시스템 소프트웨어

Homework #3

소프트웨어학부
2022041028
박유경

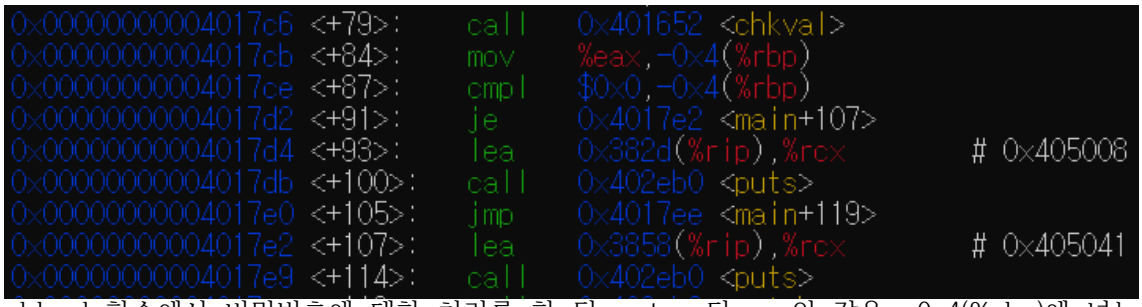
- 성공 스크린 샷

```
PS C:₩Users₩yugyeong> cd C:₩Users₩yugyeong₩hw3
PS C:₩Users₩yugyeong₩hw3> gdb .₩binarypassword2.exe
GNU gdb (GDB for MinGW-W64 x86_64, built by Brecht Sanders) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from .₩binarypassword2.exe...
(gdb) run
Starting program: C:₩Users₩yugyeong₩hw3₩binarypassword2.exe
[New Thread 12716.0x5788]
[New Thread 12716.0x5e44]
qlalfqjsgh

Correct Password
Capture this screen and write report.
```

- 과정

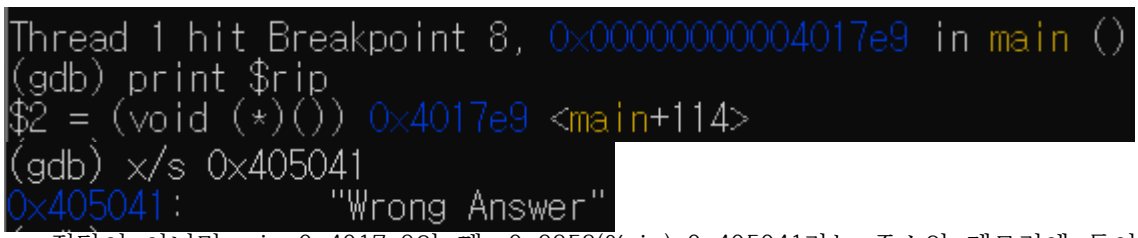
1. main

main에서 gets 함수를 통해 입력을 받고, return된 rax를 rcx에 넣는다.
현재 rcx에는 입력받은 값이 저장된 메모리의 주소가 들어있다.



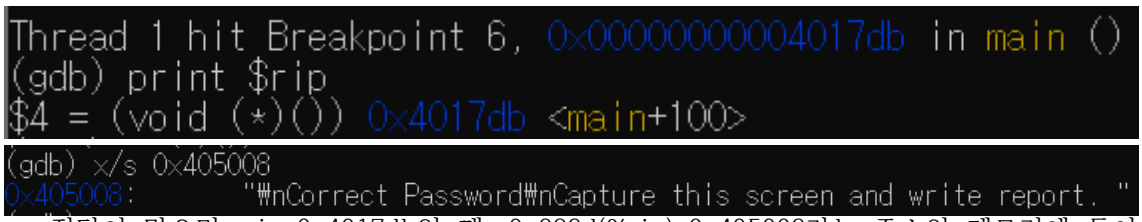
chkval 함수에서 비밀번호에 대한 처리를 한 뒤, return된 eax의 값을 -0x4(%rbp)에 넣는다.

