Offensive and Defensive Cybersecurity

23-24

Download Now! (~7GB)

https://bit.ly/ODC23VM

https://bit.ly/ODCVM23_mirror

Schedule (Tentative)

- Shellcode Writing
- Dynamic Loading
- Protection bypass
- RET to lib & ROP
- Heap Exploitation
- Reversing
- Symbolic Execution

- Dom based XSS
- Race Condition
- Serialization
- Unpacking
- Dynamic Malware Analysis
- Hardware Security

- CTF
- 7h

18/09 -> 22/12

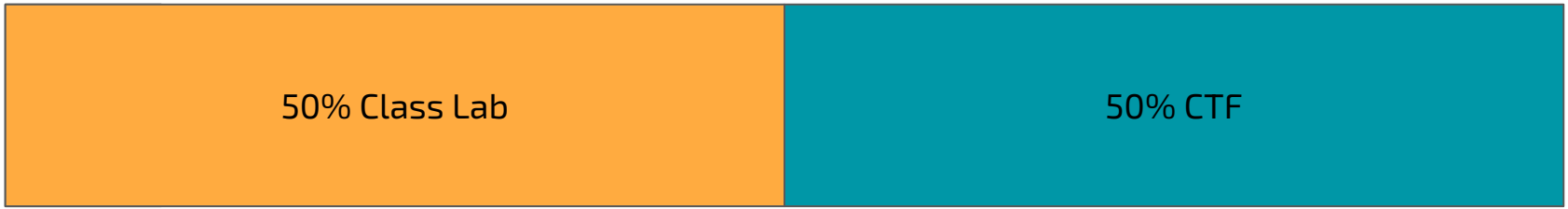
Monday 16:30 -> 18:00 Room: 3.1.5

Friday 16:30 -> 18:00 Room: 3.1.3

Beginning of January

Tentative: 08/01

Evaluation (Tentative)



3 challenge per category

work together

Ask hints!

Individual work

Points are split among solutions

A project can be discussed as an alternative if you cannot take part to the CTF

Prerequisites

- Computer Security
- x86 assembly
  - https://bit.ly/CSX86_2020_p1
  - https://bit.ly/CSX86_2020_p2
- BoF:
  - https://www.youtube.com/watch?v=T03idxny9jE
- python

How to learn exploitation techniques?

- Hands-on! (a lot)
- 30-ish minutes of explanation / demo
- You try on your own!
- Ask questions!
- Do not fall behind
- Recordings on Webeep

What do you need? (https://bit.ly/ODC23VM)

- Linux (ubuntu ~~18.04 LTS~~ 22.04 LTS recommended)
- x86 and x86_64
- GDB (pwndbg, peda, gef, etc…)
  (https://github.com/pwndbg/pwndbg)
- pwntools (`pip install pwntools`)
- ghidra (or IDA Pro) (https://ghidra-sre.org/)
- tmux (screen, terminator)

Binary Challenge Setup

- Challenges (https://training.offdef.it/)
- Remote Service
  - Running on docker on ubuntu 18.04 or 22.04
- You get the binary
- You get the library (some times)
- Read file `“/chall/flag”`

Shellcode?

Writing a Shellcode

- execute a shell!
- plan your shellcode:
  - `exec("/bin/sh")`
    - use an assembler

      (https://defuse.ca/online-x86-assembler.htm)

32 bit vs 64 bit

- Registers
- Syscalls :
  - https://w3challs.com/syscalls/
  - https://syscall.sh/
  - x86 int 80
  - x86_64 syscall
- man is your friend (even google is fine)
  - `man 2 read`

How to Assemble

- GCC (as)
- Nasm
- pwntools
- Online assembler.
  - https://defuse.ca/online-x86-assembler.htm

