第 13 章实验报告

|  |  |  |
| --- | --- | --- |
| 学号： | 姓名： 杜恩俊 | 班级： 12012202 |
| 实验题目：实现本地 Web 攻击 | | |
| 实验日期：2024.4.2 | | |
| 实验目的：实现本地Web攻击和防御  背景: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 | | |
| 硬件环境：  磁盘驱动器：NVMe KIOXIA- EXCERIA G2 SSD  NVMe Micron 3400 MTFDKBA1TOTFH  显示器：NVIDIA GeForce RTX 3070 Ti Laptop GPU  系统型号 ROG Strix G533ZW\_G533ZW  系统类型 基于 x64 的电脑  处理器 12th Gen Intel(R) Core(TM) i9-12900H，2500 Mhz，14 个内核，20 个逻辑处理器  BIOS 版本/日期 American Megatrends International, LLC. G533ZW.324, 2023/2/21  BIOS 模式 UEFI  主板产品 G533ZW  操作系统名称 Microsoft Windows 11 家庭中文版 | | |
| 软件环境：  VMware Workstation Pro  Ubuntu 18.04.6 LTS  Kali linux  Microsoft Windows 11 家庭中文版 | | |
| 实验步骤与分析：   1. 先打开终端运行sudo apt install python3安装python3环境。 2. 先采用sudo apt update和sudo apt install python3-flask -y来安装Flask框架。 3. 在kali中创建topic13文件夹，包含已经给出的代码资料。 4. 在终端上输入命令 python app.py，来运行app.py，出现如下图片即表示运行成功。可通过在浏览器访问地址localhost:5000或者127.0.0.1:5000来访问该地址。      1. 现在首先进行XSS反射型实验：在地址上输入http://127.0.0.1:5000/?content=<script>alert('XSS反射实验成功！')</script>&submit=提交，当显示如下弹窗时，则证明攻击成功。 2. 现在执行XSS持久型实验：在评论栏输入代码<a href="#" onclick="window.location='https://www.yuanshen.com';">XSS持久型攻击——原神，启动！</a>后点击提交新评论，此时网址https://www.yuanshen.com便已经被存储在评论XSS持久型攻击——原神，启动！里面了。点击该评论跳转原神官网，攻击成功！   aeebbd8bf6e50710c84bd20db1bdb80  88b21c16f85c9f2219cd45db9409d68   1. 现在修改app.py的代码来防止XSS攻击，修改后代码如下：  |  | | --- | | 1. from flask import Flask, render\_template, request, escape 2. from werkzeug.utils import escape 3. app = Flask(\_\_name\_\_) 4. # 假设的评论数据集 5. 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!"] 6. @app.route("/", methods=["GET", "POST"]) 7. def index(): 8. query = "" 9. if request.method == "POST": 10. if request.form.get("submit") == "提交新评论": 11. # 通过 escape 函数转义用户输入 12. comment = escape(request.form.get("newComment").strip()) 13. if comment: 14. dataset.append(comment) 15. elif request.method == "GET": 16. if request.args.get("submit") == "提交": 17. # 对查询参数进行转义以防止反射型XSS攻击 18. query = escape(request.args.get("content").strip()) 19. if query: 20. sub\_dataset = [x for x in dataset if query.lower() in x.lower()] 21. return render\_template("index.html", query=query, comments=sub\_dataset) 22. # 默认情况下渲染页面，展示所有评论 23. return render\_template("index.html", query=query, comments=dataset) 24. if \_\_name\_\_ == "\_\_main\_\_": 25. app.run() |   修改地方如下： 使用了“escape”函数来转义用户输入  对POST请求中的‘comment’进行转义 对GET请求中的‘query’进行转义 原因：转义可以将用户输入中的特殊HTML字符（如 <, >, &, " 等）转换为它们的HTML实体形式（例如，< 转换为 &lt;）。这意味着当浏览器渲染这些转义后的字符时，它们不会被作为HTML标签或JavaScript代码执行，而是作为普通文本显示。这样，即使用户输入了 <script>alert('XSS')</script> 这样的代码，它也只会被显示出来，而不会执行。  