Intro to Binary Exploitation
Day 1

Nathan Peercy

September 21, 2020

General Announcements & Introduction

- Assuming you are familiar with RE concepts. Watch Rowan's Day 1 presentation if you haven't yet.
- Caveat: you don't **have** to know anything to get something out of this.
- I will answer questions from chat.
- Docker container; GitHub Repo: https://github.com/b01lers/bootcamp-pwn-examples.git!

Insert a bit about me here . . .

What is Binary Exploitation? - Demo 0 [Maybe]: Demo CTF Challenge

- Making a [compiled/binary] program do something it wasn't intended to
- Most often getting arbitrary code execution
- Usually involves memory corruption

Common Vulnerabilities

Preview of many of the types of vulnerabilities we'll see and learn how to exploit:

- Bash/Shell Tricks
- Buffer Overflows
- Off-by one errors
- Function Misuse
- Race Conditions
- Input Validation

Common Defenses

Preview of defenses we will learn to bypass:

- ASLR - Address Space Layout Randomization
- PIE - Position Independent Execution
- NX - Non-eXecutable Memory
- Stack Canaries

Tooling

Tools that we will be using:

- GDB with GEF
- python3 w/ pwntools
- one_gadget, checksec, ROPGadget, a few others
- Ghidra & RE Tools

Example 1: Getting Familiar with pwntools

Demo

Example 1: Bug 1

```c
struct {
    int secret_number;      // Secret Number
    int number_list[16];    // List of known numbers
    char name[32];          // Name of Winner
    int score;              // Your score
} game_state ;

...

printf("Which number in number_list would you like to print?\n> ");
int choice;
scanf("%d", &choice);

printf("number_list[%d] = %d\n", choice, game_state.number_list[choice]);
```

Example 1: Bug 2

```c
struct {
    int secret_number;      // Secret Number
    int number_list[16];    // List of known numbers
    char name[32];          // Name of Winner
    int score;              // Your score
} game_state ;

printf("Enter your name to save your score:\n> ");
scanf("%s", game_state.name);
```

Example 1: Bugs 3 & 4

```
srand(time(NULL));

// Initialize Game state
game_state.secret_number = rand() % 1000; // Generate an impossible to guess number
for(int i = 0; i < 16; i++) {
    game_state.number_list[i] = rand() % 1000;
}
```

Example 1: Pwntools + Writing Solve Script

Pwntools is a great utility to interact with a program via Python.
Useful pwntools methods:

```
from pwn import *  # Common pattern

# Open Processes
p = process('./example01')

# Connect to remote
r = remote('127.0.0.1', 1337)

# Write To Process
p.send(message)
p.sendline(line) # Appends a '\n' to your input
p.sendlineafter(b' >', line)

# Read From Process
p.recv() # Recieves until latest character
p.recvline()
p.recvuntil(b' >')
p.recvall() # Recieves until an EOF is reached

# Manual Control
p.interactive()
```

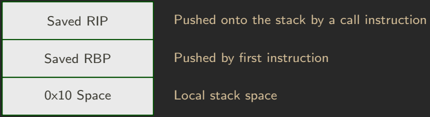
Example 02: Stack Buffer Overflow

```c
#include<stdio.h>
#include<stdlib.h>

void win1() {
    printf("You win! Now try to use this program to call /bin/sh");
}

void win2(char * arg) {
    system(arg);
}

void main() {
    setvbuf(stdin, 0, 2, 0);
    setvbuf(stdout, 0, 2, 0);
    char buffer[16];

    gets(buffer);
}
```

Example 02: Stack Frames - A Refresher

Every time a function is called, something called a stack frame is created. Lets go through an example function:

```
/* This is the program Prologue */
0x000000000040113a <+0>:        push   rbp
0x000000000040113b <+1>:        mov    rbp,rsp
0x000000000040113e <+4>:        sub    rsp,0x10
```


