API安全：CSRF

jfeiyi / 2017-11-06 15:17:00 / 浏览数 2742

好吧，这篇文章要说的是CSRF。我知道这已经是个很老的话题了，也没多少人对它感兴趣。但在做API / 微服务架构防御的时候我觉得还是有些值得探讨的地方。
先搞个demo吧，以下是两个servlet，分别处理get和post请求：

```
@WebServlet("/data/get")
public class GetData extends HttpServlet {
   private static final long serialVersionUID = 1L;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public GetData() {
       super();
       // TODO Auto-generated constructor stub
   }

   /**
    * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
       response.getWriter().print("Received data:"+request.getParameter("data"));
       System.out.println("Received data:"+request.getParameter("data"));
   }

}
```

```
@WebServlet("/data/post")
public class PostData extends HttpServlet {
   private static final long serialVersionUID = 1L;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public PostData() {
       super();
       // TODO Auto-generated constructor stub
   }

   /**
    * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
    */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
       response.getWriter().print("Received data:"+request.getParameter("data"));
       System.out.println("Received data:"+request.getParameter("data"));
   }

}
```

再来一个响应过滤器:

```
@WebFilter(filterName = "filter1")
public class CORSResponseFilter implements Filter {

   @Override
   public void destroy() {
       // TODO Auto-generated method stub

   }

   @Override
   public void doFilter(ServletRequest arg0, ServletResponse arg1,
           FilterChain arg2) throws IOException, ServletException {
       HttpServletResponse resp = (HttpServletResponse) arg1;
       HttpServletRequest request = (HttpServletRequest) arg0;
```

```
        resp.addHeader("Access-Control-Allow-Origin", "*");
        arg2.doFilter(request, resp);
    }

    @Override
    public void init(FilterConfig arg0) throws ServletException {
        // TODO Auto-generated method stub

    }

}
```

客户端代码:

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
<script>
$.ajax({
  url: "http://localhost:8080/TestWeb/data/get",
  type: 'GET',
  data: { data: "get method" },
  success: function(data) {
   alert(data);

 },
 error: function(e) {
   alert("Error");
 }
});
</script>
<body>
</body>
</html>
```

对于大多数开发者来说，都知道用Access-Control-Allow-Origin来防御CSRF，但其实还有更多的问题需要考虑，比如：
1，你的配置是否正确。错误的Access-Control-Allow-Origin配置并不少见，但这不是我想讨论的。
2，就算正确配置了，万一本站或者子站存在XSS，依旧有可能造成CSRF。而这也不是我想讨论的。。
3，再者就是，就算用了Access-Control-Allow-Origin，你觉得就够了么？
我们先来做个实验，在响应过滤器中加入以下代码，那么通常的理解应该就是除了来自abc.com的请求，其他源发来的请求应该都无效。

```
resp.addHeader("Access-Control-Allow-Origin", "abc.com");
```

运行客户端代码，得到以下结果：

如预想的一样，无法得到请求结果。但你真的以为Access-Control-Allow-Origin阻止了这个请求么？完全没有。。。在服务器端的控制台可以看到如下输出：

如果发的是POST请求也会得到一样的结果：

POST请求代码：

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
<script>
$.ajax({
  url: "http://localhost:8080/TestWeb/data/post",
  type: 'POST',
  data: { data: "post method" },
  success: function(data) {
   alert(data);

 },
 error: function(e) {
   alert("Error");
 }
});
</script>
<body>
</body>
</html>
```

也就是说，就算你正确配置了Access-Control-Allow-Origin，最多也就是防止别人从响应中偷数据。对于修改数据的请求完全没用，不知道有多少人意识到这点，我
之后呢，有人会说，restful

API■GET请求本就该设计为不修改数据（有木有见过同一个service同时支持GET和POST的？），而POST请求的话内容应该是JSON■Content-Type也应该是applicati
然而，要考虑的不仅有这些。如果把其他响应头设置错误甚至会导致以上的防御失效。来看下一个例子，服务器端检查了所有POST请求的Content-type，必须存在且必须
在响应过滤器代码中加入:

```
if (request.getMethod().equals("POST")
               && (request.getHeader("Content-Type") == null || !request
                       .getHeader("Content-Type").toLowerCase()
                       .startsWith("application/json"))) {
           resp.setStatus(HttpServletResponse.SC_TEMPORARY_REDIRECT);
           return;
       }
       if(request.getMethod().equals("OPTIONS")){
           resp.addHeader("Access-Control-Allow-Origin", "*");
       }
       else
       {
           resp.addHeader("Access-Control-Allow-Origin", "abc.com");
       }
       resp.addHeader("Access-Control-Allow-Headers", "Content-Type");
       resp.addHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
```

