

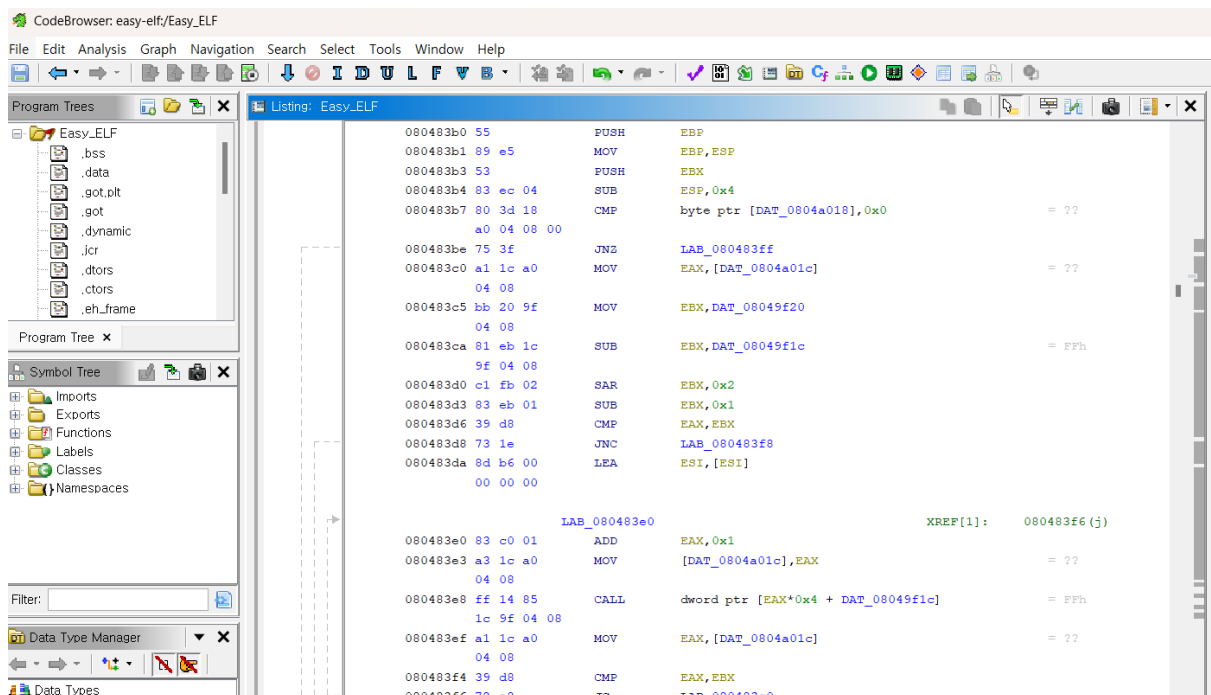
# Easy\_ELF

```
yuna0129@DESKTOP-S0HVT18:/mnt/c/users/gram/Downloads/Easy_ELF$ ./Easy_ELF
Reversing.Kr Easy ELF

1234
Wrong
```

먼저 리눅스에서 실행했을 때 해당 elf 파일은 비밀번호를 입력받는 프로그램으로 보인다. 이 문제를 해결하는 방법은 올바른 비밀번호를 입력하는 것일 것이다.

원래는 gdb로 분석하려 했으나 너무 복잡해서 풀이 방법을 잘 알 수 없었고, ida는 free버전에서는 elf 분석 기능을 제공하지 않기 때문에 새로운 도구를 찾아보았다. ghidra라는 프레임워크로 elf 파일 디버깅 기능을 제공한다고 하여 사용해보았다.



Easy\_ELF 파일을 가져오면 이렇게 초기 화면이 보인다. 먼저 프로그램이 Wrong 문자열을 출력한다는 것을 알고 있으므로 Search 기능에서 문자열 검색 기능을 이용해보았다.

