Lec04: Writing Exploits

Taesoo Kim

Scoreboard

Administrivia

- Please join Piazza
- An optional recitation at 5-7pm on every Wed (in **CoC 052**)
- Due : Lab03's deadline is on Sept 20th at midnight

Best Write-ups for Lab02

bomb201-readfirst	**nhicks6, gojha**
bomb202-objdump	**riya, seulbae**
bomb203-signal	**seulbae, nhicks6**
bomb204-minfuck	**riya, nhicks6**
env	**nhicks6, seulbae**
shellcode32	**riya, seulbae**
shellcode64	**riya, ibtehaj**
shellcode-min	**nhicks6, easdar**
shellcode-poly	**nhicks6, bwickman3**
shellcode-ascii	**stong, bwickman3**

Bomb Stats

- Bombs exploded ?? times in total?
- in ?? phases?
- ?? people exploded at least once?

Bomb Stats

- Bombs exploded 51 times in total (51 x -5 = -255 pts)
- in ALL phases!
- ~20 people exploded at least once!
    - Each lab: 20/17/07/07 times

shellcode-min

- 30 bytes? 20 bytes? 10 bytes? 5 bytes?

shellcode-min

- 1-byte: jwalsh45 (how?)

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 (phase 1)

1. What's going on the first phase?

Discussion 3 (phase 2)

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

Discussion 3 (obfuscation)

```
$ x/10i 0x555555555952
> x/10i 0x4018d0
   0x4018d0:    jmp    0x4018d3
   0x4018d2:    jmp    0x2484a41f
   0x4018d7:    or     BYTE PTR [rax-0x77],cl
   0x4018da:    (bad)
   0x4018db:    call   0x40180b
```

Discussion 3 (when tracing)

```
> x/10i 0x4018d3
!   0x4018d3:     mov     rax,QWORD PTR [rsp+0x8]
!   0x4018d8:     mov     rdi,rax
    0x4018db:     call    0x40180b
```

Discussion 3 (intention)

```
> x/10i 0x4018d0
   401c10:    jmp    401c13
   401c12:    jmp    3f7500
   401c17:    dec    DWORD PTR [rdi]
   401c19:    (bad)
   401c1a:    test   BYTE PTR [rax],al
```

Discussion 4 (phase 3)

1. What's going on the third phase?

Discussion 4 (phase 3)

```
int count = 0;
void progress_bar(int signo) {
  if (count != 0)
    printf("\b\b\b\b");
  printf("| %02d%%", count);
  count += 2;
  ...
}

phase() {
  signal(SIGTRAP, progress_bar);
  for (int i = 0; i < 50; i ++) {
    ...
    __asm__ volatile("int3");
  }
}
```

Discussion 5 (phase 4)

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]
```

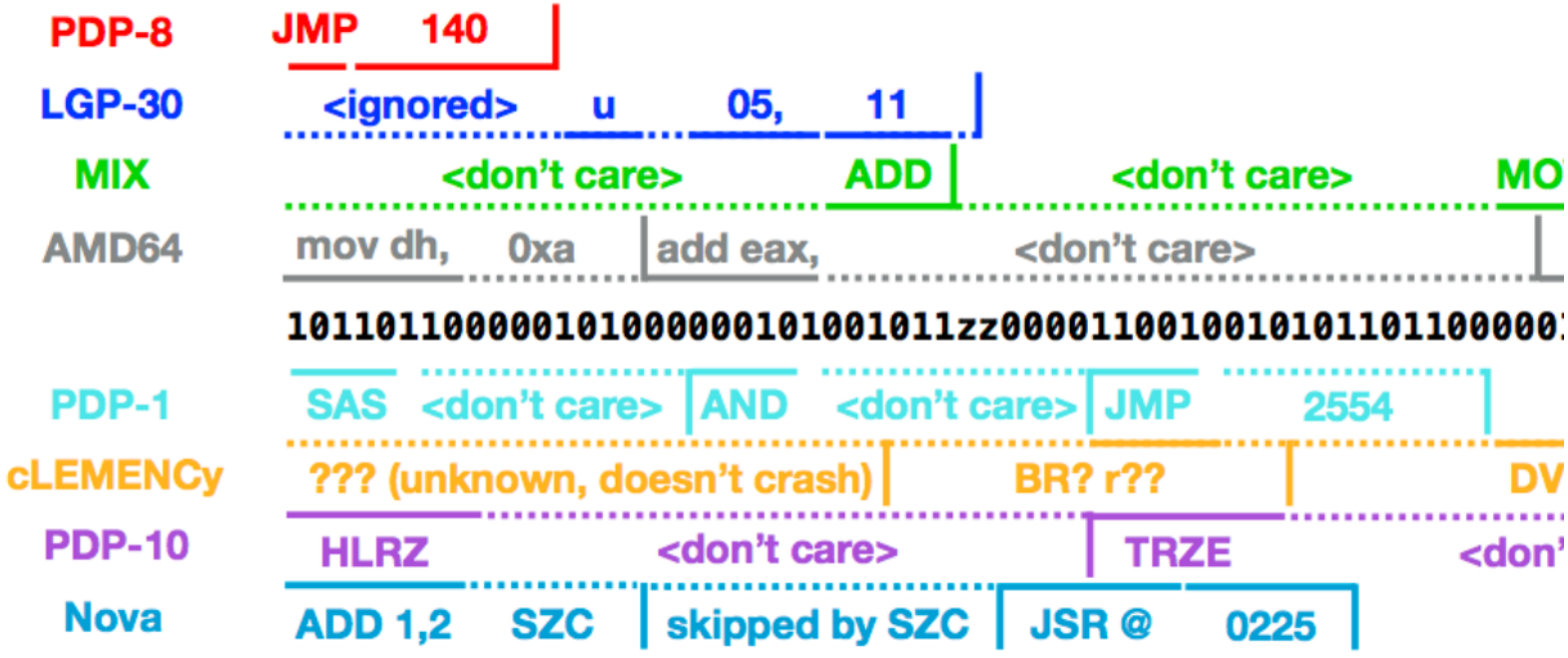
Dispatching routine

