PWNING PWNABLES

Attacking binary challenges in Capture The Flag competitions (CTFs)

Presenter: Harold Rodriguez (@superkojiman) | superkojiman@techorganic.com

WHO AM I

Harold Rodriguez || superkojiman

- University of Toronto SysAdmin
- Likes binary exploitation and CTFs
- Plays for the VulnHub CTF Team (https://www.vulnhub.com)

Contact

- Twitter: @superkojiman
- IRC : #vulnhub on Freenode
- Website: https://techorganic.com

VulnHub-CTFT

Capture The Flag Team

CTF? PWNABLES?

Capture the Flag

- Competition for hackers (solo or team)
- Goal: solve the challenge, get the flag, score points
- Challenges span various categories

Pwnables: just a program with an exploitable vulnerability

ABOUT THIS TALK

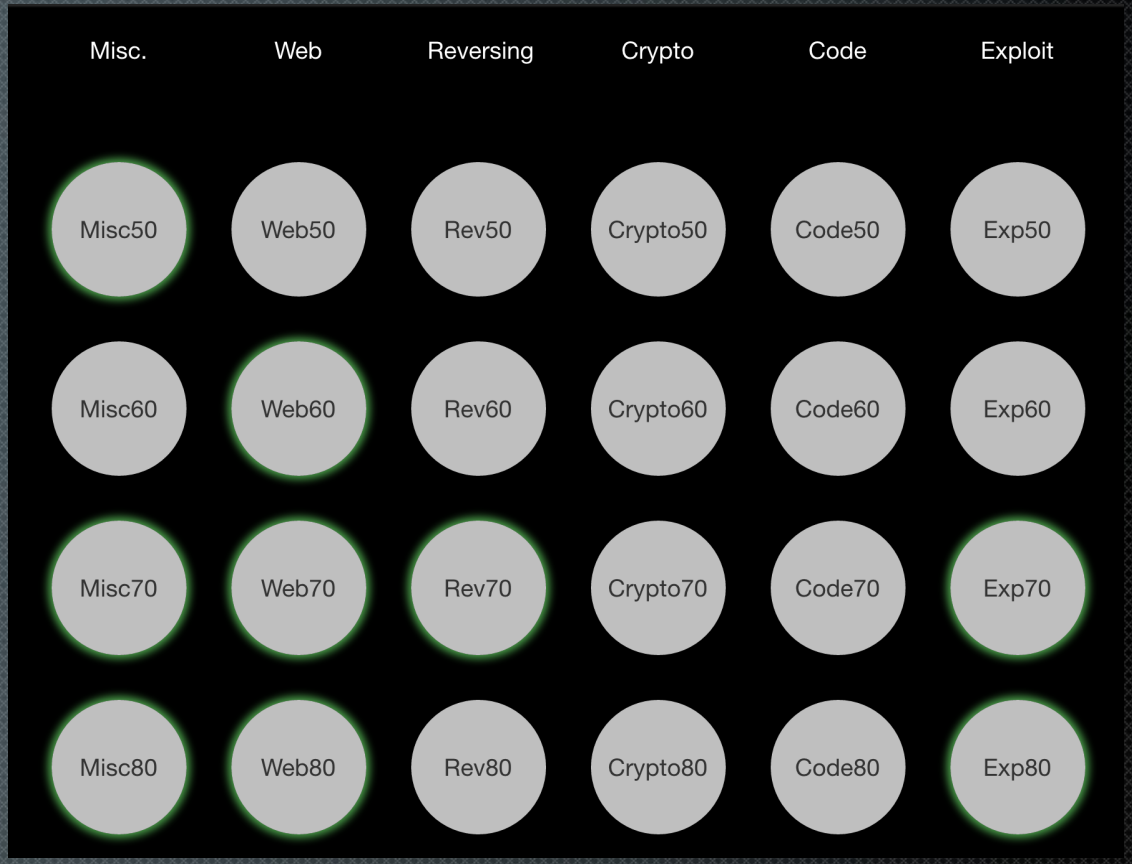
An approach to tackling pwnables in CTFs

- Pwnables can result in swearing and table flipping

(╯°□°)╯ ︵ ┻━┻

- How to get from “wtf?” to “w00t!”?

