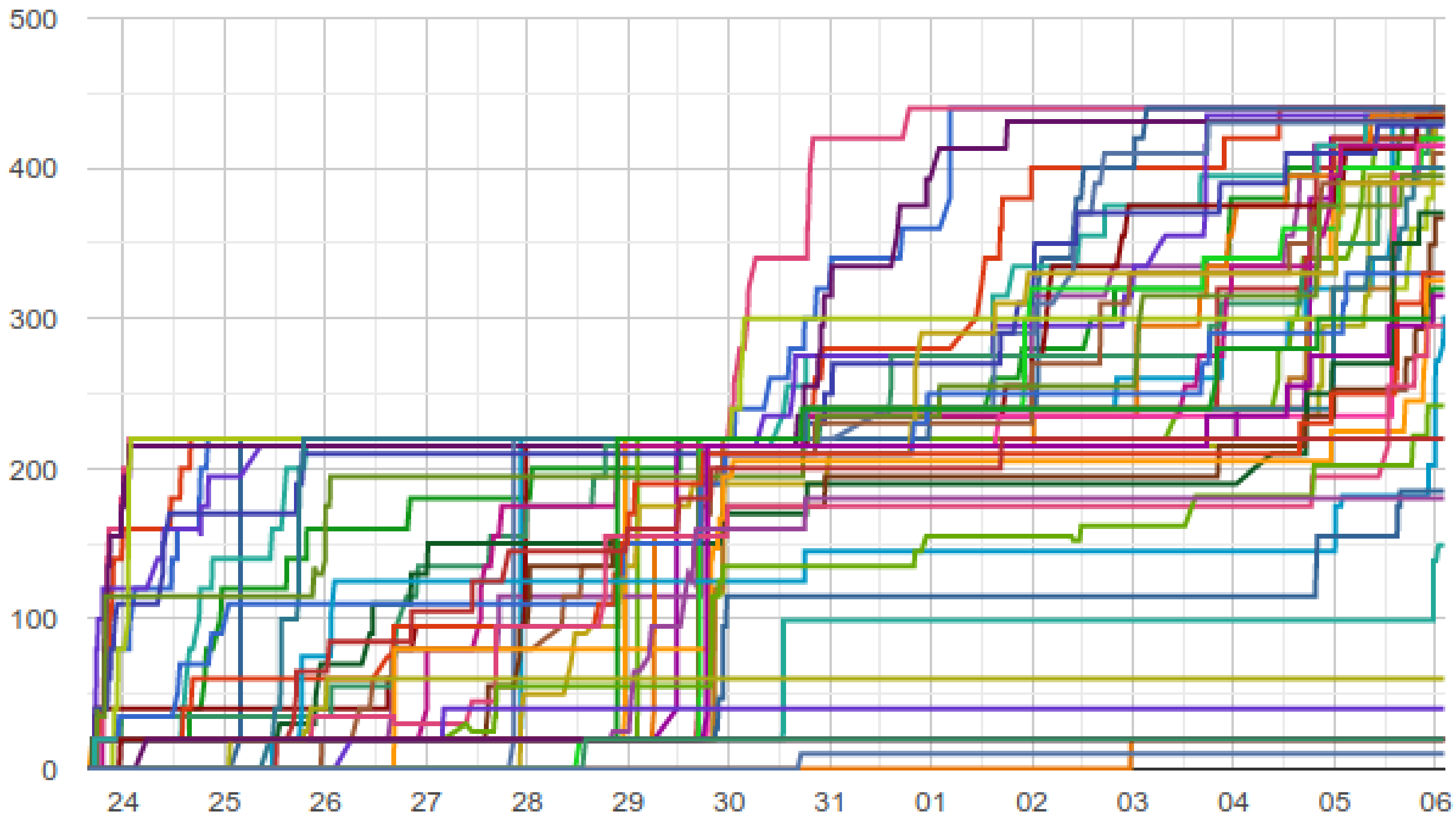
Lec03: Writing Exploits

Taesoo Kim

Scoreboard



Administrivia

- Survey: how many hours did you spend? (<3h, 6h, 10h, 15h, >20h)
- Please join Piazza
- Two optional recitations on every Mon/Wed (check piazza)!
- Lab03: stack overflow challenges are out!
- Due: Sept 19th at midnight (2 weeks)
- In-class CTF: Nov 22–Nov 23!

