## 1.漏洞简介

漏洞描述：在Nginx的range filter中存在整数溢出漏洞，可以通过带有特殊构造的range的HTTP头的恶意请求引发这个整数溢出漏洞，并导致信息泄露。

风险等级：低

受影响范围/版本：Nginx 0.5.6 – 1.13.2

造成的影响：泄露简单信息

## 2.漏洞成因分析

前置知识：

http断点续传：

也就是要从文件已经下载的地方开始继续下载。在以前版本的 HTTP 协议是不支持断点的，HTTP/1.1 开始就支持了。一般断点下载时才用到 Range 和 Content-Range 实体头。Range

用于请求头中，指定第一个字节的位置和最后一个字节的位置，一般格式：

Range:(unit=first byte pos)-[last byte pos]

Content-Range

用于响应头，指定整个实体中的一部分的插入位置，他也指示了整个实体的长度。在服务器向客户返回一个部分响应，它必须描述响应覆盖的范围和整个实体长度。一般格式：

Content-Range: bytes (unit first byte pos) - [last byte pos]/[entity legth]

请求下载整个文件:

GET /test.rar HTTP/1.1

Connection: close

Host: 116.1.219.219

Range: bytes=0-801 //一般请求下载整个文件是bytes=0- 或不用这个头

一般正常回应

HTTP/1.1 200 OK

Content-Length: 801

Content-Type: application/octet-stream

Content-Range: bytes 0-800/801 //801:文件总大小

Nginx Range Multipart：

如果一次请求有多个range，返回的数据需要multipart来组织；格式如下：

HTTP/1.1 206 Partial Content

Date: Wed, 15 Nov 1995 06:25:24 GMT

Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT

Content-type: multipart/byteranges; boundary=THIS\_STRING\_SEPARATES

--THIS\_STRING\_SEPARATES

Content-type: application/pdf

Content-range: bytes 500-999/8000

...the first range...

--THIS\_STRING\_SEPARATES

Content-type: application/pdf

Content-range: bytes 7000-7999/8000

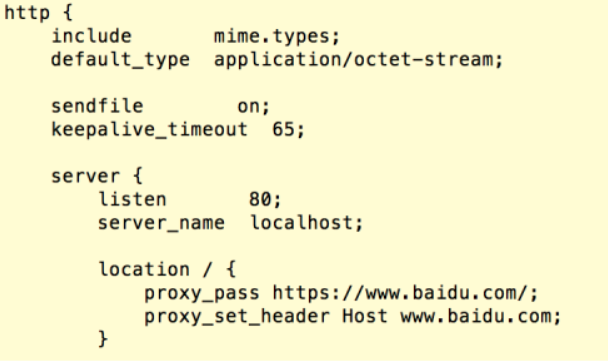
...the second range

--THIS\_STRING\_SEPARATES--

Nginx Cache：

Nginx可以作为缓存服务器，将Web应用服务器返回的内容缓存起来。如果客户端请求的内容已经被缓存，那么就可以直接将缓存内容返回，而无需再次请求应用服务器。由此，可降低应用服务器的负载，并提高服务的响应性能。

下面是使用Nginx作为缓存服务器的一个示例。假设Nginx监听本地80端口，反向代理百度，那么就有如下配置：



此时，我们访问[http://127.0.0.1](http://127.0.0.1/)，即可得到百度的返回：



检查页面资源，存在一个静态图片文件http://www.baidu.com/img/bd\_logo1.png。由于这类静态文件一般不会发生变化，我们可以将其缓存起来。