-0x4(%rbp)의 값이 0이면(틀리면), main+107로 jump하여 0x3858(%rip)의 값을 rcx에 넣고 그 값을 puts 함수를 통해 출력한다.



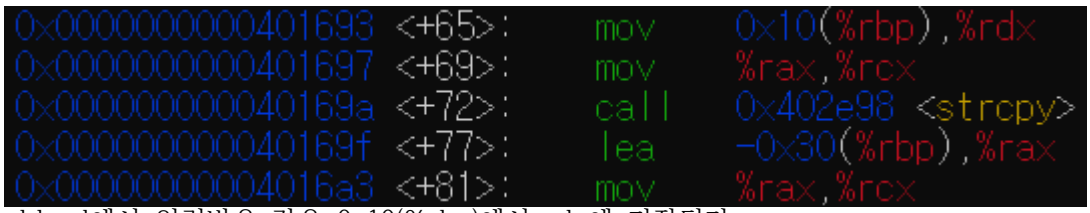
-> 정답이 아니면, rip=0x4017e9일 때, 0x3858(%rip)=0x405041라는 주소의 메모리에 들어있는 “Wrong Answer”를 출력한다.

-0x4(%rbp)의 값이 0이 아니면(맞으면), jump하지 않고 0x382d(%rip)의 값을 rcx에 넣고 그 값을 puts 함수를 통해 출력한다.



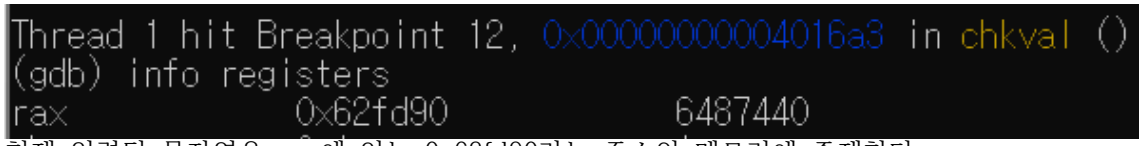
-> 정답이 맞으면, rip=0x4017db일 때, 0x382d(%rip)=0x405008라는 주소의 메모리에 들어있는 “Correct Password\nCapture this screen and write report.”를 출력한다.

2. chkval -> ab, cd, ef, gh 함수를 통해 각 비밀번호의 부분을 비교한다.

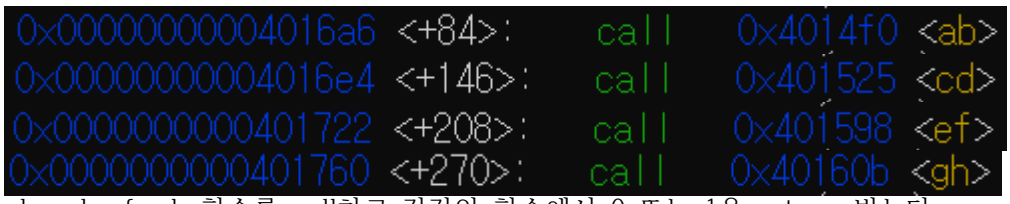


chkval에서 입력받은 값은 0x10(%rbp)에서 rdx에 저장된다.

strcpy 함수를 통해 입력된 값의 메모리 주소를 복사하여 -0x30(%rbp)에서 rax에 저장한다.



현재 입력된 문자열은 rax에 있는 0x62fd90라는 주소의 메모리에 존재한다.



ab, cd, ef, gh 함수를 call하고 각각의 함수에서 0 또는 1을 return 받는다.

