

# 과제02\_GDB를 활용하여 쉘 실행하기

강의명	시스템프로그래밍
작성자	2019136056 박세현
날짜	@2023년 9월 14일

## 목차

1. 소스코드 컴파일 후 GDB 실행
2. BreakPoint 설정
3. 쉘 실행을 위한 순서
4. 방법
5. 정리

## 1. 소스코드 컴파일 후 GDB 실행

먼저 작성한 소스코드를 디버깅 할 수 있게 아래 명령어를 이용해 컴파일 한다.

```
$ gcc quiz.c -o debug -g
```

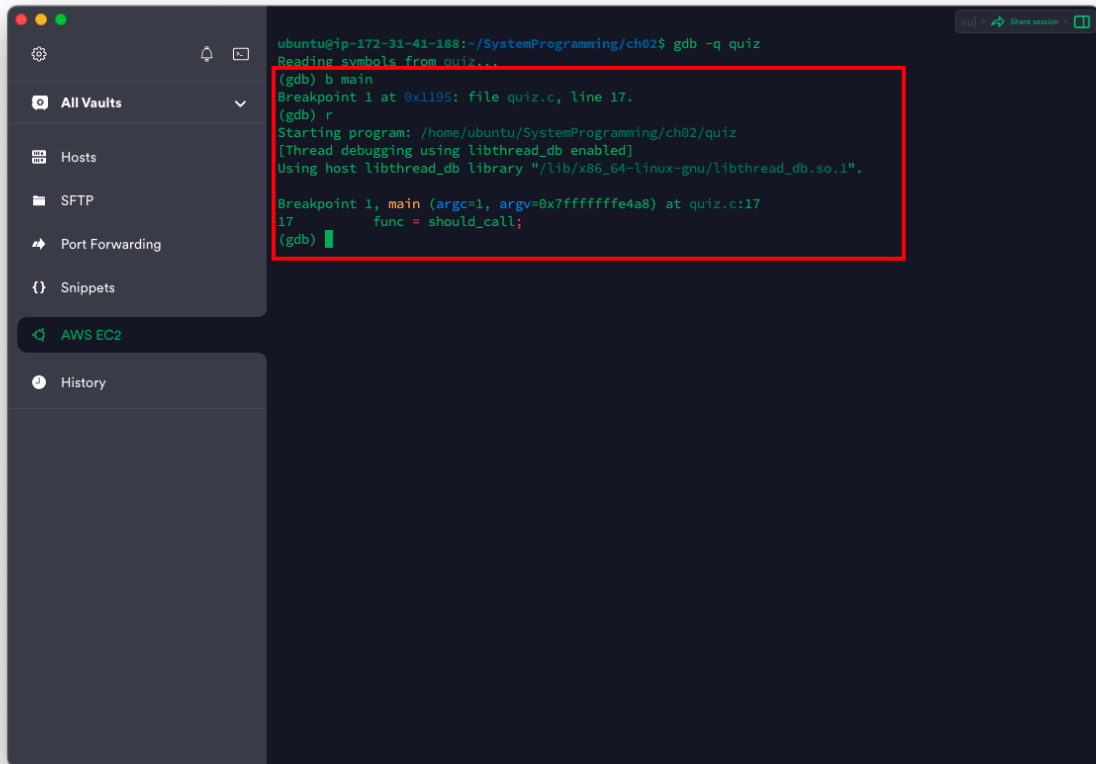
컴파일 후 아래 명령어를 통해 GDB를 실행한다.

```
$ gdb -1 ./quiz
```

## 2. BreakPoint 설정

GDB 실행 후 아래 명령어를 통해 메인 함수에 BreakPoint를 설정 후 실행한다.

```
(gdb) b main  
(gdb) r
```



위와 같이 실행 되었다면 현재 `func = should_call;` 이라는 명령어가 다음에 실행된다는 것을 의미한다.

### 3. 쉘 실행을 위한 순서

#### 1) 소스 코드의 동작

GDB 상에서 쉘을 실행하도록 실행 흐름을 바꾸기 위해서는 먼저 메인 함수의 소스 코드를 살펴봐야한다.

```
int main(int argc, char **argv)
{
    void (*func)(char *);

    func = should_call;
    func("no way\n");

    return 0;
}
```

메인 함수는 아래와 같이 동작한다.

- `func` 변수에 `should_call` 값을 넣어 해당 함수를 할당한다.

