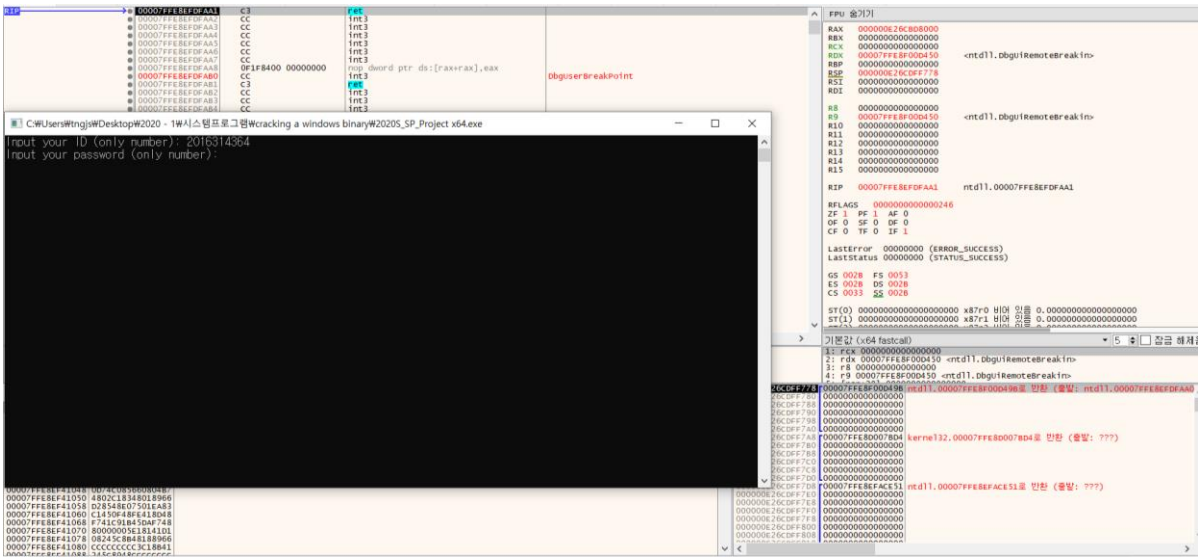
# Project(Cracking a Windows Binary) Report

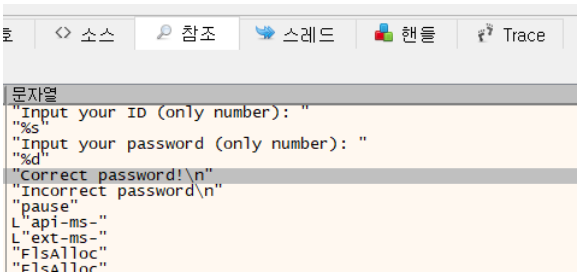
2016314364 박수헌

## • Detail description about code & screenshot

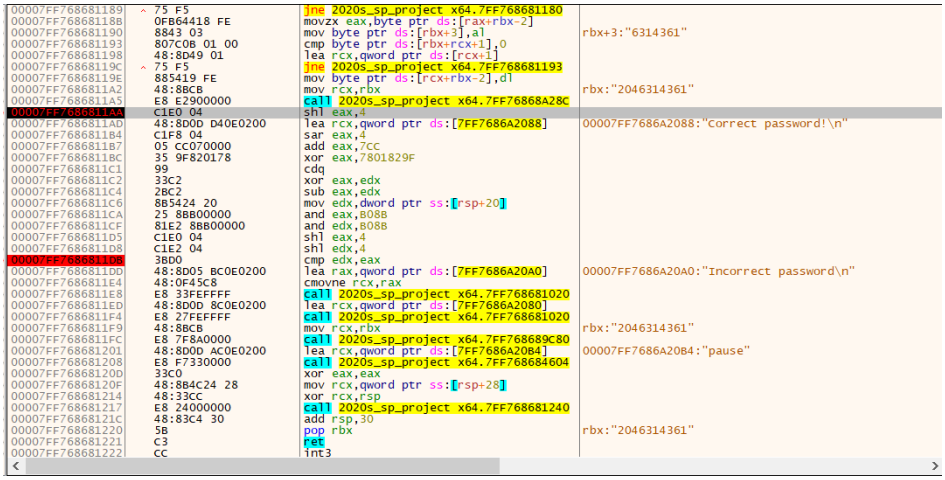
### ID가 학번일 때




.exe 프로그램을 실행 후 학번까지 입력한 후, x64dbg에 부착해 열어보았다.




문자열 참조에서, Correct Password 라고 출력되는 부분으로 들어가 breakpoint를 설정 후 임의의 password인 12345678을 입력해 보았다.