原因：  **防止反射型XSS攻击：** 通过对URL参数（如搜索查询）进行转义，即使攻击者试图通过构造含有恶意脚本的URL来发起反射型XSS攻击，这些脚本也不会执行。因为在HTML中，这些脚本已被转义，不会被浏览器解释为代码。  **防止持久型XSS攻击：** 通过对用户提交的评论内容进行转义，即使攻击者在评论中嵌入了JavaScript代码，这些代码也只会作为普通文本被存储和显示，而不会在其他用户浏览评论时执行。这样，就算这些评论包含了恶意代码，也不会对其他用户造成威胁。  测试：输入代码http://127.0.0.1:5000/?content=<script>alert('XSS反射实验成功！')</script>&submit=提交后，显示“查询评论<script>alert('XSS反射实验成功！')</script>”，故该代码未被执行，防止反射型XSS攻击成功！ 28700af69fa900e4695ce9381ba3bc5  在评论栏输入代码<a href="#" onclick="window.location='https://www.yuanshen.com';">XSS持久型攻击——原神，启动！</a>后，点击提交新评论，显示得到如下评论，故可知道转义后的代码不会被执行，代码会被当做纯文本显示在评论上。防止XSS持久型攻击成功！  89e02dfbf083c34a10ca5fd5504c4cd   1. 在终端输入命令python -c "import sqlite3"来导入sqlite3模块 2. 在Flask中创建一个名为‘init\_db.py’的Python文件，然后运行该文件创建初始化数据库和用户表‘login.db’。这里创建了一个初始账户，账号为sky，密码为sky666. 源码如下：  |  | | --- | | 1. import sqlite3 2. connection = sqlite3.connect('login.db') 3. cursor = connection.cursor() 4. *# 创建一个用户表* 5. cursor.execute(''' 6. CREATE TABLE users(id INTEGER PRIMARY KEY, username TEXT, password TEXT) 7. ''') 8. *# 插入示例用户* 9. cursor.execute(''' 10. INSERT INTO users(username, password) VALUES ('sky', 'sky666') 11. ''') 12. connection.commit() 13. connection.close() |  1. 创建一个login.html，增加登录和注册功能，源码如下：  |  | | --- | | 1. <!DOCTYPE html> 2. <html lang="en"> 3. <head> 4. <meta charset="UTF-8"> 5. <title>Web安全实验</title> 6. </head> 7. <body> 8. <h2>Web安全实验</h2> 9. <form action="" method="post"> 10. Username: <input type="text" name="username"><br> 11. Password: <input type="password" name="password"><br> 12. <input type="submit" value="Login"> 13. </form> 14. <a href="/register">Register</a> 15. {% if error %} 16. <p style="color: red">{{ error }}</p> 17. {% endif %} 18. </body> 19. </html> |  1. 再创建一个register.html，增加注册功能，源码如下：  |  | | --- | | 1. <!DOCTYPE html> 2. <html lang="en"> 3. <head> 4. <meta charset="UTF-8"> 5. <title>Register</title> 6. </head> 7. <body> 8. <h2>Register</h2> 9. <form action="/register" method="post"> 10. Username: <input type="text" name="newUsername"><br> 11. Password: <input type="password" name="newPassword"><br> 12. <input type="submit" value="Register"> 13. </form> 14. </body> 15. </html> |  1. 修改index.html，增加退出登录功能。 2. 修改app.py函数，代码原理：首先进入链接后检测是否成功登陆，如果没有成功登陆则跳转login.html，login.html包含login和register，如果没有账号可以点击register跳转register.html来进行注册。然后在login输入账号密码登录成功后跳转index.html，index.html李有logout选项，点击Logout后会退出登录返回到login.html中。源码如下：  |  | | --- | | 1. from flask import Flask, render\_template, request, redirect, url\_for, session 2. import sqlite3 3. from werkzeug.utils import escape 4. app = Flask(\_\_name\_\_) 5. app.secret\_key = 'fan\_xing\_yu\_is\_little\_boy\_mother' 6. *# 假设的评论数据集* 7. 