악성코드과제1

20190937 이혜린

```
ellie@ubuntu:~$ cat > hw1.c
#include <stdio.h>
int main(){
        char student_no[] = {"20190937"};
        printf("hello %s\n", student_no);
        return 0;
}
^C
```

20190937 % 6 = 1 이어서 gcc -O1 -o hw1 hw1.c -no-pie 로 컴파일 했습니다.

```
ellie@ubuntu:~$ gcc -O1 -o hw1 hw1.c -no-pie
```

Gdb hw1 로 디버깅합니다.

```
ellie@ubuntu:~$ gdb hw1
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
l>
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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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"...
pwndbg: loaded 192 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from hw1...
(No debugging symbols found in hw1)
pwndbg>
```

Disass main 결과

```
pwndbg> disass main
Dump of assembler code for function main:
   0x0000000000401156 <+0>:     endbr64
   0x000000000040115a <+4>:     push   rbx
   0x000000000040115b <+5>:     sub    rsp,0x20
   0x000000000040115f <+9>:     mov    ebx,0x28
   0x0000000000401164 <+14>:    mov    rax,QWORD PTR fs:[rbx]
   0x0000000000401168 <+18>:    mov    QWORD PTR [rsp+0x18],rax
   0x000000000040116d <+23>:    xor    eax,eax
   0x000000000040116f <+25>:    movabs rax,0x3733393039313032
   0x0000000000401179 <+35>:    mov    QWORD PTR [rsp+0xf],rax
   0x000000000040117e <+40>:    mov    BYTE PTR [rsp+0x17],0x0
   0x0000000000401183 <+45>:    lea    rdx,[rsp+0xf]
   0x0000000000401188 <+50>:    lea    rsi,[rip+0xe75]        # 0x402004
   0x000000000040118f <+57>:    mov    edi,0x1
   0x0000000000401194 <+62>:    mov    eax,0x0
   0x0000000000401199 <+67>:    call   0x401060 <__printf_chk@plt>
   0x000000000040119e <+72>:    mov    rax,QWORD PTR [rsp+0x18]
   0x00000000004011a3 <+77>:    xor    rax,QWORD PTR fs:[rbx]
   0x00000000004011a7 <+81>:    jne    0x4011b4 <main+94>
   0x00000000004011a9 <+83>:    mov    eax,0x0
   0x00000000004011ae <+88>:    add    rsp,0x20
   0x00000000004011b2 <+92>:    pop    rbx
```

```
   0x00000000004011b3 <+93>:    ret
   0x00000000004011b4 <+94>:    call   0x401050 <__stack_chk_fail@plt>
```

ni 한번 해본 결과

```
Starting program: /home/ellie/hw1

Breakpoint 1, 0x0000000000401156 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────[ REGISTERS ]──────────────────────────────
 RAX  0x401156 (main) ◂— endbr64
 RBX  0x4011c0 (__libc_csu_init) ◂— endbr64
 RCX  0x4011c0 (__libc_csu_init) ◂— endbr64
 RDX  0x7fffffffe148 —▸ 0x7fffffffe453 ◂— 'SHELL=/bin/bash'
 RDI  0x1
 RSI  0x7fffffffe138 —▸ 0x7fffffffe443 ◂— '/home/ellie/hw1'
 R8   0x0
 R9   0x7ffff7fe0d50 ◂— endbr64
 R10  0x0
 R11  0x7ffff7f748f0 (intel_02_known+304) ◂— 0x800003400468
 R12  0x401070 (_start) ◂— endbr64
 R13  0x7fffffffe130 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x0
 RSP  0x7fffffffe048 —▸ 0x7ffff7de90b3 (__libc_start_main+243) ◂— mov    edi, eax
 RIP  0x401156 (main) ◂— endbr64
──────────────────────────────[ DISASM ]──────────────────────────────
 ► 0x401156 <main>       endbr64
   0x40115a <main+4>     push   rbx
   0x40115b <main+5>     sub    rsp, 0x20
   0x40115f <main+9>     mov    ebx, 0x28
   0x401164 <main+14>    mov    rax, qword ptr fs:[rbx]
   0x401168 <main+18>    mov    qword ptr [rsp + 0x18], rax
   0x40116d <main+23>    xor    eax, eax
   0x40116f <main+25>    movabs rax, 0x3733393039313032
   0x401179 <main+35>    mov    qword ptr [rsp + 0xf], rax
   0x40117e <main+40>    mov    byte ptr [rsp + 0x17], 0
   0x401183 <main+45>    lea    rdx, [rsp + 0xf]
──────────────────────────────[ STACK ]──────────────────────────────
00:0000│ rsp  0x7fffffffe048 —▸ 0x7ffff7de90b3 (__libc_start_main+243) ◂— mov    edi, eax
01:0008│      0x7fffffffe050 —▸ 0x7ffff7ffc620 (_rtld_global_ro) ◂— 0x5090a00000000
02:0010│      0x7fffffffe058 —▸ 0x7fffffffe138 —▸ 0x7fffffffe443 ◂— '/home/ellie/hw1'
03:0018│      0x7fffffffe060 ◂— 0x100000000
04:0020│      0x7fffffffe068 —▸ 0x401156 (main) ◂— endbr64
05:0028│      0x7fffffffe070 —▸ 0x4011c0 (__libc_csu_init) ◂— endbr64
06:0030│      0x7fffffffe078 ◂— 0xc152dcdb91da0b9
07:0038│      0x7fffffffe080 —▸ 0x401070 (_start) ◂— endbr64
──────────────────────────────[ BACKTRACE ]──────────────────────────────
 ► f 0          401156 main
   f 1     7ffff7de90b3 __libc_start_main+243
──────────────────────────────────────────────────────────────────────────
pwndbg>
```