Jeopardy style CTF challenge board



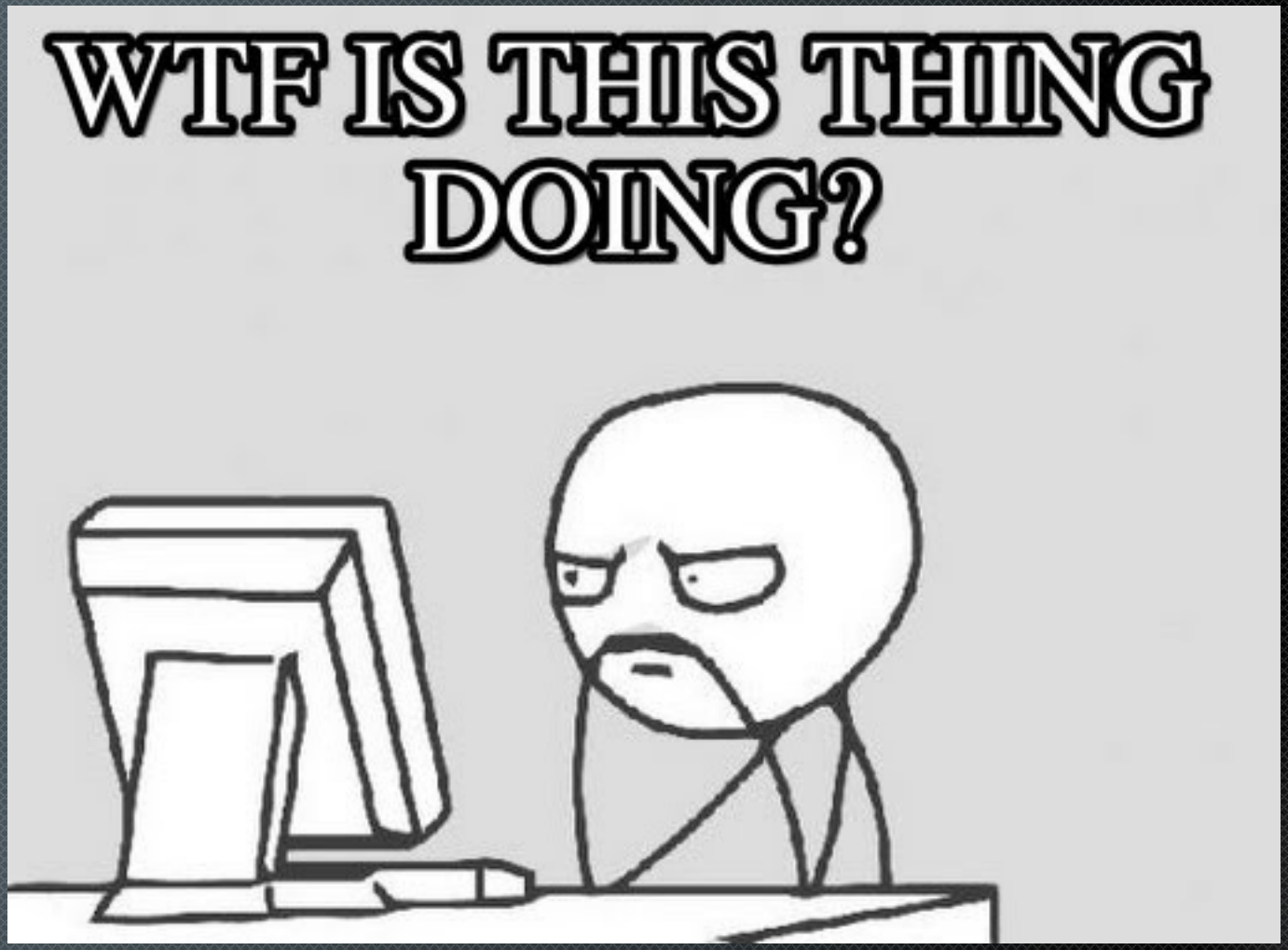
WHAT YOU SHOULD KNOW

- Basic assembly programming (usually x86)
- Using a debugger and disassembler
- Programming

