Return-Oriented Programming

Scott Hand

CS 6v81.005 Spring 2012

Background and History

Traditional Stack Overflow

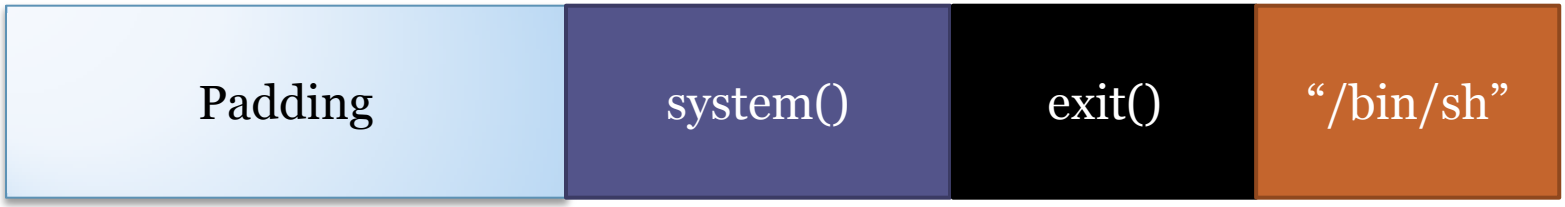
Traditional Stack Overflow

- The simplest stack overflow exploit is the one that operates as follows:
  1. Send a payload with a NOP sled, shellcode, and a pointer to the NOP sled
  2. The pointer to the NOP sled overwrites the saved return address and thereby takes over the stored EIP
  3. EIP now points to the machine code and the program executes arbitrary code

Evaluation

- Pros
  - Very easy to trigger
  - Simple to understand
  - Being able to inject code means our payloads are powerful and flexible
- Cons
  - Just make the stack non-executable
  - Lots of problems with bad characters, buffer sizes, payload detection, etc.

Return-to-libc



Return-to-libc

- Used primarily to streamline exploitation to bypass mitigation and situational limitations
- We want to spawn a shell. Send a payload that overwrites the saved EIP with the address of system(), the address of exit(), and a pointer to "/bin/sh".
- The system call will return directly to exit() which will then shut down the program cleanly.

Evaluation

- Pros
  - Does not need executable stack
  - Also pretty easy to understand and implement
- Cons
  - Relies on access to library functions
  - Can only execute sequential instructions, no branching or fancy stuff
  - Can only use code in .text and loaded libraries

ROP Basics

Basic Idea

- We want to take advantage of the code that already exists in the program
- We will use ESP as a combination of program code and memory
- Each return will be stepping along our ESP program

Gadgets

- ROP operates by executing a string of “gadgets”
- Each gadget will
  - Take values from stack to registers via POP commands
  - Once we have done what we needed to do, return to the next gadget
- Given enough code in a program, gadgets are Turing-complete

Sample Gadgets

- pop eax; ret
  - Loads esp into eax
- pop eax; pop ebx; ret
  - Loads two words from stack and returns. Is a way of incrementing the stack pointer or stepping over instructions
- mov esp, ebx; ret
  - Moves (or “pivots”) the stack into the pointer stored in ebx

Capabilities of Gadgets

- Loading a constant
- Loading from memory
- Storing to memory
- Adding
- Xor
- And, Or, Not
- Shifts, Rotates
- Unconditional and conditional jumps

Automating ROP

Finding Gadgets Automatically

- Obviously looking through a binary for gadgets manually will take a long time
- Tools to help:
  - ropeme
  - ROPgadget
  - Msfrop
- Each one will give a list of addresses of useful gadgets and allow for searching

Automating it All

- ROPgadget has an automated payload generator that transforms any given shellcode into a gadget sequence
- Ropeme has a Python library as well as command line interface that automates many tasks of crafting a ROP payload

Ropeme – Stage 0

- Stage-0
  - Creates a fixed stack at a given writable section
  - This bypasses ASLR and gives better control
- Transfer stage-1 payload to custom stack
  - Pick a byte in payload
  - Search for bytes in binary
  - Generate a strcpy() call
  - Repeat for all bytes in payload
- Transfer control to custom stack with pop ebp; ret followed by move esp, ebp; ret gadgets

Ropeme – Stage 1

- Executing chained ret-to-libc calls
  - Addresses are randomized, but offsets remain constant
  - Can resolve run-time randomized libc addresses by using ROP gadgets to calculate the distance from a known address in GOT
- Overwrite GOT entries
  - Load an offset into register
  - Add register to GOT entry
  - Return to PLT entry

Demo

- Ropeme demonstration

Conclusion

- Any questions?