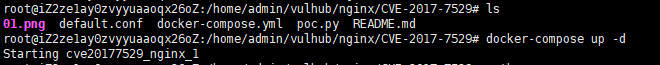
## 复现

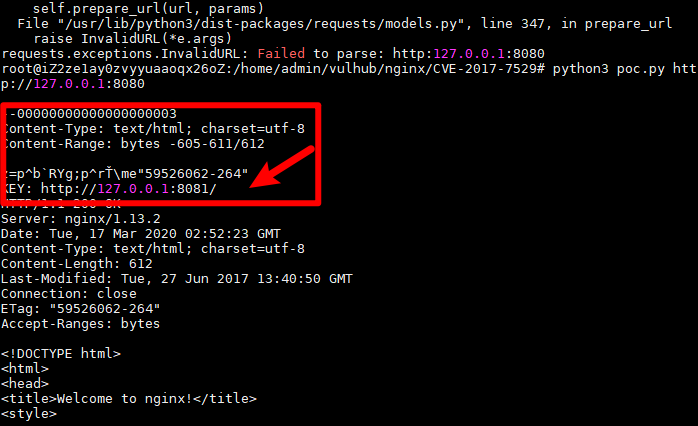
首先使用docker-composer up –d来启动vulhub下的漏洞复现环境;



然后访问localhost:8080来访问：



利用poc进行复现：执行python3 poc.py http://123.57.89.118:8080/读取返回结果：



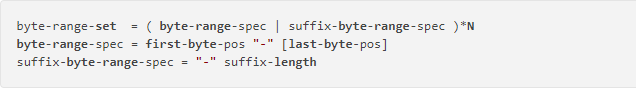
可见，越界读取到了位于“HTTP返回包体”前的“文件头”、“HTTP返回包头”等内容。

## 分析

### HTTP断点续传：Range

HTTP的Range允许客户端分批次请求资源的一部分，如果服务端资源较大，可以通过Range来并发下载；如果访问资源时网络中断，可以断点续传。

Range设置在HTTP请求头中，它是多个byte-range-spec(或suffix-byte-range-spec)的集合：



其中，first-bytes-pos指定了访问的第一个字节，last-byte-pos指定了最后一个字节，suffix-length则表示要访问资源的最后suffix-length个字节的内容；例如：

Range:bytes=0-1024 表示访问第0到第1024字节；

Range:bytes=500-600,601-999，-300 表示分三块访问，分别是500到600字节，601到600字节，最后的300字节；

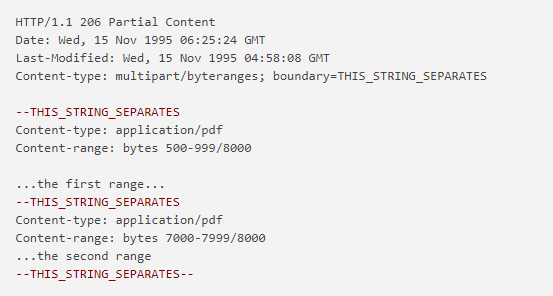
在Response头中设置:

Accept-Ranges:bytes 表示接受部分资源的请求；

Content-Range: bytes START-END/SIZE 表示返回的资源位置；其中SIZE等于Content-Length;如：Content-Range: bytes 500-600/1000

### 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：



Nginx对Range的支持包括header处理和body处理，分别用来解析客户端发送过来的Range header和裁剪返回给客户端的请求数据Body。其实现分别由ngx\_http\_range\_header\_filter\_module和ngx\_http\_range\_body\_filter\_module两个过滤模块完成。

在ngx\_http\_range\_header\_filter\_module中调用了ngx\_http\_range\_header\_filter函数，而该函数进一步调用了ngx\_http\_range\_parse函数来解析header中的Range字段；分别调用ngx\_http\_range\_singlepart\_header和ngx\_http\_range\_multipart\_header来生成single range和multi ranges的Response Header；

