Return-Oriented Programming

Andy Podgurski

Electrical Engineering & Computer Science Department
Case Western Reserve University

Overview

- Return-oriented programming (ROP) is a technique that allows an attacker to induce arbitrary behavior in a program whose control flow he has diverted.
- ROP does *not* involve injecting code.
- It links code snippets already present in memory.
  - Each snippet ends in a **ret** instruction.
- ROP defeats the W⊕X protections deployed by Microsoft, Intel, and AMD.
- It is readily exploitable on multiple architectures.
- It bypasses security measures that seek to prevent execution of malicious code.

Background

- For 20+ years, security research has focused on preventing introduction and execution of new malicious code:
  - Techniques to guarantee the integrity of control flow in existing programs, e.g.,
    - type-safe languages, stack cookies, CFI
  - Techniques to isolate “bad” code introduced into the system, e.g.,
    - W⊕X, memory tainting, virus scanners, trusted computing

Example: W⊕X

- Memory is marked as either writable or executable, never both.
- Thus an adversary can't inject data into a process and then jump to that memory.
  - Raises a processor exception
- ROP categorically evades W⊕X protections.
  - The snippets are in memory marked executable.

Gadgets

- The organizational unit of a ROP attack is the gadget:
  - An arrangement of words on the stack:
    - pointers to instruction sequences
    - immediate data words
  - Accomplishes some well-defined task
    - e.g., load, xor, conditional branch
- By assembling a Turing-complete collection of gadgets, an attacker can synthesize any malicious behavior.
- Roemer et al show how to build such gadgets using snippets from the Standard C Library on Linux and Solaris.

Background: Return-to-libc Attacks

- Attackers responded to code injection defenses by reusing code already present in the target process image.
- The standard C library, libc, was the usual target.
  - libc contains wrappers for system calls.
  - This is called a return-to-libc attack.
  - McDonald showed how libc calls could be chained together.
  - This is sufficient to defeat W⊕X on machines without immutable memory protections.
- In principle, any available code could be used.
- ROP generalizes return-to-libc to allow arbitrary computation without calling any functions.

ROP Principles (for x86)

- A RO program involves a particular layout of the stack segment. (Fig. 3)
- Each RO instruction is a stack word, pointing to an instruction sequence somewhere in memory.
- The stack pointer **%esp** governs what RO instruction sequence is fetched next:
  - Execution of **ret** induces fetch-and-decode:
    1. The target of **%esp** is read and used as the new value for **%eip** (instruction pointer).
    2. %**esp** is incremented to point to the next word on the stack.
  - If the next instruction sequence also ends in **ret**, this process is repeated.

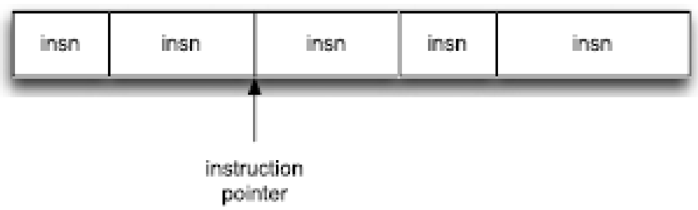

Fig. 2. Layout of an ordinary program.

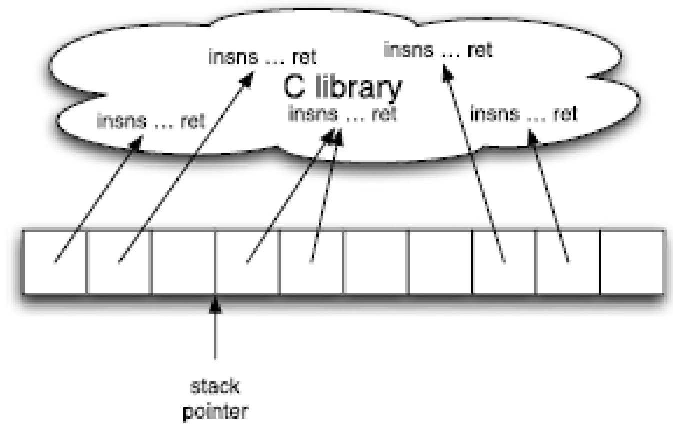

Fig. 3. Layout of a return-oriented program.

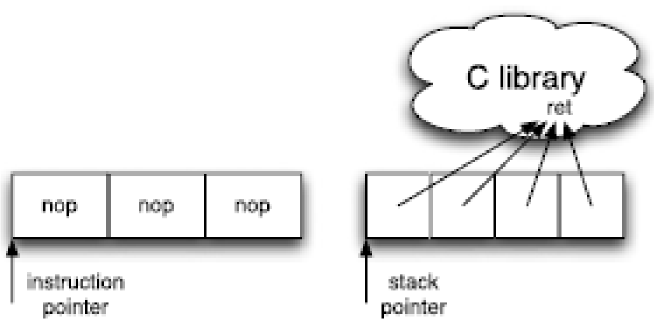

Fig. 4. Ordinary and return-oriented nop sleds.