i r 한 결과

```
pwndbg> i r
rax            0x401156            4198742
rbx            0x4011c0            4198848
rcx            0x4011c0            4198848
rdx            0x7fffffffe148      140737488347464
rsi            0x7fffffffe138      140737488347448
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffe048      0x7fffffffe048
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x0                 0
r11            0x7ffff7f748f0      140737353566448
r12            0x401070            4198512
r13            0x7fffffffe130      140737488347440
r14            0x0                 0
r15            0x0                 0
rip            0x40115a            0x40115a <main+4>
eflags         0x246               [ PF ZF IF ]
```

레지스터 정리

ax: 누산기 레지스터, 연산에 사용되는 레지스터

bx: 베이스 레지스터, 메모리의 주소를 저장하는 레지스터

dx: 데이터 레지스터, 연산에 사용되는 레지스터

si: 근원지 인덱스 레지스터, 복사할 데이터의 주소를 저장하는 레지스터

di: 목적지 인덱스 레지스터, 복사한 데이터의 목적지 주소를 저장하는 레지스터

sp: 스택 프레임에서 스택의 끝 지점 주소 (현재 스택 주소)를 저장하는 레지스터

ip: 프로세서가 읽고 있는 현재 명령의 위치를 가리키는 명령 포인터

fs: 데이터 관련 확장 레지스터

(이름이 r로 시작함: 프로세스가 64bit 환경

이름이 e로 시작함: 프로세스가 32 bit 환경)

명령어 정리

mov: source에서 destination으로 데이터를 복사한다.

lea: source 피연산자의 유효 주소를 계산하여 destination 피연산자에 복사한다.

sub: destination에서 source의 값을 빼서 destination에 저장한다.

add: destination에 source의 값을 더해서 destination에 저장한다.

xor: destination과 source 피연산자의 각 비트가 xor 된다.

push: 스택에 값을 넣는다. sp의 값이 4만큼 줄어들고 이 위치에 새로운 값이 채워진다.

pop: sp 레지스터가 가리키고 있는 위치의 스택 공간에서 4byte 만큼을 destination 피연산자에 복사한다. 그리고 sp 레지스터의 값에 4를 더한다.

ptr: 피연산자의 크기를 재설정한다.

call: 프로시저를 호출한다.

jne: Jump if NOT equal 똑같지 않거나 0이 아니면 지정된 주소로 넘어간다.

ret: 호출했던 바로 다음 지점으로 이동한다.

movabs: 임의의 64 비트 상수를 레지스터에 로드하고 임의의 상수 64 비트 주소에서 정수 레지

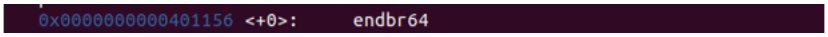
스터를 로드/저장한다.

데이터 타입 정리

QWORD: 64비트 정수

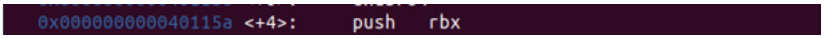
BYTE: 8비트 부호 없는 정수

분석



endbr64

end branch 64bit



push rbx

rbx의 값을 스택에 저장한다.

sub rsp, 0x20

rsp를 0x20만큼 감소시켜 스택에 데이터 공간을 확보한다.

mov ebx, 0x28

0x28을 ebx에 복사한다.

mov rax, QWORD PTR fs:[rbx]

rbx에 들어있는 메모리 주소에 들어있는 값을 rax에 복사한다.

```
0x0000000000401168 <+18>:    mov    QWORD PTR [rsp+0x18],rax
```

mov QWORD PTR [rsp+0x18], rax

rax에 들어있는 값을 rsp+0x18인 메모리 주소에 복사한다.