High-level and Machine Code

Developer

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

int foo(int a, int b) {
  int c = 14;
  c = (a + b) * c;
  return c;
}

int main(int argc, char * argv[]) {
  int avar;
  int bvar;
  int cvar;
  char * str;
  avar = atoi(argv[1]);
  bvar = atoi(argv[2]);
  cvar = foo(avar, bvar);
  gets(str);
  puts(str);
  printf("foo(%d, %d) = %d\n", avar, bvar, cvar);
  return 0;
}
```

≠

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

int32_t foo(int32_t a, int32_t b);

// From module:   layout.c
// Address range: 0x80484ac - 0x80484cd
// Line range:    5 - 10
int32_t foo(int32_t a, int32_t b) {
    int32_t c = 14 * (b + a); // 0x80484c4
    return c;
}

// From module:   layout.c
// Address range: 0x80484cf - 0x8048559
// Line range:    13 - 30
int main(int argc, char **argv) {
    int32_t apple = (int32_t)argv; // 0x80484d8
    int32_t str_as_i = atoi((int8_t *)*(int32_t *)(apple + 4));
    int32_t str_as_i2 = atoi((int8_t *)*(int32_t *)(apple + 8));
    int32_t banana = foo(str_as_i, str_as_i2); // 0x804850f
    gets(NULL);
    puts(NULL);
    printf("foo(%d, %d) = %d\n", str_as_i, str_as_i2, banana);
    return 0;
}
```

Compiler

Decompiler

```
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $32, %esp
        movl    12(%ebp), %eax
        addl    $4, %eax
        movl    (%eax), %eax
```

≠

```
push   %ebp
mov    %esp,%ebp
and    $0xfffffff0,%esp
sub    $0x20,%esp
mov    0xc(%ebp),%eax
add    $0x4,%eax
mov    (%eax),%eax
mov    %eax,(%esp)
call   80483b0 <atoi@plt>
mov    %eax,0x1c(%esp)
mov    0xc(%ebp),%eax
```

Assembler

Disassembler

```
0000000: 01111111 01000101 01001100 01000110 00000001 00000001
0000006: 00000001 00000000 00000000 00000000 00000000 00000000
000000c: 00000000 00000000 00000000 00000000 00000010 00000000
0000012: 00000011 00000000 00000001 00000000 00000000 00000000
0000018: 11000000 10000011 00000100 00001000 00110100 00000000
000001e: 00000000 00000000 10110100 00001100 00000000 00000000
0000024: 00000000 00000000 00000000 00000000 00110100 00000000
000002a: 00100000 00000000 00001000 00000000 00101000 00000000
```

Machine

No src! - What do I use?

- Objdump - Disasm
- radare2 - Disasm
- Binary Ninja - Disasm + Decompiler
- **GHIDRA - Disasm + Decompiler** (https://ghidra-sre.org/)
- rev.ng - Disasm + Decompiler (maybe one day)
- **IDA Pro - Disasm + Decompiler (de facto standard)**

Writing a Shellcode - Multi Stage

- If you do not have space, you make space.
- plan your shellcode:
  - Stage One
    - `read(·,·,·) #second stage`
  - Stage Two:
    - `exec("/bin/sh")`

Writing a Shellcode - Fork Server

- fd 0 or 1 are not always the way.
- plan your shellcode:
  - `dup2(·,·,·)`
  - `exec("/bin/sh")`

Writing a Shellcode open read write

- you may need to read bpf filters (https://github.com/david942j/seccomp-tools)
- plan your shellcode:
  - `open("/flag")`
  - `read(·,·,·)`
  - `write(·,·,·)`

Writing a Shellcode - Reverse Shell

- Connect to remote host.
- plan your shellcode:
  - `socket(·,·,·)`
  - `dup2(·,·,·)`
  - `connect(·,·,·)`
  - `exec(“/bin/sh”)`

My Comfortable Setup

Setup the environment

- You need a **comfortable** test environment
  - Learn how to use the tools
  - Build your own!
- Most **similar** env
  - The **DSA** Assumption (Last Ubuntu LTS)
- debug tools (gdb)
- **Scripting** => **Reproducibility** (pwntools)
- debug while **running** your script...

Debugging Challenges with GDB

Host the challenge:

```
socat TCP-LISTEN:4000,reuseaddr,fork EXEC:"./challenge"
```

Connect your script. (NB You script should wait.)

