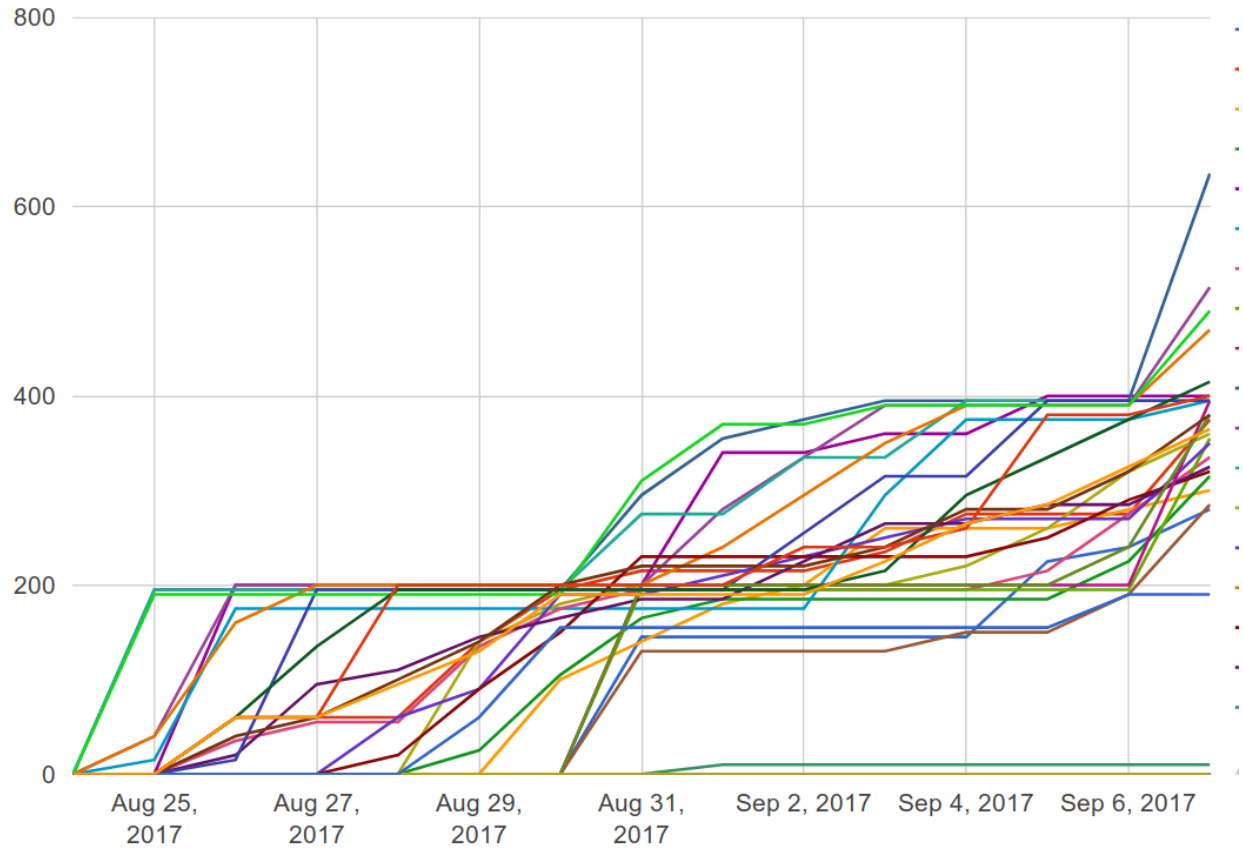
Lec03: Writing Exploits

Taesoo Kim

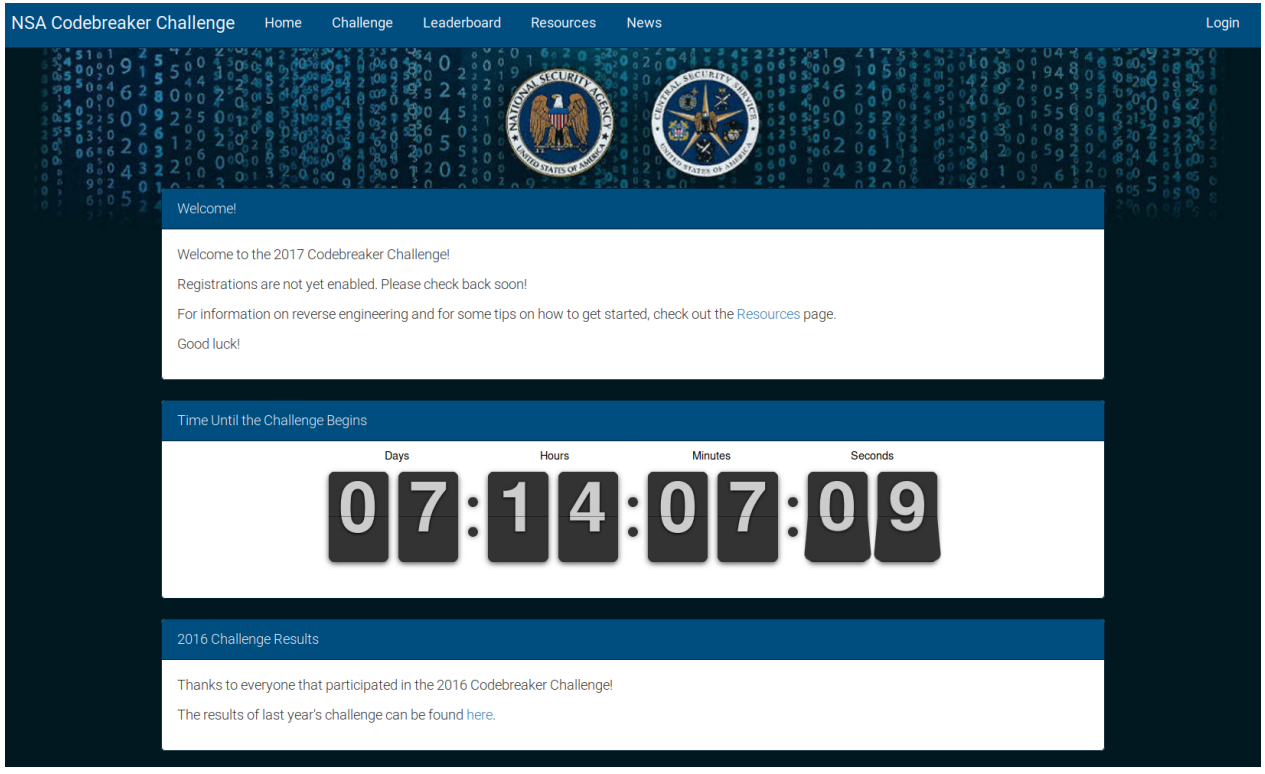
Scoreboard


800
600
400
200
0
Aug 25, 2017
Aug 27, 2017
Aug 29, 2017
Aug 31, 2017
Sep 2, 2017
Sep 4, 2017
Sep 6, 2017

Administrivia

- Survey: how many hours did you spend? (<3h, 6h, 10h, >20h)
- Join Piazza
- An optional recitation on every Wed at 4:30-5:30pm (in CBC 104A)
- Lab03: stack overflow is released!
- Lab04: NSA Codebreaker Challenge
- Due: Lab03: Sept 21 at midnight
- Due: Lab04 (until the end of this semester)

NSA Codebreaker Challenges



NSA Codebreaker Challenges

> *The NSA Codebreaker Challenge provides students with a hands-on opportunity to develop their reverse-engineering / low-level code analysis skills while working on a realistic problem set centered around the NSA's mission .*

Best Write-ups for Lab02

- shellcode32: jallen309, carterchen
- shellcode64: jallen309, rohandvora
- shellcode-ascii: dhaval, carterchen
- shellcode-min: carterchen, markwis
- shellcode-poly: carterchen, shudak3
- shortest shellcode-min: poning, spark720 (6 bytes)

Bomb Stats

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

Bomb Stats

- Bombs exploded 86 times in total (86 x -5 = -430 pts)
- in ALL phases!
- 16 people exploded at least once!
  - Each lab: 28/7/5/4 people
  - Each lab: 63/9/9/5 times

Min shellcode

- 1000 bytes? 500 bytes? 100 bytes?

Min shellcode

- 6-byte: poning
- 6-byte: spark720

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
   0x555555555952:      lea    rsp,[rsp-0x1028]
   0x55555555595a:      or     QWORD PTR [rsp],0x0
   0x55555555595f:      lea    rsp,[rsp+0x1020]
   0x555555555967:      jmp    0x55555555596a
   0x555555555969:      jmp    0x555555549b56
   0x55555555596e:      dec    DWORD PTR [rax-0x7d]
   0x555555555971:      (bad)
   0x555555555972:      or     bl,al
```

Discussion 3 (when tracing)

```
   0x555555555952:      lea    rsp,[rsp-0x1028]
