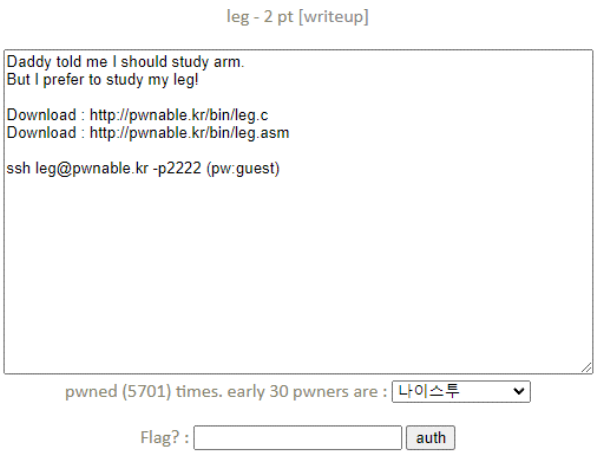
Pwnable.kr
- leg -

Download : http://pwnable.kr/bin/leg.c
Download : http://pwnable.kr/bin/leg.asm
ssh leg@pwnable.kr -p2222
pw : guest

0. 문제 살펴보기



문제에서 팔과 다리에 대한 얘기를 하고 있다. 무슨 말인지 잘 모르겠으니 일단 원격 접속을 해보겠다.

1. SSH 접속 및 살펴보기

수많은 오류 문구와 함께 쉘이 실행되었다.

일단 쉘이 실행되었으므로 디렉토리의 파일들을 살펴보자.
우리가 평소에 보던 계정 디렉토리도 아니고 쉘도 다른 느낌이다.

호기심으로 확인해본 결과 /bin/sh이 실행되고 있으며 현재 디렉토리는 /였다.

2. 파일 다운로드

이제 다른 터미널을 이용하여 링크로 주어진 파일들을 다운로드 해볼 것이다.

```
┌──(kali㉿kali)-[~]
└─$ wget http://pwnable.kr/bin/leg.c
--2024-02-04 11:14:17--  http://pwnable.kr/bin/leg.c
Resolving pwnable.kr (pwnable.kr)... 128.61.240.205
Connecting to pwnable.kr (pwnable.kr)|128.61.240.205|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 600 [text/x-csrc]
Saving to: 'leg.c'

leg.c                               100%[═══════════════════════════════════════════════════════════════════════>]     600  --.-KB/s    in 0s

2024-02-04 11:14:18 (59.4 MB/s) - 'leg.c' saved [600/600]
```

```
┌──(kali㉿kali)-[~]
└─$ wget http://pwnable.kr/bin/leg.asm
--2024-02-04 11:14:30--  http://pwnable.kr/bin/leg.asm
Resolving pwnable.kr (pwnable.kr)... 128.61.240.205
Connecting to pwnable.kr (pwnable.kr)|128.61.240.205|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3035 (3.0K)
Saving to: 'leg.asm'

leg.asm                             100%[═══════════════════════════════════════════════════════════════════════>]   2.96K  --.-KB/s    in 0s

2024-02-04 11:14:31 (333 MB/s) - 'leg.asm' saved [3035/3035]
```

하나는 c 코드 파일이었으며, 하나는 어셈블리 파일이었다.

3. leg.c 확인

```c
#include <stdio.h>
#include <fcntl.h>
int key1(){
        asm("mov r3, pc\n");
}
int key2(){
        asm(
        "push   {r6}\n"
        "add    r6, pc, $1\n"
        "bx     r6\n"
        ".code   16\n"
        "mov    r3, pc\n"
        "add    r3, $0x4\n"
        "push   {r3}\n"
        "pop    {pc}\n"
        ".code  32\n"
        "pop    {r6}\n"
        );
}
int key3(){
        asm("mov r3, lr\n");
}
int main(){
        int key=0;
        printf("Daddy has very strong arm! : ");
        scanf("%d", &key);
        if( (key1()+key2()+key3()) == key ){
                printf("Congratz!\n");
                int fd = open("flag", O_RDONLY);
                char buf[100];
                int r = read(fd, buf, 100);
                write(0, buf, r);
        }
        else{
                printf("I have strong leg :P\n");
        }
        return 0;
}
```

leg.c 파일을 확인하자 key 라는 이름을 가진 함수들이 어셈블리어를 가지고 있는 것을 볼 수 있었다.
main 함수에서는 이 어셈블리어들의 결과가 우리가 입력한 key 값과 같을 때, flag 파일을 볼 수 있는 것 같다.

