Intro to Binary Exploitation

Day 2

Nathan Peercy

September 25, 2020

General Announcements & Introduction

- RE Day 1, PWN Day 1.
- Will answer questions from chat.
- Docker container
- GitHub Repo: https://github.com/b01lers/bootcamp-pwn-examples.git

Tooling

Tools that we will be using:

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

Day 1 Review

- Intro to Pwntools
- Out of bounds array indexes
- Buffer overflows
- Return Oriented Programming
- Partial Overwrites
- Global Offset Table and Leaking Libc

Outline

- Review Defenses
- Stack Canaries & Ret2Libc
- Format Strings
- Heap

Defenses Review

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

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]
```

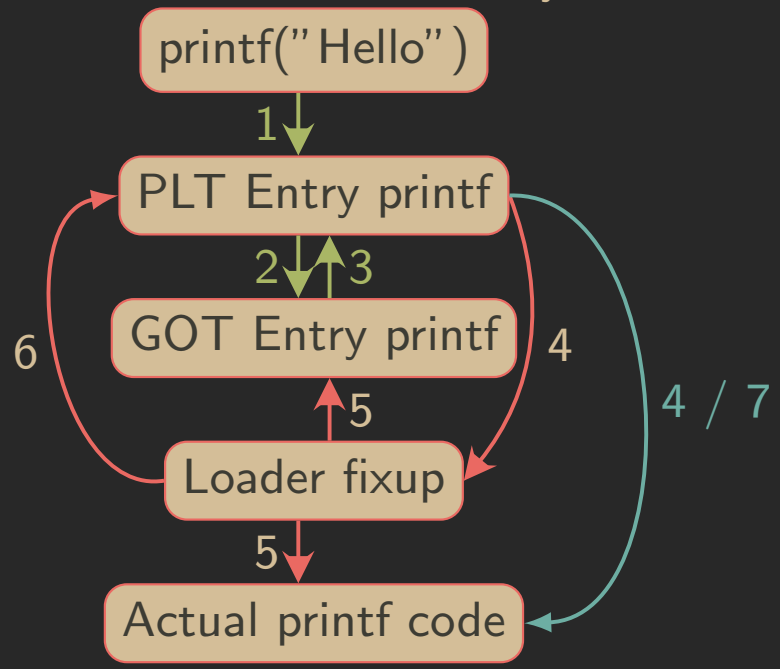
NX - Non-eXecutable Memory

Executable memory memory is never marked as writeable, and writeable memory never marked as executable.

```
> cat /proc/2375250/maps # NX Disabled
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 rwxp 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 rwxp 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]
```

RELRO - RELocation Read-Only

GOT - Global Offset Table is initialized before program enters main execution and is marked as read-only.



Stack Canaries

- AKA 'Stack Cookies'
- Random value on stack just before saved RBP & RIP
- Inserted in functions where user input is read to the stack
  - 64bits / 8 bytes on x86_64
- On 64-bit, first byte is NULL to make leaks more difficult.
- Show in GDB

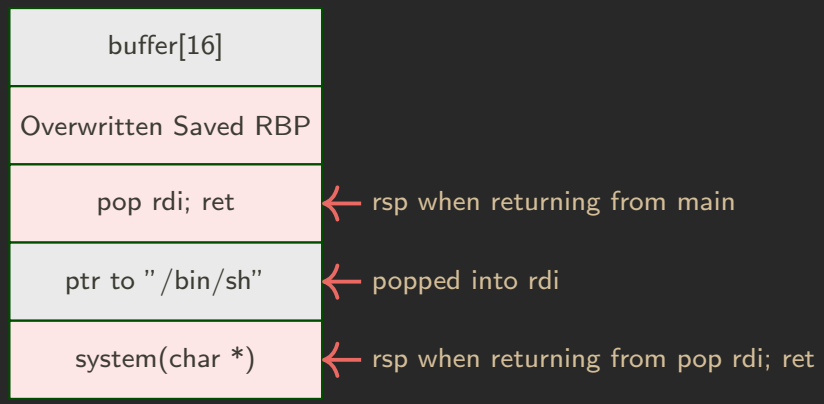
Example 5: Stack Canaries and Ret2Libc

First demo with default modern protections.

```
[*] '/bootcamp-pwn-examples/example05/example07'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

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 x86 32-bit, no ROP to set up arguments are required, since arguments are passed on the stack



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

```
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']
```

Stack Canaries

- AKA 'Stack Cookies'
- Random value on stack just before saved RBP & RIP
- Inserted in functions where user input is read to the stack
  - 64bits / 8 bytes on x86_64
- On 64-bit, first byte is NULL to make leaks more difficult.

Bypassing Stack Canaries

- Leaks
  - Format Strings
  - Print N bytes
  - Out of bounds indexes
  - Non-null terminated string
- Brute Force
  - Brute force one byte at a time
  - Applicable if the program auto restarts, forks, or has a try/catch.
- Ignoring them
  - Write directly to return pointer
  - Write to other memory location (GOT)

Example 6 - Format Strings

- What's wrong with this code? Why?

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

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 6: Demo

Demo

Heap Exploitation

Most common vulnerability class in modern production software. Common heap vulnerabilities

- Heap Overflow
- Use After Free
- Double Free
- Wild Free

About Heap

- malloc(int size): Allocates memory of at least size "size."
- free(void * mem): Frees memory to be available later.
- Different heap implementations. ptmalloc in linux glibc
- Linked list of memory allocations.
- Free memory kept available by inserting into different linked lists depending on size.
- Many security checks (size checks, double free checks, etc)
- How to understand heap better:
  - Read comments in the source https://sources.debian.org/src/glibc/2.28-10/malloc/malloc.c/#L1072
  - CS 252 implements a version of malloc

Example 7: Heap (tcache)

- Ubuntu 18.04 implemented the tcache for small allocation sizes.
- Different linked list for each size
- Tcache uses a singly linked list
- Fewer security checks, usually easier to exploit.

Example 7: Heap (tcache)

Demo

Questions?

Ask away!

References & Heap Resources

- http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf
- https://sources.debian.org/src/glibc/2.28-10/malloc/malloc.c/