Nginx配置缓存主要由以下命令完成：

proxy\_cache\_key用于区分cache文件。

proxy\_cache\_path设置cache文件的路径和参数。

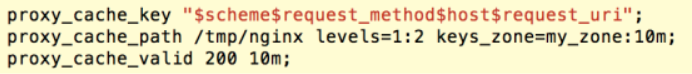
cache文件会保存在指定的目录下面，文件名是cache key的MD5值

通过level参数将cache文件分多层目录保存，以避免某个目录下存在大量文件造成的性能开销

通过keys\_zone参数指定cache key在内存中的区域及其大小，1M的区域大概可以保存8000条key的信息

proxy\_cache\_valid对不同返回状态值设定cache有效时间

例如，下面这条配置：



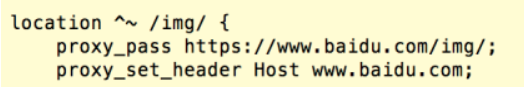
指定了以下信息：

使用协议、请求方法、域名、URI作为cache key

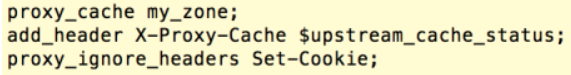
cache文件保存在目录/tmp/Nginx/下，采取两层目录，keys\_zone名称为my\_zone，大小为10

对于返回状态值为200的内容，cache有效时间为10分钟

现在，我们配置好了名为my\_zone的cache，接下来选择对目录www.baidu.com/img/下的图片做缓存。首先，仍然是设置反向代理：



接下来，我们使用下列命令对img目录下的文件进行缓存：



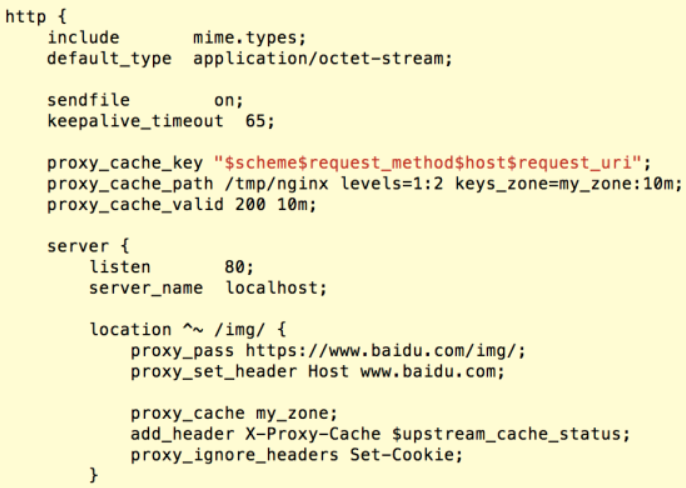
配置命令解释如下：

proxy\_cache指定使用的keys\_zone名称，就是之前的my\_zone

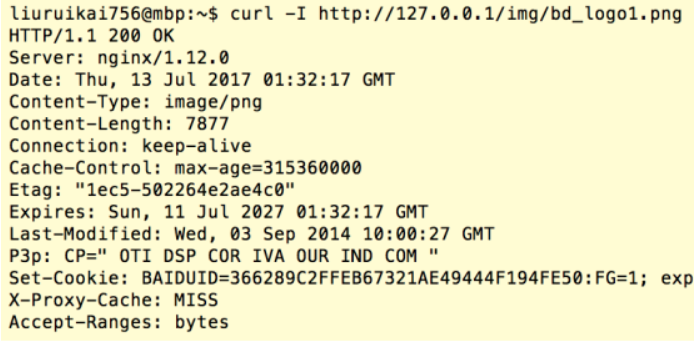
add\_header在Nginx返回的HTTP头中，增加一项X-Proxy-Cache，如果缓存命中其值为HIT，未命中则为MISS

proxy\_ignore\_headers由于百度对图片的请求也会Set-Cookie设置，而Nginx不会缓存带有Set-Cookie的返回，因此我们这里设置忽略该HTTP头

