

第十三章实验：实现本地 Web 攻击

一、实验目的

二、实验原理

三、实验环境

四、实验思路

五、实验步骤

step1: 安装flask框架、安装Mysql

step2: 运行app.py

step3: 进行XSS反射型攻击

step4: 进行XSS持久型攻击

step5: 使用防御方法，防御XSS攻击

step6: 增设登录功能，并展示SQL注入与防御措施

六、CSRF攻击与防御

1、CSRF（Cross-site request forgery）

2、攻击实现

3、防御实现

七、总结

八、参考文献

第十三章实验：实现本地 Web 攻击

一、实验目的

实现本地Web攻击和防御

二、实验原理

大多数 Web 应用程序攻击都是来源于XSS、CSRF 和 SQL 注入攻击，这些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，其中CSRF存在是指攻击者构建的恶意网站被用户访问后，返回一些攻击性代码，并发出一个请求要求访问第三方站点，从而盗用用户身份，如用户名义发送邮件、虚拟货币转账等

三、实验环境

Ubuntu虚拟机（20.04），Kali虚拟机，Python 3.10.8

四、实验思路

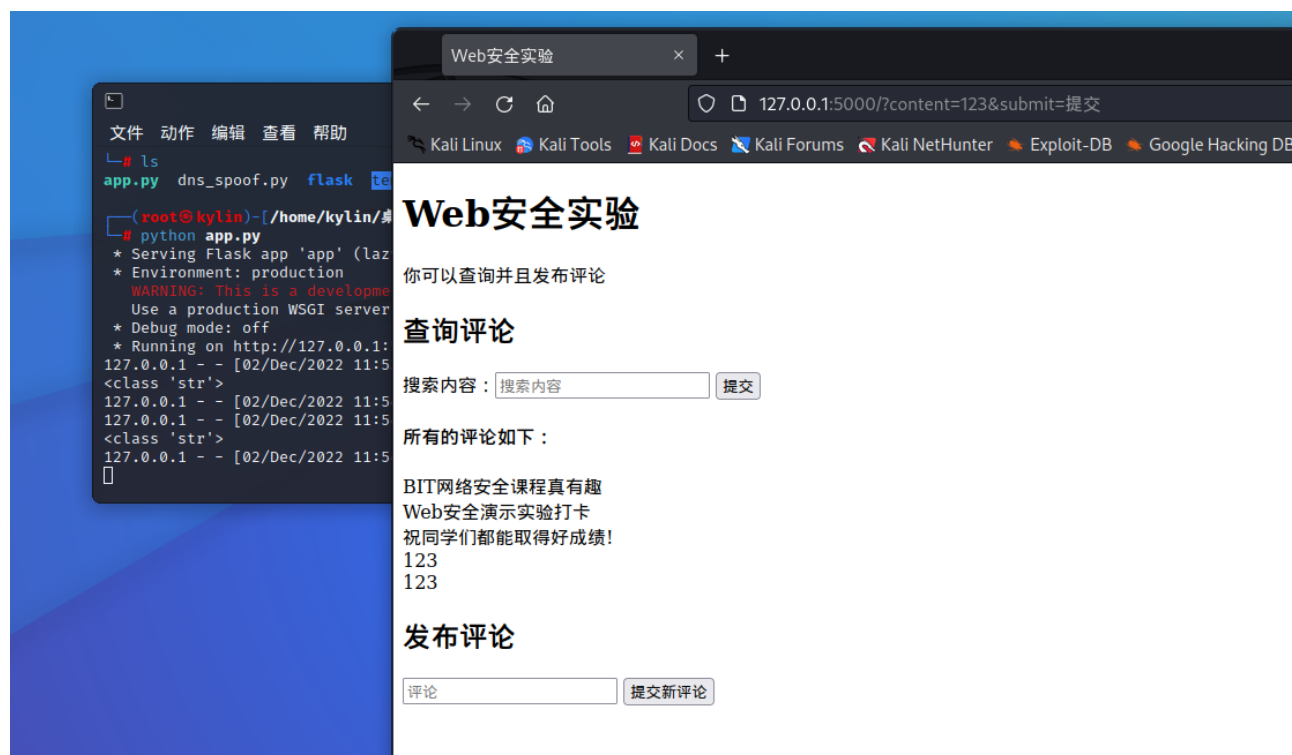
安装Flask框架并启动提供的源代码，开发一个简单的网页

- 1、访问网页并实现XSS反射型与持久型攻击
- 2、使用防御方法，防范XSS攻击
- 3、增加一个登录功能，设计有 SQL 注入隐患的代码，进行攻击，并且展示如何进行防范
- 4、设计一个 CSRF 攻击范例，并且演示如何防御（可选）