   0x55555555595a:      or     QWORD PTR [rsp],0x0
   0x55555555595f:      lea    rsp,[rsp+0x1020]
-> 0x555555555967:      jmp    0x55555555596a
 |  0x555555555969:      jmp    0x555555549b56
 |  0x55555555596e:      dec    DWORD PTR [rax-0x7d]
 |  0x555555555971:      (bad)
 |  0x555555555972:      or     bl,al
 +->   0x55555555596a:  call   0x5555555558b0
       0x55555555596f:  add    rsp,0x8
       0x555555555973:  ret
       0x555555555974:  push   rbp
```

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]
```

Discussion 6 (shellcode ascii/min)

1. Wow, what are your tricks?
2. NOTE. can be as small as zero byte..

Lab04: Stack overflow (due in 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 ..

Today's Tutorial

- Example: hijacking crackme0x00!
- A template exploit code
- In-class tutorial
  - IDA (yeah!)
  - Extending the exploit template (python)

DEMO: IDA/crackme0x00

- IDA w/ crackme0x00
- exploit writing

crackme0x00

```
$ objdump -d crackme0x00
...
8048414:       55                          push   %ebp
8048415:       89 e5                       mov    %esp,%ebp
8048417:       83 ec 28                    sub    $0x28,%esp

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

crackme0x00

```
$ objdump -d crackme0x00
...
8048448:       8d 45 e8                lea    -0x18(%ebp),%eax
804844b:       89 44 24 04             mov    %eax,0x4(%esp)
804844f:       c7 04 24 8c 85 04 08    movl   $0x804858c,(%esp)
8048456:       e8 d5 fe ff ff          call   8048330 <scanf@plt>

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

In-class Tutorial

- Step 1: Install IDA (feel free to use from now)
- Step 2: Play with your first exploit!

```
$ ssh YOURID@cyclonus.gtisc.gatech.edu -p 2023
$ ssh YOURID@cyclonus.gtisc.gatech.edu -p 2022
$ ssh YOURID@computron.gtisc.gatech.edu -p 2023
$ ssh YOURID@computron.gtisc.gatech.edu -p 2022

$ cd tut/lab03
$ cat README
```

References

- IDA Demo
- Phrack #49-14