客户端代码：

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
<script>
$.ajax({
  url: "http://localhost:8080/TestWeb/data/post",
  type: 'POST',
  contentType: "application/json",
  data: JSON.stringify({ data: "post method" }),
  success: function(data) {
   alert(data);

 },
 error: function(e) {
   alert("Error");
 }
});
</script>
<body>
</body>
</html>
```

结果就是服务器端依旧接收到了这个请求并处理了:

数据值为null是因为客户端发的是JSON格式，服务器端需要特别处理一下才会显示。

至此，你可以看到，在做API CSRF防护的时候需要很谨慎，稍微配置错就可能导致防护失效。
首先你要确保GET请求就只是用来做数据获取的，而不是修改。
其次服务器端要检查Content-typ。
然后在设置Access-Control-Allow-Origin的时候不仅要把域名范围考虑周全，还要把OPTIONS也考虑进去。
有人会说可以直接服务器端检查referral或者origin，这么做也是个解决方案，但并不是最佳实践。而如果遇到有要用API提供文件上传功能的时候，似乎也只能硬着头皮
还有。。似乎在IE上预检的触发条件不一样。。。以上的例子在我机器上就没有触发。。
此文写出来仅是希望与大家共同探讨API安全，虽然现在微服务架构的应用还不多，但未来很可能成为趋势。

上一篇：Mac下有哪些好用的日常的tips？？？ 下一篇：使用 Mimikatz 和 Pow...

1. 8 条回复

nearg1e 2017-11-06 17:40:00

这篇文章是小冰写的吗？
刚好是之前研究的内容，所以来支持一下社区的讨论氛围。

之前有考虑到一个需求，如果要使一个api可以任何域都跨域正常使用，应该满足那些条件呢？

1. "Access-Control-Allow-Origin" 为 "*"
2. "Access-Control-Allow-Methods" 上的支持

3. 该api支持option方法（preflight）

应该需要满足上面的条件。
所以如果一个网站上所有敏感的请求都是ajax(现在很多，例如基于 vuejs 或 angularjs
的网站)，而且所有ajax接口都不需要跨域的话，原则上来说，在有一个较为通用简单的CSRF防护手段是没办法bypass的:

1. 在所有路由的基础类(base router)上检测一个必须存在自定义header头, 这个header头是固定的也没事
2. angularjs和jquery等js库都可以在全局注册一个Interceptor，可以使所有ajax请求都插入一个header

e.g.

```
$(document).on('ajaxSend', function (elm, xhr, settings) {
           xhr.setRequestHeader('RequestToken', request_token);
       });
```

这种情况，攻击者是没有办法在其他域构造一个带有RequestToken header头的ajax请求的。
当然本文里面考虑到的内容是，必然会有跨域需求的api，这就另当别论了。

0 回复Ta


neargle 2017-11-06 17:40:36

发出去的回复好像不能编辑？

0 回复Ta


hades 2017-11-06 22:47:53

@neargle 不是我写的啦 我试着编辑了一下，晚点会把文章重新编辑到作者名下，后台还不太好用

0 回复Ta


hades 2017-11-06 22:49:56

@neargle 是的 不能编辑ing

0 回复Ta


neargle 2017-11-06 23:14:03

@hades 感觉回复还是能编辑好一点 像我1楼的那个回复 有点乱 想重新排版一下 但是不行的话 就感觉可不好意思

0 回复Ta


hades 2017-11-07 09:00:59

@neargle 我觉得排版还可以啊 想重新发就重新发咯，我等会删掉原来的就好

0 回复Ta


ifeiyi 2017-11-08 08:00:32

@neargle 服务器端还是要检查一下header是否存在的呀，不然fetch no-cors就可以了。

0 回复Ta

nearg1e 2017-11-08 09:27:49

@jfeiyi 对啊我提到了 第一点说的就是服务端的检测啦

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板