五、实验步骤

step1: 安装flask框架、安装Mysql

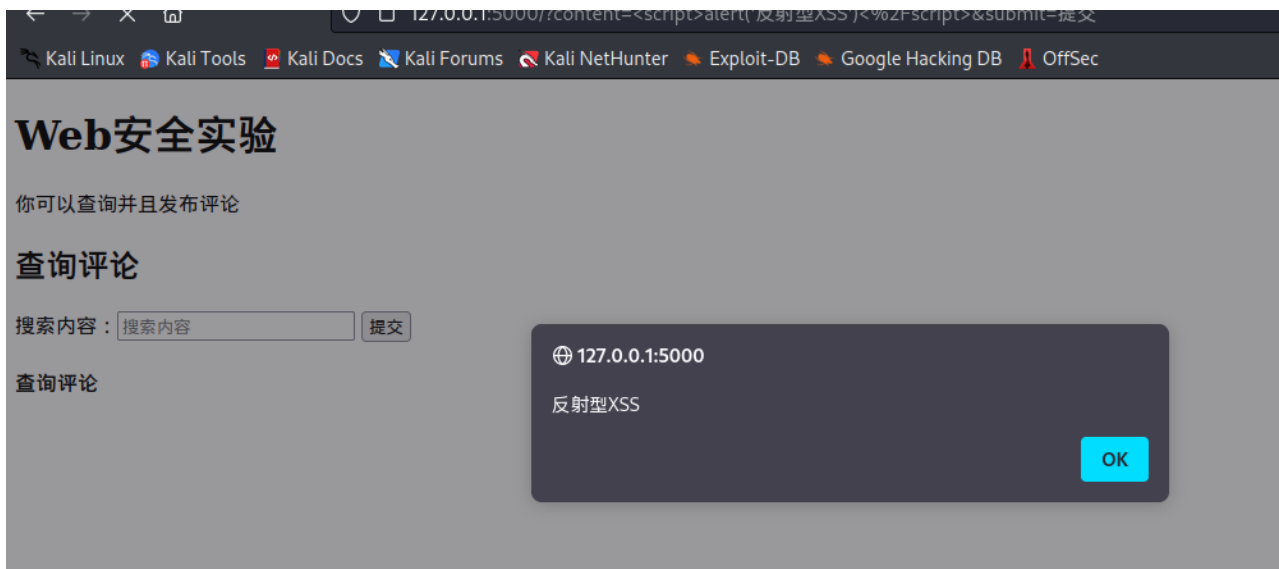
step2: 运行app.py



step3: 进行XSS反射型攻击

在搜索内容处输入命令

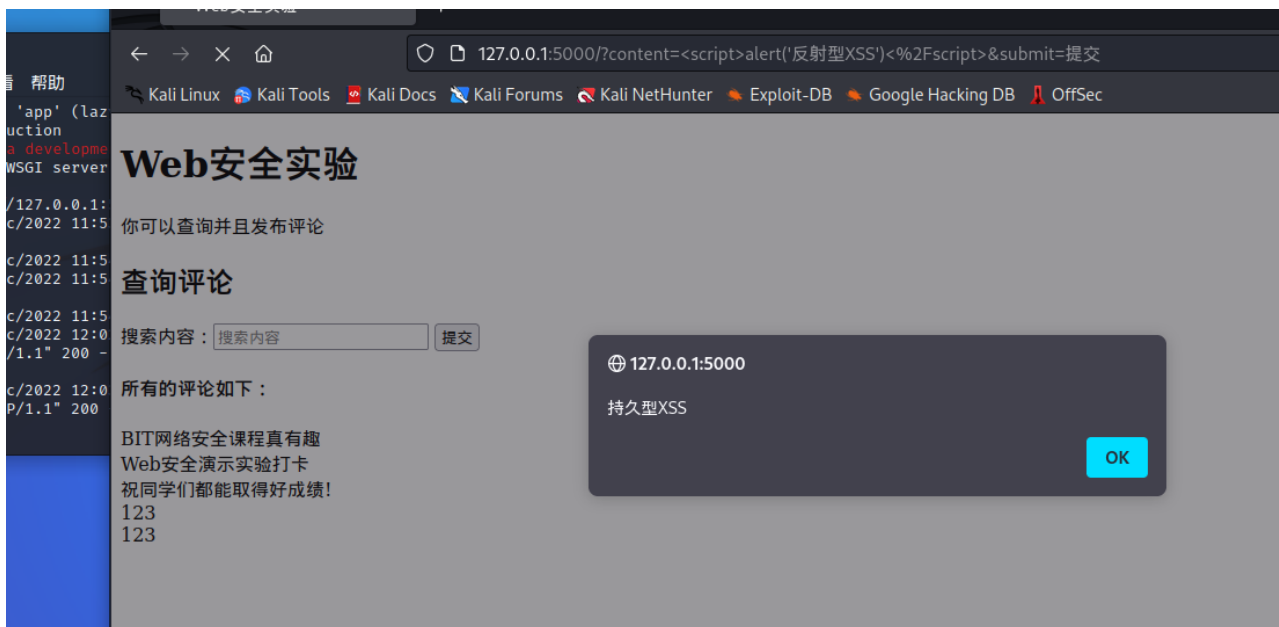
```
<script>alert('反射型XSS')</script>
```



step4、进行XSS持久型攻击

在评论框内输入命令：

```
<script>alert('持久型XSS')</script>
```



step5、使用防御方法，防御XSS攻击

防御思路：增设XSS过滤器，对文本信息进行过滤

修改app.py源码

```
from flask import Flask, render_template, request
from html.parser import HTMLParser

//写一个子类继承HTMLParser
class StripTagsHTMLParser(HTMLParser):
    data = ""
    def handle_data(self, data):
        self.data += data

    def getData(self):
        return self.data

app = Flask(__name__)

dataset=["BIT网络安全课程真有趣", "web安全演示实验打卡", "祝同学们都能取得好成绩!"]

@app.route("/", methods=["GET", "POST"])
def index():
    query = ""
    parser = StripTagsHTMLParser()
    if request.method == "POST":
        if request.form.get("submit") == "提交新评论":
            comment = request.form.get("newComment").strip()

            #对comment进行过滤
            parser.feed(comment)
            comment = parser.getData()

            print(comment)
            print(type(comment))
            if comment:
                dataset.append(comment)

    elif request.method == "GET":
        if request.args.get("submit") == "提交":
```

```

query = request.args.get("content").strip()

#对query进行过滤
parser.feed(query)
query = parser.getData()

if query:
    sub_dataset = [x for x in dataset if query in x]
    return render_template("index.html", query=query,
comments=sub_dataset)
    return render_template("index.html", query=query,
comments=dataset)

if __name__ == "__main__":
    app.run()