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!"] 8. *# 注册功能* 9. @app.route('/register', methods=['GET', 'POST']) 10. def register(): 11. if request.method == 'POST': 12. new\_username = request.form['newUsername'] 13. new\_password = request.form['newPassword'] 14. connection = sqlite3.connect('login.db') 15. cursor = connection.cursor() 16. cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (new\_username, new\_password)) 17. connection.commit() 18. connection.close() 19. return redirect(url\_for('login')) 20. return render\_template('register.html') 21. @app.route("/", methods=["GET", "POST"]) 22. def index(): 23. if not session.get('logged\_in'):  *# 检查用户是否登录* 24. return redirect(url\_for('login'))  *# 未登录，重定向到登录页面* 25. query = "" 26. if request.method == "POST": 27. if request.form.get("submit") == "提交新评论": 28. comment = escape(request.form.get("newComment").strip()) 29. if comment: 30. dataset.append(comment) 31. elif request.method == "GET": 32. if request.args.get("submit") == "提交": 33. query = escape(request.args.get("content").strip()) 34. if query: 35. sub\_dataset = [x for x in dataset if query.lower() in x.lower()] 36. return render\_template("index.html", query=query, comments=sub\_dataset) 37. return render\_template("index.html", query=query, comments=dataset) 38. @app.route('/login', methods=['GET', 'POST']) 39. def login(): 40. error = None 41. if request.method == 'POST': 42. *# 从表单中获取用户名和密码* 43. username = request.form['username'] 44. password = request.form['password'] 45. connection = sqlite3.connect('login.db') 46. cursor = connection.cursor() 47. *# 故意引入 SQL 注入的隐患（仅作为演示，实际中不应这么做）* 48. query = f"SELECT \* FROM users WHERE username = '{username}' AND password = '{password}'" 49. cursor.execute(query) 50. user = cursor.fetchone() 51. connection.close() 52. if user: 53. session['logged\_in'] = True 54. return redirect(url\_for('index')) 55. else: 56. error = "Invalid username or password" 57. return render\_template('login.html', error=error) 58. @app.route('/logout') 59. def logout(): 60. session.pop('logged\_in', None) 61. return redirect(url\_for('login')) 62. if \_\_name\_\_ == "\_\_main\_\_": 63. app.run(debug=True) |  1. SQL注入攻击演示： 打开app.py后，在login里的username中输入' OR '1'='1' --，在password中可以不作任何输入，点击登录，可以发现，页面直接跳转到了index.html。 2. 原理分析： 最开始的判定代码为：‘SELECT \* FROM users WHERE username = '{username}' AND password = '{password}'’，当我们输入' OR '1'='1' --后，代码就变为了‘SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = '任意值'’。  ' OR '：这个片段是试图打破原有 SQL 查询的逻辑，通过添加一个总是为真的条件（'1'='1'）。  此时我们可以知道，‘1’=‘1’永远判定为真  ‘--’：这是 SQL 中的注释标记。在这个上下文中，它的作用是注释掉 SQL 语句的剩余部分，特别是与密码相关的那部分，这样 SQL 服务器就不会检查密码是否正确。 