# 应用安全攻击

刘铠铭 2020030014 计14

## 实验目的

演示 Web 应用安全的 XSS 漏洞，实现 SQL 注入攻击和 CSRF 攻击。

## 环境配置

1. WSL2 Ubuntu22.04LTS
2. Python3.10

## 实验步骤

首先运行程序，可以得到如下界面：


## XSS 攻击

核心代码：

```
<!-- Search form -->
<form method="GET">
<input type="text" name="q"
         placeholder="搜索内容" autocomplete="off" />
<input type="submit" value="提交" />
</form>

<!-- Comments -->
```

```
{% if not search_query %}
<h3>所有的评论如下:</h3>
{% else %}
<h3>包含 "{{ search_query }}" 评论如下:</h3>
{% endif %}

{% for comment in comments %}
<div>
   <p>{{ comment }}</p>
</div>
{% endfor %}

<!-- Write form -->
<form action="/" method="POST">
<input type="text" name="comment"
         placeholder="评论" autocomplete="off" />
<input type="submit" value="提交新评论" />
</form>
```

1. 在搜索框输入 `<script>alert('反射型 XSS')</script>`，进行反射型 XSS 攻击，如下图：


点击提交，运行结果：


2. 在评论区输入 `<script>alert('持久型 XSS')</script>`，进行持久型 XSS 攻击，如下图：


点击提交新评论，之后在每次访问数据库数据时均会弹出：

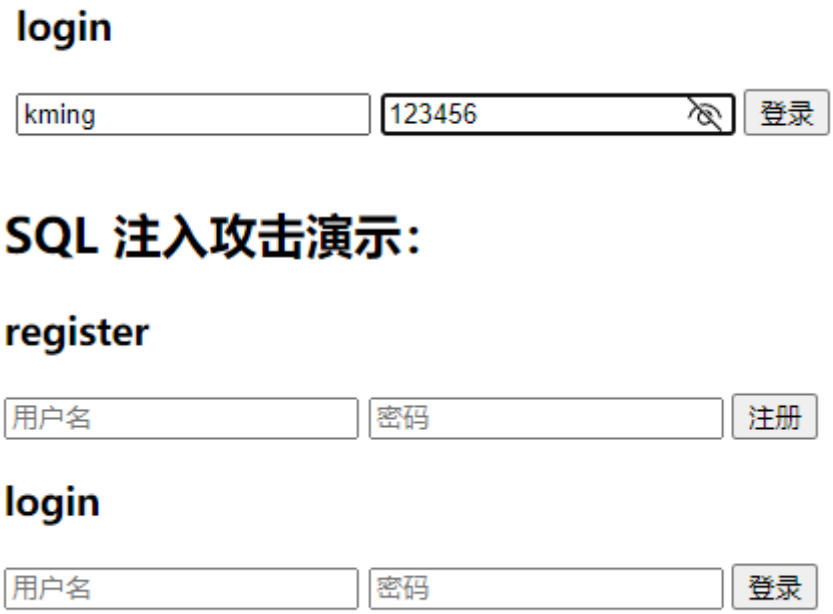

## SQL 注入攻击

核心代码：

```
<h3>register</h3>
<form action="{{ url_for('register') }}" method="POST">
<input type="text" name="username"
         placeholder="用户名" autocomplete="off" />
<input type="password" name="password"
         placeholder="密码" autocomplete="off" />
<input type="submit" value="注册" />
</form>

<h3>login</h3>
<form action="{{ url_for('login') }}" method="POST">
<input type="text" name="username"
         placeholder="用户名" autocomplete="off" />
<input type="password" name="password"
         placeholder="密码" autocomplete="off" />
<input type="submit" value="登录" />
</form>

<h3>注册&登录结果：{{ get_flashed_messages()[0] }}</h3>
```