```

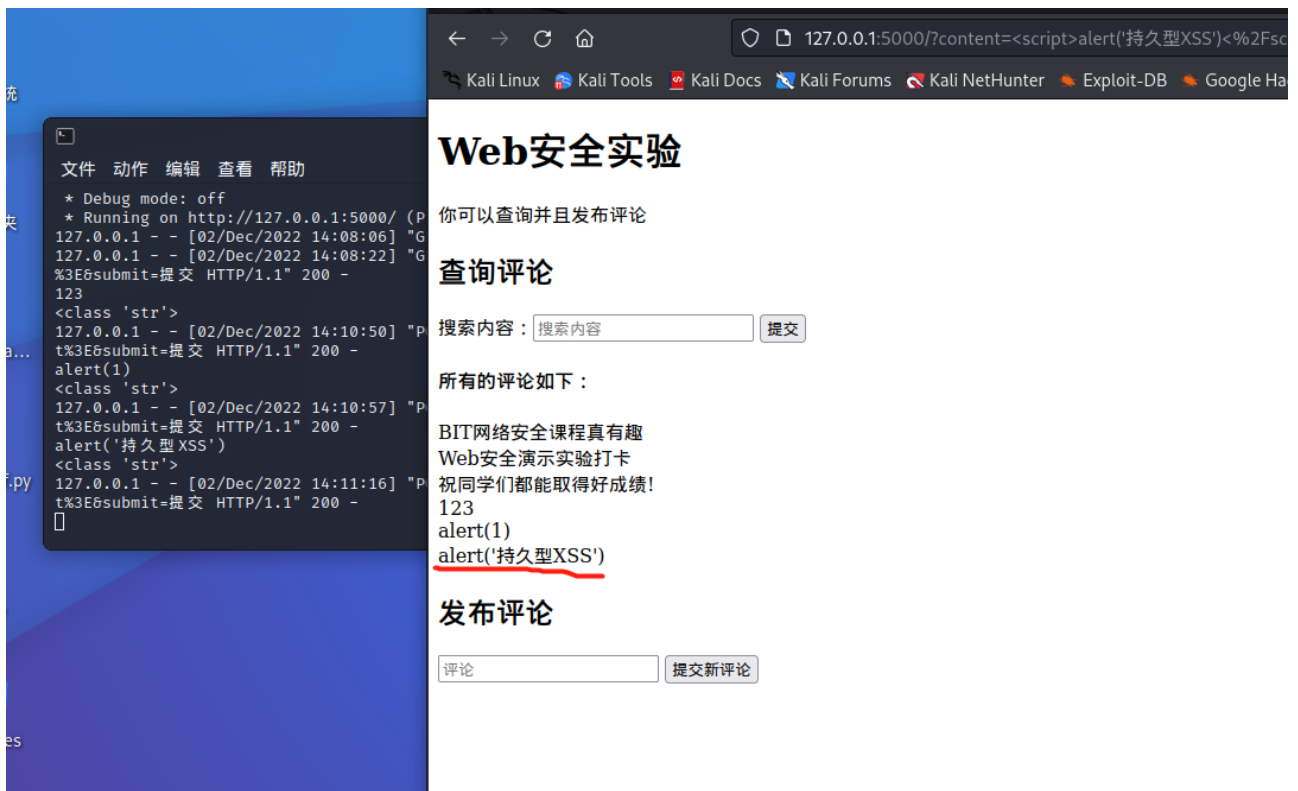
防御结果:



The screenshot displays a web browser window with the address bar showing the URL `127.0.0.1:5000/?content=<script>alert('持久型XSS')<%2Fscript>&su`. The browser's address bar also shows several tabs, including 'flask xss攻击_百度', 'XSS攻击原理、示例', 'OSError: [Errno 98]', and 'python的flask解决'. The browser's address bar also shows several links, including 'Kali Linux', 'Kali Tools', 'Kali Docs', 'Kali Forums', 'Kali NetHunter', 'Exploit-DB', and 'Google Hacking DB'.

The browser window displays the title 'Web安全实验' and the text '你可以查询并且发布评论'. Below this, there is a search bar labeled '查询评论' with the text '搜索内容:' and a '提交' button. The search results show '查询评论 alert('持久型XSS') 结果如下:'. Below the search results, there is a section labeled '发布评论' with a text input field labeled '评论' and a '提交新评论' button.

The terminal window shows the output of the Flask application. It displays the message 'alert('反射型XSS')' and the class 'str'. The terminal also shows the Flask application running on `http://127.0.0.1:5000/` and the attack payload being executed: `%3E&submit=提交 HTTP/1.1" 200 - ^C`.



