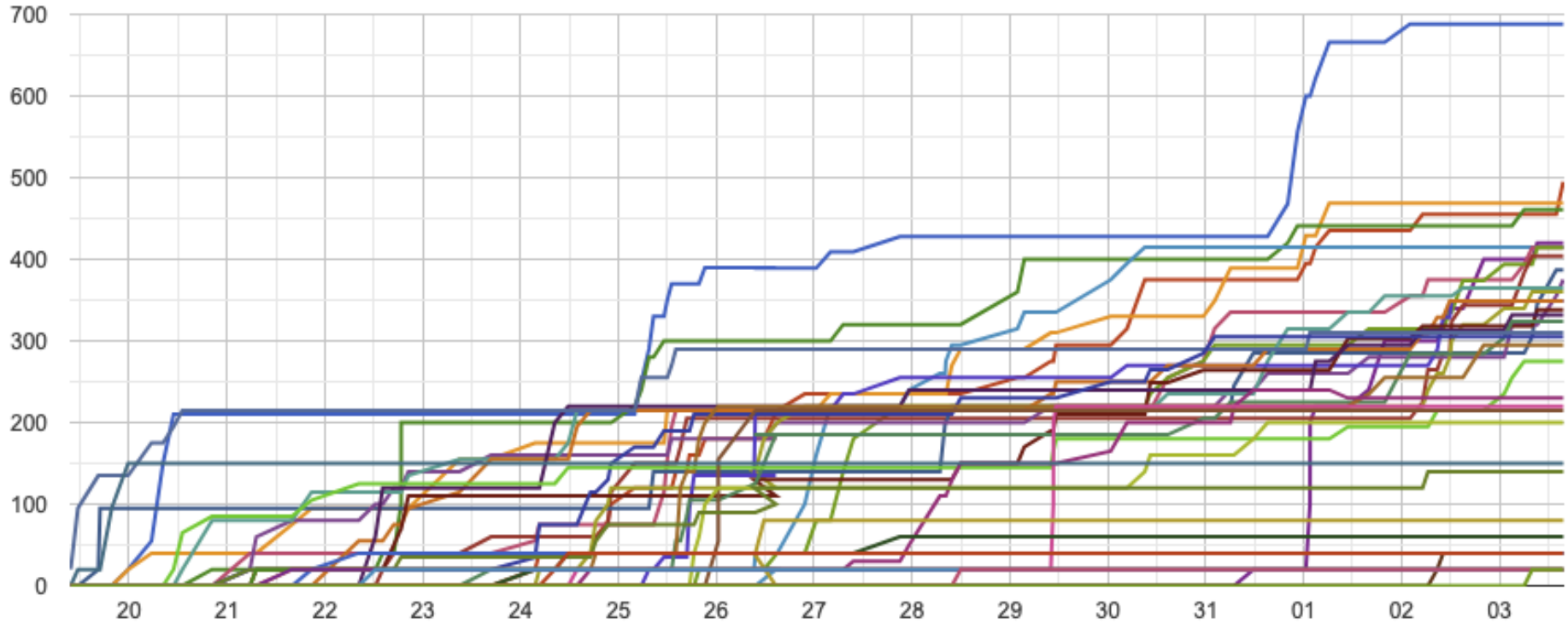
Lec03: Writing Exploits

Taesoo Kim

Scoreboard


700
600
500
400
300
200
100
0
20
21
22
23
24
25
26
27
28
29
30
31
01
02
03

Administrivia

- Survey: how many hours did you spend? (<3h, 6h, 10h, 15h, >20h)
- Please join Mattermost and Piazza!
- Lab03: stack overflow challenges are out!
- Due: Jan 11!

Survival Guide for CS6265

1. Work as a group/team (find the best ones around you!)
    - NOT each member tackles different problems
    - All members tackle the same problem (and discuss/help)
2. Ask questions wisely, concretely
    - Explain your assumption first (e.g., I expect A because …)
    - Explain your problem second (e.g., A is expected but B appears)

Thinking of Threat Model

- Story: A group of students modified “bomb” and got “flags” ..
- Why TAs think they are not correct flags?
- How does our system validate flags?
- How does a setuid binary work?

