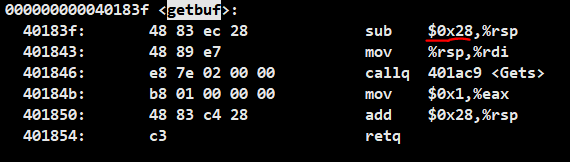
**Target : 34**

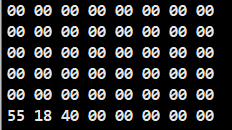
**Phase 1**

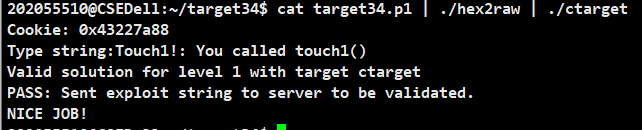
Pdf 파일에 test()함수가 getbuf()라는 함수로 입력을 받고 있으므로 disassemble한 파일로 getbuf를 보게 되면



0x28, 즉 40만큼의 버퍼의 크기로 입력을 받음을 확인 할 수 있다. 목적은 touch1함수를 실행하게 하는것이므로 단순히 40바이트 만큼의 아무 값을 채워넣고, 그 뒤에 touch1 함수의 주소인

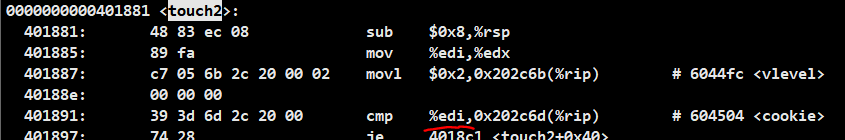
 0x401855를 little-endian 방식으로 입력하여

이처럼 파일을 생성하여 결과를 확인하면 된다.

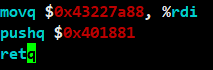


**Phase 2**

Phase 2에서는 touch2 함수를 호출하고, 내가 입력한 값이 미리 주어진 cookie의 값과 같으면 문제가 해결된다.

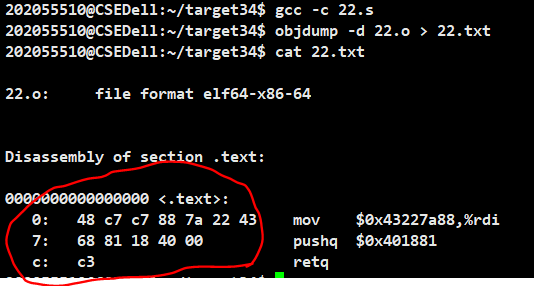
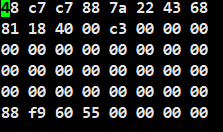
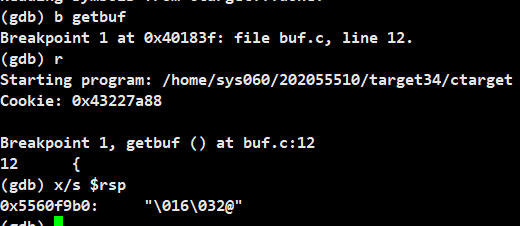
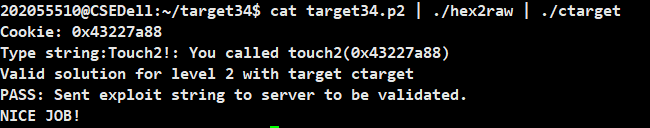


touch2 함수내에서 rdi값이 인자로 전달되어 비교를 하고 있으므로, 어셈블리어로 직접 파일을 다음과 같이 만들어준다.



%rdi에 쿠키값을 넣고, touch2의 주소를 push하여 ret이후에 touch2함수가 불러지게끔 한다.

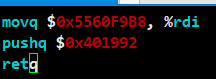
이후 컴파일과 disas과정을 거친후 얻는

  
다음과 같은 hex값들을 버퍼 상단에 넣어주고, 40바이트 중 남은 부분들을 채워 넣은 뒤에 이 코드를 버퍼에 직접 넣어야 하므로 버퍼의 주소를 gdb로 구해 %rsp의주소 0x5560f9b0에서 버퍼의 크기인 0x28을 뺀 0x5560f988이 버퍼의 주소이므로 little endian 방식으로 넣어주면 된다.  