1. 首先注册账户：


2. 正常情况下，当我们输入错误密码时无法登录：




3. 进行 SQL 注入攻击，当我们知道希望登陆的用户名时：


注册&登录结果：User kming' -- logged in

4. 实际上，我们不需要知道数据库中有哪些用户名：


## CSRF 攻击模拟

背景：一个银行转账系统，只有 sudo 账户可以进入该页面对自己的账户进行信息改动。

核心代码：

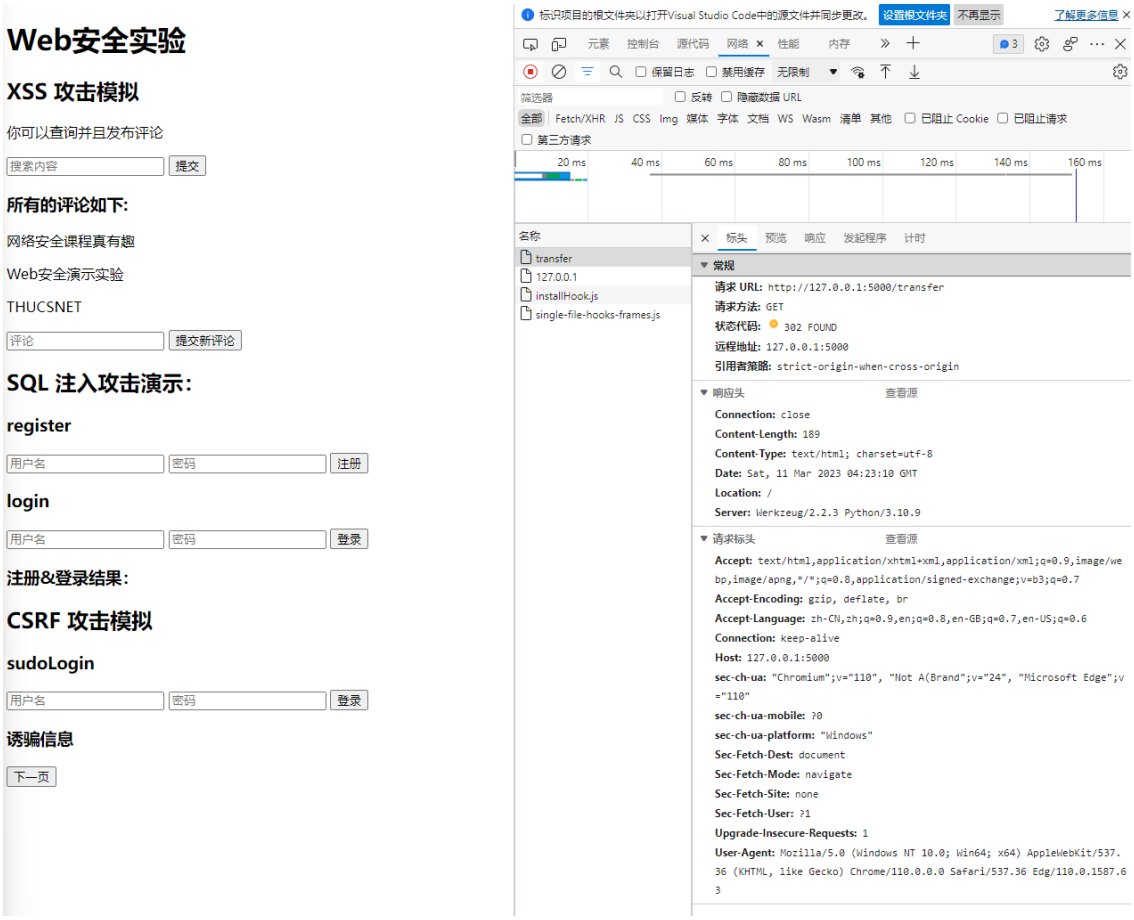
```
<h3>sudoLogin</h3>

<form action="{{ url_for('sudo_login') }}" method="POST">
<input type="text" name="username"
         placeholder="用户名" autocomplete="off" />
<input type="password" name="password"
         placeholder="密码" autocomplete="off" />
<input type="submit" value="登录" />
</form>

<h3>诱骗信息</h3>

<form action="{{ url_for('transfer') }}" method="POST">
<input type="hidden" name="username" value="attacker" />
<input type="hidden" name="amount" value="1000000" />
<input type="submit" value="下一页" />
</form>
```