OVERVIEW

- Analysis
- Exploitation
- Live demo

ANALYSIS



ANALYSIS

Goal: learn as much as possible about the binary

- What file format, architecture, 32-bit or 64-bit
- Any exploit mitigations in place
- What happens to input we pass to the binary
- What functions are called to work on the input
- Any interesting strings in the binary

ANALYSIS [FUZZING]

Send all kinds of data and see if something bad happens

Examples:

- Large strings
- Format strings
- Negative or really large numbers

Binaries behaving badly

```
root|dc416|~/work/dc416/analysis
$ ./ex1
Enter something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
root|dc416|~/work/dc416/analysis
$ ./ex2
Enter something: %p.%p.%p.%p.%p
You entered: 0x4006c3.0x7f72702fc9e0.0x203a64657265746e.0x7f727051f00f.(nil)
root|dc416|~/work/dc416/analysis
$ ./ex3
Enter a number: 999999999999999999999999
You entered: -1
root|dc416|~/work/dc416/analysis
$
```

ANALYSIS [REVERSE ENGINEERING]

- Try to understand program's flow
- Look for functions vulnerable to memory corruption, format string leaks, race conditions
- Defined functions that aren't called anywhere
- Functions that make use of the input sent

Disassembly of ex1

```
[0x00400490]> pdf@sym.main
            ;-- main:
/ (fcn) sym.main 42
|           ; var int local_20h @ rbp-0x20
|           ; DATA XREF from 0x004004ad (entry0)
|           0x0040057d      55              push rbp
|           0x0040057e      4889e5          mov rbp, rsp
|           0x00400581      4883ec20        sub rsp, 0x20
|           0x00400585      bf34064000      mov edi, str.Enter_something: ; "Enter something: " @ 0x400634
|           0x0040058a      b800000000      mov eax, 0
|           0x0040058f      e8bcfeffff      call sym.imp.printf
|           0x00400594      488d45e0        lea rax, [rbp - local_20h]
|           0x00400598      4889c7          mov rdi, rax
|           0x0040059b      e8e0feffff      call sym.imp.gets
|           0x004005a0      b800000000      mov eax, 0
|           0x004005a5      c9              leave
\           0x004005a6      c3              ret
[0x00400490]>
```

ANALYSIS [TOOLS]

Disassemblers

- IDA Pro https://www.hex-rays.com/products/ida
- Radare2 https://www.radare.org
- Hopper Disassembler http://www.hopperapp.com

