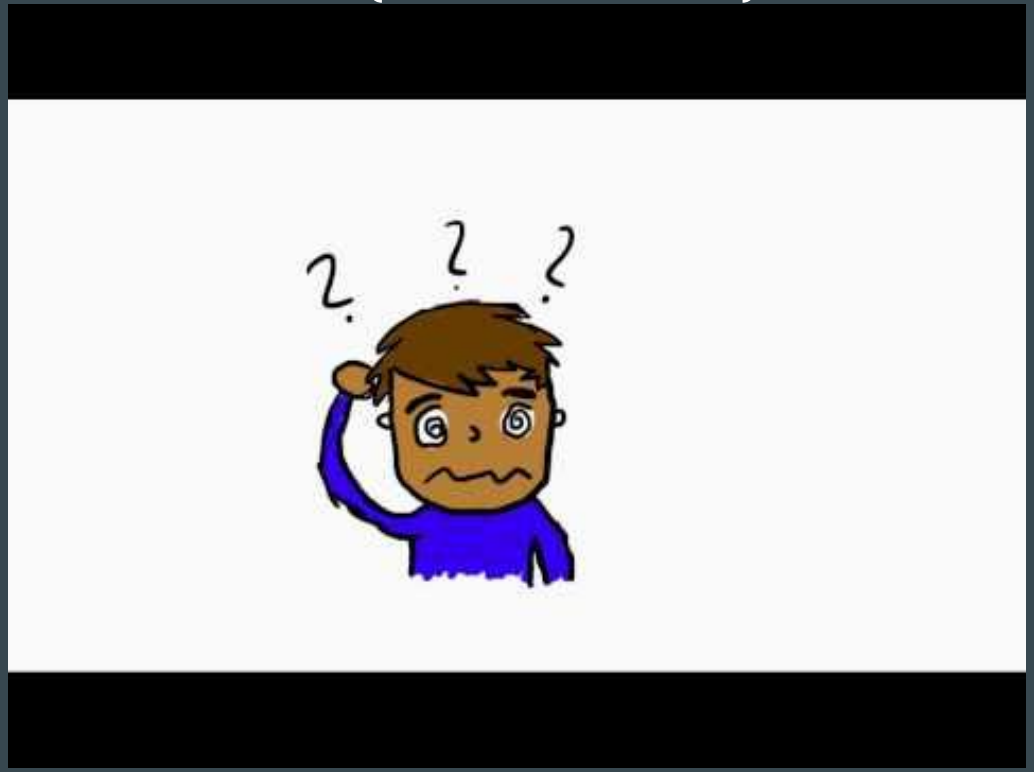
Reading, Writing, and Format String Attacks

Daniel Chen

Video from Live Overflow: (4:20 - 7:33)

