펭귄 스터디 내용11

작성자 : 류성수

작성일자 : 24.07.03

url : https://www.youtube.com/watch?v=3MRXEVs0_5o&list=PLz—ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=13

● 함수포인터

포인터 변수로 타겟의 주소값을 받아서, 해당 주소값을 참조하여, 타겟에 접근할 수 있었다.
그렇다면, 주소값을 함수의 시작주소로 받으면 어떨까?

1. 파라미터가 없는 함수를 포인터로 받아서 call

```
3       void hello()
4       {
5               printf("Hello, World!\n");
6       }
7
8       void bonjour()
9       {
10              printf("bonjour le monde!\n");
11      }
12
(gdb) list
13      void main()
14      {
15              void (*fp)(); // 반환값과 매개변수가 없는 함수포인터 fp 선언
16
17              fp = hello; // 함수의 첫시작주소 넘기기
18              fp();
19
20              fp = bonjour;
21              fp();
22      }
```

원하는 함수를 fp 함수포인터로 받아서 출력해보았다.
여기서 중요한 점은 함수포인터를 선언할 때, 타겟으로 받을 함수의 함수원형처럼 선언해줘야 하는 것이다.

```
Dump of assembler code for function main:
14      {
   0x000000000000117d <+0>:     endbr64
   0x0000000000001181 <+4>:     push   %rbp
   0x0000000000001182 <+5>:     mov    %rsp,%rbp
   0x0000000000001185 <+8>:     sub    $0x10,%rsp

15              void (*fp)(); // 반환값과 매개변수가 없는 함수포인터 fp 선언
16
17              fp = hello; // 함수의 첫시작주소 넘기기
   0x0000000000001189 <+12>:    lea    -0x47(%rip),%rax        # 0x1149 <hello>
   0x0000000000001190 <+19>:    mov    %rax,-0x8(%rbp)

18              fp();
   0x0000000000001194 <+23>:    mov    -0x8(%rbp),%rdx
   0x0000000000001198 <+27>:    mov    $0x0,%eax
   0x000000000000119d <+32>:    call   *%rdx

19
20              fp = bonjour;
   0x000000000000119f <+34>:    lea    -0x43(%rip),%rax        # 0x1163 <bonjour>
   0x00000000000011a6 <+41>:    mov    %rax,-0x8(%rbp)

21              fp();
   0x00000000000011aa <+45>:    mov    -0x8(%rbp),%rdx
   0x00000000000011ae <+49>:    mov    $0x0,%eax
   0x00000000000011b3 <+54>:    call   *%rdx

22      }
   0x00000000000011b5 <+56>:    nop
   0x00000000000011b6 <+57>:    leave
   0x00000000000011b7 <+58>:    ret
```

main 함수 부분을 어셈블리어로 분해해보았다

여기서 주목할 점은, 바로 주황색 네모이다.

- 0x1189 lea -0x47(%rip), %rax

여기서, 명령어를 다 읽으면, 0x1190 이 되고, 이 주소를 rip 레지스터가 가져간다. (여기서 -0x49를 하면 hello함수의 주소)
이를 rax 레지스터에 저장한다

- 0x1190 mov %rax, -0x8(%rbp)

그리고 해당 rax의 값을 rbp-8(fp의 주소) 에 저장한다.

그리고 초록색 네모를 주목하면,

- 0x1194 mov -0x8(%rbp), %rdx
- 0x1198 mov $0x0, %eax

fp에 저장된 함수주소를 rdx로 옮긴다.
그리고 반환값을 초기화하여 함수호출을 준비한다

- 0x119d call *%rdx

함수의 주소를 드디어 호출한다. 그 다음은 스택프레임의 기존동작과 같다.

```
Dump of assembler code for function hello:
4       {
   0x0000000000001149 <+0>:     endbr64
   0x000000000000114d <+4>:     push   %rbp
   0x000000000000114e <+5>:     mov    %rsp,%rbp

5               printf("Hello, World!\n");
   0x0000000000001151 <+8>:     lea    0xeac(%rip),%rax        # 0x2004
   0x0000000000001158 <+15>:    mov    %rax,%rdi
   0x000000000000115b <+18>:    call   0x1050 <puts@plt>

6       }
   0x0000000000001160 <+23>:    nop
   0x0000000000001161 <+24>:    pop    %rbp
   0x0000000000001162 <+25>:    ret
```