可以看到已经对标签进行了去除，对XSS攻击有一定的防御作用

step6: 增设登录功能，并展示SQL注入与防御措施

修改index.html，增加用户登录

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>web安全实验</title>

</head>
<body>
  <h1>web安全实验</h1>
  你可以查询并且发布评论
  <br>
  <h2>查询评论</h2>
  <form action="" method="get">
    搜索内容: <input type="text" name="content" placeholder="搜索内
    容">
    <input type="submit" name="submit" value="提交">
```

```
</form>
{%if query%}
<h4>查询评论 {{query|safe}} 结果如下: </h4>
{%else%}
<h4>所有的评论如下: </h4>
{%endif%}
{% for comment in comments %}
<td>{{comment|safe}}</td>
<br>
{% endfor %}
<h2>发布评论</h2>
<form action="" method="post">
    <input type="text" name="newComment" placeholder="评论">
    <input type="submit" name="submit" value="提交新评论">
</form>
<br>
    <form action="" method="post">
        用户:
        <input type="text" name="name" >
        <br>
        密码:
        <input type="password" name="password">
        <br>
        <input type="submit" name="submit" value="登录">
    </form>

</body>
</html>
```

← → ↻ 🏠 127.0.0.1:5000

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetH

Web安全实验

你可以查询并且发布评论

查询评论

搜索内容：

所有的评论如下：

BIT网络安全课程真有趣
Web安全演示实验打卡
祝同学们都能取得好成绩！

发布评论

用户：

密码：

修改app.py，处理回路信息

```
from flask import Flask, render_template, request
from html.parser import HTMLParser
import pymysql

class StripTagsHTMLParser(HTMLParser):
    data = ""
    def handle_data(self, data):
        self.data += data

    def getData(self):
        return self.data

app = Flask(__name__)
```



```
dataset=["BIT网络安全课程真有趣","web安全演示实验打卡","祝同学们都能取得好成绩!"]
```

```
@app.route("/", methods=["GET", "POST"])
```

```
def index():
```

```
    query = ""
```

```
    parser = StripTagsHTMLParser()
```

```
    if request.method == "POST":
```

```
        if request.form.get("submit") == "提交新评论":
```

```
            comment = request.form.get("newComment").strip()
```

```
            parser.feed(comment)
```

```
            comment = parser.getData()
```

```
            print(comment)
```

```
            print(type(comment))
```

```
            if comment:
```

```
                dataset.append(comment)
```

```
    #处理登录请求，登录成功后返回“hello+name”
```

```
    elif request.form.get("submit") == "登录":
```

```
        name = request.form.get("name").strip()
```

```
        password = request.form.get("password").strip()
```

```
        #链接数据库进行查询
```

```
        db =
```

```
pymysql.connect(host="localhost",user="root",password="123",db="user" )
```

```
    # 使用cursor()方法获取操作游标
```

```
    cursor = db.cursor()
```

```
    # SQL 查询语句（有缺陷）
```

```
    sql = "select * from user where user= '{ }' and password='{ }' ".format(name,password)
```

```
    try:
```

```
        # 执行sql语句
```

```
        cursor.execute(sql)
```

```
        results = cursor.fetchall()
```

```
        print(len(results))
```

```
        if len(results)==1:
```

```
            return "hello,"+name
```

```
        else:
```

```

        return '用户名或密码不正确'
    # 提交到数据库执行
    db.commit()
except:
    # 如果发生错误则回滚
    traceback.print_exc()
    db.rollback()
# 关闭数据库连接
db.close()

elif request.method == "GET":
    if request.args.get("submit") == "提交":
        query = request.args.get("content").strip()

        parser.feed(query)
        query = parser.getData()

        if query:
            sub_dataset = [x for x in dataset if query in x]
            return render_template("index.html", query=query,
comments=sub_dataset)
        return render_template("index.html", query=query,
comments=dataset)

if __name__ == "__main__":
    app.run()

```

在虚拟机上配置mysql，建立一个新的数据库user

```

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.000 sec)

MariaDB [(none)]> create database user
→ ;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| user |
+-----+
5 rows in set (0.000 sec)

MariaDB [(none)]> use user
Database changed

```

在user中添加一个表username，表里记录账号和密码

```

MariaDB [(none)]> use user
Database changed
MariaDB [user]> create table username(
→ user varchar(25),
→ password varchar(25)
→ );
Query OK, 0 rows affected (0.003 sec)

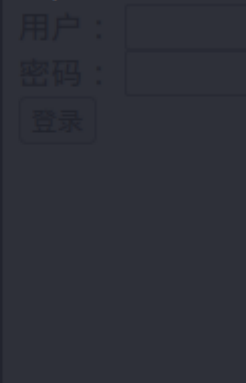
MariaDB [user]> show tables
→ ;
+-----+
| Tables_in_user |
+-----+
| username |
+-----+
1 row in set (0.000 sec)

```

在表里插入数据，记录root用户的密码

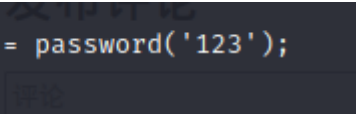
```
MariaDB [user]> insert into username(user,password)
→ value("root","root");
Query OK, 1 row affected (0.001 sec)