- `func("no way\n");` 명령어는 실제로 `should_call("no way\n");` 로 동작하며, 인자로 "no way\n" 라는 문자열을 전달한다.

## 2) 셸 실행 방법

먼저 C 코드에서 셸을 실행하기 위해서 `system("/bin/sh")` 코드를 작성하면 시스템 함수에서 셸을 호출한다.

셸을 실행하기 위해서 2가지를 수행해야한다.

- `func` 변수에 `system` 함수를 할당해야한다.
- "no way\n" 인자로 전달되는 문자열을 `"/bin/sh"` 로 변경해야한다.

## 4. 방법

위의 사진에서 `n` 을 명령어로 입력하면 다음 줄로 넘어가게 된다.

```

ubuntu@ip-172-31-41-188:~/SystemProgramming/ch02$ gdb -q quiz
Reading symbols from quiz...
(gdb) b main
Breakpoint 1 at 0x1195: file quiz.c, line 17.
(gdb) r
Starting program: /home/ubuntu/SystemProgramming/ch02/quiz
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe4a8) at quiz.c:17
17 func = should_call;
(gdb) n
18 func("no way\n");
(gdb)

```

### 1) func 변수 값 변경

현재 상태는 `func = should_call;` 이 실행된 상태이며 다음 실행할 명령은 `func("no way\n");` 인 것이다. 따라서, 지금 시점에서 `func` 변수에 들어간 값을 변경해야 한다. 이전 단계에서 값을 변경할 경우 기존 명령어가 실행되면서 변경한 값이 소용없어지기 때문이다.

```
(gdb) set var func = system
```

위 명령어를 통해 `func` 변수에 할당된 값을 변경할 수 있다. GDB에서 함수의 주소는 함수의 이름을 작성하면 되기 때문에 `system` 을 그대로 작성하면 된다.

## 2) 레지스터 확인

```
(gdb) disas main
```

위 명령어를 입력하면 메인 함수 내에서 실행되는 어셈블리 코드 명령어들이 나열된 것이 보인다.

```
ubuntu@ip-172-31-41-188:~/SystemProgramming/ch02$ gdb -q quiz
Reading symbols from quiz...
(gdb) b main
Breakpoint 1 at 0x1195: file quiz.c, line 17.
(gdb) r
Starting program: /home/ubuntu/SystemProgramming/ch02/quiz
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

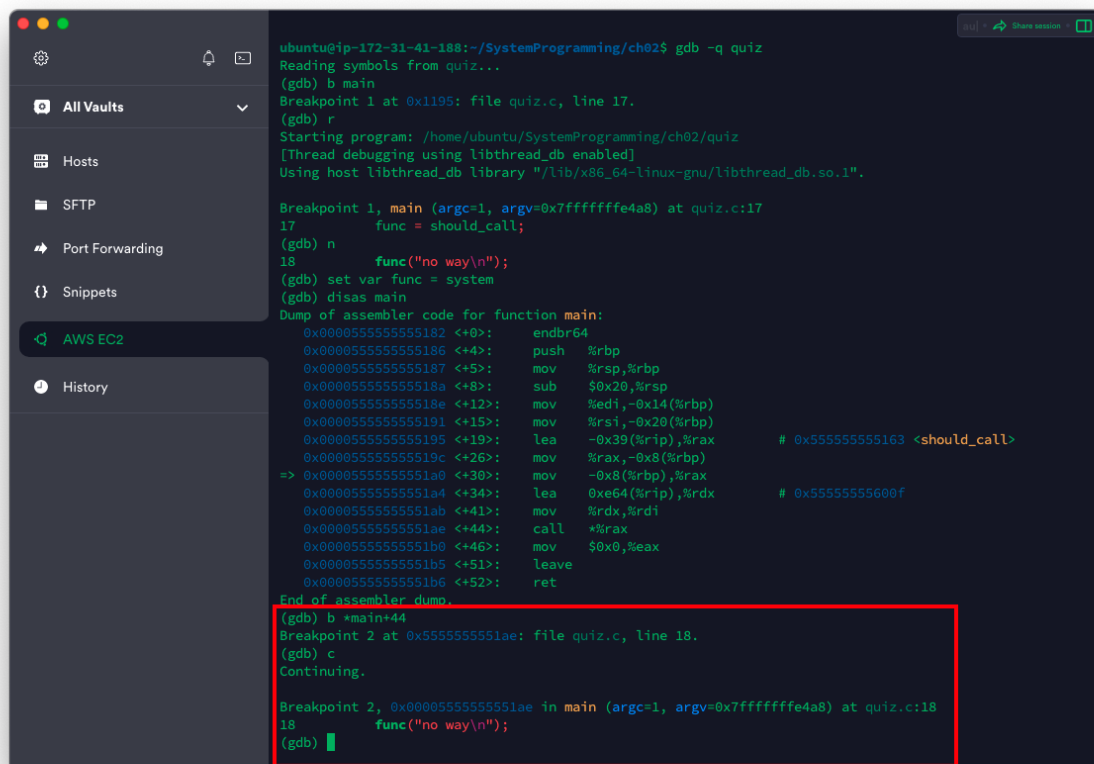
Breakpoint 1, main (argc=1, argv=0x7fffffffe4a8) at quiz.c:17
17      func = should_call;
(gdb) n
18      func("no_wav\n");
(gdb) disas main
Dump of assembler code for function main:
0x000055555555182 <+0>: endbr64
0x000055555555186 <+4>: push    %rbp
0x000055555555187 <+5>: mov     %rsp,%rbp
0x00005555555518a <+8>: sub     $0x20,%rsp
0x00005555555518e <+12>: mov     %edi,-0x14(%rbp)
0x000055555555191 <+15>: mov     %rsi,-0x20(%rbp)
0x000055555555195 <+19>: lea     -0x39(%rip),%rax          # 0x55555555163 <should_call>
0x00005555555519c <+26>: mov     %rax,-0x8(%rbp)
=> 0x0000555555551a0 <+30>: mov     -0x8(%rbp),%rax
0x0000555555551a4 <+34>: lea     0xe64(%rip),%rdx          # 0x555555555600f
0x0000555555551ab <+41>: mov     %rdx,%rdi
0x0000555555551ae <+44>: call    *%rax
0x0000555555551b0 <+46>: mov     $0x0,%eax
0x0000555555551b5 <+51>: leave
0x0000555555551b6 <+52>: ret
End of assembler dump.
(gdb)
```

