Return Oriented Programming

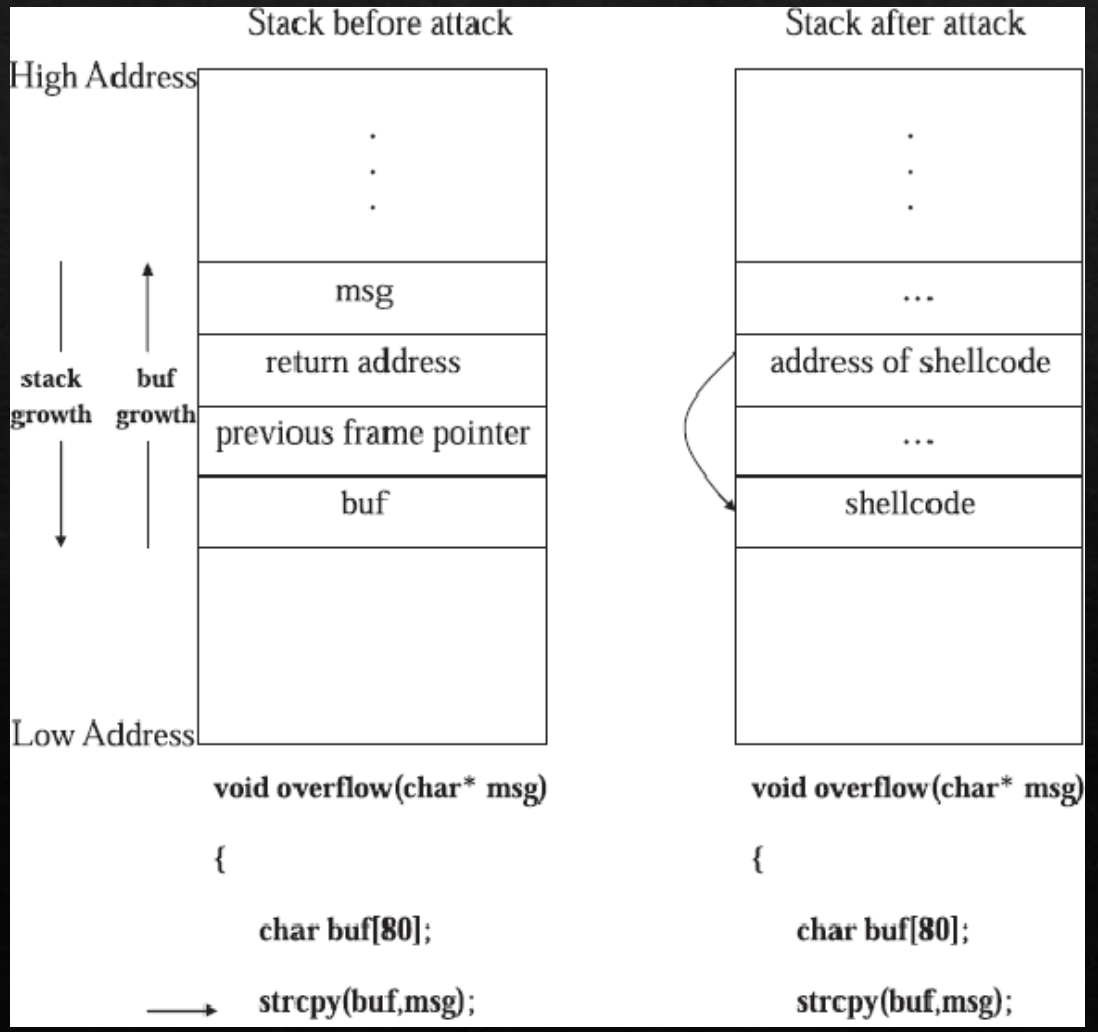
Part 4 of Binary Exploitation

4 Week Plan

- Week 1: Assembly & Shellcoding
  - Writing our own assembly, writing some shellcode. Getting used to debugging tools
- Week 2: Reverse Engineering
  - Learning some basic reversing techniques, getting used to reversing frameworks
- Week 3: Stack smashing
  - Basic program exploitation. How to exploit programs that have little/no protections
- Week 4: Return Oriented Programming
  - Exploiting programs with some modern protections enabled

Recap of Stack Smashing

- 1. Find the offset to the return pointer
  - Cyclic Patterns or Quick Math
- Write shellcode onto the stack
- "Ret" to shellcode
- Prerequisites
  - Stack leak or No ASLR
  - NX Stack Disabled



Recap of Protections

OS Protection

- Address Space Layout Randomization (ASLR)
  - Addresses on stack is randomised - harder to guess addresses in overflow

Compiler Protections

- Position Independent Execution (PIE)
  - Binary compiled using offsets for code location. Actual locations calculated on runtime
- Stack Canaries (Fortify)
  - Sets values that are monitored. If value changes the program stops
- Non Executable Stack (NX Stack)
  - Remove executable permissions from the stack
- Half/ Full Relocation Read Only (RelRO)
  - Global Offset Table Protections