MariaDB [user]> select * from username
→ ;
+-----+-----+
| user | password |
+-----+-----+
| root | root     |
+-----+-----+
1 row in set (0.000 sec)
```

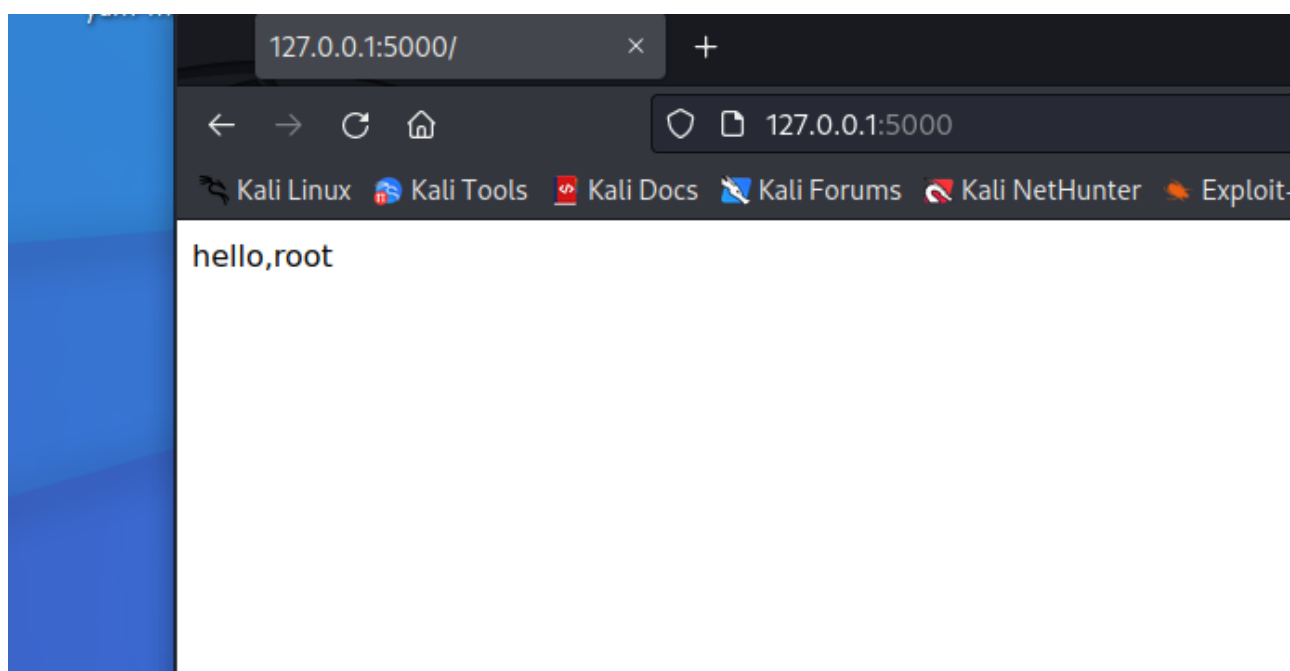


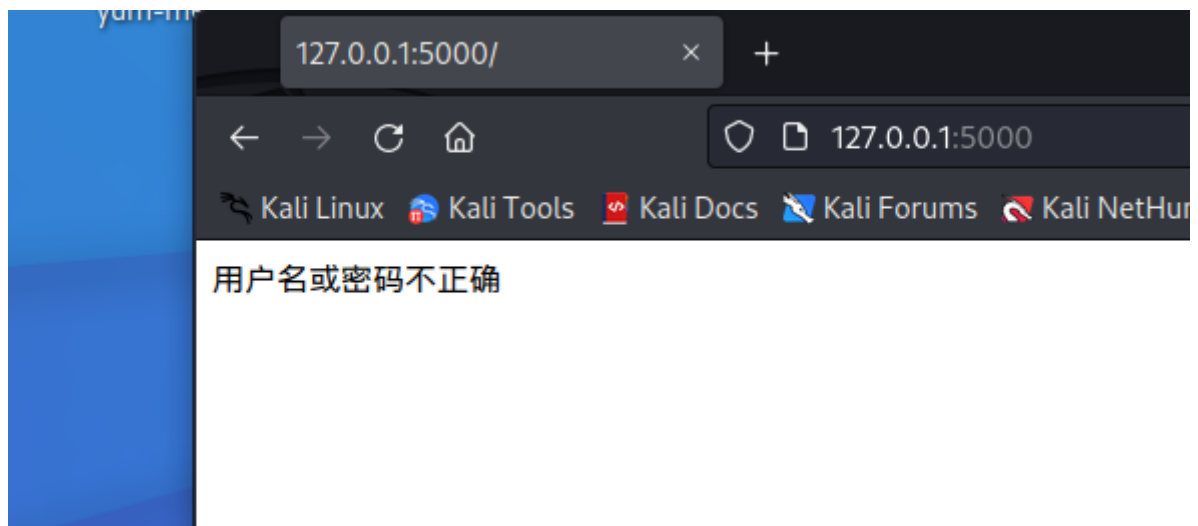
设置数据库的密码

