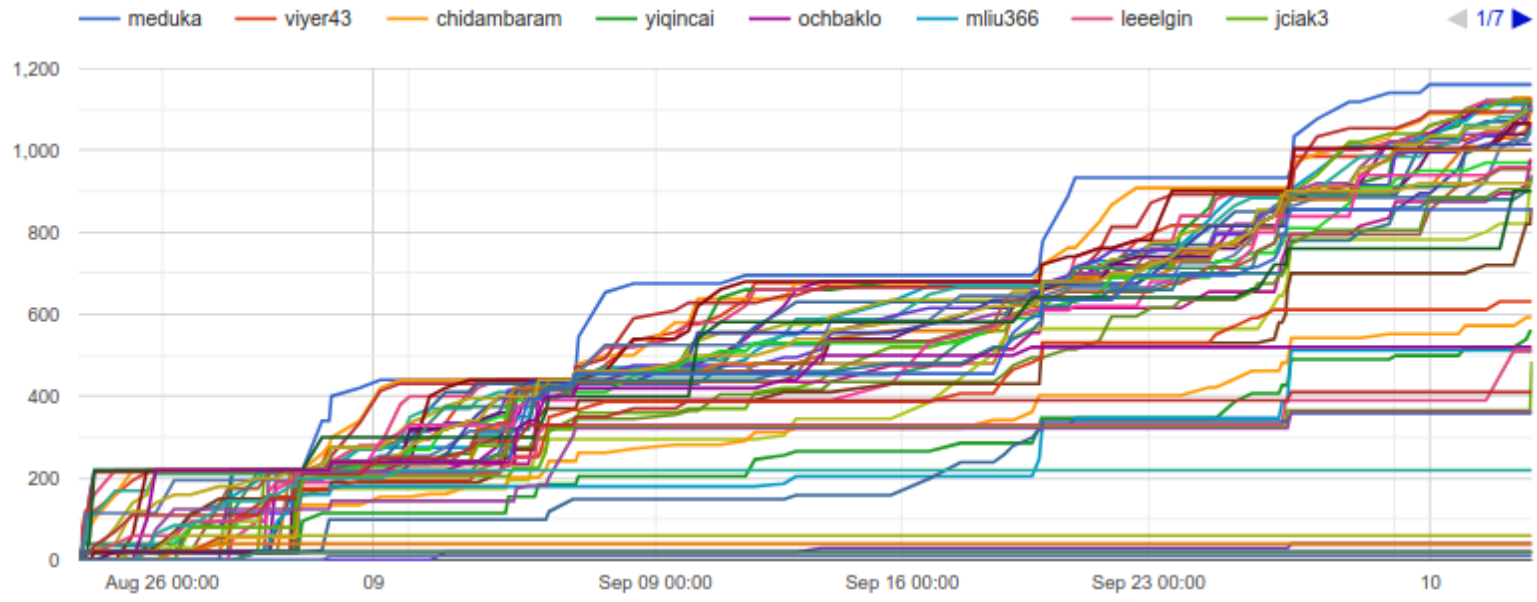
Lec07: Return-oriented Programming

Taesoo Kim

Scoreboard


meduka
viyer43
chidambaram
yiqincai
ochbaklo
mliu366
leeelgin
jciak3
1/7
1,200
1,000
800
600
400
200
0
Aug 26 00:00
09
Sep 09 00:00
Sep 16 00:00
Sep 23 00:00
10

Administrivia

- Congrats! Just completed 1/2 labs!
- Please submit both 'working exploit' and write-up on time!
- Due: Lab06 is out and its due on Oct 17 (two weeks)!
- NSA Codebreaker Challenge → Due: Dec 06