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)
3. Take advantage of four TAs standing next you to help!
   - World-class hackers give a private tutoring for you!
   - But, remember! only when you ask ..

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)
```

Best Write-ups for Lab02

bomb201-readfirst	viyer43, achang66
bomb202-objdump	abhineet, cfeng66
bomb203-signal	viyer43, mdaniel40
bomb204-minfuck	abhineet, yonghae
env	viyer43, Aditi
shellcode32	0xcoffeeda, Aditi
shellcode64	ochbaklo, Aditi
shellcode-min	viyer43, vishiswoz
shellcode-poly	vishiswoz, ochbaklo
shellcode-ascii	vishiswoz, meduka

Bomb Stats

- Bombs exploded ?? times in total?
- In ?? phases?

Bomb Stats

- Bombs exploded 6 times in total (6 x -5 = -30 pts)
- In 2/3/4 phases

```
- Each phase is solved by  : 40/37/33/33 people
- Each phase is exploded by: 00/01/01/01 people
- Each phase is exploded   : 00/03/02/01 times
```

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)

Discussion 3 (when tracing)

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

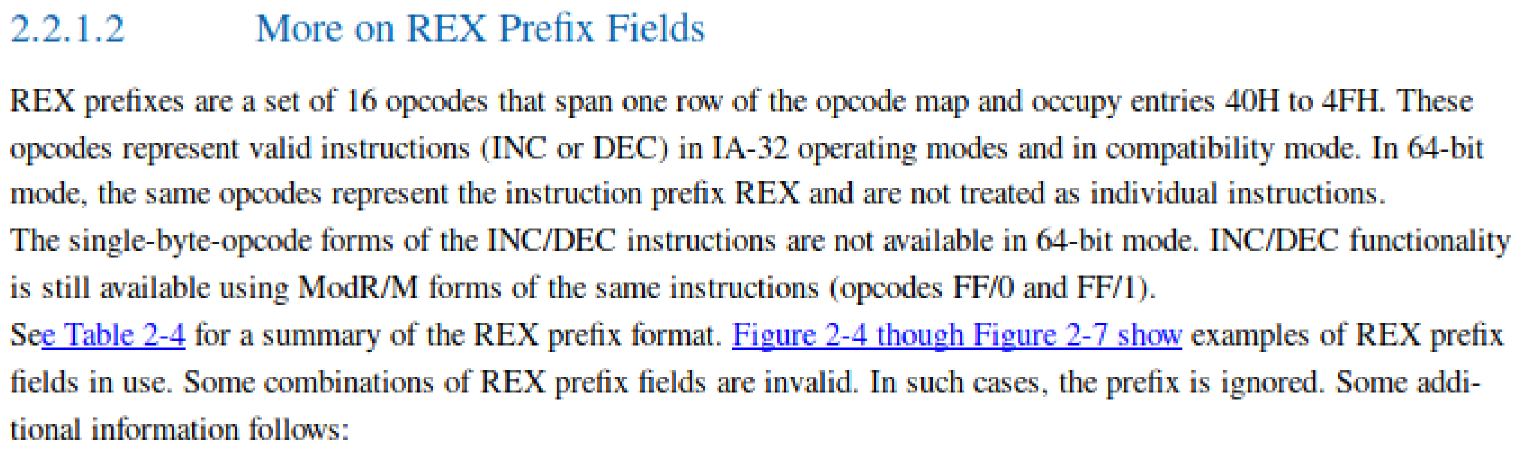
1. int $80 vs. syscall

```
$ 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).

