Web 系统测试

4.14 渗透测试—HTML5安全

河北师范大学软件学院
Software College of Hebei Normal University

目 录

HTML5概述

- HTML5 是W3C制定的新一代HTML语言的标准
  - 该标准还在不断地修改，然而大部分浏览器已经具备了某些HTML5的支持
  - HTML5 带来了新的功能，也带来了新的安全挑战

HTML5新标签安全

➢canvas妙用

- canvas是HTML5中最大的创新之一。不同于<img>标签只是加载一个图片，<canvas>标签让JS可以在页面中直接操作图片对象，也可以直接操作像素，构造出图片区域。
- canvas的出现极大地挑战了传统富客户端插件的地位

HTML5概述

➢使用HTML5中新增的<video>标签，在网页中远程加载一段视频

```
<video src="http://tinyvid.tv/file/29d6g90a204il.ogg"
        onloademetadata="alert(document.cookie);"
        ondurationchanged="alert(/xss/);"ontimeupdate="alert(/xss1/
        );"tabindex="0">
</video>
```

HTML5概述

百度空间的 XSS

目 录

HTML5新标签安全

- HTML5新增一些标签和属性，使得XSS等Web攻击产生了新的变化

HTML5新标签安全

➢iframe的sandbox

- <iframe>标签一直以来都存在隐患，如：挂马、XSS、点击劫持等；浏览器厂商也一直在想办法限制iframe执行脚本的权限，比如：跨窗口访问会有限制，以及IE中支持security属性限制脚本的执行
- HTML5中，专门为iframe定义了一个新的属性，叫sandbox。使用sandbox这一属性后，<iframe>标签加载的内容将被视为一个独立的“源”，其中的脚本将被禁止执行，表单将被禁止提交，插件被禁止加载，指向其他浏览器对象的链接也会被禁止

HTML5新标签安全

- sandbox属性可以通过参数来支持更精确的控制，有以下几个值可以选择：
  - allow-same-origin:允许同源访问
  - allow-top-navigation:允许访问顶层窗口
  - allow-forms:允许提交表单
  - allow-scripts:允许执行脚本

HTML5新标签安全

- 有的行为，即便是设置了allow-scripts，也是不允许的，比如“弹出窗口”，一个iframe的实例如下：

```
<iframe sandbox="allow-same-origin, allow-forms, allow-scripts"
src="http://maps.example.com/embedded.html"></iframe>
```

iframe的sandbox属性将极大地增强应用使用iframe的安全性

HTML5新标签安全

➢Link Types:noreferrer

- 在HTML5中为<a> 和<area>标签定义一个新的Link Types:noreferrer
  - 标签指定了noreferrer后，浏览器在请求该标签指定的地址时，将不再发送referrer
  - test
  - *rel 属性规定当前文档与被链接文档之间的关系*

HTML5新标签安全


Request Line
Request Headers
view parsed
GET http://www.optimizesmart.com/google-analytics-wrong-why/ HTTP/1.1
Host: www.optimizesmart.com
Proxy-Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/web
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHT
Referer: http://plus.url.google.com/url?sa=z&n=1400603604168&url=http%3
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: stb_box_9876=true; wordpress_test_cookie=WP+Cookie+check; wordp
-1=editor%3Dtinymce%26hidetb%3D1%26widgets_access%3Don%26m4%3Do%26m5%3D
9%3Do%26m10%3Do%26m2%3Do%26uploader%3D1%26m12%3Do%26wplink%3D1%26ed_siz
35; _wp_lead_magnet_actions_taken=a%3A2%3A%7Bs%3A17%3A%22dismiss-subscr
%22subscribe%22%3Ba%3A0%3A%7B%7D%7D; __utma=57112436.667464943.14000782
|utmcmd=referral|utmcct=/url
HTTP Request Headers

HTML5新标签安全

- 为什么这样设计
  - 这种设计是出于保护敏感信息和隐私考虑，因为通过Referer,可能会泄漏一些敏感信息
  - 这个标签需要开发者手动添加到页面的标签中，对于有需求的标签可以选择使用noreferrer

其他安全问题

➢Cross—Origin—Resource Sharing

- 浏览器的同源策略限制了脚本的跨域请求，但互联网的发展趋势越来越开放，因此跨域访问的需求也变得越来越迫切，同源策略给Web开发者带来了很多困扰
- 开发者不得不想方设法地实现一些“合法”的跨域技术，由此诞生了jsonp、iframe跨域等技巧，W3C委员会决定制定一个新的标准来解决日益迫切的跨域访问问题

其他安全问题（Cross-Origin-Resource Sharing）

➢假设从http://www.a.com/test.html发起一个跨域的XMLHttpRequest请求，请求地址为：http://www.b.com/test.php

```
<script>
        var client = new XMLHttpRequest();
        client.open(“GET”, “http://www.b.com/test.php”);
< /script >
```

其他安全问题（Cross-Origin-Resource Sharing）

- 如果是在IE8中，则需要使用XDomainRequest来实现跨域请求

```
var request = new XDomainRequest();
request.open("GET",xdomainurl);
request.send();
```

