Return-Oriented Programming without Returns

Stephen Checkoway, Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Hovav Shacham, Marcel Winandy

ACM CCS 2010, Chicago, USA
Motivation for ROP

- Ad hoc defense against code injection:
  - $W \oplus X$
  - DEP
- Code injection unnecessary for arbitrary computation
- Use existing code to synthesize new behavior
Overview of (traditional) ROP

- Stack is the program
  - Pointers to code
  - Data
- Execution proceeds by changing the stack pointer
- Turing-complete
Defenses

- Control-flow integrity
  - [Abadi et al. CCS’05, Erlingsson et al. OSDI’06]
    - Defends against an entire class of memory error vulnerabilities
- Count frequency of ret instructions
- Use LIFO invariant of the call stack
  - Maintain shadow call stack
- Modify compiler to avoid emitting ret instructions
Deconstructing the ret

- Copy top of stack to instruction pointer
  Transfers control
- Increment stack pointer
  Updates processor state
Update-Load-Branch

```
pop  %eax
jmp  *%eax

add  $4,%eax
jmp  *(%eax)

pop  %eax
jmp  *(%eax)

inc  %eax
jmp  *(%ebx,%eax,4)
```
Removing the rets

\[
\begin{align*}
\text{add} & \quad \%\text{eax}, \%\text{ecx} \\
\text{ret} & \\
\text{add} & \quad \%\text{eax}, \%\text{ecx} \\
\text{pop} & \quad \%\text{ebx} \\
\text{jmp} & \quad \%\text{ebx} \\
\end{align*}
\]
Key insight

- Only need one update-load-branch sequence

- edx points to ULB

```assembly
add %eax, %ecx
jmp *%edx

pop %ebx
jmp *%ebx
```
ARM – Overview

• ARM stands for Advanced RISC Machine
• Application area: Embedded systems
  ◆ Mobile phones, smartphones (Apple iPhone, Google Android), music players, tablets, netbooks
• Advantage: Low power consumption
• ARM features XN (eXecute Never) Bit
• Follows RISC design
  ◆ Mostly single-cycle execution
  ◆ Dedicated load and store instructions
  ◆ Fixed instruction length
ARM Registers

- ARM’s 32 Bit processor features 16 registers
- In contrast to Intel x86, each register is directly accessible
  - E.g., it is possible to directly change the program counter (r15)

Function arguments and results from function:
- r0
- r1
- r2
- r3

Register variables (callee saved):
- r4
- r5
- r6
- r7
- r8
- r9
- r10
- r11

Scratch Register:
- r12

Stack Pointer:
- r13/sp

Link Register:
- r14/lr

Program Counter:
- r15/pc

Control Program Status Register:
- cpsr
ARM Calling Convention

• AAPCS - ARM Architecture Procedure Call Standard
• No dedicated call and return instructions
  ✷ Instead any jump instruction can be used as call and return resp.
• Function Call
  ✷ BL – Branch with Link
  ✷ BLX – Branch with Link and Exchange (allows indirect calls)
  ✷ BL and BLX load the return address into the link register (r14)
• Function Return
  ✷ Loading return address into program counter
The BLX Instruction

• Candidates for an attack on ARM
  ◆ All indirect jump instructions not part of a function epilogue
    » Instructions where pc is used as destination register
    » Indirect branch instructions, e.g., BLX

• We inspected libc and libwebcore on Android 2.0
  ◆ Result: Many sequences end with a BLX instruction

BLX register

Branch to register

Store return address in link register

Instruction set exchange
Trampoline (Update-Load-Branch)

- Trampoline sequence for ARM
  - Unfortunately no POP-BLX sequence in our libraries
  - **Update-Load-Branch** sequence
    » Initialize a register \((r_j)\) so that it points to injected jump addresses
    » Update the state of \(r_j\) after each sequence
    » Load a second register \((r_s)\) with the address of the next sequence pointed by \(r_j\)
    » Branch with BLX to the address stored in \(r_s\)
**Attack Design and Memory Layout**

- **Trampoline**
- **Jump Addresses**
  - Jump Address 1
  - Jump Address 2
  - Jump Address 3
- **Arguments**
  - Argument 1
  - Argument 2
- **Memory under control of adversary**
- **Adversary**
  - Take control over pc
  - Setup r_a, and r_j

**GADGET 1**
- Sequence 1
  - Instruction
  - Instruction
  - BLX Trampoline
- Sequence 2
  - Instruction
  - Instruction
  - BLX Trampoline

**GADGET 2**
- Sequence 1
  - Instruction
  - Instruction
  - BLX Trampoline

**Terminology**
- r_a – Pointer to arguments (sp)
- r_j – Pointer to jump addresses
Conclusion

• Our results
  - Return address checkers can be bypassed
  - Showed return-oriented programming without returns
  - We derived a Turing-complete gadget set for x86 and ARM
  - Attack instantiation on Debian (x86) and Android (ARM)

• Implications
  - Return-oriented programming (without returns) is a serious problem
  - Will become crucial attack technique in future and effective countermeasures are needed
  - We show how to use it to mount a privilege escalation attack on Android (upcoming paper at ISC 2010)
Questions?

Thank you for your attention
The History of ROP

2007

2008

2009

2010

Intel x86

SPARC

Atmel AVR

Z80

PowerPc

ARM

Internet Explorer

Adobe Reader

Apple Jailbreak

Quicktime Player

System Security Lab
Ruhr-University Bochum