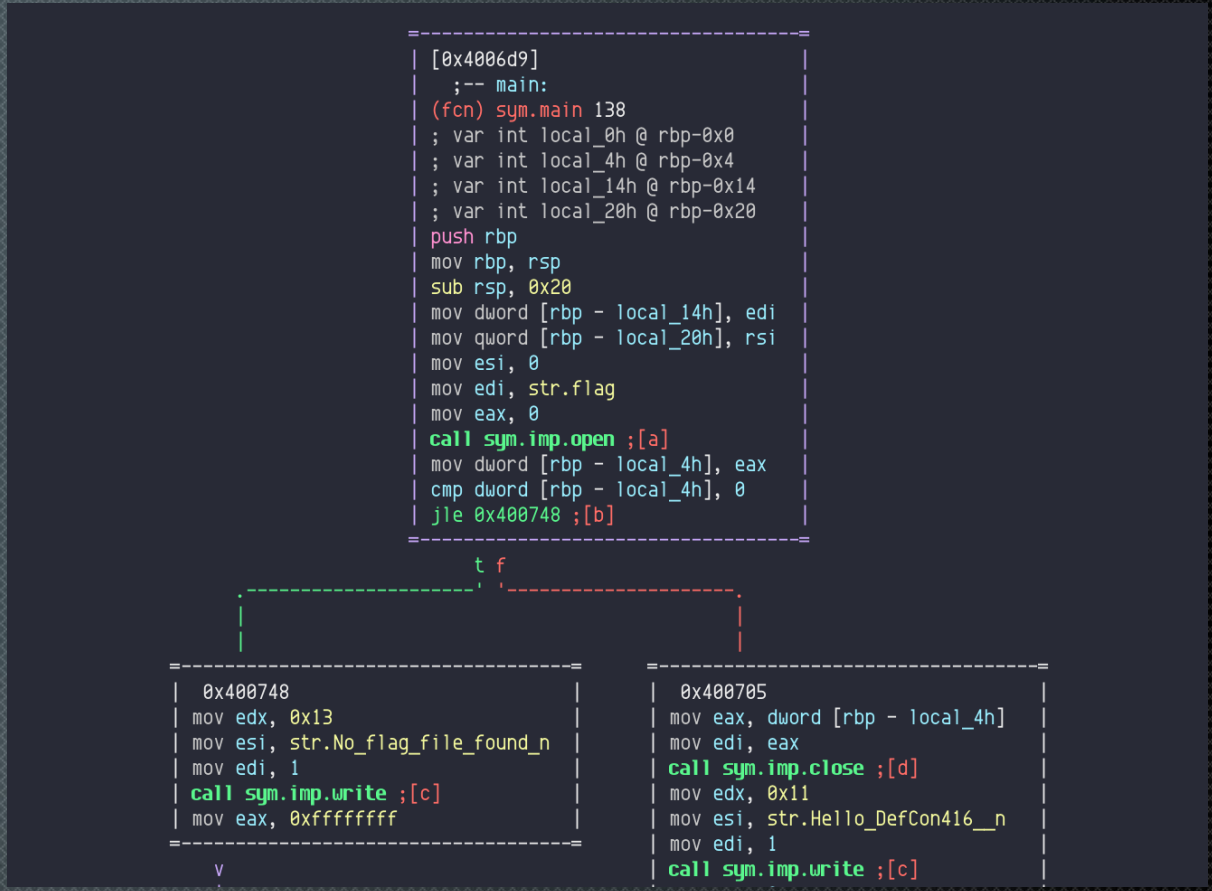
Debuggers

- gdb with PEDA https://github.com/longld/peda

Other tools

- strace, ltrace, readelf, objdump, file, xxd

Radare2 in visual mode



gdb with PEDA

```
gdb-peda$ start
[----------------------------------registers-----------------------------------]
RAX: 0x4006d9 (<main>:  push   rbp)
RBX: 0x0
RCX: 0x0
RDX: 0x7ffdd0d0c738 --> 0x7ffdd0d0deb8 ("HOSTNAME=dc416")
RSI: 0x7ffdd0d0c728 --> 0x7ffdd0d0dea1 ("/root/work/dc416/pwnme")
RDI: 0x1
RBP: 0x7ffdd0d0c640 --> 0x0
RSP: 0x7ffdd0d0c640 --> 0x0
RIP: 0x4006dd (<main+4>:        sub    rsp,0x20)
R8 : 0x7f918c8afe80 --> 0x0
R9 : 0x7f918c8c5530 (push   rbp)
R10: 0x7ffdd0d0c4d0 --> 0x0
R11: 0x7f918c511e50 (<__libc_start_main>:       push   r14)
R12: 0x400580 (<_start>:        xor    ebp,ebp)
R13: 0x7ffdd0d0c720 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x4006d8 <get_reply+98>:     ret
   0x4006d9 <main>:     push   rbp
   0x4006da <main+1>:   mov    rbp,rsp
=> 0x4006dd <main+4>:   sub    rsp,0x20
   0x4006e1 <main+8>:   mov    DWORD PTR [rbp-0x14],edi
   0x4006e4 <main+11>:  mov    QWORD PTR [rbp-0x20],rsi
   0x4006e8 <main+15>:  mov    esi,0x0
   0x4006ed <main+20>:  mov    edi,0x4007f4
[------------------------------------stack-------------------------------------]
0000| 0x7ffdd0d0c640 --> 0x0
0008| 0x7ffdd0d0c648 --> 0x7f918c511f45 (<__libc_start_main+245>:       mov    edi,eax)
0016| 0x7ffdd0d0c650 --> 0x0
0024| 0x7ffdd0d0c658 --> 0x7ffdd0d0c728 --> 0x7ffdd0d0dea1 ("/root/work/dc416/pwnme")
0032| 0x7ffdd0d0c660 --> 0x100000000
0040| 0x7ffdd0d0c668 --> 0x4006d9 (<main>:      push   rbp)
0048| 0x7ffdd0d0c670 --> 0x0
0056| 0x7ffdd0d0c678 --> 0x1cbd4b51ec24e486
