




 xiongwilee / blog


 Watch3


 Star7

 Fork1

 Code

 Issues9

 Pull requests0

 Projects0

Insights ▾

Dismiss

Join GitHub today

GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.

Sign up

前后端分离架构下CSRF防护机制 #7

New issue

Open xiongwilee opened this issue on Apr 13 · 2 comments



xiongwilee commented on Apr 13 • edited

Owner

### 背景

#### 1、什么是CSRF攻击？

这里不再介绍CSRF，已经了解CSRF原理的同学可以直接跳到：“3、前后端分离下有何不同？”。

不太了解的同学可以看这两篇对CSRF介绍比较详细的参考文章：

- CSRF 攻击的应对之道
- 浅谈CSRF攻击方式

如果来不及了解CSRF的原理，可以这么理解：有一个人发给你一个搞(me)笑(nv)图片链接，你打开这个链接之后，便立刻收到了短信：你的银行里的钱已经转移到这个人的帐户了。

#### 2、有哪些防御方案？

上面这个例子当然有点危言耸听，当然可以确定的是确实会有这样的漏洞：你打开了一个未知域名的链接，然后你就自动发了条广告帖子、你的Gmail的邮件内容就泄露了、你的百度登录状态就没了.....

防御方案在上面的两篇文章里也有提到，总结下，无外乎三种：

- 用户操作限制，比如验证码；
- 请求来源限制，比如限制HTTP Referer才能完成操作；
- token验证机制，比如请求数据字段中添加一个token，响应请求时校验其有效性；

第一种方案明显严重影响了用户体验，而且还有额外的开发成本；第二种方案成本最低，但是并不能保证100%安全，而且很有可能会埋坑；第三种方案，可取！

token验证的CSRF防御机制是公认最合适的方案，也是本文讨论的重点。

#### 3、前后端分离下有何不同？

《CSRF 攻击的应对之道》这篇文章里有提到：

要把所有请求都改为 XMLHttpRequest 请求，这样几乎是要重写整个网站，这代价无疑是不能接受的

我们前端架构早已经告别了服务端语言（PHP/JAVA等）绑定路由、携带数据渲染模板引擎的方式（毕竟是2011年的文章了，我们笑而不语）。

Assignees

No one assigned

Labels

Backend

Nodejs

Projects

None yet

Milestone

No milestone

2 participants



当然，前端不要高兴的太早：前后端分离之后，Nodejs不具备完善的服务端SESSION、数据库等功能。

总结一下，在“更先进”的前端架构下，与以往的架构会有一些区别：

- Nodejs层不处理SESSION，无法直接实现会话状态数据保存；
- 所有的数据通过Ajax异步获取，可以灵活实现token方案；

## 实现思路

如上文提到，这里仅仅讨论在“更先进”的前后端架构背景下的token防御方案的实现。

### 1、可行性方案

token防御的整体思路是：

- 第一步：后端随机产生一个token，把这个token保存在SESSION状态中；同时，后端把这个token交给前端页面；
- 第二步：下次前端需要发起请求（比如发帖）的时候把这个token加入到请求数据或者头信息中，一起传给后端；
- 第三步：后端校验前端请求带过来的token和SESSION里的token是否一致；

上文提到过，前后端分离状态下，Nodejs是不具备SESSION功能的。那这种token防御机制是不是就无法实现了呢？

肯定不是。我们可以借助cookie把这个流程升级下：

- 第一步：后端随机产生一个token，基于这个token通过SHA-56等散列算法生成一个密文；
- 第二步：后端将这个token和生成的密文都设置为cookie，返回给前端；
- 第三步：前端需要发起请求的时候，从cookie中获取token，把这个token加入到请求数据或者头信息中，一起传给后端；
- 第四步：后端校验cookie中的密文，以及前端请求带过来的token，进行正向散列验证；

当然这样实现也有需要注意的：

- 散列算法都是需要计算的，这里会有性能风险；
- token参数必须由前端处理之后交给后端，而不能直接通过cookie；
- cookie更臃肿，会不可避免地让头信息更重；

现在方案确定了，具体该如何实现呢？

### 2、具体实现

我们的技术栈是 koa(服务端) + Vue.js(前端)。有兴趣可以看这些资料：

- [趣店前端团队基于koajs的前后端分离实践](#)
- [koa-grace——基于koa的标准前后端分离框架](#)
- [grace-vue-webpack-boilerplate](#)

