Computer Security HW7 Write-Up

Student ID: B07902143
Account: soyccan
Name: 陳正康

casino++

FLAG{Y0u_pwned_me_ag4in!_Pwn1ng_n3v3r_di4_!}

Reconnaissance

```
gdb-peda$ checksec
RELRO:    Partial RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      No PIE
FORTIFY:  Enabled
```

- no PIE

Vulnerability: out-of-bound access of array

```
printf( "Change the number? [1:yes 0:no]: " );
if( read_int() == 1 ){
    printf( "Which number [1 ~ 6]: " );
    idx = read_int() - 1; // NOTE: no check idx range
    printf( "Chose the number %d: " , idx );
    guess[idx] = read_int(); // NOTE: out-of-bound read
}
```

1. `guess` is a global variable, lies in .bss section
2. specifying (possibly negative) index will lead to writing everywhere in .bss section
3. no PIE protection, .bss section address is fixed

=> GOT hijacking is possible

Vulnerability: buffer overflow

```
    puts( "Show me your passport." );
    printf( "Your name: " );
    read( 0 , name , 0x100 );
```

this allows us to control `seed`
memory allocation in .bss:

```
$ objdump -j .bss -t casino++
0000000000602078 g       .bss   0000000000000000 __bss_start
0000000000602080 l    d  .bss   0000000000000000 .bss
0000000000602080 g     O .bss   0000000000000008 stdout@@GLIBC_2.2.5
0000000000602090 g     O .bss   0000000000000008 stdin@@GLIBC_2.2.5
00000000006020a0 g     O .bss   0000000000000008 stderr@@GLIBC_2.2.5
00000000006020a8 l     O .bss   0000000000000001 completed.7697
00000000006020b0 g     O .bss   0000000000000018 lottery
00000000006020d0 g     O .bss   0000000000000018 guess
00000000006020f0 g     O .bss   0000000000000010 name <- from here
0000000000602100 g     O .bss   0000000000000004 seed <- overflows to here
0000000000602104 g     O .bss   0000000000000004 age
0000000000602108 g       .bss   0000000000000000 _end
```

GOT Hijacking

```
void casino(){

    srand( seed );
    for( int i = 0 ; i < 6 ; ++i ) lottery[i] = rand() % 100;

    int try = 2, idx;

    while( try-- ){
        printf( "\n$$$$$$$ Lottery $$$$$$$\n " );

        for( int i = 0 ; i < 6 ; ++i ){
            printf( "Chose the number %d: " , i );
            guess[i] = read_int();
        }

        printf( "Change the number? [1:yes 0:no]: " );
        if( read_int() == 1 ){
            printf( "Which number [1 ~ 6]: " );
            idx = read_int() - 1;
            printf( "Chose the number %d: " , idx );
            guess[idx] = read_int();
        }

        for( int i = 0 ; i < 6 ; ++i ){
            if( guess[i] != lottery[i] ) break;
            if( i == 5 ){
                puts( "You win! Hacker don't need luck :P" );
            }
        }
    }

    printf( "You lose.\nBye~\n " );
}
```

- we will hijack `srand()` at the beginning, and `puts()` at the end of `casino()`
- it requires two round to write an address to GOT, each to modify 4 bytes
- at the second round, we need to pass the lottery check, so that `puts()` would be called
- if we points `puts@GOT` to `casino()`, `casino()` will be recursively called

Exploition Flow

1. in main(): seed := setvbuf@got (0x602050)

    - by overflow on `name`
2. first round in casino(): puts@GOT (0x602020) -> casino (0x40095d)
    - "->" mean "points to"
3. second round in casino(): srand@GOT (0x602040) -> printf@plt (0x400706)
    - `srand(seed)` is same as : `printf(setvbuf@GOT)`
    - LIBC address will leak the next round!
4. third round in casino(): srand@GOT (0x602040) -> gets@LIBC
    - since LIBC address is leaked, address of `gets()` can be obtained
    - we'll send "/bin/sh" in the next round
    - Note: setvbuf@got will be overwritten by "/bin/sh"
5. fourth round in casino(): srand@GOT (0x602040) -> system@LIBC
6. fifth round in casino(): get shell!

Passing Lottery Check

```
void init(){
    setvbuf(stdout,0,2,0);
    setvbuf(stdin,0,2,0);
    setvbuf(stderr,0,2,0);
    seed = time(0);
}
```

```
int main(){

    init();
    welcome();

    puts( "Show me your passport." );
    printf( "Your name: " );
    read( 0 , name , 0x100 );
```

- the seed is generated at the beginning of program
    - later overwritten by overflow on `name`
    - lottery is generated by `rand()` , AFTER `seed` is overwritten
    - seed is controllable
- as we know seed, we can re-run the random process to get what lottery is
    - see `rand.c` in my code