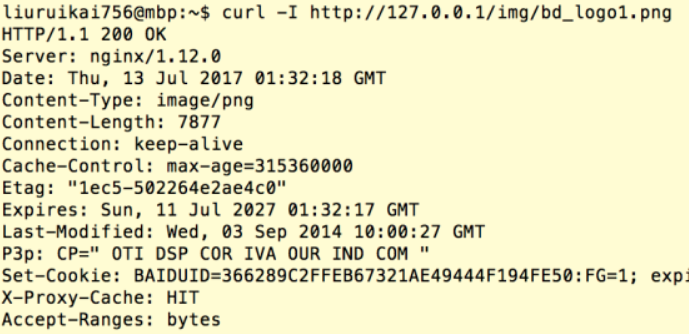
现在，对图片的缓存配置就完成了，完整的配置内容如下



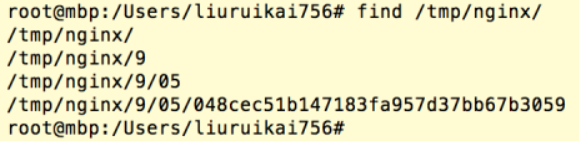
我们使用curl命令进行实验，访问http://127.0.0.1/img/bd\_logo1.png。由于是第一次访问，可以看到返回内容中X-Proxy-Cache的值为MISS：



再次访问时，此时缓存命中，X-Proxy-Cache的值为HIT了：



那么现在的Cache文件是什么样的呢？我们检查设置的缓存目录/tmp/Nginx，发现存在以下Cache文件：



可见，确实使用了2层目录保存了Cache文件。Cache文件保存了Nginx请求得到的返回内容：



可以看到，cache key的内容保存在了Cache文件的头部，此外还有Nginx请求后端返回的HTTP头，如后端（这里是www.baidu.com）的服务器为Apache。正常情况下，这些信息是不会返回给客户端的。而本次的的漏洞，就是由于负数偏移量，导致Cache文件的头部信息也被返回，从而造成信息泄漏。

## 3.漏洞复现

### 3.1 复现环境

本漏洞使用vulnub-master开源项目下的漏洞集成环境测试，搭建在ubuntu16，04上

需要：python3 pip3 docker.io docker-compose

POC程序来源地址：<https://github.com/vulhub/vulhub/tree/master/nginx/CVE-2017-7529>

### 3.1 复现过程

配置好vulhub集成环境:

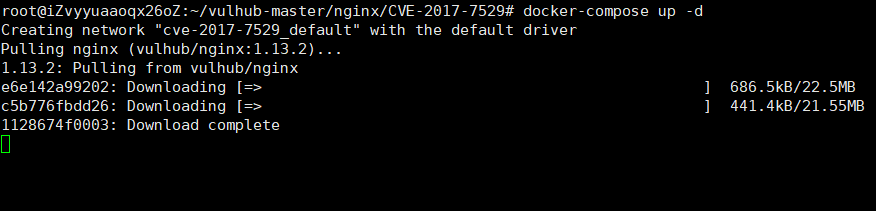
wget https://github.com/vulhub/vulhub/archive/master.zip -O vulhub-master.zip

unzip vulhub-master.zip

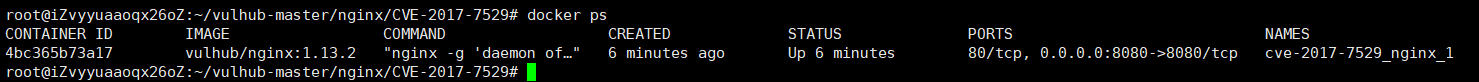
cd vulhub-master



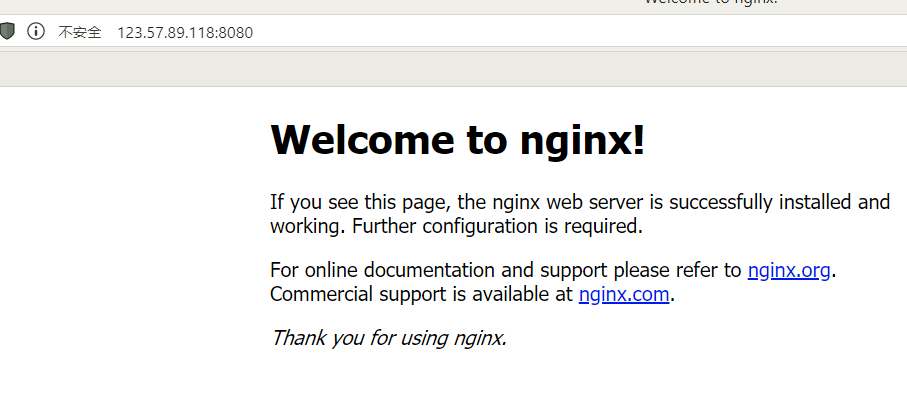
进入到Nginx的文件夹执行docker-compose up -d 自动下载并启动环境：