```
MariaDB [user]> set password for root@localhost = password('123');
Query OK, 0 rows affected (0.001 sec)
```



登录效果，成功后会显示hello,root，失败后会显示“用户名或密码不正确”





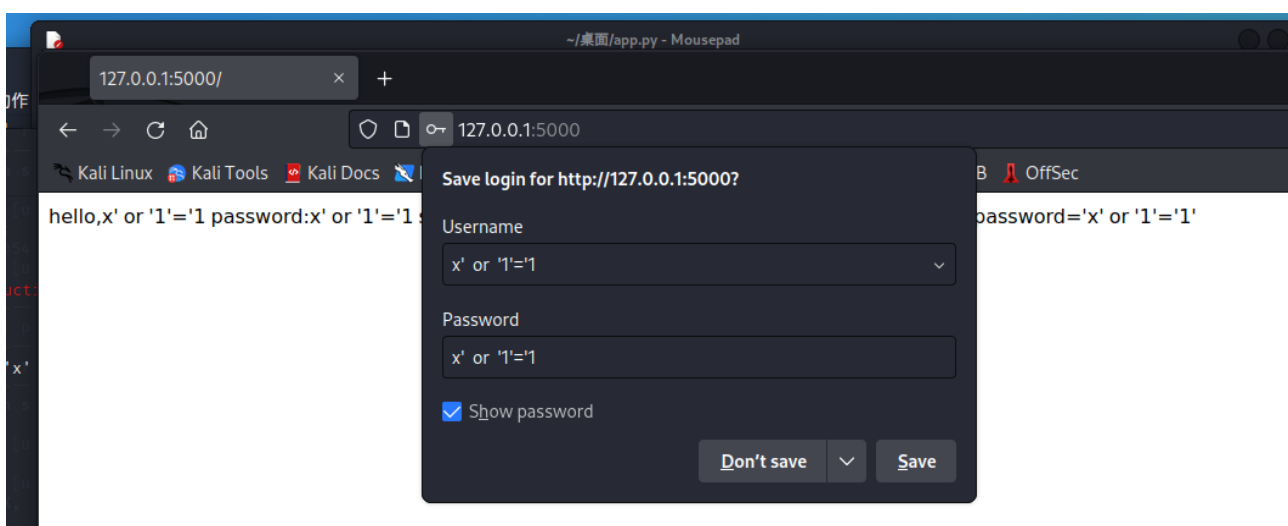
进行SQL注入攻击，为了展示攻击效果，在返回页面输出登录用户名和密码，以及查询语句sql

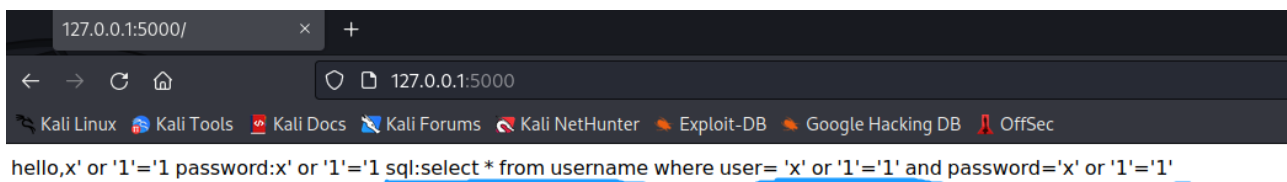
```
# 执行sql语句
cursor.execute(sql)
results = cursor.fetchall()
print(len(results))
if len(results)==1:
    return "hello,"+name+'      password:'+password+'      sql:'+sql
else:
    return '用户名或密码不正确'
# 提交到数据库执行
```

已知查询语句为：

```
"select * from username where user= '{} ' and
password='{} ' ".format(name,password)
```

可以构造攻击语句 `x' or '1'='1`，即可完成攻击





可以看到查询语句是：

```
select * from username where user= 'x' or '1'='1' and  
password='x' or '1'='1'
```

等价于：

```
select * from username
```

对SQL注入攻击进行防护：采取字符过滤的手段，不让攻击者更改sql查询语句，

在链接数据库查询前，首先对name、password进行筛查，若发现敏感拼写，则返回“请输入规范的参数”

更改app.py代码：

```
from flask import Flask, render_template, request  
from html.parser import HTMLParser  
import pymysql  
import re  
  
class StripTagsHTMLParser(HTMLParser):  
    data = ""  
    def handle_data(self, data):  
        self.data += data  
  
    def getData(self):  
        return self.data  
  
app = Flask(__name__)  
  
dataset=["BIT网络安全课程真有趣", "web安全演示实验打卡", "祝同学们都能取得好成绩!"]  
  
@app.route("/", methods=["GET", "POST"])  
def index():
```

```

query = ""
parser = StripTagsHTMLParser()

if request.method == "POST":
    if request.form.get("submit") == "提交新评论":
        comment = request.form.get("newComment").strip()

        parser.feed(comment)
        comment = parser.getData()

        print(comment)
        print(type(comment))
        if comment:
            dataset.append(comment)

    #处理登录请求，登录成功后返回“hello+name”
    elif request.form.get("submit") == "登录":

        name = request.form.get("name").strip()
        password = request.form.get("password").strip()
        #在链接数据库查询前，首先对name、password进行筛查，若发现敏感拼
        #写，则返回“请输入规范的参数”
        data = [name, password]
        for v in data:
            v = str(v).lower()
            pattern =
r"\b(and|like|exec|insert|select|drop|grant|alter|delete|update|cou
nt|chr|mid|master|truncate|char|declare|or)\b|(\*|;)"
            r = re.search(pattern, v)
            if r:
                return '请输入规范的参数！'

        #链接数据库进行查询
        db =
pymysql.connect(host="localhost", user="root", password="123", db="use
r" )

        # 使用cursor()方法获取操作游标
        cursor = db.cursor()
        # SQL 查询语句（有缺陷）

```

```

        #sql= "SELECT * FROM username WHERE (name = '" + name +
        "') and (password = '"+ password +"'");"
        sql = "select * from username where user= '{}{}' and
        password='{}{}'".format(name,password)
        print(sql)
        try:
            # 执行sql语句
            cursor.execute(sql)
            results = cursor.fetchall()
            print(len(results))
            if len(results)==1:
                return "hello,"+name
            else:
                return '用户名或密码不正确'
            # 提交到数据库执行
            db.commit()
        except:
            # 如果发生错误则回滚
            traceback.print_exc()
            db.rollback()
        # 关闭数据库连接
        db.close()

    elif request.method == "GET":
        if request.args.get("submit") == "提交":
            query = request.args.get("content").strip()