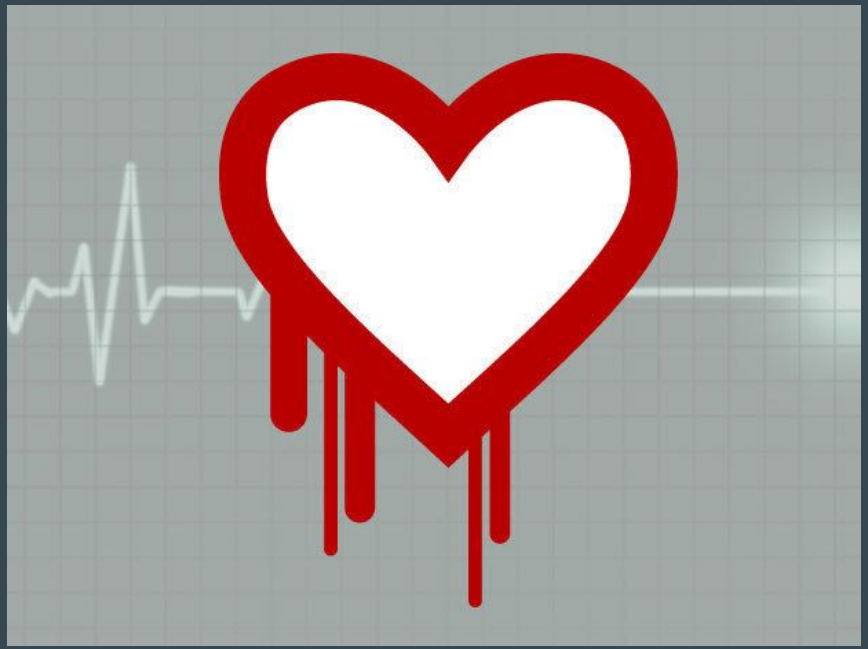


Objectives in exploit development

- Leak out memory - Reading
  - View decision making points
  - Values
  - Addresses
- Control execution flow - Writing
  - Change decision making points
  - Global Offset Table
  - Function pointers
  - Return addresses
- Hence why arbitrary reads and writes are very powerful!

Case study: Heartbleed

- Missing input validation check on the length of the heartbeat TLS response
- Result: Leaking of sensitive data, passwords, cookies, etc from memory



Ways to get the arbitrary read/write

- Use after free
- Double free
- Improper dynamic memory allocation/Heap grooming
- Format string attacks
  - What we will be focusing on today

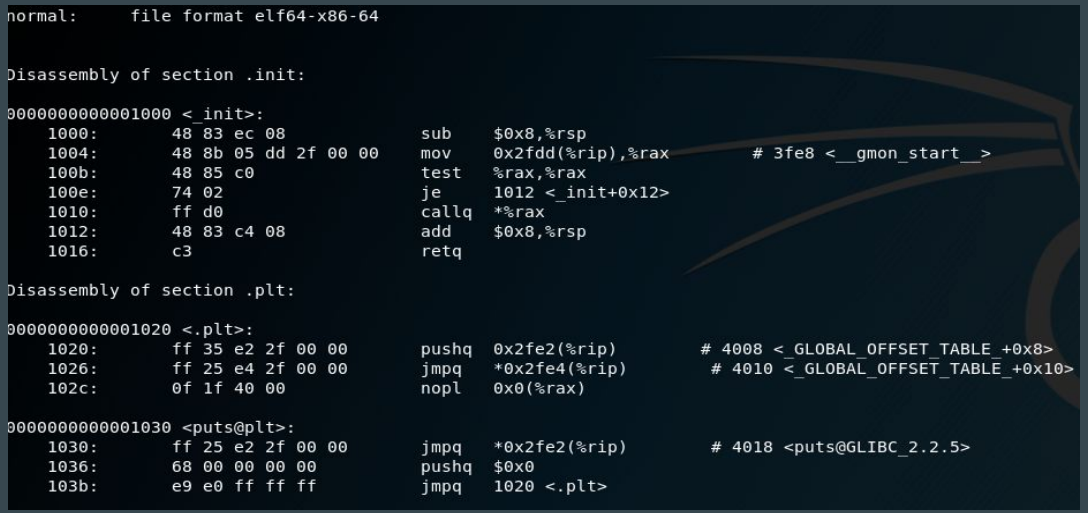
Binary Security

Stack Canaries?

- Little to no impact against arbitrary reads/writes
  - Only useful against attacks that will corrupt a large amount of the stack memory (e.g. buffer overflows)

Position Independent Executable (PIE)

- Randomizes the addresses of the .data and .text sections
  - All the function locations and global variable addresses are randomized!
- Makes it infeasible to pinpoint where to write, or what to leak.
- Enabled by default, use the -no-pie option on gcc to disable



Relocation Read-Only (RELRO)

- Changes the Global Offset Table permissions
- Partial Relocation Read-only:
  - Prevents buffer overflows on global variables from overwriting the Global Offset Table
  - Little to no effect on arbitrary read/write attacks
  - Enabled by default
- Full Relocation Read-only:
  - Makes the Global Offset Table read-only, thereby preventing GOT overwrite attacks.
  - Increase program startup time
  - Disabled by default, to enable, add -W1,-z,relro,-z,now to gcc during program compilation

Address Space Layout Randomization (ASLR)

- Randomizes the addresses of library functions, heap addresses, and stack addresses.
- Enabled by default as a kernel setting
- HOWEVER: While the addresses are randomized, the offsets between the addresses remain constant.
  - Within a given C library, the distance between &printf() and %system() is the same!
  - Hence, if you leak a single C library function address, you can calculate the address of all other C library function addresses!

