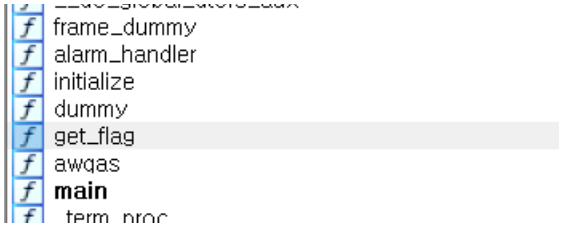
1. welcome

```
v6 = dummy;
v4 = 0LL;
printf("dummy : %p\n", dummy);
printf("dummy key: %llu\n", ((unsigned __int64)dummy ^ 0x65) + 101);
puts("input key: ");
__isoc99_scanf("%llu", &v4);
v5 = (void (*)(void))awqas(v4);
v5();
return 0;
```

```
__int64 __fastcall awqas(__int64 a1)
{
  return (a1 - 101) ^ 0x65;
}
```

Dummy key를 (dummy ^ 0x65) + 101로 암호화 해서 출력하고 있고 awqas를 살펴보니 사용자의 입력을 역으로 (al - 101) ^ 0x65로 연산하고 있다.

실행하고자 하는 함수의 주소를 Dummy key의 연산과 같이 ^0x65 + 101로 하여 입력하면 원하는 함수를 실행할 수 있을 것이다.



함수 목록을 살펴보면 flag를 따주는 get_flag함수가 존재함을 알 수 있다.

$ checksec welcome

[*] '/home/ubuntu/cau/welcome'

Arch: amd64-64-little

RELRO: Partial RELRO

Stack: No canary found

NX: NX enabled

PIE: PIE enabled

PIE가 걸려있어 함수 주소는 계속 바뀌지만 프로그램 실행 때마다 dummy함수의 주소를 출력해 주므로 이를 통해 code base를 구하고 get_flag함수를 실행시킬 수 있다.

익스플로잇

```
from signal import signal, SIGPIPE, SIG_DFL
signal(SIGPIPE,SIG_DFL)
from pwn import *
context.log_level = "debug"
def slog(n, m): return success(": ".join([n, hex(m)]))
p = remote("rev.isangxcaution.xyz", 30000)
elf = ELF("./welcome")


slog("dummy", elf.symbols["dummy"])
slog("get_flag", elf.symbols["get_flag"])
```

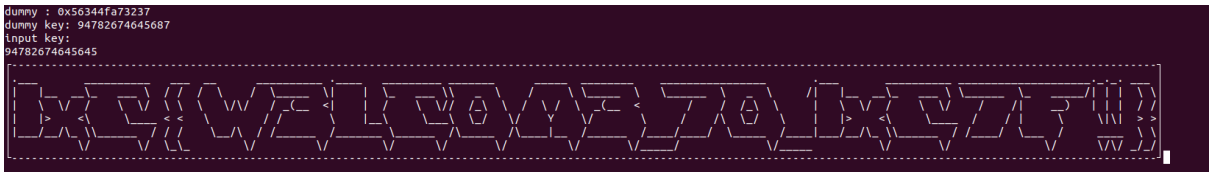
[+] dummy: 0x1237

[+] get_flag: 0x124d

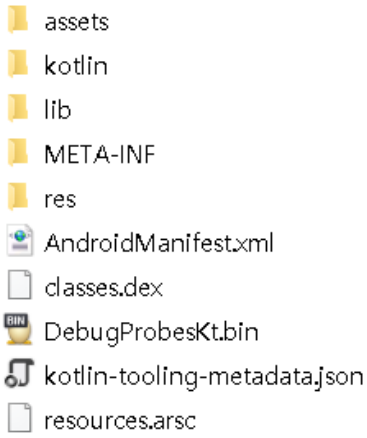
#offset은 항상 동일

```
dummy_addr = int(input("dummy_addr: "),16)
code_base = dummy_addr - 0x1237
print("code_base: " + hex(code_base))
get_flag = code_base + 0x124d
get_flag = get_flag ^ 0x65
get_flag += 101
print("get_flag: " + hex(get_flag))
print(get_flag)
```

```
dummy_addr: 0x56344fa73237
code_base: 0x56344fa72000
get_flag: 0x56344fa7328d
94782674645645
```



2. ChatFlag



문제 의도가 절대 이게 아닌 것 같은데 CTF에서 처음 보는 파일 제공 형식이라 파일들을 하나씩 뒤지다가 assets -> flutter_assets -> images -> img.png 경로에서 이미지 파일을 보니 flag가 적혀 있었다...