$, Thomas R. Gross$^2$

$^1$ University of California, Berkeley
$^2$ ETH Zurich
$^3$ Purdue University
Background
Background

Memory Corruption \rightarrow Control Flow Integrity
Main Result:
CFI does not stop all attacks.
Control-Flow Integrity
Control-Flow Integrity

\((*fn)(x)\); return 7;
Control-Flow Integrity

CHECK(fn);
(*fn)(x);

return 7;
Control-Flow Integrity

CHECK(fn);
(*fn)(x);
return 7;
Control-Flow Integrity

CHECK(fn);
(*fn)(x);

CHECK_RET();
return 7;
Control-Flow Integrity

CHECK(fn);
(*fn)(x);
CHECK_RET();
return 7;
Shadow Stacks

- Dynamic restrictions on return instructions
Shadow Stacks

- Dynamic restrictions on return instructions
Shadow Stacks

- Dynamic restrictions on return instructions
Two ways to weaken CFI:

1. Coarsen CFG
2. Remove Shadow Stack
Prior Work: Weak CFI is broken

- Göktaş et al. IEEE S&P '14
  “Out Of Control: Overcoming Control-Flow Integrity”

- Carlini et al. USENIX Security '14
  “ROP is Still Dangerous: Breaking Modern Defenses”

- Davi et al. USENIX Security '14
  “Stitching the Gadgets: On the Ineffectiveness of Coarse-Grained Control-Flow Integrity Protection”

- Göktaş et al. USENIX Security '14
  “Size Does Matter: Why Using Gadget-Chain Length to Prevent Code-Reuse Attacks is Hard”
Our question:
How secure are other CFI variants?
Our question:
How secure are other CFI variants?

1. CFI with no shadow stack
2. Fully unweakened CFI
Fully-Precise Static CFI: CFI in its best possible form
A transfer from X to Y is allowed if and only if some benign execution uses it.
How secure is half-weakened CFI?
(Shadow stack removed)
Can we just do ROP?
Can we just do ROP?

No.
Can we just do ROP?

No.

How about return-to-libc?
Return to Libc: Challenges

1. Find path to system() in CFG.

2. Divert control flow down this path.

3. Control system() arguments.
What does a CFG look like?
That picture is a lie
Exploit

memcpy

system
So simple paths between arbitrary points exist. But are they feasible?
Dispatcher Functions

memcpy(dst, src, 8)

Stack

Old stack data
Dispatcher Functions

memcpy(dst, src, 8)
Dispatcher Functions

memcpy(dst, src, 8)
Dispatcher Functions

memcpy(dst, src, 8)
Dispatcher Functions

1. Commonly called

2. Arguments under attacker control

3. Can overwrite its own return address
Return to Libc: Challenges

1. Find path to system() in CFG. ✓

2. Divert control flow down this path. ✓

3. Control system() arguments. ✓
Evaluation (part 1)

- Analyzed six vulnerable binaries, assuming fully-precise static CFI was in place.
- What attacks are possible?
  - Direct arbitrary code execution in 3 of the 6
  - File system access in 2 of the 6
  - Attack prevented in 1 of the 6
CFI without shadow stack is broken – even with fully precise static CFI.
How secure is un-weakened CFI?
This time: no dispatcher functions
Return to Libc: Challenges

1. Find path to `system()` in CFG.

2. Divert control flow down this path.

3. Control `system()` arguments.
Evaluation (part 2)

• Evaluate same 6 binaries; with shadow stack

• Direct arbitrary code execution in 1 of the 6
• File system access in 2 of the 6
• Confined code execution in 2 of the 6
• Attack prevented in 1 of the 6
ROP gives us Turing-complete attacks. Can we do this with CFI and a shadow stack?
Printf-Oriented Programming

- A Turing complete domain-specific language

- Program $\rightarrow$ Format String

- Program Counter $\rightarrow$ Format String Counter
Printf-Oriented Programming

- Memory Reads → %s
- Memory Writes → %n
- Conditional → %.d
- Loops? Write over the format specifier counter.
Conclusion

CFI with shadow stack stops some attacks.
Conclusion

CFI, in its best form, cannot stop all attacks.
Conclusion

Shadow stacks significantly complicates attacks.
Conclusion

Enforcing memory integrity may be more effective than control-flow integrity.
Questions?
... but wait, you didn't break CFI!

- The goal of CFI is to make programs secure.
- The mechanism CFI uses is enforcing a CFG.
- We do not break the mechanism.
- We do break the goal.