hello 함수를 분해해보았다.
해당 함수의 시작주소 : 0x1149

```
Dump of assembler code for function bonjour:
9       {
   0x0000000000001163 <+0>:     endbr64
   0x0000000000001167 <+4>:     push   %rbp
   0x0000000000001168 <+5>:     mov    %rsp,%rbp

10              printf("bonjour le monde!\n");
   0x000000000000116b <+8>:     lea    0xea0(%rip),%rax        # 0x2012
   0x0000000000001172 <+15>:    mov    %rax,%rdi
   0x0000000000001175 <+18>:    call   0x1050 <puts@plt>

11      }
   0x000000000000117a <+23>:    nop
   0x000000000000117b <+24>:    pop    %rbp
   0x000000000000117c <+25>:    ret
```

bonjour 함수를 분해해보았다.
해당 함수의 시작주소 : 0x1163

2. 그렇다면, 파라미터가 있는 함수를 포인터로 받으면?

```c
#include <stdio.h>

int add(int a, int b)     // int형 반환값, int
{
    return a + b;
}

int mul(int a, int b)     // int형 반환값, int
{
    return a * b;
}

void main()
{
    int (*fp)(int, int);     // int형 반환값,

    fp = add;
    printf("%d\n", fp(10, 20));

    fp = mul;
    printf("%d\n", fp(10, 20));
}
```

두 개의 파라미터를 받고 반환하는 함수를 선언하였다.

```
15          int (*fp)(int, int);     // int형 반환값, int형 매개변수 두 개가 있
16
17          fp = add;
   0x0000000000001184 <+12>:    lea    -0x42(%rip),%rax        # 0x1149 <add>
   0x000000000000118b <+19>:    mov    %rax,-0x8(%rbp)

18          printf("%d\n", fp(10, 20));
   0x0000000000001193 <+27>:    mov    $0x14,%esi
   0x0000000000001198 <+32>:    mov    $0xa,%edi
   0x000000000000119f <+39>:    mov    %eax,%esi
   0x00000000000011a1 <+41>:    lea    0xe5c(%rip),%rax        # 0x2004
   0x00000000000011a8 <+48>:    mov    %rax,%rdi
   0x00000000000011ab <+51>:    mov    $0x0,%eax
   0x00000000000011b0 <+56>:    call   0x1050 <printf@plt>
```

add 함수를 받고 호출하는 부분

주황 네모를 확인해 본다면, 이전과는 다른모습을 볼 수 있다.
파라미터가 함수를 호출할 때, 파라미터 0x14(=20), 0xa(=10) 오른쪽 순서대로 레지스터로 옮겨가는 부분을 확인할 수 있다.
이러한 호출함수의 파라미터를 전달하는 방식은 6개의 파라미터까지만 가능하다.

3. 7개의 파라미터는?

```c
#include <stdio.h>

int test(char p1, char p2, char p3, char p4, char p5, char p6, char p7)
{
        return p1 + p2 +p3 +p4+ p5+ p6+p7;
}

void main()
{
        int (*fp)(char,char,char,char,char,char,char);

        fp = test;

        printf("%d\n",fp(1,2,3,4,5,6,7));
}
```

7개의 1바이트 변수를 7개받아서 그 합을 반환하는 함수 test

```
14              printf("%d\n",fp(1,2,3,4,5,6,7));
   0x00000000000011bc <+23>:    sub    $0x8,%rsp
   0x00000000000011c0 <+27>:    push   $0x7
   0x00000000000011c2 <+29>:    mov    -0x8(%rbp),%rax
   0x00000000000011c6 <+33>:    mov    $0x6,%r9d
   0x00000000000011cc <+39>:    mov    $0x5,%r8d
   0x00000000000011d2 <+45>:    mov    $0x4,%ecx
   0x00000000000011d7 <+50>:    mov    $0x3,%edx
   0x00000000000011dc <+55>:    mov    $0x2,%esi
   0x00000000000011e1 <+60>:    mov    $0x1,%edi
   0x00000000000011e6 <+65>:    call   *%rax
```

main 쪽을 분해했을시, 오른쪽부터 읽기 때문에, 맨 오른쪽 p7만 test 함수의 스택프레임에 푸시되고, 나머지 6개는 레지스터에 담기는걸 확인할 수 있다.