현재 `func` 변수의 값은 `rax` 레지스터에 저장되어 있으며, 44번째 줄에서 함수 호출이 일어난다. 그 전 단계에서 `rdi` 레지스터에 "no way\n" 라는 문자열을 넣어주고 있다. 따라서 함수 호출이 실행되기 직전에 BreakPoint를 설정하고 "no way\n"라는 문자열을 `"/bin/sh"` 로 변경해주면 셸을 실행할 수 있다.

## 3) BreakPoint 설정

아래 명령어를 통해 함수 호출이 실행되기 직전인 메인 함수에서 +44번째 줄에 BreakPoint를 설정하고, c를 입력해 continue 한다.

```
(gdb) b *main+44
(gdb) c
```



```
ubuntu@ip-172-31-41-188:~/SystemProgramming/ch02$ gdb -q quiz
Reading symbols from quiz...
(gdb) b main
Breakpoint 1 at 0x1195: file quiz.c, line 17.
(gdb) r
Starting program: /home/ubuntu/SystemProgramming/ch02/quiz
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argv=0x7fffffe4a8) at quiz.c:17
17      func = should_call;
(gdb) n
18      func("no way\n");
(gdb) set var func = system
(gdb) disas main
Dump of assembler code for function main:
0x000055555555182 <+0>: endbr64
0x000055555555186 <+4>: push %rbp
0x000055555555187 <+5>: mov %rsp,%rbp
0x00005555555518a <+8>: sub $0x20,%rsp
0x00005555555518e <+12>: mov %edi,-0x14(%rbp)
0x000055555555191 <+15>: mov %rsi,-0x20(%rbp)
0x000055555555195 <+19>: lea -0x39(%rip),%rax # 0x55555555163 <should_call>
0x00005555555519c <+26>: mov %rax,-0x8(%rbp)
=> 0x0000555555551a0 <+30>: mov -0x8(%rbp),%rax
0x0000555555551a4 <+34>: lea 0xe64(%rip),%rdx # 0x5555555560f
0x0000555555551ab <+41>: mov %rdx,%rdi
0x0000555555551ae <+44>: call *%rax
0x0000555555551b0 <+46>: mov $0x0,%eax
0x0000555555551b5 <+51>: leave
0x0000555555551b6 <+52>: ret
End of assembler dump.
(gdb) b *main+44
Breakpoint 2 at 0x555555551ae: file quiz.c, line 18.
(gdb) c
Continuing.

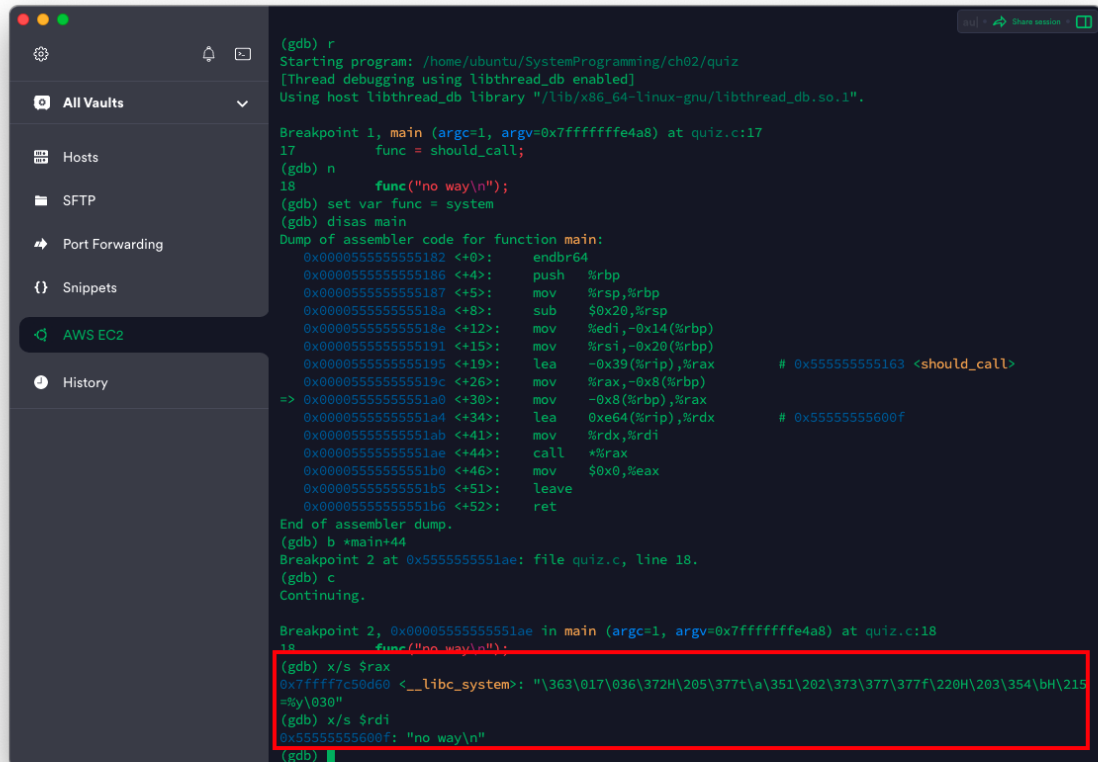
Breakpoint 2, 0x0000555555551ae in main (argv=0x7fffffe4a8) at quiz.c:18