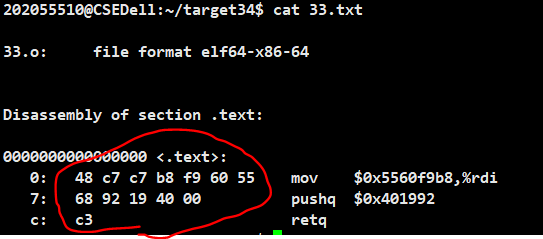
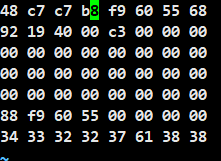
**Phase 3**

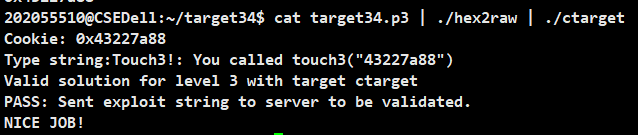
Phase 3에서는 매개변수를 char \* 형인 문자열을 입력으로 받고, 내부에서 hexmatch() 함수를 통해 문자열의 hex값이 cookier값과 같은지 확인하므로 %rdi값에 문자열을 hex값으로 바꾼 값의 주소(입력할 문자열의 주소)를 넣어주면 된다. 내가 입력할 문자열의 주소는, 아까 getbuf()함수가 호출될때의 %rsp의주소 0x5560f9b0에서 오버플로우로 인해 첫번째 return address의 위치에 덮어 씌워지게 때문에 0x5560f9b0 + 0x8 = 0x5560f9b8의 값을 %rdi값에 넣어주면 된다.

그렇게되면 어셈블리어 코드는

 0x401992(touch3)의 주소 또한 phase2처럼 push해준다.

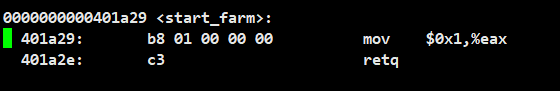
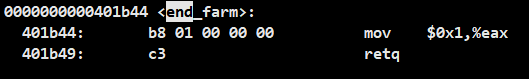
이후 컴파일과 어셈블리로 변환을 해준 뒤, 만든 코드를 버퍼에 넣어주고, 버퍼의 남은 부분을 padding후에, phase 2에서 구한 버퍼의 주소인 0x5560f988과 쿠키 값 43227a88을 hex값으로 바꾼 34 33 32 32 37 61 38 38을 넣어주면 해결이 된다.



**Phase 4**

Phase 4부터는 ctarget이 아닌 rtarget을 쓰게 되는데, ROP기법을 사용하여 start\_farm과 end\_farm 사이에 있는 명령어들을 찾아서 써야한다.

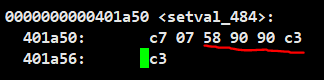
 

Phase 2처럼 touch 2함수를 풀면 되므로, %rdi에 쿠키값을 넣어주어야 한다.

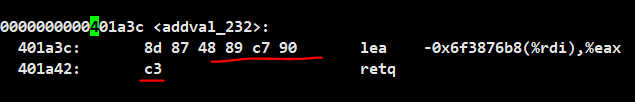
Movq, popq, ret 같은 가젯들을 모아서, 필요한 일련의 식을 짜보게 되면

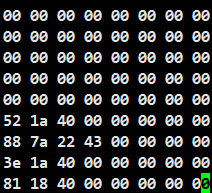
Popq %rax, ret이후 쿠키값을 넣어주어 %rax에 쿠키값을 넣어주고, movq %rax, %rdi를 수행후 최종적으로 touch2 함수의 주소를 넣어주면 인자값으로 쿠키가 전달되어 풀리게 될것이다.

Popq %rax, ret의 hex값을 pdf에서 찾아보게 되면 58 c3이므로

 0x90은 nop이므로 무시하고, 58 c3을 0x401a50 + 0x2 주소인 0x401a52에서 찾을 수 있다.

이후 또 필요한 movq %rax, %rdi의 hex 값은 48 89 c7이므로

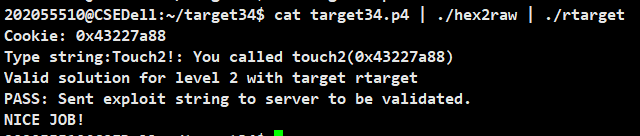
 0x401a3c + 0x2 주소인 0x401a3e에서 찾을 수 있었다. 이를 조합하여 파일을 만들어보고, 실행하면 해결됨을 확인하였다.