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Temporary breakpoint 5, 0x00000000004006dd in main ()
gdb-peda$
```

Found the vulnerability, time to pwn it

EXPLOITATION

EXPLOITATION

Things to try

- Replicate the target environment if possible
- Cyclic patterns to find offsets for overwritten pointers/registers
- Check permission of memory location where input is stored
- Identify bad characters in the payload

EXPLOITATION [TECHNIQUES]

GOT overwrite

- Commonly used in format string exploitation
- Overwrite pointer in GOT with pointer to another location

Code re-use (ret2libc, ret2plt, ROP)

- Make use of existing code and instructions to exploit the binary

Jump to payload

- ret2reg or jump to payload if the stack is executable and addresses aren't randomized

EXPLOITATION [MITIGATIONS]

ASLR (Address Space Layout Randomization)

- Look for non-randomized location to store payload
- Leak a stack or libc address

NX (No-eXecute):

- Code re-use attacks like ROP to make a memory location executable

Stack canary:

- If the binary calls fork(), brute force the canary
- Leak the canary

Code-reuse attack to bypass NX from CSAW 2015: Autobots

```
poprax = p64(libc_base + 0x000293b8)
poprsi = p64(libc_base + 0x00005365)
poprdi = p64(libc_base + 0x0000367a)
poprdx = p64(libc_base + 0x0009da40)

# rop chain starts here
# mprotect chain
buf += poprax
buf += p64(0xa)
buf += poprdi
buf += p64(0x00601000)
buf += poprsi
buf += p64(0x1000)
buf += poprdx
buf += p64(0x7)
buf += syscal
```

EXPLOITATION [TOOLS]

Exploit frameworks

- pwntools https://github.com/Gallopsled/pwntools
- libformatstr https://github.com/hellman/libformatstr

ROP tools

- Ropper https://github.com/sashs/Ropper
- ROPGadget https://github.com/JonathanSalwan/ROPgadget