08048276	utf8 u8"write"	"write"	string	6 t
0804827c	utf8 u8"GLIBC_2.7"	"GLIBC_2.7"	string	10 f
08048286	utf8 u8"GLIBC_2.0"	"GLIBC_2.0"	string	10 t
08048653	s_Correct!_0...	ds "Correct!\n"	"Correct!\n"	string 10 t
0804865d	s_Reversing....	ds "Reversing.Kr Easy EL..."	"Reversing.Kr Easy ELF\n\n"	string 24 t
08048675	s_Wrong_08...	ds "Wrong\n"	"Wrong\n"	string 7 t

그러면 이렇게 파일에서 사용하는 문자열 목록이 보인다. 그중 이미 위에서 봤던 Wrong과 새롭게 보인 Correct라는 문자열이 보였다. 아마 Correct는 올바른 비밀번호를 입력했을 때 출력되는

문자열로 보인다. 일단 더블클릭하여 해당 부분으로 이동해보았다.

```

s_Correct!_08048653
08048653 43 6f 72 ds "Correct!\n"
72 65 63
74 21 0a 00
XREF[1]: FUN_080484f7:08048505(*)

```

옆에 표시된 xref 즉 상호 참조 함수를 클릭해보았다.

```

1
2 void FUN_080484f7(void)
3
4 {
5     write(1,"Correct!\n",9);
6     return;
7 }
8

```

해당 함수가 디컴파일된 결과까지 보면 일단은 Correct를 출력하는 단순한 함수이다. 아무래도 main함수를 찾으려면 좀 더 상호 참조를 따라가야 할 것 같다.

```

undefined FUN_080484f7()
undefined <UNASSIGNED> <RETURN>
undefined4 Stack[-0x14]:4local_14
undefined4 Stack[-0x18]:4local_18
undefined4 Stack[-0x1c]:4local_1c
FUN_080484f7
XREF[1]: 080484fd(W)
XREF[1]: 08048505(W)
XREF[1]: 0804850d(*)
XREF[3]: FUN_0804851b:0804854f(c),
080486a0, 0804874c(*)

084f7 55 PUSH EBP
084f8 89 e5 MOV EBP,ESP
084fa 83 ec 18 SUB ESP,0x18
084fb 74 04 JZ 084fd

0804852c 00 00 MOV dword ptr [ESP + local_1c],a_Reversing.Kr_Easy...= "Reversing.Kr Easy
04 5d 86
04 08
08048534 c7 04 24 MOV dword ptr [ESP]=local_20,0x1
01 00 00 00
0804853b e9 20 fe CALL <EXTERNAL>::write ssize_t write(int _
ff ff
08048540 e8 ef fe CALL FUN_08048434 undefined FUN_080484
ff ff
08048545 e8 07 ff CALL FUN_08048451 undefined FUN_080484
ff ff
0804854a 83 f8 01 CMP EAX,0x1
0804854d 75 0c JNZ LAB_0804855b
0804854f e8 a3 ff CALL FUN_080484f7 undefined FUN_080484
ff ff
08048554 b8 00 00 MOV EAX,0x0
00 00
08048559 eb 21 JMP LAB_0804857c

LAB_0804855b
0804855b c7 44 24 MOV dword ptr [ESP + local_18],0x6 XREF[1]: 0804854d(j)
08 06 00
00 00
08048563 c7 44 24 MOV dword ptr [ESP + local_1c],a_Wrong_08048675 = "Wrong\n"
04 75 86
04 08
0804856b c7 04 24 MOV dword ptr [ESP]=local_20,0x1

```

080484f7번 함수의 상호 참조 함수 목록을 계속해서 이동하고 디컴파일 결과를 보다가 main함수로 보이는 0804851b 함수를 발견했다.

이 함수는 다음과 같이 동작한다고 볼 수 있다.

1. 프로그램 정보를 출력한다.
2. FUN\_08048434 함수를 호출하여 입력을 받는다.
3. FUN\_08048451 함수를 호출하여 검사를 수행하고 그 결과를 iVar1에 저장합니다. iVar1의 값에 따라 성공 메시지 (FUN\_080484e7 호출) 또는 실패 메시지("Wrong Wn" 출력)를 출력한다.
4. 0을 반환하며 종료한다.