Best Write-ups for Lab05

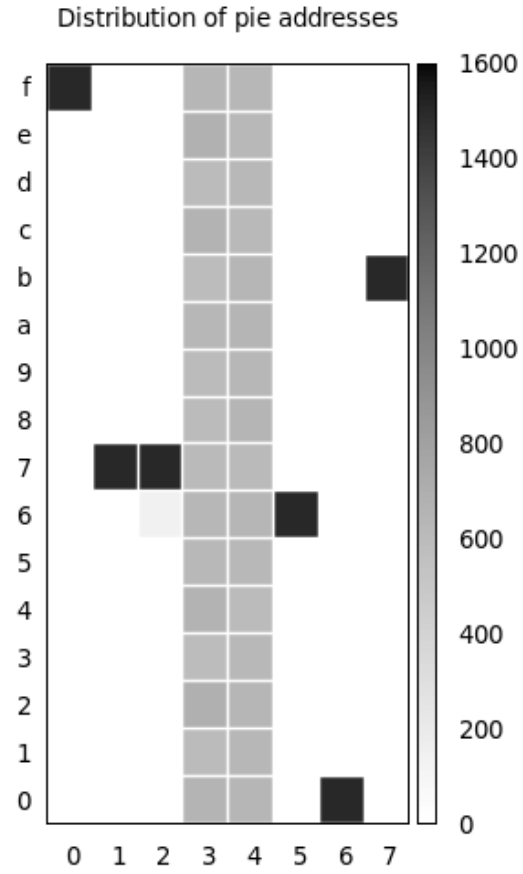
libbase	**achang66, mliu366**
moving-target	**yonghae, achang66**
fmtstr-digging	**viyer43, mliu366**
fmtstr-read	**mliu366, viyer43**
fmtstr-write	**viyer43, mliu366**
brainfxxk	**chidambaram, yiqincai**
fd-const	**abhineet, mliu366**
fmtstr-heap	**vishiswoz, chidambaram**
profile	**yiqincai, viyer43**
mini-sudo	**chidambaram, mliu366**

Discussion: moving-target

- What's `check-aslr.sh` and pie.c?
- How many times should we try to exploit?

Lack of entropy is a fundamental limitation (e.g., 32-bit x86).

Discussion: moving-target



Discussion: fmtstr-*?

- fmtstr-read/write/digging are relatviely easy

But, How to Prevent fmtstr-*?

But, How to Prevent fmtstr-*?

1. Non-POSIX compliant (e.g., Windows)
    - Discarding %n
    - Limiting width (e.g., "%.512x" in XP, "%.622496x" in 2000)
2. Dynamic: enabling FORTIFY in gcc (e.g., Ubuntu)
3. Static: code annotation (e.g., Linux)

FORTIFY (-D_FORTIFY_SOURCE=2)

- Ensuring that all positional arguments are used
  - e.g., %2$d is not ok without %1$d
- Ensuring that fmtstr is in the read-only region (when %n)
  - e.g., "%n" should not be in a writable region

```
$ ./fortify-yes %2$d
*** invalid %N$ use detected ***

$ ./fortify-yes %n
*** %n in writable segment detected ***
```

Discussion: mini-sudo (CVE-2012-0809)

- What is '-D9' for?

Discussion: mini-sudo (CVE-2012-0809)

```
void sudo_debug(int level, const char *fmt, ...) {
  va_list ap;
  char *fmt2;

  if (level > debug_level) return;

  /* Backet fmt with program name and a newline to make it
     a single write */
  easprintf(&fmt2, "%s: %s\n", getprogname(), fmt);
  va_start(ap, fmt);
  vfprintf(stderr, fmt2, ap);
  va_end(ap);
  efree(fmt2);
}
```

CVE-2013-1848: Linux ext3

```
void ext3_msg(struct super_block *sb, const char *prefix,
        const char *fmt, ...)
{
  struct va_format vaf;
  va_list args;

  va_start(args, fmt);

  vaf.fmt = fmt;
  vaf.va = &args;

  printk("%sEXT3-fs (%s): %pV\n", prefix, sb->s_id, &vaf);

  va_end(args);
}
```

CVE-2013-1848: Linux ext3

- Intended/correct usages:

```
ext3_msg(sb, KERN_ERR, "Invalid uid value %d", option);
```