3. SQL防御： 在app.py里将login函数修改为：  |  | | --- | | 1. @app.route('/login', methods=['GET', 'POST']) 2. def login(): 3. error = None 4. if request.method == 'POST': 5. username = request.form['username'] 6. password = request.form['password'] 7. connection = sqlite3.connect('login.db') 8. cursor = connection.cursor() 9. # 使用参数化查询防止 SQL 注入 10. cursor.execute("SELECT \* FROM users WHERE username = ? AND password = ?", (username, password)) 11. user = cursor.fetchone() 12. connection.close() 13. if user: 14. session['logged\_in'] = True 15. return redirect(url\_for('index')) 16. else: 17. error = "Invalid username or password" 18. return render\_template('login.html', error=error) |   再次运行后，输入' OR '1'='1' --后发现显示Invalid username or password。   1. 原理分析：为了防御SQL攻击，我们应该避免直接将用户的输入拼接到SQL查询中。故我采用参数化查询，这种方式可以确保传入的参数不会被解释为 SQL 代码的一部分，而是作为数据处理。在上述login函数的修改中，我使用参数化查询替代字符串格式化。  这么设置的好处是应用程序不再对 SQL 注入攻击易受攻击，因为用户输入被安全地处理为查询的一部分，而不是作为 SQL 代码执行。 2. CSRF攻击设计： 设计CSRF\_attack.html源码如下：  |  | | --- | | 1. <!DOCTYPE html> 2. <html> 3. <head> 4. <title>恶意网站</title> 5. </head> 6. <body> 7. <script> 8. // 构建目标网站的URL，包括恶意的XSS代码 9. var targetUrl = "http://127.0.0.1:5000/?content=%3Cscript%3Ealert('XSS%E5%8F%8D%E5%B0%84%E5%AE%9E%E9%AA%8C%E6%88%90%E5%8A%9F%EF%BC%81')%3C%2Fscript%3E&submit=%E6%8F%90%E4%BA%A4"; 10. // 利用Image对象发起GET请求，绕过同源策略 11. var img = new Image(); 12. img.src = targetUrl; 13. </script> 14. </body> 15. </html> |  1. 该源码使用转义，将想要输入的url“http://127.0.0.1:5000/?content=<script>alert('XSS反射实验成功！')</script>&submit=提交”转义为“"http://127.0.0.1:5000/?content=%3Cscript%3Ealert('XSS%E5%8F%8D%E5%B0%84%E5%AE%9E%E9%AA%8C%E6%88%90%E5%8A%9F%EF%BC%81')%3C%2Fscript%3E&submit=%E6%8F%90%E4%BA%A4";” 构造思路：  构建恶意URL：targetUrl变量中存储了目标网站的URL，其中包括了一个查询参数content。这个参数通过URL编码嵌入了恶意的JavaScript代码，这段代码是<script>alert('XSS反射实验成功！')</script>。当目标网站接收到这个请求并处理content参数时，假设它没有对这个参数进行适当的清理或转义，那么这段JavaScript代码将在用户的浏览器中执行。   利用Image对象发起GET请求：通过创建一个Image对象并将其src属性设置为构建好的恶意URL，实际上发起了一个对目标URL的GET请求。这个请求是在不需要用户交互的情况下自动完成的。由于是通过<img>标签加载资源的方式，它可以绕过一些简单的同源策略限制。   1. 结果展示：在运行app.py后浏览器打开CSRF\_attack.html，可以发现app.py终端显示被XSS反射型攻击成功。 2. 选做之如何防御CSRF攻击：   为了防御CSRF攻击，需要使用POST请求而不是GET，并且实施CSRF保护。   1. 首先在终端运行pip install Flask-WTF来下载Flask-WTF 进行 CSRF 保护。 2. 接着修改app.py代码如下：  |  | | --- | | 1. from flask import Flask, render\_template, request, session, redirect, url\_for 2. from flask\_wtf import FlaskForm 3. from wtforms import StringField, SubmitField 4. from flask\_wtf.csrf import CSRFProtect 5. app = Flask(\_\_name\_\_) 6. app.config['SECRET\_KEY'] = 'fan\_xing\_yu\_is\_little\_boy\_mother' 7. csrf = CSRFProtect(app) 8. *# 定义评论表单类* 9. class CommentForm(FlaskForm): 10. newComment = StringField('评论') 11. submit = SubmitField('提交新评论') 12. 