Saved RIP
Pushed onto the stack by a call instruction
Saved RBP
Pushed by first instruction
0x10 Space
Local stack space

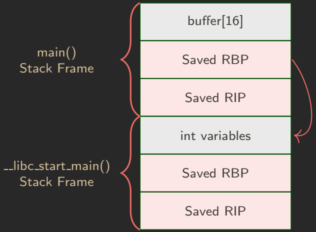
The program body would go in here . . .

```
/* This is the program Epilogue */
0x0000000000401169 <+47>:        leave
0x000000000040116a <+48>:        ret
```

These instructions pop the saved RBP from the prologue back into RBP to return the stack frame's context back to the caller of the current function.

Example 02: Function Calls

Given that we can overflow buffer[16], what does this mean for us?



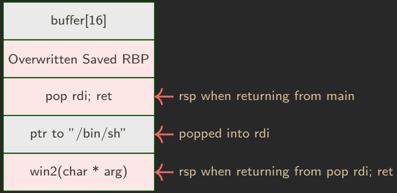
Demo.

Example 02: Calling win2

- How do we call win2(char * arg) with the correct argument?
- Remember the System V calling convention in x86_64:
  - Arguments are passed in registers rdi, rsi, rdx, rcx, r8, r9, in order.
  - Additional arguments are passed on the stack.
  - Return values are returned in rax.
  - Functions must preserve rbx, rsp, rbp, r12-15.
- We need to set rdi to a pointer to "/bin/sh" before calling win2.

Example 02: Return Oriented Programming

- It turns out that we can chain 'returns'
- Gadget: Short but useful instruction(s) usually followed by a return
  - pop rdi; ret
  - inc rax; ret
  - syscall
- Tools like ROPGadget help
- If the gadget RIP does not modify the stack, ret will return to the next address that RSP is pointing to.



Demo in GDB.

Shellcoding ASLR, and NX

- I totally ignored exploits that use an executable stack.
- They are rare now, but they had their place in history.
- Before ASLR and NX protections, it was the easiest technique to use. You could overflow a buffer, put your shellcode in it, then jump directly to it by overwriting the return pointer.
- ASLR: We don't know the address of the stack to jump to.
- NX: We can't execute code on the stack.

ASLR - Address Space Layout Randomization

Randomly arranges addresses of stack, heap, and libraries.

```
> cat /proc/673685/maps
00400000-00401000 r--p 00000000 08:05 673685                 /home/pwn/example01/example01
00401000-00402000 r-xp 00001000 08:05 673685                 /home/pwn/example01/example01
00402000-00403000 r--p 00002000 08:05 673685                 /home/pwn/example01/example01
00403000-00404000 r--p 00002000 08:05 673685                 /home/pwn/example01/example01
00404000-00405000 rw-p 00003000 08:05 673685                 /home/pwn/example01/example01
7ff2768e1000-7ff2768e4000 rw-p 00000000 00:00 0
7ff2768e4000-7ff276906000 r--p 00000000 08:05 1979523        /usr/lib/libc-2.31.so
7ff276906000-7ff276a4a000 r-xp 00022000 08:05 1979523        /usr/lib/libc-2.31.so
7ff276a4a000-7ff276a99000 r--p 00166000 08:05 1979523        /usr/lib/libc-2.31.so
7ff276a99000-7ff276a9d000 r--p 001b4000 08:05 1979523        /usr/lib/libc-2.31.so
7ff276a9d000-7ff276a9f000 rw-p 001b8000 08:05 1979523        /usr/lib/libc-2.31.so
7ff276a9f000-7ff276aa5000 rw-p 00000000 00:00 0
7ff276aa5000-7ff276aa6000 r--p 00000000 08:05 1979514        /usr/lib/ld-2.31.so
7ff276aa6000-7ff276ac5000 r-xp 00001000 08:05 1979514        /usr/lib/ld-2.31.so
7ff276ac5000-7ff276acd000 r--p 00020000 08:05 1979514        /usr/lib/ld-2.31.so
7ff276ace000-7ff276acf000 r--p 00028000 08:05 1979514        /usr/lib/ld-2.31.so
7ff276acf000-7ff276ad0000 rw-p 00029000 08:05 1979514        /usr/lib/ld-2.31.so
7ff276ad0000-7ff276ad1000 rw-p 00000000 00:00 0
7ffe07926000-7ffe07947000 rw-p 00000000 00:00 0              [stack]
7ffe079af000-7ffe079b2000 r--p 00000000 00:00 0              [vvar]
7ffe079b2000-7ffe079b3000 r-xp 00000000 00:00 0              [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0      [vsyscall]
```