`python x.py` (OR `ncat 127.0.0.1 4000`)

Attach with gdb:

```
ps aux | grep challenge
```

```
sudo gdb attach 25209
```

Debugging Challenges with GDB the pwntools way

```
1. context.terminal = ['tmux', 'splitw', '-h']
2. r = process("./multistage")
3. gdb.attach(r,'''
4. # b * 0x004000b0
5. # b *0x4000DD
6. ''')
7. input("wait")
```

Debugging Challenges with GDB the pwntools way

```
1. context.terminal = ['tmux', 'splitw', '-h']
2. ssh = ssh("jinblack", "192.168.56.102")
3. r = ssh.process("./multistage")
4. gdb.attach(r,'''
5. # b * 0x004000b0
6. # b *0x4000DD
7. ''')
8. input("wait")
```

Load another Library (libc-2.xx.so)

- **env LD_PRELOAD**
  - `LD_PRELOAD=./libc-2.23.so ./binary`
- **ld.so**
  - `./ld-2.23.so --library-path ./lib ./binary`
  - `lib contains libc.so.6`
- **patchelf** (https://github.com/NixOS/patchelf)
  - `patchelf --set-interpreter ./ld-2.23.so --replace-needed libc.so.6 ./libc-2.23.so ./binary`
- **YOLO** (Do not use this!)
  - Replace system library
- **Docker/Virtual Machine**

If DSA Fails (You do NOT know LibC)

- **Standard** LibC
  - Two Symbols
  - LibC DB: https://libc.blukat.me/
- **Custom** LibC (needs a leak)
  - Read Out Libc
  - pwntools dynelf

pwndbg Heap Inspection

```
pwndbg> heap
Top Chunk: 0x6020a0
Last Remainder: 0

0x602000 PREV_INUSE
{
  prev_size = 0x0,
  size = 0x31,
  fd = 0xaaaaaaaaaaaaaaaa,
  bk = 0xaaaaaaaaaaaaaaaa,
  fd_nextsize = 0xaaaaaaaaaaaaaaaa,
  bk_nextsize = 0xaaaaaaaaaaaaaaaa
}
0x602030 PREV_INUSE
{
  prev_size = 0x0,
  size = 0x21,
  fd = 0x0,
  bk = 0x5555555555555555,
  fd_nextsize = 0x0,
  bk_nextsize = 0x51
}
```

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x606850 ← 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x602070 ← 0x7ffff7dd37b8
smallbins
0x190: 0x602150 ← 0x7ffff7dd3938
0x210: 0x602320 ← 0x7ffff7dd39b8
0x290: 0x602570 ← 0x7ffff7dd3a38
largebins
0x3000: 0x602d50 ← 0x7ffff7dd3eb8
```

libc Debugging Symbols

- `sudo apt install` **`libc6-dbg`**
- **`pwninit/spwn`** (https://github.com/io12/pwninit https://github.com/MarcoMeinardi/spwn)
- **`eu-unstrip`** `libc-2.23.so libc-2.23.so.dbg` from the `elfutils` package
- Load it in GDB:

```
(gdb) add-symbol-file ./libc-2.23.so.debug -o 0x7ffff7a0d000 0x7ffff7a0d000
```

- GDB Auto Load:

https://sourceware.org/gdb/onlinedocs/gdb/Separate-Debug-Files.html

Get libc Debugging Symbols

```
file libc-2.23.so -> BuildID

curl -X POST -H 'Content-Type: application/json' --data \
    '{"buildid": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"}' \
    'https://libc.rip/api/find'
```

http://archive.ubuntu.com/ubuntu/pool/main/g/glibc/libc6-dbg_2.23-0ubuntu11.3_amd64.deb

```
data.tar.xz
usr/lib/debug/.build-id/ or usr/lib/debug/lib/
```

Get libc Debugging Symbols - DEBUGINFOD

A online DB of debugging symbol from linux distros.
https://sourceware.org/elfutils/Debuginfod.html

Automatically (or manually) download debugging symbols.

Get libc Debugging Symbols - DEBUGINFOD

A online DB of debugging symbol from linux distros.
https://sourceware.org/elfutils/Debuginfod.html

```
export DEBUGINFOD_URLS="https://debuginfod.elfutils.org/"
(or setup /etc/debuginfod/)

file libc-2.23.so -> BuildID

debuginfod-find -v debuginfo BuildID
```

from **gdb** >10.1 autoload debuginfo:

```
set debuginfod enabled on
```