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!"] 13. @app.route("/", methods=["GET", "POST"]) 14. def index(): 15. form = CommentForm() 16. if form.validate\_on\_submit(): 17. comment = form.newComment.data.strip() 18. if comment: 19. dataset.append(comment) 20. return redirect(url\_for('index')) 21. query = request.args.get('content', '') 22. sub\_dataset = [x for x in dataset if query.lower() in x.lower()] 23. return render\_template("index.html", form=form, comments=sub\_dataset, query=query) 24. if \_\_name\_\_ == "\_\_main\_\_": 25. app.run(debug=True) |  1. 修改index.html代码如下：  |  | | --- | | 1. <!DOCTYPE html> 2. <html lang="en"> 3. <head> 4. <meta charset="UTF-8"> 5. <title>Web安全实验</title> 6. </head> 7. <body> 8. <h1>Web安全实验</h1> 9. <h2>查询评论</h2> 10. <form action="" method="get"> 11. 搜索内容：<input type="text" name="content" placeholder="搜索内容"> 12. <input type="submit" name="submit" value="提交"> 13. </form> 14. {% if query %} 15. <h4>查询评论 "{{ query }}" 结果如下：</h4> 16. {% else %} 17. <h4>所有的评论如下：</h4> 18. {% endif %} 19. {% for comment in comments %} 20. <div>{{ comment }}</div> 21. <br> 22. {% endfor %} 23. <h2>发布评论</h2> 24. <form action="" method="post"> 25. {{ form.hidden\_tag() }} 26. {{ form.newComment.label }}: {{ form.newComment(size=20) }}<br> 27. {{ form.submit() }} 28. </form> 29. </body> 30. </html> |  1. 在这个修改后的版本中，我们使用了 Flask-WTF 来定义一个简单的表单，它自动包括 CSRF 保护。{{ form.hidden\_tag() }} 负责渲染 CSRF 令牌字段。这样，每次用户提交表单时，Flask-WTF 将验证 CSRF 令牌，确保请求是合法的，从而防御 CSRF 攻击。   我们此时运行app.py后再点击CSRF\_attack.py，显示攻击失败。 | | |
| 结论与体会：  通过本次实验，我深入理解了 Web 安全中几种常见攻击方式：XSS 和 CSRF，以及 SQL 注入的概念和防御策略。通过实际操作实验，我不仅加深了对这些概念的认识，也学会了如何在实际开发中应用相关防御技术。  XSS 攻击及防御：通过构造特殊的 URL 或输入恶意脚本代码到评论中，我成功模拟了反射型和持久型 XSS 攻击。这一过程让我认识到了用户输入验证和转义的重要性。通过使用 escape 函数转义用户输入，我学会了一种有效防御 XSS 攻击的方法。这个方法能够防止恶意脚本执行，保护网站和用户免受攻击。  SQL 注入及防御：我通过在登录框中输入特殊的 SQL 语句成功实现了 SQL 注入攻击，绕过了登录验证。攻击成功后，通过修改代码，采用参数化查询的方式，我学会了如何防御 SQL 注入攻击。这种方法通过将输入作为参数传递给 SQL 语句，有效避免了恶意输入被解释执行的风险。  CSRF 攻击及防御：我尝试通过构造恶意网页自动提交表单的方式发起 CSRF 攻击，但最终通过引入 Flask-WTF 提供的 CSRF 保护机制成功防御了这种攻击。通过实验，我了解到 CSRF 令牌的作用及其在 Web 应用安全中的重要性。  **结论与体会：**  安全意识的重要性：作为一名 Web 开发者，必须具备安全意识，了解常见的 Web 攻击方式和防御策略，这对于开发安全的 Web 应用至关重要。  持续学习和实践：随着 Web 技术的发展，新的安全威胁不断出现。只有不断学习和实践，才能有效地应对这些安全挑战。  工具和库的作用：Flask-WTF 等工具和库提供了方便的防御机制，如 CSRF 保护、用户输入的转义等，能够大大提高开发效率和应用安全性。合理利用这些工具和库，可以让我们更加专注于业务逻辑的开发。  安全策略的综合运用：在实际开发中，应综合运用各种安全策略，如输入验证、参数化查询、使用 HTTPS、设置合理的 HTTP 安全头等，来构建一个多层次的安全防御体系。  通过本次实验，我不仅掌握了一些具体的防御技术，也对 Web 安全的重要性有了更深刻的认识。在未来的学习和工作中，我将继续关注 Web 安全领域的最新动态，不断提高自己的安全防御能力。 | | |
|  | | |