- 如果服务器www.b.com返回一个HTTP Header

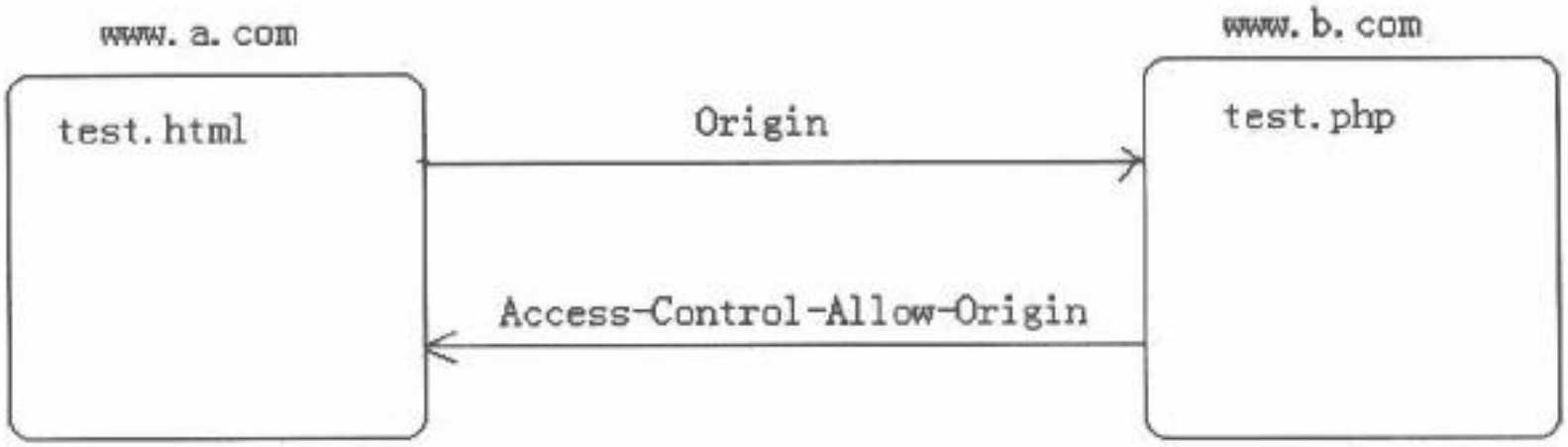
```
Access-Contrlo-Allow-Origin:http://www.a.com
<?php
    header("Access-Control-Allow-Origin:*");
?>
```

- 来自http://www.a.com/test.html的跨域请求就会被通过

其他安全问题（Cross-Origin-Resource Sharing）

- 在这个过程中，http://www.a.com/test.html发起的请求还必须带上一个Origin Header

- **Origin：http://www.a.com**



跨域请求的访问过程

其他安全问题（Cross-Origin-Resource Sharing）

- Origin Header 用于标记HTTP发起的“源”，服务器通过识别浏览器自动带上的这个Origin Header，来判断浏览器的请求是否来自一个合法的“源”。Origin Header可以用于防范CSRF，它不像Referer那么容易被伪造或清空

其他安全问题（Cross-Origin-Resource Sharing）

- 如上服务器返回Access-Control-Allow-Origin:*，这里使用通配符“*”，极其危险，它将允许来自任意域的跨域请求访问，等于没有做任何安全限制
  - 可以精确控制
    - Access-Control-Allow-Origin HTTP Response Header
    - Access-Control-Max-Age HTTP Response Header
    - Access-Control-Allow-Credentials HTTP Response Header
    - Access-Control-Allow-Methods HTTP Response Header
    - ……

其他安全问题—postMessage

- postMessage跨窗口传递消息
  - window.name跨窗口、跨域传递信息。实际上，window这个对象几乎不受同源策略限制，很多脚本攻击都巧妙的利用window对象的这一特点
  - 在HTML5中为了丰富Web开发者的能力，制定了一个新的API:postMessage(允许每个窗口往其他窗口发送文本消息)

其他安全问题—postMessage

- 发送窗口

```
<iframe src=http://dev.jquery.com/_john/message/
id="iframe"></iframe>

<form id="form">
    <input type = "text" id="msg" value="Message to send" />
    <input type="submit"/>
</form>
```

其他安全问题—postMessage


```
<script>
        window.onload=function(){
                var win = document.getElementById(“iframe”).contentWindow;
                document.getElementById(“form”).onsubmit =
        function(e){win.postMessage(document.getElementById(‘msg’).value)
        e.preventDefault();
}
};
</ script >
```

其他安全问题—postMessage

➢接收窗口

```
<script>
        document.addEventListener(“message”,function(e){documen
t.getElementById(“test”).textContent = e.domain+ “said:” +
e.data;},false);
</ script >
```

其他安全问题—postMessage

➢两点注意

- 在必要时，可以在接收窗口验证Domain，甚至验证URL，以防止来自非法页面的消息，这实际上是在代码中实现一次同源策略的验证过程
- 接收消息写入textContent，但实际应用中，如果将消息写入innerHTML，甚至直接写入script中，则可能会导致DOM based XSS的产生。根据Secure By Default原则，在接收窗口不应该信任接收到的消息，而需要对消息进行安全检查

内容总结

- HTML5概述
- HTML5新标签安全
- 其他安全问题

Question