Lec07: Exploiting Format String

Chengyu Song

Slides based on CS6265 taught by Prof. Taesoo Kim
Scoreboard
Administrative

- Almost there!!
- Due: Lab07 is out and its due on Mar 9 at midnight
- 6 normal challenges + 4 bonus
- Survey: exam week
Lab06: ASLR

- libbase, moving-target: 9
- uninit-stack, heap-spray, puzzle: 7
- heap-spray2, brainfxxk, upto-retaddr, find-gadget, heap-spray3: 4
Discussion: Lab06

• What's the most "interesting" bug or challenge?
• Has ASLR makes exploit more difficult?
• Any none-randomized part could be a problem.
• Too many ways to leak information, right?
  • We didn't even touch side-channels!
Discussion: uninit-stack

- Where does the leaked information come from?
- Did you always initialize stack variables when you coding?
- Much trickier than you think!
  - Read our paper UniSan section 2.2
Discussion: heap-spray2

- Is it more challenging than heap-spray1?
- What is the major problem?
Discussion: brainfxxk

- Arbitrary read/write
Discussion: puzzle

• Is printing still possible?
Discussion: upto-retaddr/heap-spray3

- Where is your payload?
Discussion: find-gadget

• Which gadgets do you use?
Discussion: Lab04-Lab06

- Three most widely deployed defense mechanisms: stack canary, DEP, ASLR
- Exploits are more difficult but not impossible
- Take-aways
  - Details matter a lot for defense mechanisms!
  - Memory errors are too powerful and flexible
Format string: *printf

```c
int printf(const char *format, ...);

1) printf("hello: %d", 10);
2) printf("hello: %d/%d", 10, 20);
```
Format string: variable length args

```c
int __printf (const char *format, ...) {
    va_list arg;
    int done;

    va_start (arg, format);
    done = vfprintf (stdout, format, arg);
    va_end (arg);

    return done;
}
```
Format string: va_*

typedef void *va_list;
#define va_start(list, param) \n   (list = (((va_list)&param) + sizeof(param)))
#define va_arg(list, type) \n   (*(type *)((list += sizeof(type)) - sizeof(type)))
Format string: *printf

printf("%d/%d/%d", a1, a2 ...)

+----(n)----+
|           v
[ra][fmt][a1][a2][a3][..]
(1) (2) (3) ....
Format string specifiers: common ones

printf(fmt);

%p: pointer
%s: string
%d: int
%x: hex
Format string specifiers: complete definition

%[parameter][flags][width][.precision][length]type

parameter: n$
flags: -,+,space,0,#
width: number,*
length: hh,h,l,ll,L,z,j,t,I,I32,I64,q,etc
type: %, integer, float, string, char, n
Format string vulnerabilities

- What if the attackers can control the `fmt`
- What if the arguments is not enough, or the type is wrong
Arbitrary Read

printf("\xaa\xbb\xcc\xdd%3$s")

+---(3rd)---+
|         v 
[ra][fmt][a1][a2][\xaa\xbb\xcc\xdd%3$s]
  (1) (2) (3) .

-> "\xaa\xbb\xcc\xdd[value]"
Write (sth) to an Arbitrary Location

printf("\xaa\xbb\xcc\xdd%3$n")

+---(3rd)---+
|          v
[ra][fmt][a1][a2][\xaa\xbb\xcc\xdd%3$n]
   (1) (2) (3) ....

-> "\xaa\xbb\xcc\xdd" = 4
Arbitrary Write

printf("\xaa\xbb\xcc\xdd%6c%3$n")

+---(3rd)---+
|     v
[ra][fmt][a1][a2][\xaa\xbb\xcc\xdd%6c%3$n]
(1) (2) (3) ....

-> *(int *)(0xddccbbaa) = strlen("\xaa\xbb\xcc\xdd......") = 10
Lab07
In-class Tutorial

• Step1: Format string to arbitrary read
• Step2: Format string to arbitrary write
• Step3: (optional) Format string to arbitrary execution
References

- Format string vulnerability