여기까지가 buffer를 padding한것이고,

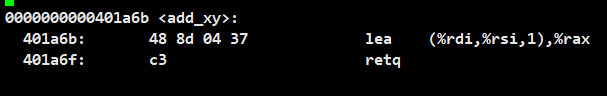
popq %rax, ret의 주소

movq %rax, %rdi의 주소

touch2 함수의 주소 

**Phase 5**

Phase 4와 마찬가지로 ROP기법을 사용하여 touch3함수를 실행시키면 된다. 내가 입력할 cookie의 hex값 문자열이 저장될 주소를 연산하여야 하는데, farm안에 있는 add\_xy() 함수를 이용하여 주소를 연산해야 겠다고 생각했다.



%rdi, %rsi에 각각 %rsp의 주소값, %rsp를 %rdi로 전달할때의 cookie와의 거리(offset) 즉 이 값들을 넣어주게 되면 더하기 연산을 하여 cookie의 주소값이 되므로 %rdi, %rsi값을 설정하는 식으로 코드를 짜보면,

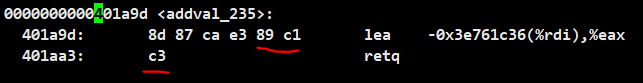
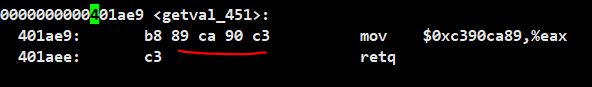
Popq %rax이후, cookie와의 거리(offset)을 넣어주어 %rax에 offset값을 넣어주고, movq %rax %rsi 한뒤, movq %rsp, %rsi이후 add\_xy를 호출하고, movq %rax, %rdi를 하면 add\_xy에서 return된 계산한 주소값이 rdi에 담기고, touch3 함수의 주소를 넣어주어 호출시키고, 마지막에 cookie의 hex값을 적어주면 앞서 계산한 주소에 이 값이 담아져있어 해결될 것이다.

하지만 실제로 farm 내에 있는 여러 gadget들이 다 존재하는 것이 아니어서, 바로바로 필요한 레지스터에 전달하지 못하고 몇번 돌아서 수행해야 했다.

처음 rax -> rdx -> rcx -> rsi를 통해 offset값을 전달하고

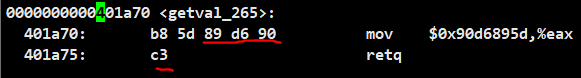
이후 rsp -> rax -> rdi 를 통해 rsp의 주소값을 전달하였다. 그 과정에서 사용한 코드들을 설명하면,

popq %rax는 phase 4에서 사용한 그대로 0x401a52이고,

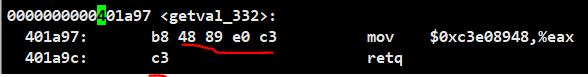
movl %eax, %ecx -> 89 c1이므로 0x401a9d + 0x4 = 0x401aa1에서 찾고

movl %ecx, %edx -> 89 ca이므로

0x401ae9 + 0x1 = 0x401aea에서 찾고

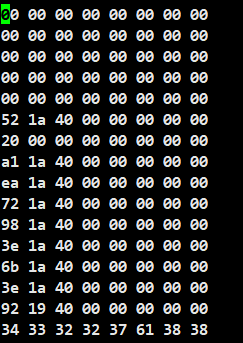
movl %edx, %esi -> 89 d6이므로

0x401a70 + 0x2 = 0x401a72에서 찾고

여기까지 %rsi에 offset 값을 넣어 준것이고,

movq %rsp, %rax -> 48 89 e0이므로

0x401a97 + 0x1 = 0x401a98에서 찾고

이렇게 구한 코드들을 바탕으로 파일을 작성해보면

여기까지 buffer padding

popq %rax

offset = (mov rax rdi에서 cookie 거리8byte\*4)32 => 0x20

mov %eax %ecx

mov %ecx %edx

mov %edx %esi

movq %rsp %rax

movq %rax %rdi (phase 4에서 구함)

add\_xy 호출

movq %rax %rdi

touch3 주소

cookie hex값

이후 결과를 확인하면 정상적으로 실행이 된다.