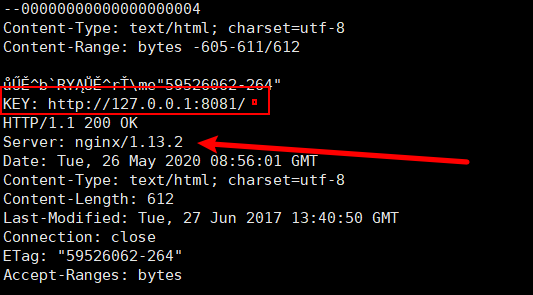
环境创建成功:



访问<http://123.57.89.118:8080>成功：



使用POC查看返回消息：



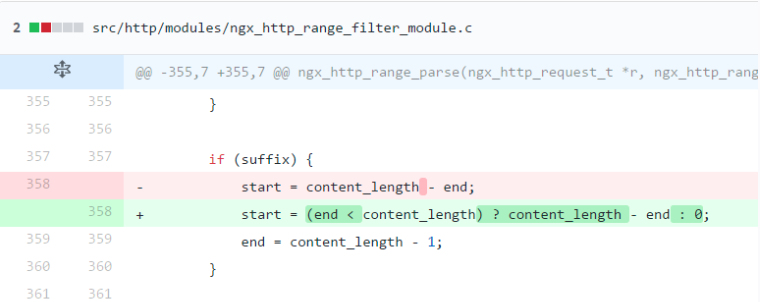
可以看到敏感信息。

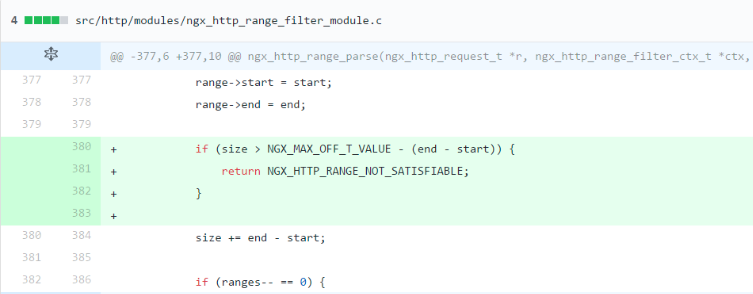
## 4.漏洞利用分析

Nginx在反向代理站点的时候，通常会将一些文件进行缓存，特别是静态文件。缓存的部分存储在文件中，每个缓存文件包括“文件头”+“HTTP返回包头”+“HTTP返回包体”。如果二次请求命中了该缓存文件，则Nginx会直接将该文件中的“HTTP返回包体”返回给用户。

如果我的请求中包含Range头，Nginx将会根据我指定的start和end位置，返回指定长度的内容。而如果我构造了两个负的位置，如(-600, -9223372036854774591)，将可能读取到负位置的数据。如果这次请求又命中了缓存文件，则可能就可以读取到缓存文件中位于“HTTP返回包体”前的“文件头”、“HTTP返回包头”等内容。

首先，我们看这次漏洞修复的commit:





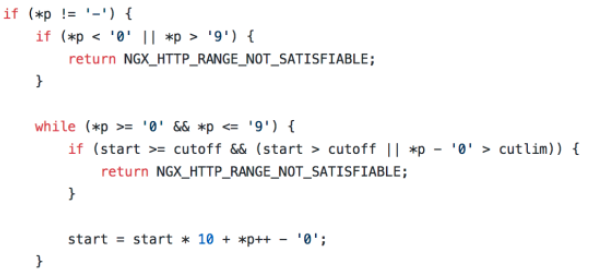
可以看到，在ngx\_http\_range\_filter\_module.c的ngx\_http\_range\_parse函数中做了两处修复：