```
// 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. NOTE. can be as small as zero byte..

Lab03: Stack overflow!

.oO Phrack 49 Oo.

Volume Seven, Issue Forty-Nine

File 14 of 16

BugTraq, r00t, and Underground.Org
bring you

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

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.

Lab03: Stack overflow!

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

Today's Tutorial

- Example: exploit crackme0x00 to get a shell/flag!
- Explore a template exploit code (PwnTool)
- In-class tutorial
  - Learning PwnTool
  - Writing your first stack overflow exploit!

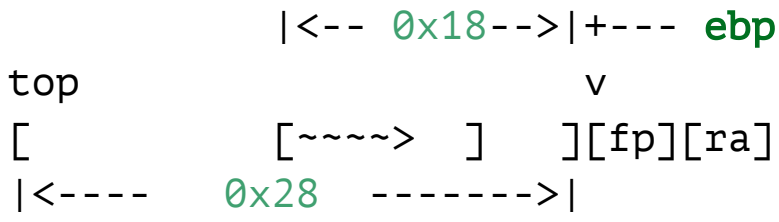
Reminder: crackme0x00

```
$ objdump-intel -d crackme0x00
...
8048414:       push   ebp
8048415:       mov    ebp,esp
8048417:       sub    esp,0x28

                          +--- ebp
top                       v
[                       ][fp][ra]
|<---    0x28   ------->|
```

Reminder: crackme0x00

```
$ objdump-intel -d crackme0x00
...
8048448:       lea    eax,[ebp-0x18]
804844b:       mov    DWORD PTR [esp+0x4],eax
804844f:       mov    DWORD PTR [esp],0x804858c
8048456:       call   8048330 <scanf@plt>
```



Reminder: crackme0x00

```
main() {
  char s1[16];
  ...
  scanf("%s", &s1);
  ...
}
```

Reminder: crackme0x00

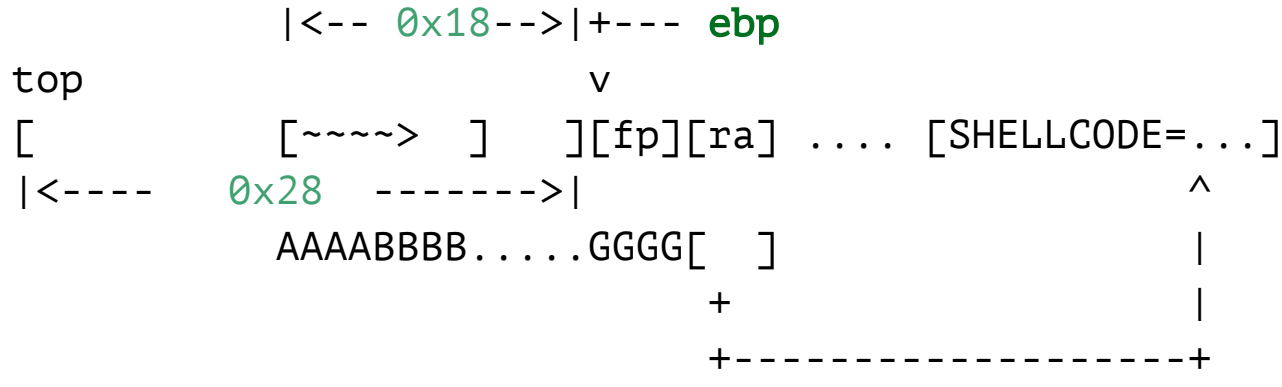
```
            |<-- 0x18-->|+--- ebp
top                     v
[           [~~~~>  ]   ][fp][ra]
|<----   0x28  ------->|
            AAAABBBB.....GGGGHHHH
```

Where to put Shellcode?

- stack (today's tutorial)
- commandline argument
- environment vars

Example: Injecting Shellcode (e.g., env)

1. How to decide the address of an environment variable? (changing?)
2. How to inject (or manipulate) environment variables?



In-class Tutorial

- Step 1: Learn PwnTool
- Step 2: Play with your first exploit!

```
$ ssh lab03@computron.gtisc.gatech.edu -p 9003
$ ssh lab03@cyclonus.gtisc.gatech.edu -p 9003
Password: lab03

$ cd tut03-stackovfl
$ cat README
```

References

- IDA Demo
- Phrack #49-14