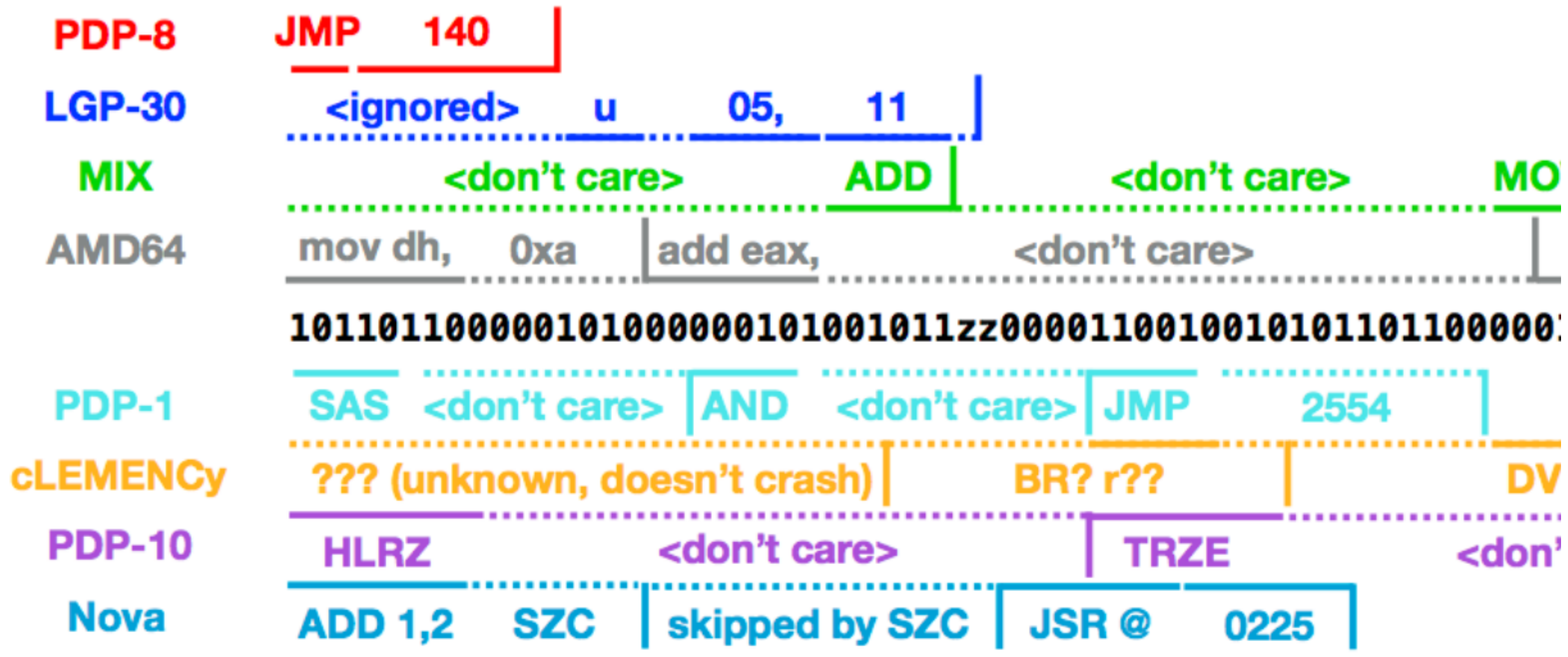
See Table 2-4 for a summary of the REX prefix format. Figure 2-4 though Figure 2-7 show examples of REX prefix fields in use. Some combinations of REX prefix fields are invalid. In such cases, the prefix is ignored. Some additional information follows:

Dispatching routine

Dispatching routine

DEFCON18 CTF Doublethink (8 Arch!)

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


PDP-8
JMP 140
LGP-30
<ignored> u 05, 11
MIX
<don't care> ADD <don't care> MO
AMD64
mov dh, 0xa add eax, <don't care>
1011011000000101000000101001011zz00001100100101011011000000
PDP-1
SAS <don't care> AND <don't care> JMP 2554
cLEMENCy
??? (unknown, doesn't crash) BR? r?? DV
PDP-10
HLRZ <don't care> TRZE <don'
Nova
ADD 1,2 SZC skipped by SZC JSR @ 0225

Discussion 6 (shellcode ascii/min)

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

Discussion 6 (shellcode ascii/min)

Lab03: Stack Overflow (Two Weeks)

- 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!

```
                     .oO Phrack 49 Oo.

                 Volume Seven, Issue Forty-Nine

                         File 14 of 16

              BugTraq, r00t, and Underground.Org
                          bring you

            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            Smashing The Stack For Fun And Profit
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

                      by Aleph One
                  aleph1@underground.org

`smash the stack` [C programming] n. On many C implementations
it is possible to corrupt the execution stack by writing past
the end of an array declared auto in a routine.  Code that does
this is said to smash the stack, and can cause return from the
routine to jump to a random address.  This can produce some of
the most insidious data-dependent bugs known to mankind.
Variants include trash the stack, scribble the stack, mangle
the stack; the term mung the stack is not used, as this is
never done intentionally. See spam; see also alias bug,
fandango on core, memory leak, precedence lossage, overrun screw.
```

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@3.223.237.92
Password:

$ cd tut03-stackovfl
$ cat README
```

References

- Phrack #49-14