PIE - Position Independent Execution

Randomly chooses addresses the program is loaded into memory, and uses relative jumps for control flow instead of full addresses.

```
> cat /proc/2375250/maps
55c2f52c5000-55c2f52c6000 r--p 00000000 08:05 2375250      /home/pwn/example03/example03
55c2f52c6000-55c2f52c7000 r-xp 00001000 08:05 2375250      /home/pwn/example03/example03
55c2f52c7000-55c2f52c8000 r--p 00002000 08:05 2375250      /home/pwn/example03/example03
55c2f52c8000-55c2f52c9000 r--p 00002000 08:05 2375250      /home/pwn/example03/example03
55c2f52c9000-55c2f52ca000 rw-p 00003000 08:05 2375250      /home/pwn/example03/example03
7f17102ab000-7f17102ae000 rw-p 00000000 00:00 0
7f17102ae000-7f17102d0000 r--p 00000000 08:05 1979523      /usr/lib/libc-2.31.so
7f17102d0000-7f1710414000 r-xp 00022000 08:05 1979523      /usr/lib/libc-2.31.so
7f1710414000-7f1710463000 r--p 00166000 08:05 1979523      /usr/lib/libc-2.31.so
7f1710463000-7f1710467000 r--p 001b4000 08:05 1979523      /usr/lib/libc-2.31.so
7f1710467000-7f1710469000 rw-p 001b8000 08:05 1979523      /usr/lib/libc-2.31.so
7f1710469000-7f171046f000 rw-p 00000000 00:00 1979514      /usr/lib/ld-2.31.so
7f1710470000-7f171048f000 r-xp 00001000 08:05 1979514      /usr/lib/ld-2.31.so
7f171048f000-7f1710497000 r--p 00020000 08:05 1979514      /usr/lib/ld-2.31.so
7f1710498000-7f1710499000 r--p 00028000 08:05 1979514      /usr/lib/ld-2.31.so
7f1710499000-7f171049a000 rw-p 00029000 08:05 1979514      /usr/lib/ld-2.31.so
7f171049a000-7f171049b000 rw-p 00000000 00:00 0
7fff6bc59000-7fff6bc7a000 rw-p 00000000 00:00 0            [stack]
7fff6bd3f000-7fff6bd42000 r--p 00000000 00:00 0            [vvar]
7fff6bd42000-7fff6bd43000 r-xp 00000000 00:00 0            [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0    [vsyscall]
```

Example 03 - Defeating ASLR and PIE

- Checksec: Tool to check protections
- Demo.

Example 03 - Partial Overwrite: Summary

- The less significant bits don't change with ASLR and PIE
- The last 12 bits are from the ELF itself.
  - This from page alignment in virtual memory
- Overwrite least significant byte of return address
- Can reasonably use a brute force to overwrite up to 3 total bytes
  - 1.5 bytes from PIE, brute force other 1.5 bytes.
  - 4+ if you rent a VPS in the same ec2 region :)
  - Sometimes used to brute the address of libc.
- Watch out for null bytes at the end of your input!