这次的问题就出现在多个range时，ngx\_http\_range\_parse函数对suffix-length的处理。

### 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，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：



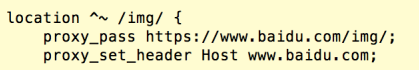
指定了以下信息：

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10M

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：



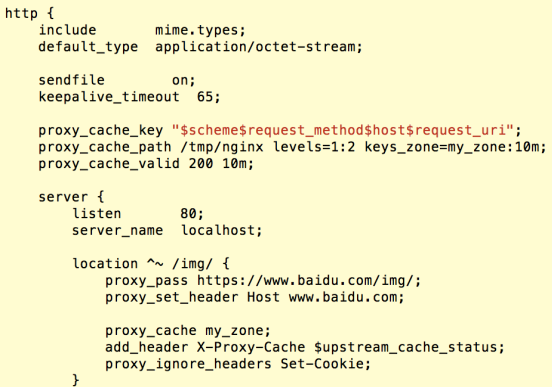
配置命令解释如下：

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

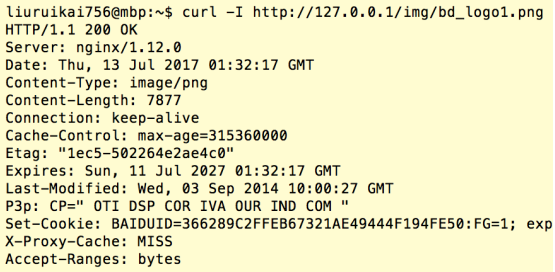
add\_header在Nginx返回的HTTP头中，增加一项X-Proxy-Cache，如果缓存命中其值为HIT，未命中则为MISS

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

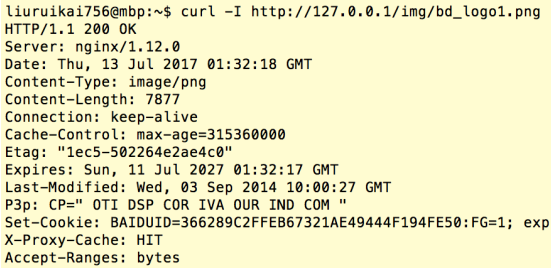
现在，对图片的缓存配置就完成了，完整的配置内容如下



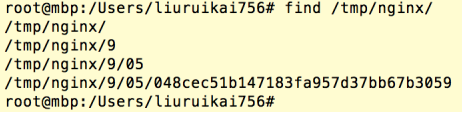
我们使用curl命令进行实验，访问http://127.0.0.1/img/bd\_logo1.png。由于是第一次访问，可以看到返回内容中X-Proxy-Cache的值为MISS：



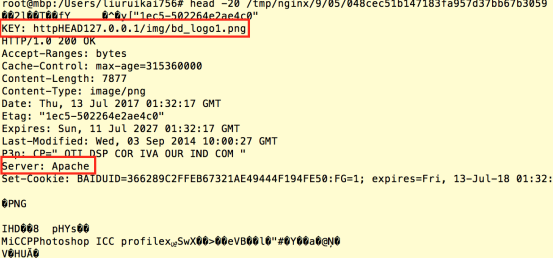
再次访问时，此时缓存命中，X-Proxy-Cache的值为HIT了：



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：



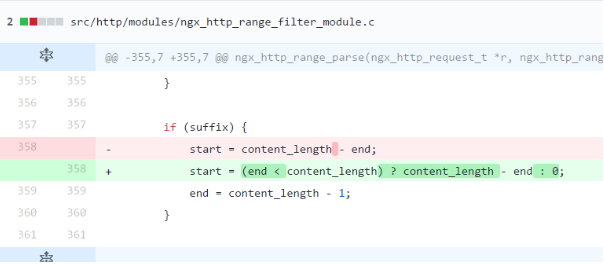
可见，确实使用了2层目录保存了Cache文件。Cache文件保存了Nginx请求得到的返回内容：