그리고 Correct password인지 Incorrect password인지를 확인하는 부분이라고 판단되는 부분까지 실행시켜 보았다.
그리고 저 상황에서의 레지스터 값은 아래와 같았다.

```
RAX   0000000079F84779
RBX   0000026E4E9B0770    "2046314361"
RCX   00000000FFFFFFFF
RDX   0000000000000005
RBP   0000000000000000
RSP   000000A3168FFB30
RSI   0000000000000000
RDI   0000026E4E9B5E10

R8    0000000019999999
R9    0000026E4E9BA501
R10   0000000000000000
R11   000000A3168FFA20
R12   0000000000000000
R13   0000000000000000
R14   0000000000000000
R15   0000000000000000
```

RBX에는 2046314361이라는 문자열이 들어있고, RAX는 0x79F84779값이 들어있는데, 이 16진수 수는 10진수로 2046314361이다. 처음 입력한 학번인 2016314364에서 몇 자리의 숫자가 변한 것을 알 수 있었다. 일단 첫번째 문제는 password를 찾는 문제였기에 넘어갔다. 그 이후의 코드들을 살펴보았다.

```
shl eax,4
lea rcx,qword ptr ds:[7FF7686A2088]     00007FF7686A2088:"Correct password!\n"
sar eax,4
add eax,7CC
xor eax,7801829F
cdq
xor eax,edx
sub eax,edx
mov edx,dword ptr ss:[rsp+20]
and eax,B08B
and edx,B08B
shl eax,4
shl edx,4
cmp edx,eax
lea rax,qword ptr ds:[7FF7686A20A0]     00007FF7686A20A0:"Incorrect password\n"
cmovne rcx,rax
call 2020s_sp_project_x64.7FF768681020
lea rcx,qword ptr ds:[7FF7686A2080]
```

shl eax, 4 : 0x79F84779 를 4만큼 left shift 한다. → 0x9F847790

lea rcx, qword ptr ds: [7FF7686A2088] : rcx → “Correct password!\n” 문자열이 담긴다

sar eax, 4 : 0x9F847790 을 4만큼 arithmetic right shift한다. → 0xF9F84779

add eax, 7CC : 0xF9F84779 + 0x7CC → 0xF9F84F45

xor eax, 7801829F : 0x81F9CDDA

cdq : 제수가 32bit data라면 피제수는 64bit가 되어야한다. 그래서 피제수의 bit를 확장해주는 연산이다. 즉, EAX의 부호 비트를 EDX 레지스터까지 확장한다.

xor eax, edx : 0x81F9CDDA^ 0xFFFFFFFF(edx 값은 0xFFFFFFFF) → 0x7E063225

sub eax, edx : eax → 0x7E063226

mov edx, dword ptr ss[rsp+0x20] : edx 값은 [rsp+0x20] = 0xBC614E(12345678, password input)이 된다.

and eax, 0xB08B : eax & 0xB08B → 0x3002

and edx, 0xB08B : edx & 0xB08B → 0x200A

shl eax, 0x4 : eax를 4만큼 left shift한다 → 0x30020

shl edx, 0x4 : edx를 4만큼 left shift한다 → 0x200A0

cmp edx, eax : edx, eax 값이 같으면 ZF : 1 다르면, ZF : 0으로 set 된다.

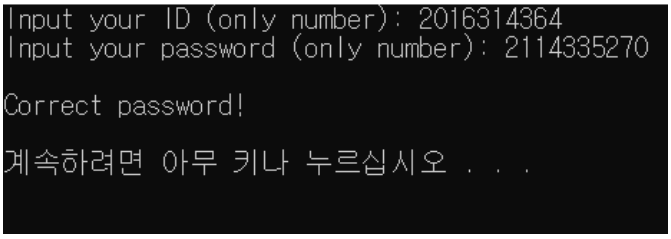
lea rax, ds:[0x00007FF7686A20A0] : rax → “Incorrect password\n” 문자열이 담긴다.

cmovne rcx, rax : ZF = 0이면 “Incorrect password” 문자열이 “Correct password”를 덮어쓴다.

즉, 이 부분에서 Correct password 부분이 덮어씌워지지 않으려면, cmp edx, eax에서 ZF가 1로 set 돼야 하므로, password를 0xB08B와 and연산 후, 4만큼 left shift 한 값이 0x30020과 같아야 한다.

순서대로 돌아가보면, 4만큼 left shift한 값이 0x30020이어야 하므로, 그 전 값은 0x3002이어야 한다.