银行转账系统页面：

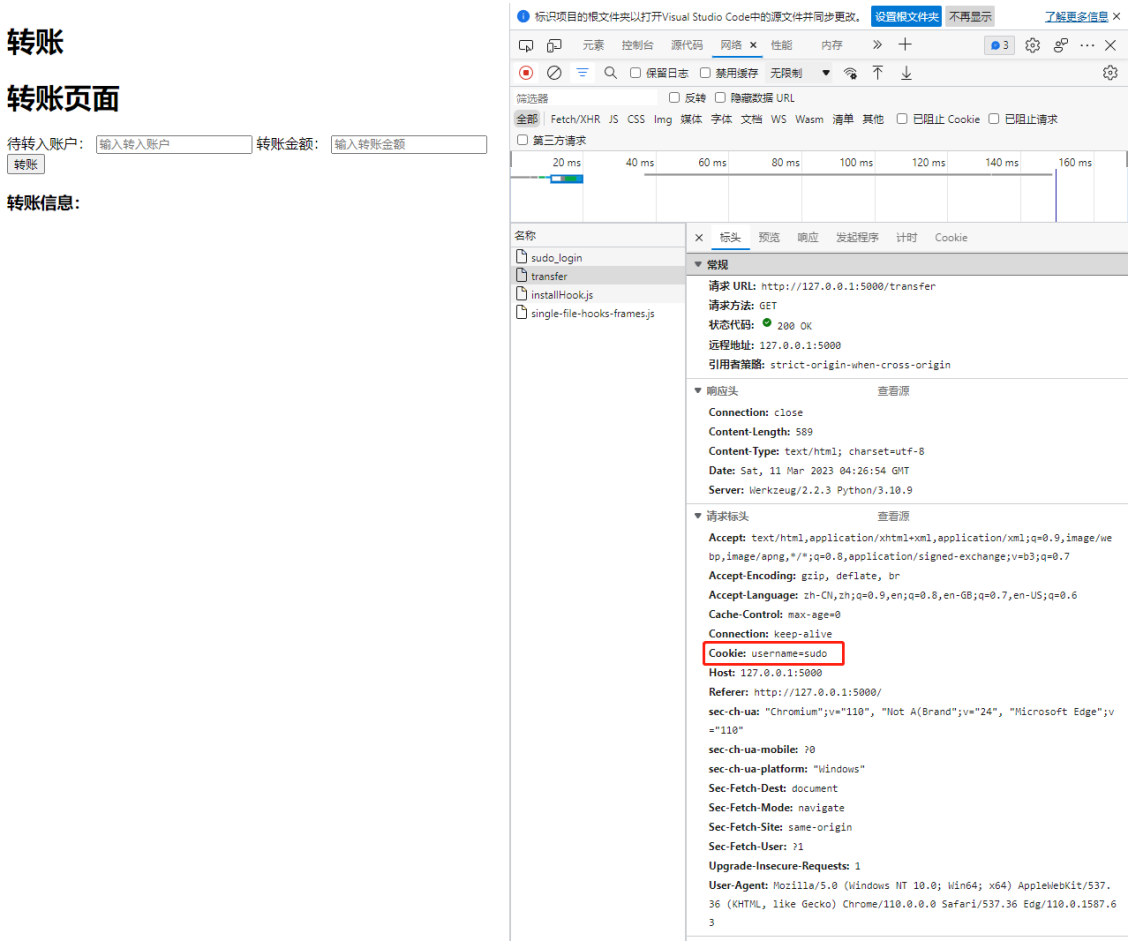
```
<header>
<h1>转账</h1>
</header>

<h1>转账页面</h1>

<form action="{{ url_for('transfer') }}" method="post">
<label>待转入账户：</label>
<input type="text" name="username" placeholder="输入转入账户"/>
<label>转账金额：</label>
<input type="text" name="amount" placeholder="输入转账金额"/>
<input type="submit" value="转账"/>
```

```
</form>

<h3>转账信息：{{ get_flashed_messages()[0] }}</h3>
```

1. 正常情况下，当我们访问 `127.0.0.1:5000/transfer` 时，可以看到无法进入转账页面，而会返回根目录。当我们点击“下一页”按钮时同理：




2. 登录 sudo 账户，此时检查表头可以发现 header 中已经有了用户 cookie 信息：




此时可以正常进入 transfer 页面并进行转账操作：


3. 此后再点击“下一页”，会发现进入 transfer 页面并且已经向非法账户进行转账操作：


## 影响因素分析

针对 XSS 注入攻击的防护手段有：

- 输入验证和过滤：对于所有的用户输入，包括表单数据、URL 参数、Cookie 等，进行严格的输入验证和过滤。
- 输出编码：在将用户输入的数据输出到 Web 页面上时，确保对输出内容进行适当的编码。
- Content Security Policy（CSP）：通过实施 CSP，可以指定哪些内容被浏览器信任并允许加载到页面中。
- HTTP Only Cookie：将敏感的会话 Cookie 标记为"HTTP Only"，这样 JavaScript 将无法访问该 Cookie。
- 避免在页面中直接嵌入用户输入的 HTML 等。

针对 SQL 注入攻击的防护手段有：

- 使用参数化查询（Prepared Statements）或存储过程：使用参数化查询（Prepared Statements）或存储过程来执行数据库查询，而不是直接拼接用户输入的数据作为查询语句的一部分。
- 输入验证和过滤：对于用户输入的数据，进行严格的输入验证和过滤。
- 避免动态拼接 SQL 语句：避免在代码中直接拼接用户输入的数据来构建 SQL 查询语句。
- 最小权限原则：为数据库用户分配最小的权限，确保应用程序只能执行必要的数据库操作。

针对 CSRF 攻击的防护手段有：

- 使用 CSRF 令牌：为每个用户会话生成唯一的CSRF令牌，并将其嵌入到表单或请求中。
- 验证来源（Referer）：在服务器端验证请求的来源 Referer 头部信息。
- 添加验证码（CAPTCHA）：在敏感操作前，要求用户输入验证码。
- SameSite Cookie 属性：将 Cookie 的 SameSite 属性设置为 Strict 或 Lax，限制 Cookie 的跨站传递，降低 CSRF 攻击的风险。
- 双重因素认证（2FA）：引入双重因素认证机制，要求用户在敏感操作前进行额外的身份验证。

本实验代码比较简单，未执行任何防护措施，因此均容易被攻击。

## 完整代码

完整代码请参考 https://cloud.tsinghua.edu.cn/d/1a96b198a593475abf7f/

## 参考资料


- 《网络空间安全原理与实践》