18      func("no way\n");
(gdb) █
```

위 명령어를 실행하면 함수 호출 직전에 브레이크가 걸린 것을 확인할 수 있다.

#### 4) 레지스터 값 확인

이제 레지스터가 담고 있는 값을 확인해본다. 실행할 함수의 주소를 담고 있는 **rax** 와 인자 값을 담고 있는 **rdi** 레지스터의 값을 아래 명령어를 통해 확인한다.

```
(gdb) x/s $rax
(gdb) x/s $rdi
```



```
(gdb) r
Starting program: /home/ubuntu/SystemProgramming/ch02/quiz
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffff4a8) at quiz.c:17
17      func = should_call;
(gdb) n
18      func("no way\n");
(gdb) set var func = system
(gdb) disas main
Dump of assembler code for function main:
0x000055555555182 <+0>:    endbr64
0x000055555555186 <+4>:    push    %rbp
0x000055555555187 <+5>:    mov     %rsp,%rbp
0x00005555555518a <+8>:    sub     $0x20,%rsp
0x00005555555518e <+12>:   mov     %edi,-0x14(%rbp)
0x000055555555191 <+15>:   mov     %rsi,-0x20(%rbp)
0x000055555555195 <+19>:   lea     -0x39(%rip),%rax      # 0x55555555163 <should_call>
0x00005555555519c <+26>:   mov     %rax,-0x8(%rbp)
=> 0x0000555555551a0 <+30>:   mov     -0x8(%rbp),%rax
0x0000555555551a4 <+34>:   lea     0xe64(%rip),%rdx      # 0x55555555600f
0x0000555555551ab <+41>:   mov     %rdx,%rdi
0x0000555555551ae <+44>:   call    *%rax
0x0000555555551b0 <+46>:   mov     $0x0,%eax
0x0000555555551b5 <+51>:   leave
0x0000555555551b6 <+52>:   ret
End of assembler dump.
(gdb) b *main+44
Breakpoint 2 at 0x555555551ae: file quiz.c, line 18.
(gdb) c
Continuing.

Breakpoint 2, 0x0000555555551ae in main (argc=1, argv=0x7fffffff4a8) at quiz.c:18
18      func("no way\n");
(gdb) x/s $rax
0x7ffff7c58d60 <__libc_system>: "\363\017\036\372H\205\377t\351\202\373\377\377f\220H\203\354\bH\215
=%y\030"
(gdb) x/s $rdi
0x55555555600f: "no way\n"
(gdb)
```