Return Oriented Programming

- Another exploitation method stemming from a buffer overflow
- The use of small snippets of assembly code that exist within the binary to cause it to do unintended things.
- We call these snippets “gadgets”
- Works irrespective of NX Stack
- Made only slightly harder in the presence of ASLR

Common ROP Exploitation Methods

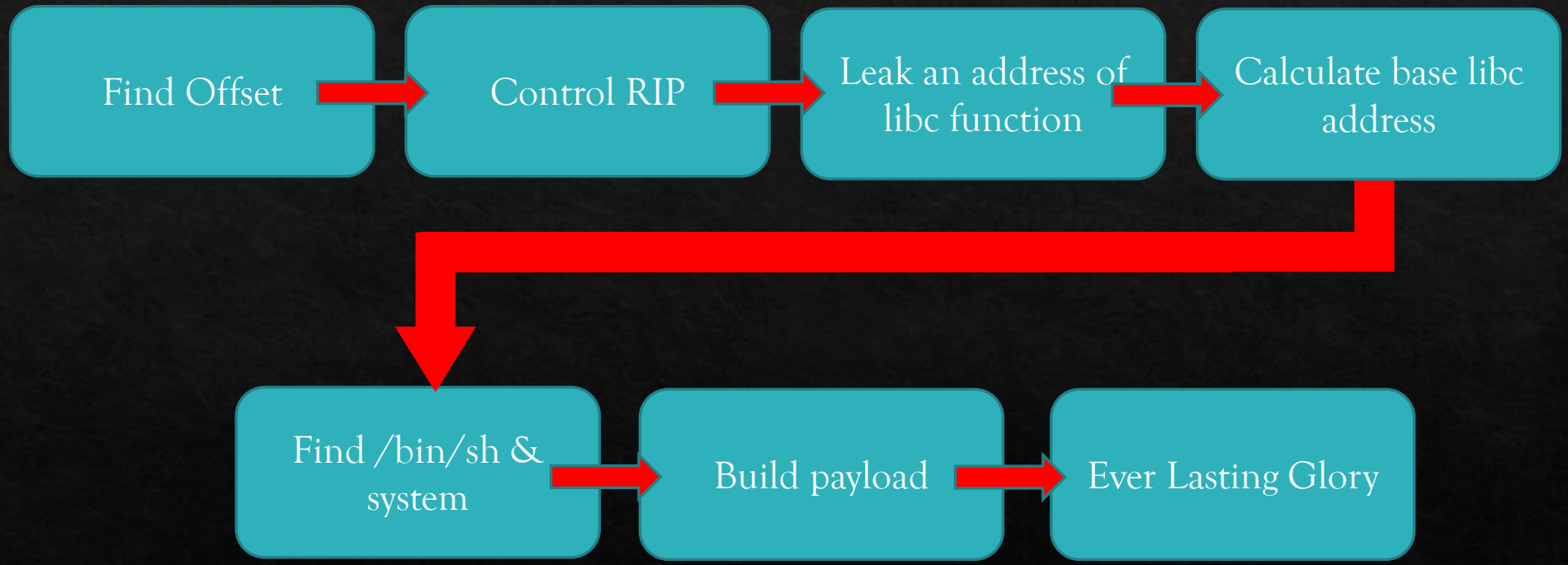
- Classic ROP
- ret2libc
  - Requires the Dynamic Linker
- ret2win
- Arbitrary Write Primitive

```
[*] '/tmp/rop/vuln'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] '/tmp/rop/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Leaked puts :  0x7f9b610ea9c0
[+] Libc at   :  0x7f9b6106a000
[*] Switching to interactive mode
Enter access password:
access denied.
$ $ id & whoami
margo
$ uid=1002(margo) gid=1002(margo) groups=1002(margo)
$
```

The Dynamic Linker

- To save space, programs are linked dynamically into libc
- The Procedural Linkage Table (PLT) resolves the address of LIBC (using dlresolve)
- Dynamic Linking is done through lazy loading
- The Global Offset Table (GOT) stores the address of a libc function
  - If resolved the points to the actual address of the libc function
  - If not resolved it points to the PLT stub for that function

ret2libc worked example




Demo

More Advanced Software Exploitation

- Bypassing PIE
- Format String Exploits
- SigReturn Oriented Programming
- ret2csu
- ret2dlresolve
- ret2vDSO
- Heap Exploitation

Where to Go From Here?

- 3 Challenges
  - ret2libc (no leak required)
  - ret2libc (leak required)
  - ret2libc with a stack canary
- ROP Emporium https://ropemporium.com/
- CTF 101 https://ctf101.org/binary-exploitation/return-oriented-programming/