4. leg.asm 확인

```
┌──(kali㉿kali)-[~]
└─$ cat leg.asm
(gdb) disass main
Dump of assembler code for function main:
   0x00008d3c <+0>:     push    {r4, r11, lr}
   0x00008d40 <+4>:     add     r11, sp, #8
   0x00008d44 <+8>:     sub     sp, sp, #12
   0x00008d48 <+12>:    mov     r3, #0
   0x00008d4c <+16>:    str     r3, [r11, #-16]
   0x00008d50 <+20>:    ldr     r0, [pc, #104]  ; 0x8dc0 <main+132>
   0x00008d54 <+24>:    bl      0xfb6c <printf>
   0x00008d58 <+28>:    sub     r3, r11, #16
   0x00008d5c <+32>:    ldr     r0, [pc, #96]   ; 0x8dc4 <main+136>
   0x00008d60 <+36>:    mov     r1, r3
   0x00008d64 <+40>:    bl      0xfbd8 <__isoc99_scanf>
   0x00008d68 <+44>:    bl      0x8cd4 <key1>
   0x00008d6c <+48>:    mov     r4, r0
   0x00008d70 <+52>:    bl      0x8cf0 <key2>
   0x00008d74 <+56>:    mov     r3, r0
   0x00008d78 <+60>:    add     r4, r4, r3
   0x00008d7c <+64>:    bl      0x8d20 <key3>
   0x00008d80 <+68>:    mov     r3, r0
   0x00008d84 <+72>:    add     r2, r4, r3
   0x00008d88 <+76>:    ldr     r3, [r11, #-16]
   0x00008d8c <+80>:    cmp     r2, r3
   0x00008d90 <+84>:    bne     0x8da8 <main+108>
   0x00008d94 <+88>:    ldr     r0, [pc, #44]   ; 0x8dc8 <main+140>
   0x00008d98 <+92>:    bl      0x1050c <puts>
   0x00008d9c <+96>:    ldr     r0, [pc, #40]   ; 0x8dcc <main+144>
   0x00008da0 <+100>:   bl      0xf89c <system>
   0x00008da4 <+104>:   b       0x8db0 <main+116>
   0x00008da8 <+108>:   ldr     r0, [pc, #32]   ; 0x8dd0 <main+148>
   0x00008dac <+112>:   bl      0x1050c <puts>
   0x00008db0 <+116>:   mov     r3, #0
   0x00008db4 <+120>:   mov     r0, r3
   0x00008db8 <+124>:   sub     sp, r11, #8
   0x00008dbc <+128>:   pop     {r4, r11, pc}
   0x00008dc0 <+132>:   andeq   r10, r6, r12, lsl #9
   0x00008dd0 <+148>:   andeq   r10, r6, r4, asr #9
End of assembler dump.
```

```
(gdb) disass key1
Dump of assembler code for function key1:
   0x00008cd4 <+0>:     push    {r11}           ; (str r11, [sp, #-4]!)
   0x00008cd8 <+4>:     add     r11, sp, #0
   0x00008cdc <+8>:     mov     r3, pc
   0x00008ce0 <+12>:    mov     r0, r3
   0x00008ce4 <+16>:    sub     sp, r11, #0
   0x00008ce8 <+20>:    pop     {r11}           ; (ldr r11, [sp], #4)
   0x00008cec <+24>:    bx      lr
End of assembler dump.
```

```
(gdb) disass key2
Dump of assembler code for function key2:
   0x00008cf0 <+0>:     push    {r11}           ; (str r11, [sp, #-4]!)
   0x00008cf4 <+4>:     add     r11, sp, #0
   0x00008cf8 <+8>:     push    {r6}            ; (str r6, [sp, #-4]!)
   0x00008cfc <+12>:    add     r6, pc, #1
   0x00008d00 <+16>:    bx      r6
   0x00008d04 <+20>:    mov     r3, pc
   0x00008d06 <+22>:    adds    r3, #4
   0x00008d08 <+24>:    push    {r3}
   0x00008d0a <+26>:    pop     {pc}
   0x00008d0c <+28>:    pop     {r6}            ; (ldr r6, [sp], #4)
   0x00008d10 <+32>:    mov     r0, r3
   0x00008d14 <+36>:    sub     sp, r11, #0
   0x00008d18 <+40>:    pop     {r11}           ; (ldr r11, [sp], #4)
   0x00008d1c <+44>:    bx      lr
End of assembler dump.
```

```
(gdb) disass key3
Dump of assembler code for function key3:
   0x00008d20 <+0>:     push    {r11}           ; (str r11, [sp, #-4]!)
   0x00008d24 <+4>:     add     r11, sp, #0
   0x00008d28 <+8>:     mov     r3, lr
   0x00008d2c <+12>:    mov     r0, r3
   0x00008d30 <+16>:    sub     sp, r11, #0
   0x00008d34 <+20>:    pop     {r11}           ; (ldr r11, [sp], #4)
   0x00008d38 <+24>:    bx      lr
End of assembler dump.
```

leg.asm 파일을 살펴보자 각 함수에 대한 어셈블리어가 등장하였다.