CVE-2013-1848: Linux ext3

- What's wrong?

```
// @get_sb_block()
ext3_msg(sb, "error: invalid sb specification: %s", *data);

// @ext3_blkdev_get()
ext3_msg(sb, "error: failed to open journal device %s: %ld",
        __bdevname(dev, b), PTR_ERR(bdev));
```

CVE-2013-1848: Linux ext3

- Annotating fmtstr for automatic checking at compilation time

```
extern __printf(3, 4)
void ext3_msg(struct super_block *sb, const char *prefix,
        const char *fmt, ...);
```

Summary of Lab05

- Fundamental limitations of DEP/ASLR (e.g., lack of entropy, info leaks)
  - brainfxxk: leaking code pointer
  - logic error: incorrectly using fd
  - profile: uninitialized use + leaking code pointers
- Powerful format string vulnerability
  - fmtstr-heap: data pointers not in the input buffer
  - mini-sudo: overwriting local variables

Take-outs from DEP/ASLR?

- Are DEP/ASLR bullet-proof defenses? No!
- Do you think DEP/ASLR make attackers' life more difficult?

Although we can't place shellcode into stack/heap, we can still hijack the control flow of a program in many interesting ways

Modern Exploit on ASLR (w/ PIE)

- Leak (or infer) code pointers (so map into library or code)
- Construct return-oriented programming (ROP!)

Today's Tutorial

- In-class tutorial:
  - Ret-to-libc
  - Code pointer leakage / gadget finding
  - First ROP!

Reminder: crackme0x00

```
void start() {
  printf("IOLI Crackme Level 0x00\n");
  printf("Password:");

  char buf[32];
  memset(buf, 0, sizeof(buf));
  read(0, buf, 256);

  if (!strcmp(buf, "250382"))
    printf("Password OK :)\n");
  else
    printf("Invalid Password!\n");
}
```

Reminder: crackme0x00

```
$ checksec ./target
 [*] '/home/lab/tut-rop/target'
  Arch:     i386-32-little
  RELRO:    Partial RELRO
  Stack:    No canary found
  NX:       NX enabled
  PIE:      No PIE (0x8048000)
```

Reminder: crackme0x00

```
int main(int argc, char *argv[])
{
  setvbuf(stdout, NULL, _IONBF, 0);
  setvbuf(stdin, NULL, _IONBF, 0);

  void *self = dlopen(NULL, RTLD_NOW);
  printf("stack   : %p\n", &argc);
  printf("system(): %p\n", dlsym(self, "system"));
  printf("printf(): %p\n", dlsym(self, "printf"));

  start();

  return 0;
}
```

Ret-to-libc: printf

```
[buf  ]
[.....]
[ra   ] => printf
[dummy]
[arg1 ] => "Password OK :)"
```

Ret-to-libc: system

```
[buf  ]
[.....]
[ra   ] => system
[dummy]
[arg1 ] => "/bin/sh"
```

Chaining Two Function Calls

```
printf("Password OK:)")
system("/bin/sh")
```

Chaining Two Function Calls

```
[buf      ]
[.....    ]
[old-ra   ] => 1) printf
[ra       ] ------------------=> 2) system
[old-arg1 ] => 1) "Password OK :)"
[arg1     ] => "/bin/sh"
```

Chaining N Function Calls

```
[buf      ]
[.....    ]
[old-ra   ] => 1) printf
[ra       ] ------------------=> pop/ret gadget
[old-arg1 ] => 1) "Password OK :)"
[ra       ] => 2) system
[ra       ] ------------------=> pop/ret gadget
[arg1     ] => "/bin/sh"
[ra       ] ...
```

Tutorial Goal: Chaining Three Calls

```
open("/proc/flag", O_RDONLY)
read(3, tmp, 1024)
write(1, tmp, 1024)
```

In-class Tutorial

- Step1: Ret-to-libc
- Step2: Understanding module base
- Step3: First ROP

```
$ ssh lab06@3.95.14.86
Password: <password>

$ cd tut07-rop
$ cat README
```

References

- ROP