0xB08B와 and 연산한 값이 0x3002이어야 하므로, password는 0x7E063226(2114335270)이어야 한다.




## Password가 학번일 때

위의 ID가 학번일 때의 경우에서 봤듯이, 2016314364를 input으로 넣었지만, correct인지 incorrect인지 확인하는 부분에 도달했을 때, eax의 값이 2046314361이었다. 어떤 규칙으로 변하는지 알기 위해, ID의 input을 다르게 입력 후 이 부분까지 실행해 확인해 보았다.

1234567890을 넣었을 때, 0x48188A63(1209567843)

0123456789을 넣었을 때, 0xBD4359C(0198456732)의 값이 나왔다.

즉 10자리의 10진수 숫자라고 보았을 때, 3,4번째의 숫자가 각각 10, 9번째의 자리로 들어가고, 9, 10번째의 숫자가 각각 4, 3번째의 자리로 들어갔다.

또한 위에서 봤듯이, 만약 password가 학번인 2016314364였다면, edx 값인 0x782e83fc(2016314364)를 0xB08B와 and연산, 4만큼 left shift 한 값과 rax의 값을 4만큼 left shift, 4만큼 arithmetic right shift, +0x7CC, 7801829F와 xor, 0xFFFFFFFF와 xor, -0xFFFFFFFF, 0xB08B와 and연산, 4만큼 left shift한 값이 같아야 한다.

### <Password>

and 0x782E83FC, 0xB08B → 0x8088

shl 0x8088, 0x4 → 0x80880

### <ID> (구해야하는 값을 answer이라고 하겠다)

거꾸로 올라가며 연산해 보았다.

shl answer, 0x4 → 0x80880

answer = 0x8088…….. ①

and answer, 0x808B → 0x8088…….. ②

answer의 첫번째 byte는 4번째 bit는 1이어야하며, 1,2번째 bit는 1이어서는 안된다.

두 번째 네 번째 byte는 네번째 bit가 1이어야 한다.

그래서 answer는 0xF088로 잡았다

mov edx, dword ptr ss:[rsp+0x20] (“Correct Password\n”)

sub answer, edx→ 0xF088

④에 의하여 edx는 eax의 MSB가 0이라면 0, 1이라면 0xFFFFFFFF의 값을 갖는다.

0이라고 가정한다면 sub answer, 0 → 0xF088이어야 하므로, answer = 0xF088이다.

xor answer, edx → 0xF088…….. ③

edx를 0이라고 가정했으므로, xor answer, 0 → 0xF088이어야 한다.

따라서 answer = 0xF088이다.

cdq…….. ④

cdq를 실행 한 후의 eax 값이 위에서 가정한 듯이 0xF088이 나와야하지만, 이 경우 MSB가 1이므로, 가정과 맞지 않다. 즉 잘못된 가정이었다. ②번부터 다시 진행해 보았다.

and answer, 0x808B → 0x8088

그래서 answer는 0x1000F088로 잡았다…….. ②

sub answer, edx→ 0x1000F088

④에 의하여 edx는 eax의 MSB가 0이라면 0, 1이라면 0xFFFFFFFF의 값을 갖는다.

0이라고 가정한다면 sub answer, 0 → 0xF088이어야 하므로, answer = 0x1000F088이다.

xor answer, edx → 0xF088…….. ③

edx를 0이라고 가정했으므로, xor answer, 0 → 0x1000F088이어야 한다.

따라서 answer = 0x1000F088이다.

cdq…….. ④

cdq가 실행된 후의 eax 값은 0x1000F088이라면 edx의 값이 0이 맞으므로 맞는 경우이다.

xor answer, 7801829F → 0x1000F088

answer = 0x68017217…….. ⑤

add answer, 7CC → 0x68017217

answer = 0x68016A4B(1744923211)

sar answer, 4 → 0x87FE861B

answer = 불가능하다. MSB가 1이라면 결과값의 MSB가 F이어야하고, MSB가 0이라면 결과값의 MSB가 0이어야 한다.이는 ⑤에서 0x7801829F와 xor했을 시의 answer의 MSB가 F나 0이어야한다는 뜻이고 그러기 위해선 ③의 answer의 MSB가 1000(2)이거나 0111(2)이어야 한다는 의미이다. 이를 위해서는 ②의 answer이 MSB가 1000(2), 0111(2)이어야 한다. 이 조건을 만족하며 ①번부터 다시 진행해 보았다.

shl answer, 0x4 → 0x80880

answer = 0x8088…….. ①

and answer, 0x808B → 0x8088…….. ②

answer=0x7FFF8088

mov edx, dword ptr ss:[rsp+0x20] (“Correct Password\n”)

sub answer, edx→ 0x7FFF8088

④에 의하여 edx는 eax의 MSB가 0이라면 0, 1이라면 0xFFFFFFFF의 값을 갖는다.