Thinking of Threat Model


```
# Q0. can we get a flag like this?
$ cat /proc/flag
# Q1. how is this flag different from what bomb prints out?
$ echo "phase2" > /proc/flag
$ cat /proc/flag
# Q2. what about under a tracer?
$ strace -- cat /proc/flag
# Q3. what about this and print flag?
$ gdb ./bomb
# Q4. are they different? why?
$ diff <(cat /proc/flag) <(cat /proc/flag)
# Q5. what about this?
$ diff <(cat /proc/flag) <(sleep 1; cat /proc/flag)
```

Discussion 0

1. How different is the bomb binary this time?

Discussion 1

1. How did you start exploring the “bomb” (no symbol)?

Discussion 2 (bomb201-readfirst)

1. What's going on the first phase?

Discussion 3 (bomb202-objdump)

1. What's going on the second phase?
    - Did you find the main() function (i.e., dispatcher?)

Discussion 3 (obfuscation)


```
4017c0:       eb 01                   jmp    4017c3 <usleep@plt+0xb23>
4017c2:       e9 e9 58 ff ff          jmp    3f70b0 <getenv@plt-0x9a50>
4017c7:       ff 0f                   dec    DWORD PTR [rdi]
4017c9:       1f                      (bad)
4017ca:       84 00                   test   BYTE PTR [rax],al
```

Discussion 3 (when tracing)


```
> x/10i 0x4017c3
   0x4017c3:    jmp    0x401720
   ..
```

Discussion 4 (bomb203-signal)

1. What's going on the third phase?

Discussion 5 (bomb204-minfuck)

1. What's going on the last phase? (nothing special!)

32/64 Shellcode

1. int $80 vs. syscall

```
$ man syscall
```

What's about poly shellcode?

1. What's your general idea?

Discrepancy b/w 32 vs 64

2.2.1.2 More on REX Prefix Fields

REX prefixes are a set of 16 opcodes that span one row of the opcode map and occupy entries 40H to 4FH. These opcodes represent valid instructions (INC or DEC) in IA-32 operating modes and in compatibility mode. In 64-bit mode, the same opcodes represent the instruction prefix REX and are not treated as individual instructions.

The single-byte-opcode forms of the INC/DEC instructions are not available in 64-bit mode. INC/DEC functionality is still available using ModR/M forms of the same instructions (opcodes FF/0 and FF/1).

See Table 2-4 for a summary of the REX prefix format. Figure 2-4 though Figure 2-7 show examples of REX prefix fields in use. Some combinations of REX prefix fields are invalid. In such cases, the prefix is ignored. Some additional information follows:

Dispatching routine

```
          +-------------+
          |             v
[dispatcher][x86      ][x86_64   ]

e.g., 0x40 0x90
 - x86       inc eax
 - x86_64    REX + nop

x86   : [ * ][goto x86 shellcode]
x86-64: [nop][ * ][goto x86_64 shellcode]
arm   : [nop][nop][ * ][goto arm shellcode]
MIPS  : [nop][nop][nop][ * ][goto MIPS shellcode]
```

Dispatching routine

```
// x86        xor eax, eax
// x86_64     xor eax, eax
xorl %eax, %eax

// x86        inc eax
// x86_64     REX + nop
.byte 0x40
nop
jz _x86_64
```

DEFCON18 CTF Doublethink (8 Arch!)

- Ref. https://www.robertxiao.ca/hacking/defcon2018-assembly-polyglot/

Discussion 6 (shellcode ascii/min)

1. Wow, what are your tricks?
2. shellcode-min: 30 bytes? 20 bytes? 10 bytes? 5 bytes?

Lab03: Stack Overflow

- Finally! It's time to write real exploits (i.e., control hijacking)
- TONS of interesting challenges!
  - e.g., lack-of-four, frobnicated, upside-down ..

Lab03: Stack Overflow!

Today's Tutorial

- Example: hijacking crackme0x00!
- A template exploit code
- In-class tutorial
  - Your first stack overflow!
  - Extending the exploit template (python)

DEMO: Ghidra/crackme0x00

- Ghidra w/ crackme0x00
- Exploit writing

crackme0x00


```
$ objdump -M intel-mnemonic -d crackme0x00
...
0804869d <start>:
804869d: 55                             push   ebp
804869e: 89 e5                          mov    ebp,esp
80486a0: 83 ec 18                       sub    esp,0x18
80486a3: 83 ec 0c                       sub    esp,0xc
...
```

```
           |<=- -0x18-=>|+--- ebp
top                     v
[          [buf ..  ]   ][fp][ra]
|<=---  0x18+0xc -----=>|
```

crackme0x00


```
$ objdump -M intel-mnemonic -d crackme0x00
...
80486c6: 8d 45 e8                lea    eax,[ebp-0x18]
80486c9: 50                       push   eax
80486ca: 68 31 88 04 08           push   0x8048831
80486cf: e8 ac fd ff ff           call   8048480 <scanf@plt>

           |<=- -0x18-=>|+--- ebp
top                     v
[          [~~~~>   ]   ][fp][ra]
|<=---  0x18+0xc -----=>|
           [*****************XXXX]
```

crackme0x00

- How can we bypass the password check w/o putting the correct password?

In-class Tutorial

- Step 1: Navigate the binary with your Ghidra!
- Step 2: Play with your first exploit!
- Step 3: Using an exploit template!

```
$ ssh lab03@ss.snucse.org
Password: <password>

$ cd tut03-stackovfl
$ cat README
```

References

- Phrack #49-14