3. ab -> “ql” 비교

```
0x00000000004014f9 <+9>:     mov    %rcx,0x10(%rbp)
0x00000000004014fd <+13>:    mov    0x10(%rbp),%rax
0x0000000000401501 <+17>:    movzbl (%rax),%eax
0x0000000000401504 <+20>:    cmp    $0x71,%al
Thread 1 hit Breakpoint 17, 0x00000000004014fd in ab ()
(gdb) info registers
rax            0x62fd90            6487440
rbx            0x1                 1
rcx            0x62fd90            6487440
```

입력받은 값의 메모리 주소는 rcx에서 0x10(%rbp)에 넣고, 다시 0x10(%rbp)에서 rax에 넣는다.

```
(gdb) print/x $eax
$2 = 0x71
(gdb) print/x $al
$3 = 0x71
```

movzbl를 통해 rax가 가리키는 메모리의 값에서 1byte만큼의 데이터를 eax에 저장한다.
즉 “qlalfqjsgh”에서 “q”의 아스키코드 값을 eax에 저장한다.
입력받은 첫 번째 글자인 al의 값과 0x71(아스키코드를 바꾸면 “q”)을 비교한다.
이때, rax는 64비트 레지스터이고, rax는 eax(rax의 하위 32비트 레지스터)와 al(rax의 하위 8비트 레지스터)을 모두 포함한다.

```
0x0000000000401506 <+22>:    jne    0x40151e <ab+46>
0x0000000000401508 <+24>:    mov    0x10(%rbp),%rax
0x000000000040150c <+28>:    add    $0x1,%rax
0x0000000000401510 <+32>:    movzbl (%rax),%eax
0x0000000000401513 <+35>:    cmp    $0x6c,%al
```

al의 값과 “q”가 같지 않으면, 0x40151e로 jump하여 eax에 0을 넣고 함수를 끝낸다.
al의 값과 “q”가 같으면, rax에 1을 더하여 rax에 넣는다.

```
(gdb) x/s 0x62fd91
0x62fd91:       "lalfqjsgh"
```

현재 rax에는 0x62fd91가 들어있으며, 이 주소의 메모리 값은 첫 번째 글자를 제외한 나머지 문자열이다.

```
Thread 1 hit Breakpoint 3, 0x0000000000401513 in ab ()
(gdb) info registers
rax            0x6c                108
```

movzbl을 통해 rax의 1byte만큼의 데이터를 eax에 저장한다.
입력받은 두 번째 글자인 al의 값과 0x6c(아스키코드를 바꾸면 “l”)을 비교한다.

```
0x0000000000401515 <+37>:    jne    0x40151e <ab+46>
0x0000000000401517 <+39>:    mov    $0x1,%eax
0x000000000040151c <+44>:    jmp    0x401523 <ab+51>
0x000000000040151e <+46>:    mov    $0x0,%eax
```

al의 값과 “l”이 같지 않으면, 0x40151e로 jump하여 eax에 0을 넣고 함수를 끝낸다.

al의 값과 “l이 같으면, eax에 1을 저장하고 0x40151e로 jump하여 함수를 끝낸다.

4. cd -> “alf” 비교

```
0x000000000040155b <+54>:    lea    -0x20(%rbp),%rax
0x000000000040155f <+58>:    mov    0x10(%rbp),%rdx
0x0000000000401563 <+62>:    mov    %rax,%rcx
```

rax, rdx, rcx를 초기화한다.

```
Thread 1 hit Breakpoint 3, 0x0000000000401566 in cd ()
(gdb) info registers
rax            0x62fd40            6487360
rbx            0x1                 1
rcx            0x62fd40            6487360
rdx            0x62fd90            6487440
(gdb) x/s 0x62fd90
0x62fd90:       "alfqjsgh"
```

현재 rdx에는 0x62fd90라는 주소가 들어있고, 그 주소의 메모리 값은 입력값에서 ab 함수에서 확인한 글자를 제외한 ”alfqjsgh“이다.

```
0x0000000000401566 <+65>:    call   0x402e98 <strcpy>
0x000000000040156b <+70>:    movb   $0x0,-0x1d(%rbp)
0x000000000040156f <+74>:    lea    -0x20(%rbp),%rax
0x0000000000401573 <+78>:    lea    0x2a96(%rip),%rdx        # 0x404010 <something>
0x000000000040157a <+85>:    mov    %rax,%rcx
```

strcpy 함수를 call하여 입력받은 값을 복사해온다.

```
Thread 1 hit Breakpoint 6, 0x0000000000401573 in cd ()
(gdb) info registers
rax            0x62fd40            6487360
(gdb) x/s 0x62fd40
0x62fd40:       "alf"
```