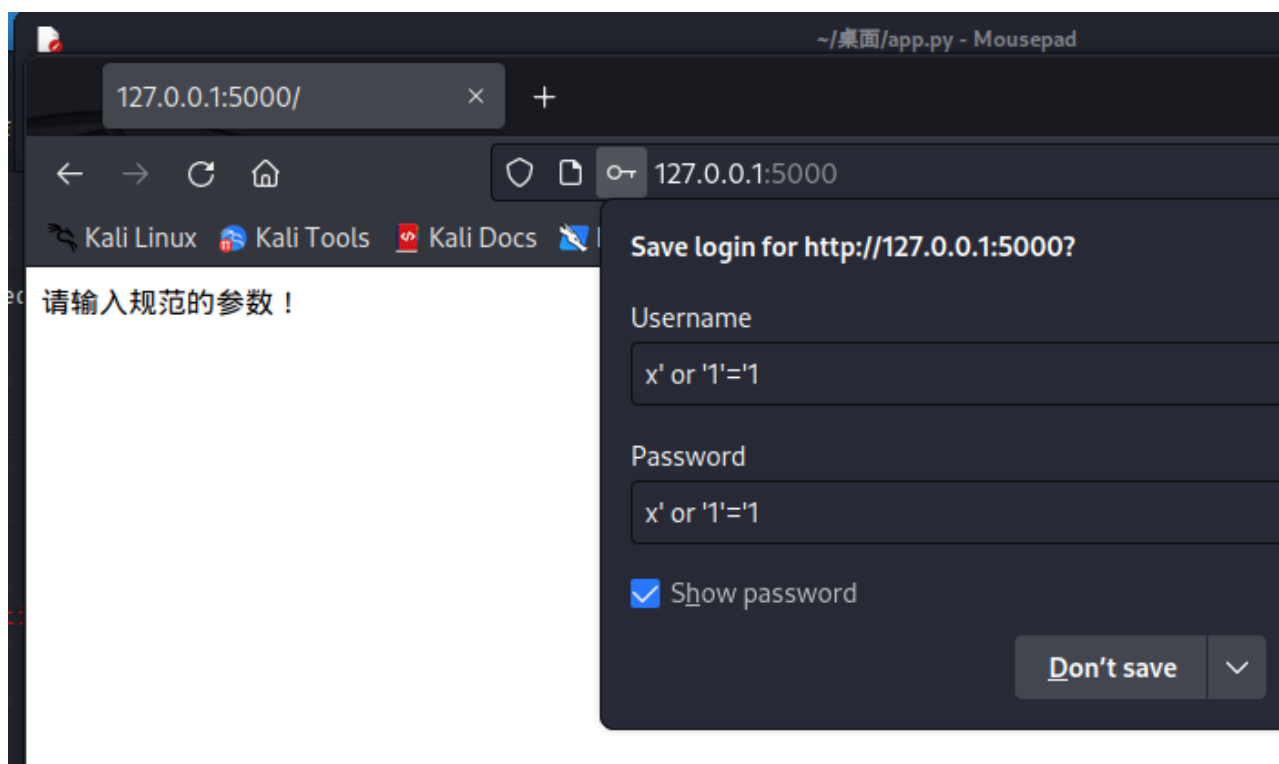
            parser.feed(query)
            query = parser.getData()

            if query:
                sub_dataset = [x for x in dataset if query in x]
                return render_template("index.html", query=query,
                comments=sub_dataset)
            return render_template("index.html", query=query,
            comments=dataset)