LIBC database

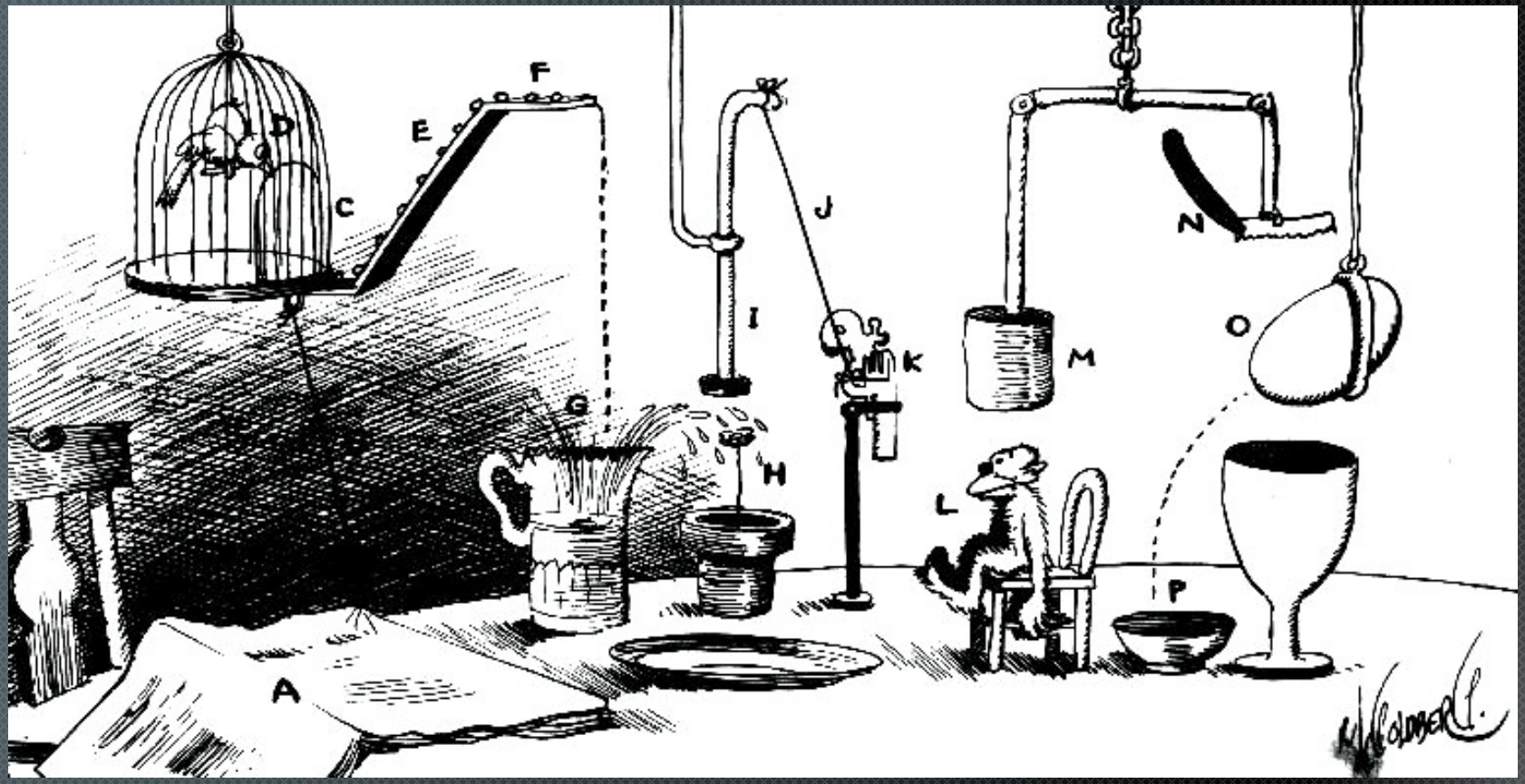
- https://github.com/niklasb/libc-database

EXPLOITATION [GOT SHELL]

So you got a shell. Explore and pillage!

- Get target's libc
- Look for poorly protected flags
- Identify flag names and locations

Exploit just has to work. Doesn't need to look pretty.



RESOURCES

CTF Events: https://ctftime.org

CTF Field Guide: https://trailofbits.github.io/ctf

OpenToAll CTF Team: https://opentoallctf.com

Team VulnHub: https://github.com/VulnHub/ctf-writeups

Solo CTF/boot2root/wargame challenges

- VulnHub: https://vulnhub.com
- OverTheWire: https://overthewire.org
- SmashTheStack: https://smashthestack.org
- Pwnable Kr: http://pwnable.kr