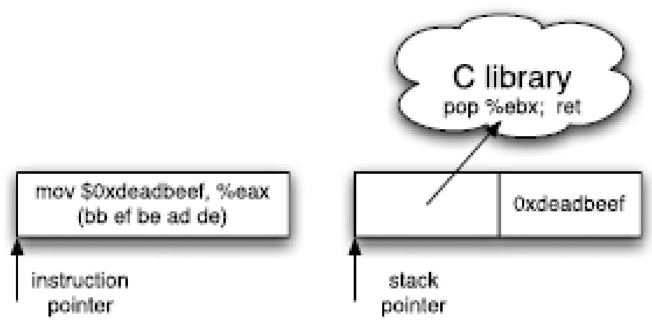

Fig. 5. Ordinary and return-oriented immediates.

No-op Instructions

- In ROP, a no-op is simply a stack word containing the address of a **ret**.
- These can be composed to form a no-op sled. (Fig. 4)

Encoding Immediate Constants

- In ROP, instructions encoding immediate constants can be approximated using a **pop *reg*** instruction
  - e.g., **pop %ebx; ret** will store the next stack word in **%ebx** and advance the stack pointer past it. (Fig. 5)

Control Flow

- In ROP, control flow is affected by perturbing the stack pointer **%esp**.
- Unconditional jumps: **pop %esp; ret** may be used
  - This is a form of immediate load. (Fig. 6)
- Conditional and indirect jumps are more difficult to implement.

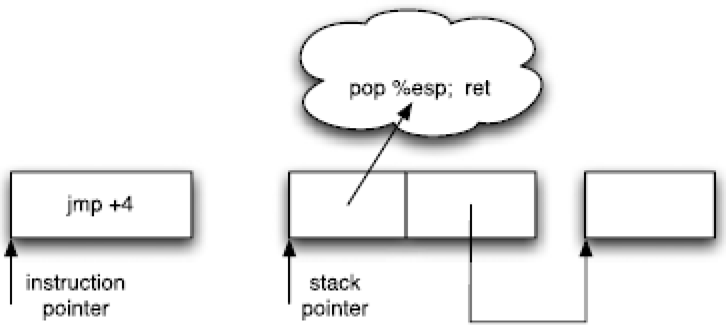

Fig. 6. Ordinary and return-oriented direct jumps.

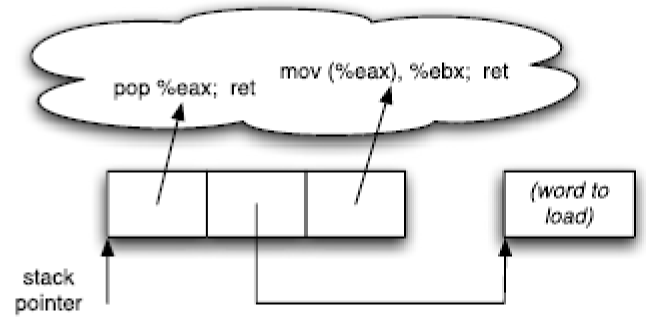

Fig. 7. A memory-load gadget.

Use of Gadgets

- Often, more than one instruction sequence is needed to encode a logical operation.
  - e.g., loading a value from memory may require first reading its address into a register, then reading memory.
- A gadget may include multiple sequence pointers and immediate values. (Fig. 7)
- They act like a return-oriented instruction set.

ROP Exploitation

- A RO program is one or more gadgets arranged to carry out an attack.
- The payload containing it is placed in the target's program's memory.
- The stack pointer must be redirected to point to the first gadget.
- The easiest way to accomplish these steps is a BOF on the stack.
  - The gadgets are placed so the first has overwritten the saved return address.
  - When the active function tries to return, the RO program is executed.
- The ROP payload could also be on the heap.
- The attacker could trigger its execution by overwriting a function pointer with the address of a snippet that sets **%esp** to the address of the first gadget and returns.

ROP Exploitation cont. (2)

- The gadgets need not be placed contiguously.
- With control flow gadgets, an attack can transfer control between the stack and heap.

Mitigations

- Address-space layout randomization (ASLR) defeats attacks that require knowledge of addresses in the target program image.
- It applies to code injection and code reuse attacks equally.
- Control-flow integrity systems also can prevent ROP attacks.

Sources

- R. Roemer et al, *Return-Oriented Programming: Systems, Languages, and Applications*, ACM TISSEC 15, 1, 2012.