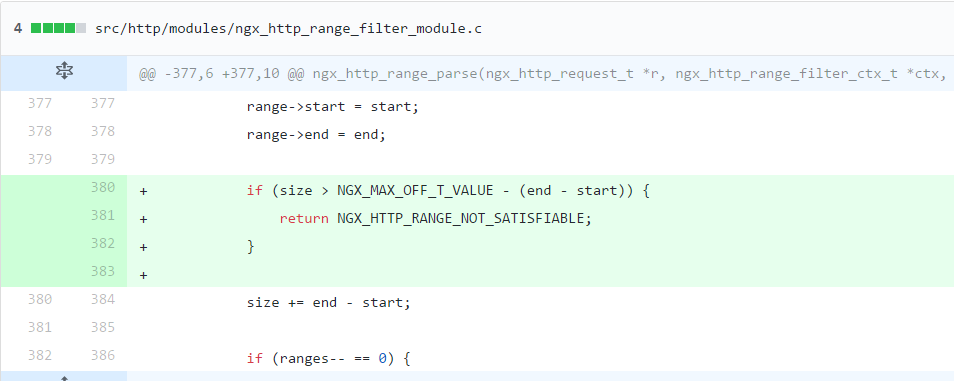


可以看到，cache key的内容保存在了Cache文件的头部，此外还有Nginx请求后端返回的HTTP头，如后端（这里是www.baidu.com）的服务器为Apache。正常情况下，这些信息是不会返回给客户端的。而本次的的漏洞，就是由于负数偏移量，导致Cache文件的头部信息也被返回，从而造成信息泄漏。

### 漏洞原理

首先，我们看这次漏洞修复的commit:





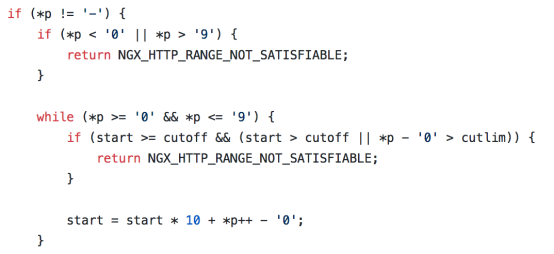
可以看到，在ngx\_http\_range\_filter\_module.c的ngx\_http\_range\_parse函数中做了两处修复：

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

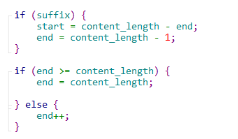
则直接限定了使用后缀的情况下，start不能为负的，最小只能是0，也就是说使用“-xxx”的方式对Cache文件的读取只能从0开始到结束。

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。

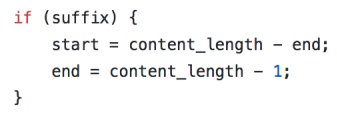
首先，如果传入完整的range参数，如start-end，则在ngx\_http\_range\_parse()中会检查start，确保其不会溢出为负值：



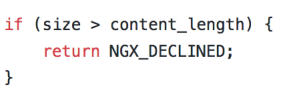
因此，如果需要将start解析为负数，只能通过-end这类后缀型range参数实现:



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：

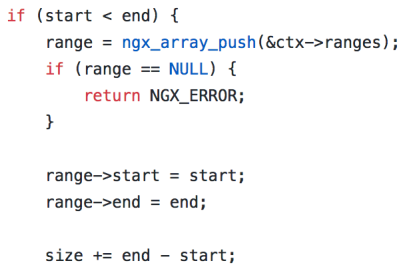


同时注意到，在这类情况下，最终end的值会被设定为content-length-1。所以这块range的总长度就超过了content-length。而Nginx对range总长度会有检查：



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。

注：0×8000000000000000为第二段溢出值，第一段查看长度之后，选择一个大的长度指定为end来查看前n个字节的值，为了使整个长度溢出，第二段end则应指定为0×8000000000000000-n ，此时即可绕过总的长度检查。

## 漏洞修复

综合来看，这个漏洞就是整数溢出漏洞的利用，能够从Cache文件中获取Cache头的信息。在某些配置的情况下Cache头中会存在IP地址信息，造成信息泄露。

就Nginx模块以及常用的第三方模块本身来说，无法通过这个整数溢出来对内存进行操作或者远程执行。

建议升级到1.13.3和1.12.1版本；如果不能升级，可以在Nginx配置文件中添加

max\_ranges 1，从而禁用multipart range。