Computer Security HW8 Write-Up

Student ID: B07902143
Account: soyccan
Name: 陳正康

election

FLAG{Wh0_h4cked_my_v0t1ng_sys7em_:P}

Reconnaissance

```
$ checksec
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    FORTIFY:  Enabled
```

- full protection

Vulnerability: buffer overflow

```
char msg[0xe0]; // rbp-0xf0
read( 0 , msg , candidates[idx].votes );
// Note: overflow, msg(224)+padding(8), votes <= 255
```

read 幾個 byte 取決於票數 `candidates[idx].votes`，最大可以到 0xff，
每次註冊都有 10 票，只要重複註冊並投票，
就能讓 `msg` overflow 至多到 return address 的倒數第二個 byte，控 rip 和 rbp

Hindrance

- stack canary protection
- unsufficient space for ROP chain
  - => stack pivoting

Vulnerability: boolean-based stack leak

```
int len = read( 0 , buf , sizeof( buf ) );
// Note: len(buf) = 0xc8; len(token) = 0xb8
if( memcmp( buf , token , len ) ){
    puts( "Invalid token." );
    break;
}
```

只要 buf 和 token 的前綴相同就會通過，但 buf 的 size 又大於 token
buf 長度相當於 token 加上 canary和 __lib_csu_init 的 return address
剛好可以暴力 try 出 canary 和 bypass PIE

Achievement

- bypass stack canary
- bypass PIE

Attack: Stack Pivoting + ROP chain

利用前面 bof 漏洞，把 stack 搬到已知位址且可控的全域變數 `buf`
在那裡放 ROP chain (size <= 0xb8)

Failed Workflow

原本嘗試的 workflow 是：
first round:

1. overflow -> stack migration
2. ROP chain:
    1. `puts(puts@got)`
    2. return to main
3. got libc address

second round:

1. overflow -> stack migration
2. ROP chain:
    1. `system("/bin/sh")`
    2. PWNED!

problem

再次回到 main() 之後，rsp 指在 .bss 段 的全域變數 `buf`
main() 有 call 其它 function，讓 stack frame 往前蓋掉 .data 段的東西 (e.g. stdin FILE object pointer) 就會壞掉，或是更往前寫到不能寫的 .rodata 段也會壞掉

Correct Workflow

1. overflow -> stack migration
2. ROP chain:
    1. puts(puts@got)
    2. read(0, &func, 8)
        - func 是 .bss 段一個沒用的地方，要放 one gadget 的指標
        - 用 csu gadget 來控 rdx
    3. (*func)()
        - 用 csu gadget 來 call function pointer: `call QWORD PTR [r12+rbx*8]`
        - PWNED!

Note++

FLAG{Heap_exp1oit4ti0n_15_fun}

Vulnerability: dangling pointer

```
void delete() {
    //...
    free( notes[idx].data ); // Note: dangling pointer
    notes[idx].is_freed = 1;
}
```

Vulnerability: information leak

```
void list(){
    for( int i = 0 ; i < MAX ; ++i ){
        if( notes[i].data && !notes[i].is_freed ){
            printf( "Note %d:\n  Data: %s\n  Desc: %s\n" , i , notes[i].data , notes[
        }
    }
    puts("");
}
```

只要能把 `is_freed` 洗成0，就會印出即使已經free掉的chunk
得到fd, bk之類的資訊

Vulnerability: double free

```
void delete() {
    // ...
    if( notes[idx].is_freed ){
        puts( "Double free! Bad hacker :(" );
        _exit(-1);
    }
    free( notes[idx].data ); // Note: dangling pointer
}
```

只要能把 `is_freed` 洗成0，就能 double free
使得 fastbin attack 成立

Vulnerability: off-by-one null byte overflow

```
struct Note{
    int is_freed;
    char *data;
    char description[48];
};

void add() {
    //...
    // fixed overflow
    // scanf( "%s" , notes[i].description ) // overflow
    scanf( "%48s" , notes[i].description ); // safe; Note: appending null-byte
}
```

`description` 剛好 48 bytes，`scanf` 會讀 48 bytes「再加」一個 null-byte 當字串結尾
因為48剛好是16的倍數
所以 `description` 的下一個 byte 剛好是下一個 note 的 `is_freed`
就會被洗成0
造成 use after free 和 double free