在服务端，实现了一个token生成的中间件，[koa-grace-csrf](#)：

```
// 注意：代码有做精简

const tokens = require('./lib/tokens');
return function* csrf(next) {
  let curSecret = this.cookies.get('密文的cookie');
  // 其他如果要获取参数，则为配置参数值
  let curToken = '请求http头信息中的token';

  // token不存在
  if (!curToken || !curSecret) {
    return this.throw('CSRF Token Not Found!', 403)
  }

  // token校验失败
  if (!tokens.verify(curSecret, curToken)) {
    return this.throw('CSRF token Invalid!', 403)
  }
}
```

```
yield next;

// 无论何种情况都种两个cookie
// cookie_key: 当前token的cookie_key, httpOnly
let secret = tokens.secretSync();
this.cookies.set(options.cookie_key, secret);
// cookie_token: 当前token的content, 不需要httpOnly
let newToken = tokens.create(secret);
this.cookies.set(options.cookie_token, newToken)
}
```

在前端代码中，对发送ajax请求的封装稍作优化：

```
this.$http.post(url, data, {
  headers: {
    'http请求头信息字段名': 'cookie中的token'
  }
}).then((res) => {})
```

总结一下：

- Nodejs生成一个随机数，通过随机数生成散列密文；并将随机数和密文存到cookie；
- 客户端JS获取cookie中的随机数，通过http头信息交给Nodejs；
- Nodejs响应请求，校验cookie中的密文和头信息中的随机数是否匹配；

这里依旧有个细节值得提一下：**Nodejs**的上层一般是**nginx**，而**nginx**默认会过滤头信息中不合法的字段（比如头信息字段名包含“\_”的），这里在写头信息的时候需要注意。

## "One more thing..."

上文也提到，通过cookie及http头信息传递加密token会有很多弊端；有没有更优雅的实现方案呢？

### 1、cookie中samesite属性

回溯下CSRF产生的根本原因：cookie会被第三方发起的跨站请求携带，这本质上是HTTP协议设计的漏洞。

那么，我们能不能通过cookie的某个属性禁止cookie的这个特性呢？

好消息是，在最新的RFC规范中已经加入了“samesite”属性。细节这里不再赘述，可以参考：

1. [SameSite Cookie, 防止 CSRF 攻击](#)
2. [Same-site Cookies](#)

### 2、更优雅的架构

当然，目前为止，客户端对samesite属性的支持并不是特别好；回到前后端分离架构下，我们明确下前后端分离框架的基本原则：

后端（Java / PHP）职责：

- 服务层颗粒化接口，以便前端Nodejs层异步并发调用；
- 用户状态保存，实现用户权限等各种功能；

前端（Nodejs + Javascript）职责：

- Nodejs层完成路由托管及模板引擎渲染功能
- Nodejs层不负责实现任何SESSION和数据库功能

我们提到，前端Nodejs层不负责实现任何SESSION和数据库功能，但有没有可能把后端缓存系统做成公共服务提供给Nodejs层使用呢？想想感觉前端整条路都亮了有木有？！这里先挖一个坑，后续慢慢填。

### 3、延伸

这里再顺便提一下，新架构下的XSS防御。

犹记得，在狼厂使用PHP的年代，经常被安全部门曝出各类XSS漏洞，然后就在smaty里添加各种 `escape` 滤镜，但是添加之后发现竟然把原始数据也给转义了。

当然，现在更多要归功于各种 `MVVM` 单页面应用：使得前端完全不需要通过读取URL中的参数来控制VIEW。

不过，还有一点值得一提：前后端分离框架下，路由由Nodejs控制；我自己要获取的后端参数和需要用在业务逻辑的参数，在主观上前端同学更好把握一些。

所以，在 `koa`(服务端) + `Vue.js`(前端) 架构下基本不用顾虑XSS问题（至少不会被安全组追着问XSS漏洞啥时候修复）。

## 总结

要不学PHP、看Java、玩Python做全栈好了？

  xiongwilee added **Backend** Nodejs labels on May 7



LikeDege commented on Jun 27

完全客户端渲染，使用ajax请求后端数据。没有nodejs作为服务端，这怎么生成页面的token？让静态服务器生成？再提交到后端服务器？



xiongwilee commented 29 days ago

Owner

@LikeDege 好问题，一般页面初始化的时候，不都有一个get请求来获取数据嘛，可以在这里获取token

[Sign up for free](#) to join this conversation on GitHub. Already have an account? [Sign in to comment](#)