Example 04 - Global Offset Table and Leaking Libc

- Find the Bug
- Ret2Libc Overview
- Got Refresher
- Demo

Example 04 - Find the Bug

```c
#include <stdio.h>
#include <stdlib.h>

long numbers[16];

int main() {
    setvbuf(stdin, 0, 2, 0);
    setvbuf(stdout, 0, 2, 0);

    char * inp;
    int c;

    while(1) {
        inp = malloc(16);
        printf("Which number would you like to write to?\n> ");
        fgets(inp, 16, stdin);
        c = atol(inp);
        printf("What value should be written?\n> ");
        fgets(inp, 16, stdin);
        numbers[c] = atol(inp);
        printf("Which number would you like to print?\n> ");
        fgets(inp, 16, stdin);
        c = atol(inp);
        printf("%lu\n", numbers[c]);
        free(inp);
    }
}
```

Example 04 - Ret2Libc

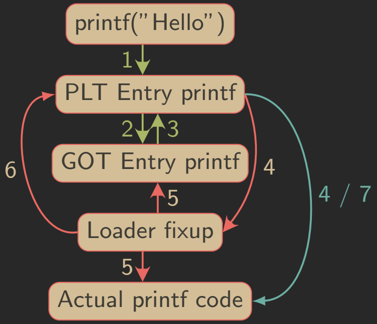
- Overwrite the return address to an address of libc, having corerectly set up arguments
- Example02 'win2', but instead of win2, call the function 'system(char *)' from libc
- The string "/bin/sh" is in libc, so if we have the address of system, we have the address to "/bin/sh".
  - Even if the full string isn't available, the string "sh" will also often work.
- In all modern CTFs, a libc leak is required
- In x86 32-bit, no ROP to set up arguments are required, since arguments are passed on the stack

Example 04 - GOT Refresher

Whenever a shared library function is called for the first time...

- 1. The program jumps to the Procedure Linkage Table (PLT) entry for the shared library function (like printf).
- 2. The PLT stub looks up the entry for printf in the Global Offset Table (GOT) and jumps to it.
- 3. The GOT entry for printf is initialized to the next instruction of the PLT, so we jump back.
- 4. The PLT stub jumps absolutely to the loader fixup.
- 5. The loader looks up the function and replaces the dummy GOT entry with the actual address.
- 6. The loader restarts the call to printf with the newly patched GOT.
- 7. printf executes and returns back to our code, and execution continues.
- 8. On subsequent calls, we don't go to the loader (blue/green only).

The first time printf is called, the following happens (red: first call only, green: always, blue: subsequent only):



Example 04 - Demo

Demo.

Example 04 - Libc Offsets

- Libc offsets and finding functions in libc has been mentioned a few times.
- If you have the address of one thing in a shared library, you can calculate the address of any other address
- Pwntools helps

```python
libc = ELF('./libc')
strtol_leak = read(leak)

# Get base of libc
libc_base = strtol_leak - libc.symbols['strtol']

# Any address in libc can be calculated with this:
system_addr = libc_base + libc.symbols['system']
```

Example 05 - Format Strings

- What's wrong with this code?

```
fgets(input, 512, stdin);
printf("Hello ");
printf(input);
```

Example 05 - Variadic Arguments

```
int printf(const char *format, ...)
```

- No obvious way to pass the number of arguments.
- Examples:
  - "%d" prints the second argument to printf as an integer
  - "%hhd%hhd" prints the second and third arguments as single byte integers.
  - "%3$x" reads the fourth argument as a hex integer
  - "%2$s" dereferences the third argument and prints the text pointed to as a string.
  - "%300c" prints 300 spaces, then the 2nd argument as a character.
  - "AAAA%12$n" dereferences the 13th argument and writes the total number of characters written so far (4) to the integer pointed to.
- Calling Convention System V:
  - Arguments are passed in registers rdi, rsi, rdx, rcx, r8, r9, in order.
  - Additional arguments are passed on the stack.
- What happens when we control the format string?
- What happens when your input is also on the stack?

Example 05 - Demo

Demo

Questions?

Ask away!