코드가 스택 검사를 위해 QWORD PTR fs:[rbx]와 QWORD PTR [rsp+0x18]에 같은 값을 넣었다.

```
0x000000000040116d <+23>:    xor    eax,eax
```

xor eax, eax

eax의 값을 0으로 초기화시킨다.

```
0x000000000040116f <+25>:    movabs rax,0x3733393039313032
```

movabs rax, 0x3733393039313032

```
*RAX  0x3733393039313032 ('20190937')
```

0x37333930393130320에 있는 값 (학번 '20190937')을 rax에 복사한다.

```
0x0000000000401179 <+35>:    mov    QWORD PTR [rsp+0xf],rax
```

mov QWORD PTR [rsp+0xf], rax

rax에 들어있는 값('20190937')을 QWORD PTR [rsp+0xf]에 복사한다.

```
0x000000000040117e <+40>:    mov    BYTE PTR [rsp+0x17],0x0
```

mov BYTE PTR [rsp+0x17], 0

0을 byte ptr [rsp+0x17]에 복사한다.

```
0x0000000000401183 <+45>:    lea    rdx,[rsp+0xf]
```

lea rdx, [rsp+0xf]

Rsp+0xf 값을 rdx에 복사한다.

x/s 해서 살펴보면 [rsp+0xf]에 "20190937"이라는 값이 저장되어 있음을 알 수 있다.

```
pwndbg> x/s $rsp+0xf
0x7fffffffe02f: "20190937"
```

```
  0x0000000000401188 <+50>:    lea    rsi,[rip+0xe75]        # 0x402004
```

lea rsi, [rip+ 0xe75]

[rip+0xe75]의 값을 rsi에 복사한다.

```
pwndbg> x/s $rsi
0x402004:       "hello %s\n"
```

rsi의 메모리 주소 상에 "hello %s₩n"이라는 문자열이 저장되어 있음을 알 수 있다.

```
  0x000000000040118f <+57>:    mov    edi,0x1
```

mov edi, 1

1을 edi에 복사한다.

```
  0x0000000000401194 <+62>:    mov    eax,0x0
```

mov eax, 0

eax의 값을 0으로 초기화 시킨다.

```
  0x0000000000401199 <+67>:    call   0x401060 <__printf_chk@plt>
```

call 0x401060 <__printf_chk@plt>

0x401060 주소를 호출하며 해당 주소는 printf()함수의 주소이다.

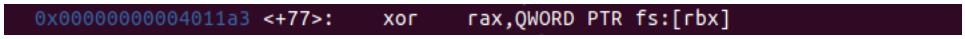
```
pwndbg> x/s $rsi
0x4052a0:       "hello 20190937\n"
```

실행한 후 rsi의 메모리 주소 상에 "hello 20190937₩n"이 들어가 있는 모습을 확인할 수 있다.

```
  0x000000000040119e <+72>:    mov    rax,QWORD PTR [rsp+0x18]
```

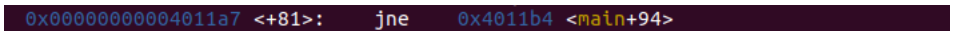
mov rax, QWORD PTR [rsp+0x18]

rax에 QWORD PTR [rsp+0x18]에 들어있는 값을 복사한다.



xor rax, QWORD PTR fs:[rbx]

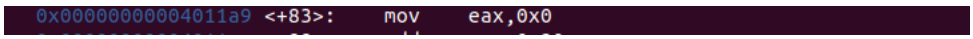
QWORD PTR fs: [rbx]의 값을 rax의 비트와 xor 한다.



jne 0x4011b4, <main+94>

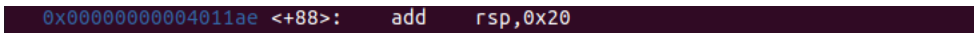
QWORD PTR [rsp+0x18]와 QWORD PTR fs:[rbx]이 같지 않아서 xor한 값이 0이 아니면 0x4011b4로 넘어간다.

코드가 QWORD PTR fs:[rbx]와 QWORD PTR[rsp+0x18]에 같은 값을 넣고 둘이 같지 않으면 스택에 문제가 있다고 판단하고 특정 함수로 넘어간다. 여기서는 값이 같아서 넘어가지 않는다.



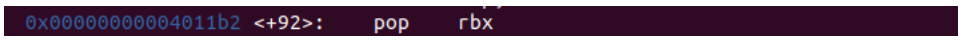
mov eax, 0

eax의 값을 0으로 초기화시킨다.



add rsp, 0x20

rsp의 값을 +0x20 해서 rsp를 main함수 들어오기 전으로 복귀시킨다.



pop rbx

rbx의 값을 스택에서 해제시킨다.

ret

메인 함수를 종료시킨다.

mov edi, eax

eax의 값을 edi로 복사하여 edi의 값을 0으로 초기화시킨다.

call exit <exit>

프로세스를 종료시킨다.