-0x20(%rbp)의 값을 rax에 넣는다. rax에는 입력한 값에서 세 번째, 네 번째, 다섯 번째 글자가 저장된 주소가 들어있다. 즉 rax에는 0x62fd40라는 주소가 들어있고 이 주소의 메모리 값을 출력하면 입력했던 ”alf“가 들어있는 것을 알 수 있다.

```
Thread 1 hit Breakpoint 1, 0x000000000040157a in cd ()
(gdb) info registers
rax            0x62fd40            6487360
rbx            0x1                 1
rcx            0xffffffffffffffb0  -80
rdx            0x404010            4210704
(gdb) x/s 0x404010
0x404010 <something>:   "alf"
```

0x2a96(%rip)값을 rdx에 넣는다. rdx에는 something 표시되어있는 0x404010라는 주소가 들어있고, 이 주소의 메모리 값을 출력하면 비교할 문자열인 ”alf“가 들어있는 것을 알 수 있다.

```
Thread 1 hit Breakpoint 4, 0x000000000040157d in cd ()
(gdb) info registers
rax            0x62fd40            6487360
rbx            0x1                 1
rcx            0x62fd40            6487360
(gdb) x/s 0x62fd40
0x62fd40:       "alf"
```

rax의 값을 rcx에 넣는다. rcx에는 0x62fd40라는 주소가 들어있고, 이 주소의 메모리 값을 출력하면 입력받은 값인 ”alf“가 들어있는 것을 알 수 있다.

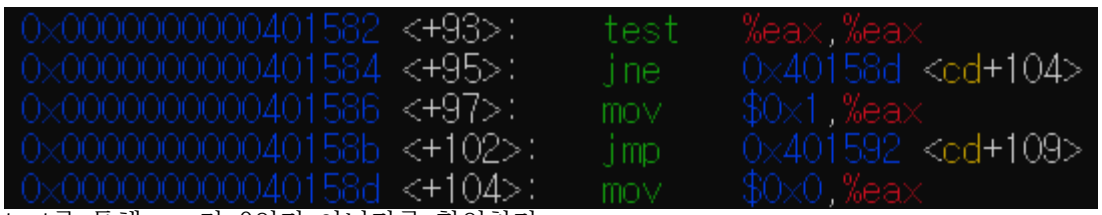
즉 현재 rcx에는 사용자에게 입력받은 값 중 세 번째, 네 번째, 다섯 번째 글자의 주소가 들

어있고, rdx에는 cd함수에서 비교할 문자열인 “alf”가 들어있다.

strcmp 함수를 call하여 rcx와 rdx를 비교한다.

breakpoint로 확인한 결과 strcmp를 call하면 0x00007ff9224d8360로 이동하는 것을 알 수 있었다.

strcmp에서는 cd에서 가져온 rcx(사용자에게 입력받은 값)와 rdx(비교할 문자열)를 비교한다. 문자열을 비교하여 같으면 0을, 다르면 0이 아닌 값을 반환한다.



test를 통해 eax가 0인지 아닌지를 확인한다.

eax가 0이 아니면, 0x40158d로 jump하여, eax에 0을 넣고 함수를 끝낸다.(두 문자열이 다르다.)

eax가 0이면 eax에 1을 넣고 0x401592로 jump하여 함수를 끝낸다.(두 문자열이 같다.)

5. ef -> “qjs” 비교

```
0x00000000004015ce <+54>:    lea    -0x20(%rbp),%rax
0x00000000004015d2 <+58>:    mov    0x10(%rbp),%rdx
0x00000000004015d6 <+62>:    mov    %rax,%rcx
```

rax, rdx, rcx를 초기화한다.

```
Thread 1 hit Breakpoint 9, 0x00000000004015d9 in ef ()
(gdb) info registers
rax            0x62fd40            6487360
rbx            0x1                 1
rcx            0x62fd40            6487360
rdx            0x62fd90            6487440
(gdb) x/s 0x62fd90
0x62fd90:       "qjsgh"
```