Format string parameter overview:

- Used as a placeholder that translates parameters to values
- %[parameter][width][length]type
  - Type: Output format
    - %x
  - Parameter: Specify which parameter to print
    - %5$x
  - Width: Specify minimum characters to print out
    - %10x
  - Length: Specify the size of the parameter to print out
    - %hhx
- Used in I/O functions within many programming languages.

Format string parameter overview:

- Parameters:
  - “d$”
    - Where d is the position of the parameter to print out (in decimal)
- Length:
  - “” - Prints out a 4 byte value
  - “h” - Prints out a 2 byte value
  - “hh” - Prints out a byte value
  - “ll” - Prints out a 8 byte value
- Width:
  - “d”
    - Where d is the number of bytes to print out (in decimal)
- Type:
  - “x” - Prints in hex format
  - “d” - Prints in signed decimal format
  - “u” - Prints in unsigned decimal format
  - “s” - Prints out a null-terminated string representation of a POINTER
- Special type: “n”
  - Does not print anything out, but writes the number of characters successfully printed so far into the **location** of the next parameter.
  - The parameter and length attributes affect this type.

Overview: Global Offset Table

- Table of addresses stored in the .data section
  - Includes pointers to C library functions!
- Used in dynamically linked binaries where global addresses are unknown until runtime
- The Procedure Linking Table provides assembly code that tells the program to jump to the address stored in the Global Offset Table

```
08049040 <printf@plt>:
 8049040:       ff 25 0c c0 04 08       jmp    *0x804c00c
 8049046:       68 00 00 00 00          push   $0x0
 804904b:       e9 e0 ff ff ff          jmp    8049030 <.plt>

08049050 <fgets@plt>:
 8049050:       ff 25 10 c0 04 08       jmp    *0x804c010
 8049056:       68 08 00 00 00          push   $0x8
 804905b:       e9 d0 ff ff ff          jmp    8049030 <.plt>
```

```
(gdb) x/lx 0x804c00c
0x804c00c:      0xf7e21830
(gdb) print &printf
$2 = (<text variable, no debug info> *) 0xf7e21830 <printf>
```

Demo time!

Your turn!

- Before we begin:
  - All challenges use 32-bit binaries, so take that into account when counting parameters!
  - All challenges are running on an Ubuntu 18.04 server
  - All challenges have the PIE security setting disabled.
  - Other security settings are left as default.
  - For format7, the designated memory region is readable, writable, and executable.
- Reminders:
  - All flags are in flag{XXXXXXXXXXXXX} format.
  - Linux binary memory are **little endian**, so take that into account when reading/writing data!
  - printf() stops when it reaches a null byte
  - The max value for a variable depends on its type, if you go past it, you will get the min value for its type! This applies to format strings! (especially the length attribute)

Connect to: nc 128.143.67.98 <port_number>

- Format0: port 30000
  - Can you read me?
- Format1: port 30001
  - Guess my numbers! The C library's rand() function is very secure!
- Format2: port 30002
  - Time to do some writing!
- Format3: port 30003
  - Learning how to write, part 2
- Format4: port 30004
  - Aim carefully and overflow!
- Format5: port 30005
  - Implement a Global Offset Table overwrite attack!
- Format6: port 30006
  - How fast can you overwrite the GOT?
- Format7: port 30007
  - Shellcode time!

References

https://en.wikipedia.org/wiki/Heartbleed

https://www.youtube.com/watch?v=akCce7vSSfw

https://medium.com/@HockeyInJune/relro-relocation-read-only-c8d0933faef3

https://ctf101.org/