if __name__ == "__main__":
    app.run()

```


可以看到再次输入 `x' or '1'='1`，就会显示如下页面，攻击失败



六、CSRF攻击与防御

1、CSRF（Cross-site request forgery）

CSRF定义：

跨站请求伪造（英语：Cross-site request forgery）是一种对网站的恶意利用，也被称为 **one-click attack** 或者 **session riding**，通常缩写为 **CSRF** 或者 **XSRF**，是一种挟制用户在当前已登录的Web应用程序上执行非本意的操作的攻击方法。**CSRF**跨站点请求伪造 (Cross—Site Request Forgery) 跟XSS攻击一样，存在巨大的危害性。

这样来理解：攻击者盗用了你的身份，以你的名义发送恶意请求，对服务器来说这个请求是完全合法的，但是却完成了攻击者所期望的一个操作，比如以你的名义发送邮件、发消息，盗取你的账号，添加系统管理员，甚至于购买商品、虚拟货币转账等。

简单地说，是攻击者通过一些技术手段欺骗用户的浏览器去访问一个自己曾经认证过的网站并执行一些操作（如发邮件，发消息，甚至财产操作如转账和购买商品）。由于浏览器曾经认证过，所以被访问的网站会认为是真正的用户操作而去执行。这利用了web中用户身份验证的一个漏洞：简单的身份验证只能保证请求发自某个用户的浏览器，却不能保证请求本身是用户自愿发出的。

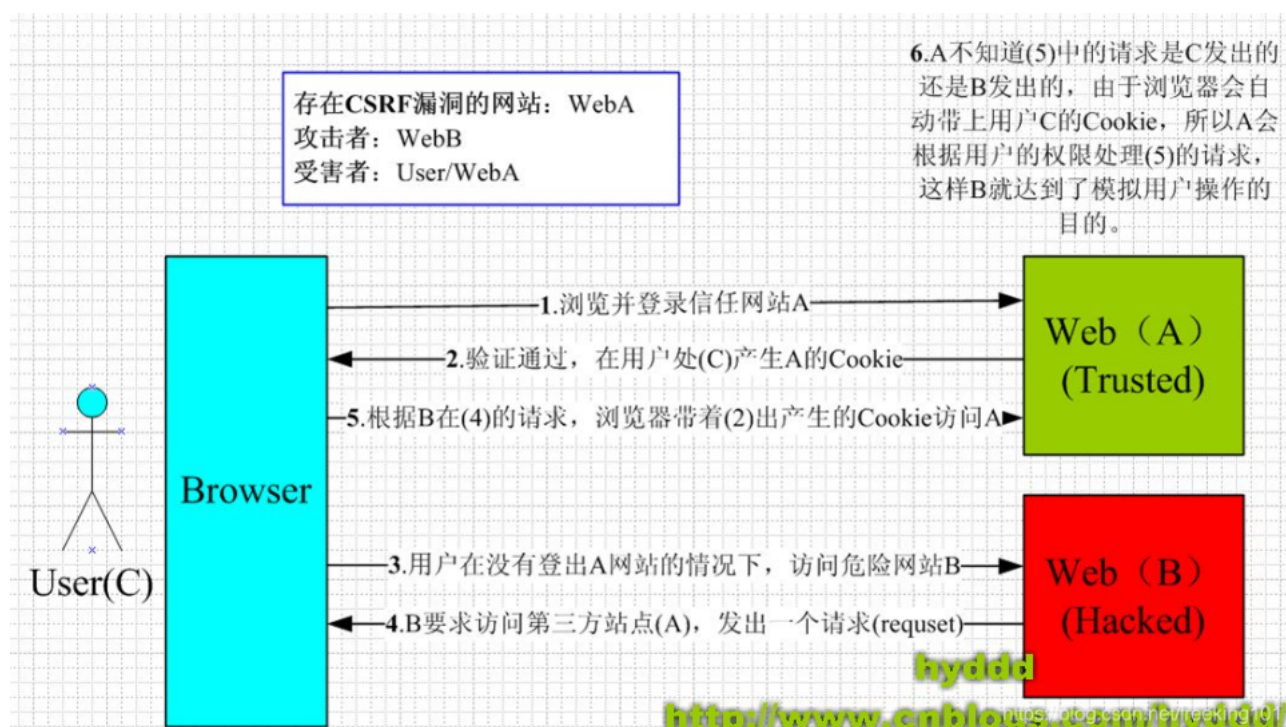
CSRF地位:

是一种网络攻击方式，是互联网重大安全隐患之一，NYTimes.com（纽约时报）、Metafilter，YouTube、Gmail和百度HI都受到过此类攻击。

对比XSS:

跟跨网站脚本(XSS)相比，XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户网页浏览器的信任。

如下：其中Web A为存在CSRF漏洞的网站，Web B为攻击者构建的恶意网站，User C为Web A网站的合法用户。



第一方和第三方cookie概念:

Cookie是一个域服务器存储在浏览器中的一小段数据块，只能被这个域访问，谁设置则谁访问。

第一方Cookie: 比如，访问www.a.com这个网站，这个网站设置了一个Cookie，这个Cookie也只能被www.a.com这个域下的网页读取。

第三方Cookie: 比如，访问www.a.com这个网站，网页里有用到www.b.com网站的一张图片，浏览器在www.b.com请求图片的时候，www.b.com设置了一个Cookie，那这个Cookie只能被www.b.com这个域访问，反而不能被www.a.com这个域访问，因为对我们来说，我们实际是在访问www.a.com这个网站被设置了一个www.b.com这个域下的Cookie，所以叫第三方Cookie。

CSRF 原理:



- 1.用户C打开浏览器，访问受信任网站A，输入用户名和密码请求登录网站A;
- 2.在用户信息通过验证后，网站A产生Cookie信息并返回给浏览器，此时用户登录网站A成功，可以正常发送请求到网站A;
- 3.用户未退出网站A之前，在同一浏览器中，打开一个TAB页访问网站B;
- 4.网站B接收到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点A;
- 5.浏览器在接收到这些攻击性代码后，根据网站B的请求，在用户不知情的情况下携带Cookie信息，向网站A发出请求。网站A并不知道该请求其实是由B发起的，所以会根据用户C的Cookie信息以C的权限处理该请求，导致来自网站B的恶意代码被执行。

简而言之：通过访问恶意网址，恶意网址返回来js自动执行访问你之前登陆的网址，因为你已经登录了，所以再次访问将会携带cookie，因为服务器只认有没有cookie，无法区分是不是用户正常的访问，所以会欺骗服务器，造成伤害

CSRF攻击防御

CSRF攻击防御的重点是利用cookie的值只能被第一方读取，无法读取第三方的cookie值。

防御方法:

预防csrf攻击简单可行的方法就是在客户端网页上再次添加一个cookie，保存一个随机数，而用户访问的时候，先读取这个cookie的值，hash一下这个cookie值并发送给服务器，服务器接收到用户的hash之后的值，同时取出之前设置在用户端的cookie的值，用同样的算法hash这个cookie值，比较这两个hash值，相同则是合法。（如果用户访问了病毒网站，也想带这个cookie去访问的时候，此时，因为病毒网站无法获取第三方cookie的值，所以他也就无法hash这个随机数，所以也就会被服务器校验的过滤掉）

2、攻击实现

无

3、防御实现

无

七、总结

通过这次实验了解了XSS攻击以及SQL攻击的多种方式，了解了前端和后端的编程知识，实现了一些简单的攻击与防御方式，切身体会到了网络安全的重要性，进一步加深了对网络传输的理解，收获颇丰！

八、参考文献

[1] <https://www.jianshu.com/p/57475845f3ad>

[2] (22条消息) python+flask+mysql实现登录注册_如风少年的博客-CSDN博客

[3] Python进阶知识全篇-MySQL（PyMySQL） - 知乎 (zhihu.com)

[4] (22条消息) python flask框架，在请求前加入参数过滤，防止sql注入cwg0508的博客-CSDN博客flask 防止sql注入

[5] SQL注入攻击与防御 - 腾讯云开发者社区-腾讯云 (tencent.com)

[6] (22条消息) CSRF攻击与防御（写得非常好）_擒贼先擒王的博客-CSDN博客