현재 rdx에는 0x62fd90 주소가 들어있고, 그 주소의 메모리 값은 입력값에서 ab. cd 함수에서 확인한 글자를 제외한 "qjsgh"이다.

```
0x00000000004015d9 <+65>:    call   0x402e98 <strcpy>
0x00000000004015de <+70>:    movb   $0x0,-0x1d(%rbp)
0x00000000004015e2 <+74>:    lea    -0x20(%rbp),%rax
0x00000000004015e6 <+78>:    lea    0x3a13(%rip),%rdx        # 0x405000
0x00000000004015ed <+85>:    mov    %rax,%rcx
```

strcpy 함수를 call하여 입력받은 값을 복사해온다.

```
Thread 1 hit Breakpoint 5, 0x00000000004015e6 in ef ()
(gdb) info registers
rax            0x62fd40            6487360
(gdb) x/s 0x62fd40
0x62fd40:       "qjs"
```

-0x20(%rbp)의 값을 rax에 넣는다. rax에는 입력한 값에서 여섯 번째, 일곱 번째, 여덟 번째 글자의 주소가 들어있다. 즉 rax에는 0x62fd40라는 주소가 들어있고 이 주소의 메모리 값을 출력하면 입력했던 “qjs”가 들어있는 것을 알 수 있다.

```
Thread 1 hit Breakpoint 6, 0x00000000004015ed in ef ()
(gdb) info registers
rax            0x62fd40            6487360
rbx            0x1                 1
rcx            0xffffffffffffffb0  -80
rdx            0x405000            4214784
```

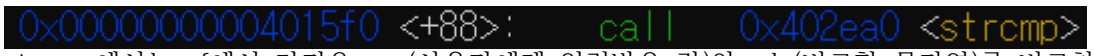
0x3a13(%rip)값을 rdx에 넣는다. rdx에는 0x404010라는 주소가 들어있고, 이 주소의 메모리 값을 출력하면 비교할 문자열인 ”qjs“가 들어있는 것을 알 수 있다.

```
Thread 1 hit Breakpoint 7, 0x00000000004015f0 in ef ()
(gdb) info registers
rax            0x62fd40            6487360
rbx            0x1                 1
rcx            0x62fd40            6487360
(gdb) x/s 0x62fd40
0x62fd40:       "qjs"
```

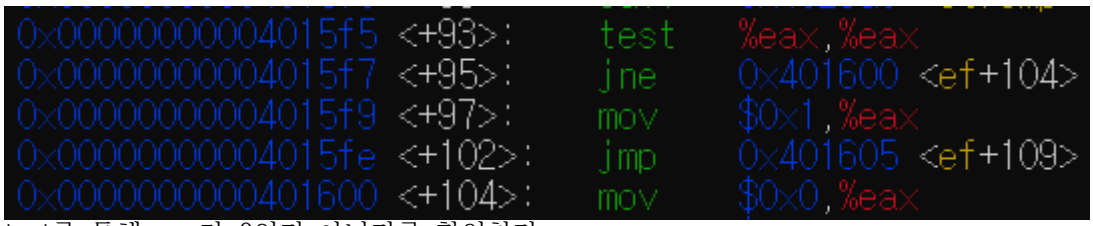
rax의 값을 rcx에 넣는다. rcx에는 0x62fd40라는 주소가 들어있고, 이 주소의 메모리 값을 출력하면 입력받은 값인 ”qjs“가 들어있는 것을 알 수 있다.

즉 현재 rcx에는 사용자에게 입력받은 값 중 여섯 번째, 일곱 번째, 여덟 번째의 주소가 글자

가 들어있고, rdx에는 ef 함수에서 비교할 문자열인 “qjs”가 들어있다.



strcmp에서는 ef에서 가져온 rcx(사용자에게 입력받은 값)와 rdx(비교할 문자열)를 비교한다.
문자열을 비교하여 같으면 0을, 다르면 0이 아닌 값을 반환한다.



test를 통해 eax가 0인지 아닌지를 확인한다.
eax가 0이 아니면, 0x401600로 jump하여, eax에 0을 넣고 함수를 끝낸다.(두 문자열이 다르다.)
eax가 0이면 eax에 1을 넣고 0x401605로 jump하여 함수를 끝낸다.(두 문자열이 같다.)