进一步检测了size的溢出情况，防止size溢出后造成小于content-length这条判断的绕过

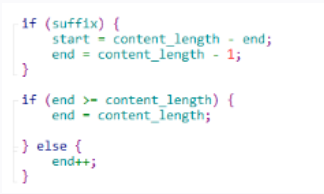
则直接限定了使用后缀的情况下，start不能为负的，最小只能是0，也就是说使用“-xxx”的方式对Cache文件的读取只能从0开始到结束。

根据漏洞修复commit的注释，我们知道这次漏洞的主要成因就是bytes-range读取的起始范围可能为负数，从而读取缓存文件头部。

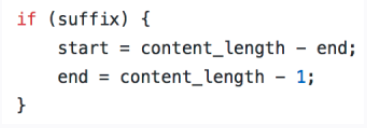
首先，如果传入完整的range参数，如start-end，则在ngx\_http\_range\_parse()中会检查start，确保其不会溢出为负值：



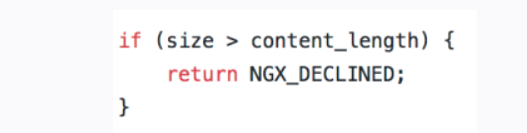
因此，如果需要将start解析为负数，只能通过-end这类后缀型range参数实现:



此时的start等于content-length减去读入的end值，所以如果传入的end比实际长度还要长，就可以使start变为负数，而这就是第二处修复所处理的情形：

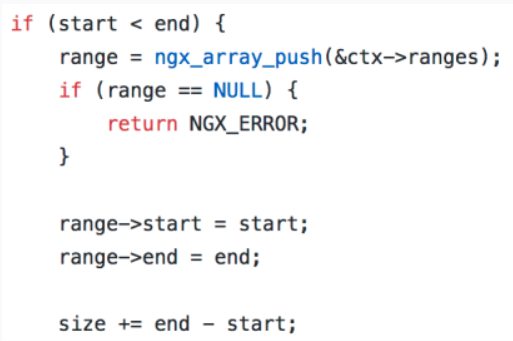


同时注意到，在这类情况下，最终end的值会被设定为content-length-1。所以这块range的总长度就超过了content-length。而Nginx对range总长度会有检查：



一般来说，range作为原始文件的一部分，其长度应该是小于content-length的。所以一旦计算得到的size比content-length还大，那就直接将原始文件返回了，不再进行range处理。为了绕过这一限制，我们就需要利用到第一处修复所处理的情形。

具体而言，检查用到的size是将multipart的全部range长度相加得到的：



因此，一个range是不够的，我们至少需要两个range，其长度之和溢出为负数，就可以绕过总长度的检查了。

要得到一个很大长度的range，同样可以采用-end这种后缀型，将end设置为一个非常大的数即可。此处的start, end, size均为64位有符号整形，所以只需要最终相加得到的size为0×8000000000000000即可。

利用：首先不指定range得到图片长度为a，然后利用a – x = -635,此处x为第一个rang，即图片在Cache文件偏移之前的623 bytes也会被返回，而这623 bytes中就包含了Cache文件头部。然后因为两个rang加起来需要的size等于0×8000000000000000，所以第二个range y=0×8000000000000000 – X。此时 CURL -I ip:port -r x,y 即可成功利用。

## 5.漏洞检测方法

（1）检查Nginx版本：Nginx 0.5.6 – 1.13.2

（2）通过POC验证

## 6.漏洞修复方案

规避方法：在Nginx配置文件中添加max\_ranges 1，从而禁用multipart range。

修复方法：升级到1.13.3和1.12.1版本

## F&Q

1.注意文件的大小，以便进行计算。

## 参考文献

<https://www.freebuf.com/articles/terminal/140402.html>

<https://blog.csdn.net/qq_27103091/article/details/75009846>

<https://blog.csdn.net/weixin_34315189/article/details/90356971>

<https://paper.seebug.org/353/>

<https://www.shungg.cn/post/173>