**rax** 레지스터에는 위에서 변경한 **system** 함수가 정상적으로 들어가 있는 것을 확인할 수 있고, 이제 아래 명령어를 통해 **rdi** 레지스터의 값을 **"/bin/sh"** 로 변경하면 된다.

```
(gdb) set $rdi = "/bin/sh"
```

```

0x000055555555187 <+5>: mov    %rsp,%rbp
0x00005555555518a <+8>: sub    $0x20,%rsp
0x00005555555518e <+12>: mov    %edi,-0x14(%rbp)
0x000055555555191 <+15>: mov    %rsi,-0x20(%rbp)
0x000055555555195 <+19>: lea    -0x39(%rip),%rax      # 0x55555555163 <should_call>
0x00005555555519c <+26>: mov    %rax,-0x8(%rbp)
=> 0x0000555555551a0 <+30>: mov    -0x8(%rbp),%rax      # 0x55555555600f
0x0000555555551a4 <+34>: lea    0xe64(%rip),%rdx      # 0x55555555600f
0x0000555555551ab <+41>: mov    %rdx,%rdi
0x0000555555551ae <+44>: call   *%rax
0x0000555555551b0 <+46>: mov    $0x0,%eax
0x0000555555551b5 <+51>: leave  %eax
0x0000555555551b6 <+52>: ret

End of assembler dump.
(gdb) b *main+44
Breakpoint 2 at 0x555555551ae: file quiz.c, line 18.
(gdb) c
Continuing.

Breakpoint 2, 0x0000555555551ae in main (argc=1, argv=0x7ffffffe4a8) at quiz.c:18
18      func("no way\n");
(gdb) x/s $rax
0x7ffffffc50d60: <__libc_system>: "\363\017\036\372H\205\377t\a\351\202\373\377f\220H\203\354\bH\215
= %y\030"
(gdb) x/s $rdi
0x55555555600f: "no way\n"
(gdb) set $rdi = "/bin/sh"
(gdb) x/s $rdi
0x5555555592a0: "/bin/sh"
(gdb) c
Continuing.
[Detaching after vfork from child process 3870]
$ ps
  PID TTY          TIME CMD
 3808 pts/0        00:00:00 bash
 3855 pts/0        00:00:00 gdb
 3858 pts/0        00:00:00 quiz
 3870 pts/0        00:00:00 sh
 3871 pts/0        00:00:00 sh
 3872 pts/0        00:00:00 ps

```

위와 같은 방법으로 레지스터에 값이 정상적으로 들어갔는지 확인한 후, 실행해보면 셸이 실행된다. 그리고 프로세스 목록을 확인하는 `ps` 명령어를 입력해보면 셸이 실행되어 있는 것을 확인할 수 있다.

## 5) 종료

정상적으로 셸을 실행했다면, `exit` 를 입력해 셸을 빠져나오고 GDB 역시 빠져나온다. 다시 한 번 `ps` 명령어를 입력해 프로세스가 종료되었는지 확인하면 된다.

## 5. 정리

GDB에서 실행 흐름을 바꿔 셸을 실행하는 과정을 정리하면 다음과 같다.

- `func` 변수에 `system` 함수를 저장하도록 값을 변경한다.
- 레지스터를 확인하고 함수 호출이 진행되기 직전에 BreakPoint를 설정한다.
- 함수의 인자 값을 저장하고 있는 레지스터의 값을 `"/bin/sh"` 로 변경한다.
- 값이 정상적으로 들어갔는지 확인 후 셸을 실행한다.
- 셸을 종료한다.