따라서 검사를 수행하는 함수인 FUN\_08048451을 따라가 보았다.

```
1
2 undefined4 FUN_08048451(void)
3
4 {
5     undefined4 uVar1;
6
7     if (DAT_0804a021 == '1') {
8         DAT_0804a020 = DAT_0804a020 ^ 0x34;
9         DAT_0804a022 = DAT_0804a022 ^ 0x32;
10        DAT_0804a023 = DAT_0804a023 ^ 0x88;
11        if (DAT_0804a024 == 'X') {
12            if (DAT_0804a025 == '\0') {
13                if (DAT_0804a022 == 0x7c) {
14                    if (DAT_0804a020 == 0x78) {
15                        if (DAT_0804a023 == 0xdd) {
16                            uVar1 = 1;
17                        }
18                        else {
19                            uVar1 = 0;
20                        }
21                    }
22                    else {
23                        uVar1 = 0;
24                    }
25                }
26                else {
27                    uVar1 = 0;
28                }
29            }
30            else {
31                uVar1 = 0;
32            }
33        }
```

```

34     else {
35         uVar1 = 0;
36     }
37 }
38 else {
39     uVar1 = 0;
40 }
41 return uVar1;
42 }
43

```

특정 메모리 주소 (DAT\_0804a020부터 DAT\_0804a025)에 저장된 값들을 순차적으로 검사하고, 특정 조건을 만족하는 경우에만 1을 반환하고, 그렇지 않으면 0을 반환하는 함수이다.

이 함수는 특정 메모리 영역의 6바이트 데이터 (DAT\_0804a020 ~ DAT\_0804a025)가 특정 값과 패턴을 가지는지 검사하는 것으로 보인다. 처음 3바이트는 특정 값과 XOR 연산을 거친 후 특정한 값이 되어야 하며, 마지막 2바이트는 특정 문자 값 ('X', 'W0')이어야 한다.

input[1] = '1'

input[0] ^ 0x34 == 0x78

input[2] ^ 0x32 == 0x7c

input[3] ^ 0x88 == 0xdd

input[4] == 'X'

input[5] == 'W0'

이렇게 정리할 수 있다.

이제 0,2,3 부분을 알아내기 위해 파이썬 코드를 작성해본다. 내가 작성한 코드는 다음과 같다.

```

C: > Users > gram > Desktop > Untitled-3.py > ...
1  def find_password():
2      password = [''] * 6 # input은 6바이트(DAT_0804a020 ~ DAT_0804a025)
3
4      # 조건 1: input[1] == '1'
5      password[1] = '1'
6
7      # 조건 2: input[0] ^ 0x34 == 0x78
8      password[0] = chr(0x78 ^ 0x34)
9
10     # 조건 3: input[2] ^ 0x32 == 0x7c
11     password[2] = chr(0x7c ^ 0x32)
12
13     # 조건 4: input[3] ^ 0x88 == 0xdd
14     password[3] = chr(0xdd ^ 0x88)
15
16     # 조건 5: input[4] == 'X'
17     password[4] = 'X'
18
19     # 조건 6: input의 길이는 5이다.
20     # DAT_0804a025 == '\0' 조건이 있으므로 널 문자('\0')
21     password[5] = '\0'
22
23     return "".join(password)
24
25 if __name__ == "__main__":
26     found_password = find_password()
27     print(f"찾아낸 문자열: {found_password}")
28
29

```

```

문제 풀이 디버그 콘솔 디버깅 노트
PS C:\Users\gram> & C:/Users/gram/AppData/Local/Programs/Python/Python39-64/Python.exe C:/Users/gram/Desktop/Untitled-3.py
찾아낸 문자열: L1NUX

```

그러면 이렇게 출력된다.

```

yuna0129@DESKTOP-S0HVT18:/mnt/c/users/gram/Downloads/Easy_ELF$ ./Easy_ELF
Reversing.Kr Easy ELF

L1NUX
Correct!

```

이렇게 찾아낸 플래그가 옳음을 확인 가능하다.