Constraint: malloc size limit

```
void add() {
    // ...
    if( size > 0x78 ){
        puts( "Too big!" );
        return;
    }
    notes[i].data = malloc( size );
}
```

malloc 大小有限制，不能用到 unsorted bin

Exploition

use after free: leak heap address

對一個在 fastbin linked list 中間的 chunk
先 free 掉，再把 is_freed 洗成 0
再 list 就會看到 fd

fastbin attack: leak LIBC address

因為不能 malloc 出 unsorted bin 的大小
所以先 malloc 出一個 fastbin-size 的 chunk
再用 fastbin attack 把它的 size 欄位改大
再 free 掉它，就會被放入 unsorted bin 中，並被填上 main_arena 的 address

fastbin attack: spawn shell and hindrance

- attempt: one gadget
  - 沒有條件滿足的 gadget
- attempt: __malloc_hook
  - 參數只有 4 bytes 可控
- attempt: Google “pwn malloc_hook”
  - good article: https://bbs.pediy.com/thread-246786.htm

- 用 realloc_hook + malloc_hook 微調 stack 位置

realloc_hook + malloc_hook

https://bbs.pediy.com/thread-246786.htm

- 把 malloc_hook 指向 __libc_realloc 開頭的某一個 push 指令
  - push 次數少了，stack 位置會往後微調，有機會讓 one gadget 可以用
- 把 realloc_hook 指向 one gadget
  - __libc_realloc 會 call realloc_hook 指向的地方
  - PWNED !!

heap layout

Chunk 0:

00	**…**	**71**
10	…	…
20	…	…
30	…	…
40	…	…
50	…	…
60	…	(fake_chunk0_size) 71

Chunk 1:

70	**…**	**71 -> (after fastbin attack) 91**
80	…	…
90	…	…
a0	…	…
b0	…	…
c0	…	…
d0	…	…

Chunk 2:

e0	**…**	**71**
f0	…	…
100	…	(fake chunk size to avoid unsorted bin merge) 51
110	…	…
120	…	…
130	…	…
140	…	…

Workflow in Detail

operation	**fastbin**	**assign**
add		note[0] = chunk0
add		note[1] = chunk1
add		note[2] = chunk2
delete 2	-> chunk2	
delete 1	-> chunk1 -> 2	
delete 0	-> chunk0 -> 1 -> 2	
add	-> chunk1 -> 2	note[0] = chunk0 overflow: note[1].is_freed = 0
list	(leak heap address)	(note[1].data is printed)
delete 0	->chunk0 -> 1 -> 2	
delete 1	->chunk1 -> 0 ->1 -> 0 …	
add	->chunk0 -> 1 ----> fake_chunk0 (fastbin attack)	note[0] = chunk1
add	-> chunk1 -> fake_chunk0	note[1] = chunk0
add	-> fake_chunk0	note[2] = chunk1
add		note[3] = fake_chunk0 (chunk1 size becomes 0x90)
delete 2	(unsorted bin) <-> chunk1	
delete 1	-> chunk0	
add		note[1] = chunk0 (overflow: note[2].is_freed=0)
list	(leak libc address)	(note[2].data is printed)
add		note[4] = chunk1
add		note[5] = chunk3
delete 5	-> chunk3	
delete 4	-> chunk1->3	

operation	fastbin	assign
add	-> chunk3	note[4] = chunk1 (overflow:note[5].is_freed=0)
delete 4	-> chunk1->3	
delete 5	-> chunk3 -> 1 -> 3 …	
add	-> chunk1 -> 3 ----> fake_chunk1 (fastbin attack)	note[4] = chunk3
add	-> chunk3 -> fake_chunk1	note[5] = chunk1
add	-> fake_chunk1	note[6] = chunk3
add		note[7] = fake_chunk1 (write to relloc_hook and malloc_hook)

Reference

Google “pwn malloc_hook”

https://bbs.pediy.com/thread-246786.htm
https://medium.com/@ktecv2000/詳談heap-exploit-9ba957e27ee8
https://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/unsorted_bin_attack-zh/

Not yet read:
http://look3little.blogspot.com/2017/01/tstack-based-buffer-overflowleak-address.html