0xFFFFFFFF이라고 가정한다면 sub answer, 0xFFFFFFFF → 0x7FFF8088이어야 하므로, answer = 0x7FFF8087이다.

xor answer, edx → 0x7FFF8087…….. ③

edx를 0xFFFFFFFF이라고 가정했으므로, xor answer, 0xFFFFFFFF → 0x70008087이어야 한다.

따라서 answer = 0x80007F78이다.

cdq…….. ④

cdq를 실행 한 후의 eax 값이 위에서 가정한 듯이 0x80007F78이 나온다면, edx값도 0xFFFFFFFF이 맞다.

xor answer, 7801829F → 0x80007F78

answer = 0xF801FDE7…….. ⑤

add answer, 7CC → 0x0xF801FDE7

answer = 0xF801F61B(4160878107)

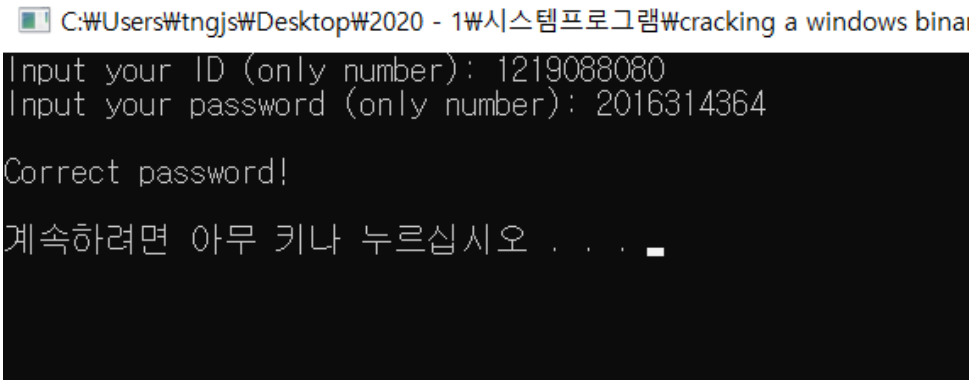
sar answer, 4 → 0xF801F61B

answer = 0x801F61B0

shl eax, 4 → 0x801F61B0

answer = 0x4801F61B(1208088091) : 열 자리를 맞춰주기 위해 MSB의 byte를 0100(2)로 설정했다.

결과적으로 나온 10진수 숫자는 1208088091이다. 하지만, 처음 입력값이 위의 규칙과 같이 달라지므로, 입력해야하는 ID 값은 1219088080이다.



## • Progress and unique experience of work

Bomb Lab과 어느정도 비슷한 부분이 많이 있는 과제여서 그런지, Bomb lab만큼 시간이 오래 걸리진 않았다. Bomb lab은 6단계로 이루어져 있고, 이 과제는 두 개의 문제로 이루어져 있어서 그런 지도 모르겠다.

우선 이 과제는 x64dbg 프로그램을 사용한 부분에서 매우 흥미로웠다. 과제 프로그램뿐만이 아닌, 다른 .exe 파일도 어셈블리어로 본 후 같은 과정을 통해 경험해보고 싶다는 생각이 매우 들었다. 방학에 한번쯤은 꼭 다시 켜보고싶은 프로그램이었다. 그리고 bomb lab을 하던 때 보지 못했던 명령어들을 많이 보아 새로웠다. Cdq나, nop같은 명령어들에 대해도 알게 되어 재미가 있었다. 또한, bomb lab에서처럼, 그저 cmp후 맞으면 ret, 다르면 explode같은 상황이 아니고, 정말 긴 코드들의 연결이었다는 점에서 더 복잡해 보이기도 하였다. 하지만, 결국 어떤 부분에서 input값이 어떻게 변형되고, 어떤 부분에서 정답인지 아닌 지가 판별되는지를 찾아낸 이후부터는 간단한 문제였다. 결국 이 지점을 찾아내는 것이 crack의 첫 시작이자 모든 것이라고 볼 수 있었다. 그 후는 bit manipulation이라든지, 간단한 알고리즘이었다.

한 학기간 시스템 프로그램 수업을 들으며 배운 모든 지식들을 동원해야 풀 수 있는 과제였다는 생각이 들어, 한 학기의 교안들도 많이 읽어보며 돌아보는 과제였다. 굉장히 의미 있었던 시간이었고, 이를 통해 많은 것을 익힐 수 있어 다행이었다.