6. gh -> “gh” 비교

```
(gdb) run
Starting program: C:₩Users₩yugyeong₩hw3₩binarypassword2.exe
[New Thread 8008.0x5bc8]
[New Thread 8008.0x54c4]
abcdefghij
```

입력 값과 비교 값을 구분하기 위해 입력을 “abcdefghij”로 하였다.

```
0x0000000000401614 <+9>:     mov    %rcx,0x10(%rbp)
0x0000000000401618 <+13>:    mov    0x10(%rbp),%rax
0x000000000040161c <+17>:    add    $0x1,%rax
0x0000000000401620 <+21>:    movzbl (%rax),%eax
0x0000000000401623 <+24>:    movsbl %al,%edx
(gdb) info registers
rax            0x62fd90            6487440
rbx            0x1                 1
rcx            0x62fd90            6487440
$1 = 0x62fd90
(gdb) x/s 0x62fd90
0x62fd90:       "ij"
```

rcx에는 사용자로부터 입력받은 값의 주소인 0x62fd90가 들어있고, 그 주소의 메모리 값은 rcx에는 입력 값의 아홉 번째, 열 번째 글자인 “ij”이다.

rcx의 값을 0x10(%rbp)에 넣고, 0x10(%rbp)의 값을 다시 rax에 넣는다.

```
(gdb) info registers
rax            0x62fd91            6487441
(gdb) x/s 0x62fd90
0x62fd90:       "ij"
(gdb) x/s 0x62fd91
0x62fd91:       "j"
```

add 명령어를 통해 rax의 값에 1을 더하고 그 값을 rax에 넣는다.

0x62fd90의 주소에 1을 더하여 rax에는 0x62fd91라는 주소가 들어있다.

char 형식은 1byte이고 문자열은 연속된 메모리의 주소에 저장되어 있기 때문에, 0x62fd91의 주소에는 “ij”에서 1byte 넘어간 “j”가 들어있다.

```
(gdb) print/x $eax
$1 = 0x6a
```

movzbl 명령어를 통해 rax에 저장되어 있는 0x62fd91라는 주소의 메모리의 값에서 1byte만큼의 데이터를 rax의 하위 32비트 레지스터인 eax에 저장한다.

즉 eax에는 두 글자 중 두 번째 글자인 “j”의 아스키코드 0x6a가 들어있다.

```
(gdb) print/x $edx
$2 = 0x6a
```

movsbl 명령어를 통해 rax의 하위 8비트 레지스터인 al의 값을 edx에 넣는다.

즉 edx에는 두 글자 중 두 번째 글자인 “j”의 아스키코드 0x6a가 들어있다.

```
0x0000000000401626 <+27>:    mov    0x10(%rbp),%rax
0x000000000040162a <+31>:    movzbl (%rax),%eax
0x000000000040162d <+34>:    movsbl %al,%eax
```

```
Thread 1 hit Breakpoint 4, 0x000000000040162a in gh ()
(gdb) info registers
rax            0x62fd90            6487440
```

```
(gdb) x/s 0x62fd90
0x62fd90:       "ij"
```

다시 rax에 0x10(%rbp)의 값을 넣는다.

rax에는 사용자로부터 입력받은 값의 주소인 0x62fd90이 들어있고, 그 주소의 메모리 값은 입력 값의 아홉 번째, 열 번째 글자인 “ij”이다.

```
(gdb) print/x $eax
$3 = 0x69
```

movzbl 명령어를 통해 rax에 저장되어 있는 0x62fd90라는 주소의 메모리의 값에서 1byte만큼의 데이터를 rax의 하위 32비트 레지스터인 eax에 저장한다.