자세히 살펴보면 main 함수에서 key 값과 비교할 때, key1(), key2(), key3()의 리턴 값이 모두 r0 이라는 것을 알 수 있다. 이후, key1()+key2()+key3()의 값은 r2 에 저장되고, 우리가 입력한 key 값은 r11 에 저장되어 있다가 r3 에 저장되는 것을 볼 수 있다.
그런 다음 r2 와 r3 를 비교하게 된다.

그렇다면 우리는 각 함수에서 r0 의 값만 알아내면 된다.

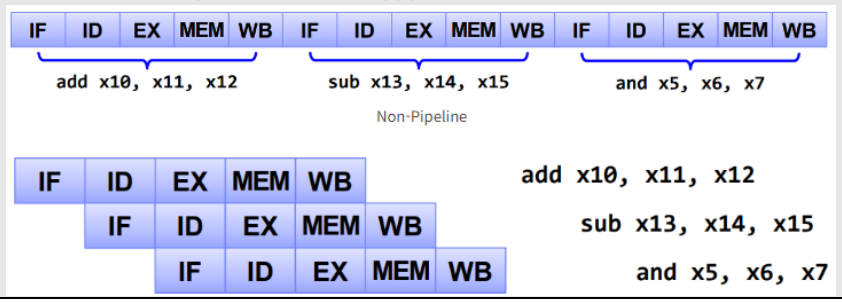
5. 값 구하기

1) key1

```
0x00008cdc <+8>:     mov     r3, pc
0x00008ce0 <+12>:    mov     r0, r3
0x00008ce4 <+16>:    sub     sp, r11, #0
```

pc의 값을 r3에 저장하고, 다시 r3의 값을 r0에 저장하는 것을 볼 수 있다.
pc는 다음 실행할 명령의 주소를 가지고 있다.
0x00008cdc에서의 pc 값은 다음 명령인 0x00008ce0이다.
하지만, 여기서 하나 알아야 할 것이 있다. 바로, 파이프라인이다.

※ 명령어 파이프라인 (instruction pipeline)
명령어를 읽어 순차적으로 실행하는 프로세서에 적용되는 기술이다.
한 번에 하나의 명령어만 실행하는 것이 아니라 하나의 명령어가 실행되는 도중에 다른 명령어 실행을 시작하는 방식으로 동시에 여러 개의 명령어를 실행하는 기법이다.



이 방법에 의하여 0x00008cdc에서의 pc 값은 다음 줄인 0x00008ce0인 것이 맞지만
0x00008cdc가 실행되고 있다면, pc 값은 바로 다음 줄로 넘어가 0x00008ce4가 된다는 것이다.
현재 <+8> 라인이 실행되어 r3에 pc의 값을 넣으려고 한다. 이 때, 명령이 이미 실행된 상태이기 때문에 다음 명령의 주소인 <+12>은 파이프라인에 저장하고, pc의 값은 그 다음 줄인 <+16>을 가리키는 것이다.

따라서, 위의 어셈블리어가 실행되고 난 후에 r0의 값은 <+16>의 주소인 0x00008ce4이다.

2) key2

```
0x00008d04 <+20>:    mov     r3, pc
0x00008d06 <+22>:    adds    r3, #4
0x00008d08 <+24>:    push    {r3}
0x00008d0a <+26>:    pop     {pc}
0x00008d0c <+28>:    pop     {r6}
0x00008d10 <+32>:    mov     r0, r3
```

이 함수에서는 pc의 값을 r3에 저장한 후에 4를 더하고, 다시 r0에 저장하고 있다.
이 역시도 파이프라인에 의하여 r3에 저장되는 pc의 값은 <+22>가 아닌 <+24>의 주소인 0x00008d08이다.
여기에 4를 더하므로 그 값은 0x00008d0c이다.

3) key3

```
0x00008d28 <+8>:     mov     r3, lr
0x00008d2c <+12>:    mov     r0, r3
```

이 함수에서는 lr을 r3에 저장한 후에 r3를 다시 r0에 저장하고 있다.
lr은 이 함수가 끝나고 다시 돌아갈 곳에 대한 주소이다.
pc가 이동할 명령의 주소라고 생각하면 된다.

```
0x00008d7c <+64>:    bl      0x8d20 <key3>
0x00008d80 <+68>:    mov     r3, r0
0x00008d84 <+72>:    add     r2, r4, r3
```

main 함수에서 그 위치를 찾아보면 key3 함수가 끝난 후에 갈 위치이므로 그 다음 줄인 <+68>의 주소인 0x00008d80이다.

4) 합

위에서 구한 모든 값을 더하면
0x8ce4 + 0x8d0c + 0x8d80 = 0x0001a770
이므로 이를 10진수로 변환하면 108,400이라는 수가 나오게 된다.

6. 공격

```
/ $ ./leg
Daddy has very strong arm! : 108400
Congratz!
```

다시 우리가 원격 접속했던 터미널로 돌아가 leg 프로그램을 실행시킨다.
우리가 얻은 수인 108400을 입력으로 넣으면 flag가 나오게 된다.