즉 eax에는 두 글자 중 첫 번째 글자인 “i”의 아스키코드 0x69가 들어있다.

```
(gdb) print/x $eax
$5 = 0x69
```

movsbl 명령어를 통해 rax의 하위 8비트 레지스터인 al의 값을 eax에 넣는다.

즉 eax에는 두 글자 중 첫 번째 글자인 “i”의 아스키코드 0x69가 들어있다.

```
0x0000000000401630 <+37>:    sub    %eax,%edx
0x0000000000401632 <+39>:    mov    %edx,%eax
```

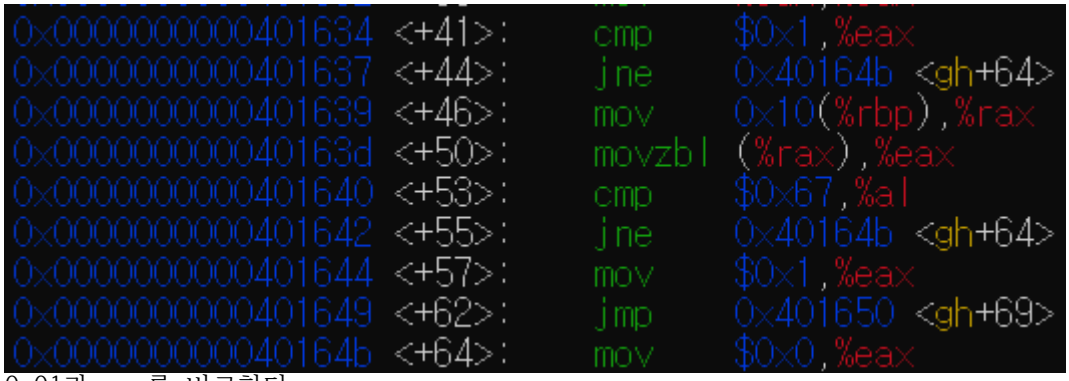
```
Thread 1 hit Breakpoint 7, 0x0000000000401632 in gh ()
(gdb) print/x $edx
$6 = 0x1
```

edx(“j”의 아스키코드 0x6a)에서 eax(“i”의 아스키코드인 0x69)를 빼서 다시 edx에 넣는다.

즉 edx에는 0x6a - 0x69 = 0x01이 들어있다.

```
Thread 1 hit Breakpoint 8, 0x0000000000401634 in gh ()
(gdb) info registers
rax            0x1                 1
rbx            0x1                 1
rcx            0x62fd90            6487440
rdx            0x1                 1
```

edx의 값 0x01을 eax에 넣는다.



0x01과 eax를 비교한다.
같지 않으면(eax가 1이 아니면), 0x40164b로 jump하여 eax에 0을 넣고 함수를 끝낸다.

같으면(eax가 1이면), 다시 rax에 0x10(%rbp)의 값을 넣고, movzbl 명령어를 통해 1byte만큼의 데이터를 가져와 eax에 입력받은 글자의 두 글자 중 첫 번째 글자인 “i”의 아스키코드 0x69를 넣는다.
그리고 비교할 글자 중 첫 번째 글자인 0x67(아스키코드로 “g”)와 비교한다.

같지 않으면(al이 “g”가 아니면), 0x40164b로 jump하여 eax에 0을 넣고 함수를 끝낸다.

같으면(al이 “g”이면) eax에 1을 넣고 0x401650로 jump하여 함수를 끝낸다.

7. 정리

정리하면, 각 ab, cd, ef, gh 함수에서 0 또는 1을 return 하는데,
0이면, 입력받은 값과 해당 부분의 비밀번호가 다르기 때문에 “Wrong Answer”를 출력하고,
1이면, 입력받은 값과 해당 부분의 비밀번호가 같기 때문에 “Correct Password\nCapture this screen and write report.”를 출력하는 로직의 프로그램이다.

함수 ab, cd, ef, gh에서 비교하는 부분(cmp, strcmp)를 주의 깊게 살펴보면서, breakpoint를 찍고 레지스터의 값들을 출력해보며 코드를 이해하고 정답을 알아낼 수 있었다.