MICHAEL MEDEIROS FERNANDES DA SILVA

MED22557757

DATE: 18/02/24

Lab: 117

**Table of Contents**

**1. Introduction**

- Overview of OWASP Juice Shop and Security Testing

**2. Requirement 1: Security Testing Environment Setup**

- 1.1 Getting Started

- 1.2 Virtual Machine Specifications

- 1.2.1 OWASP Juice Shop Server

- 1.2.2 Security Testing Workstation

**3. Requirement 2: OWASP Juice Shop Exploration**

- 2.1 Features and Functions Tested

- 2.2 Inputs and Outputs

- 2.3 Unexpected Behaviours and Technical Issues

**4. Vulnerability Exploitation**

- 4.1 Requirement 3: Exploit a Cross-Site Scripting Vulnerability

- 4.2 Requirement 4: Exploit a SQL Injection Vulnerability

- 4.3 Requirement 5: Exploit a Broken Access Control Vulnerability

- 4.4 Requirement 6: Exploit an Authentication Bypass Vulnerability

- 4.5 Requirement 7: Exploit an Improper Input Validation Vulnerability

- 4.6 Requirement 8: Exploit a Sensitive Data Exposure Vulnerability

**5. Overall Insights**

- 5.1 Importance of Secure Coding Practices

- 5.2 Utilization of Tools and Resources

- 5.3 Ethical Hacking Techniques

**6. References**

- OWASP Juice Shop GitHub Repository

- Kali Linux and ParrotOS Security Edition

- YouTube Tutorials and GitHub Write-ups

**Requirement 2. OWASP Juice Shop Functionality Testing**

**Features and Functions**

**Checking Out Stuff**: You can see all the items they are selling, get details about a product and see what others think about it.

A screenshot of a computer

Description automatically generated

**Finding What You Want**: There's a search box to help you find specific items by their name or what they're about.

A screenshot of a computer

Description automatically generated

**Getting Into Your Account**: To buy something, you need to log in. You can use your usual login stuff or your Google account if you're new or coming back.

A screenshot of a computer login

Description automatically generated

**shopping cart**: When you find something you want to buy, you throw it in the shopping cart. You can change as many as you want or abandon something if you change your mind.

A screenshot of a computer

Description automatically generated

**Real checkout**: tells you how to choose where your things go, how they get to you and how you're paying. They also talk about not using real information because they are just a test site.

A screenshot of a credit card

Description automatically generated

**Security Stuff**: They've got features to help if you forget your password, want to change it, or make your account extra secure. They also let you see where you last logged in from.

A screenshot of a computer

Description automatically generated

**Keeping Track of Your Order**: You can check where your order's at and when it's supposed to get to you.

A screenshot of a computer

Description automatically generated

**Feedback and Help**: You can tell them what you think, ask for help in a chat, or complain if something wasn't right.

A screenshot of a computer

Description automatically generated

A screenshot of a chat

Description automatically generated

**About us**: Is a brief explanation about the company.

A screenshot of a computer screen

Description automatically generated

**Inputs and Outputs**

**Inputs**: Stuff like your Delivery Address, how many days will take to arrive etc.

A screenshot of a computer

Description automatically generated

**Outputs**: You get to see all the products, find out if your login worked, see your shopping cart updates, get through the checkout, and manage your account.

**What's Not Supposed to Happen (Unexpected Behaviours)**

* **Mistake**: Things like the website messing up when you're trying to do something important.

A screenshot of a login box

Description automatically generated

**Security Vulnerabilities**: Identifying weaknesses that could lead to unauthorized access or other security breaches.

**Technical Issues:** Occurrences when the website fails to function as expected, such as issues with updating your shopping cart or changing your profile picture.

Complementary pictures

A screenshot of a qr code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a black belt

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A green screen with black lines

Description automatically generated

A screenshot of a computer

Description automatically generated

**Requirement 3. Exploit a Cross-Site Scripting vulnerability.**

I successfully exploited a Reflected XSS vulnerability in the OWASP Juice Shop. After accessing the scoreboard to get the right code, I made a mock purchase and signed up. At the order tracking stage, I manipulated the URL by injecting `<iframe src="javascript:alert(`xss`)">` into the URL parameter. This broke the system by executing a JavaScript alert, confirming the XSS attack worked**.**

A screenshot of a computer

Description automatically generated

**Requirement 4. Exploit a SQL injection vulnerability.**

I found a SQL injection vulnerability in Juice Shop by entering `something'` into one of the input fields. If the input is directly included in a SQL query, this extra quote can break the query structure, potentially causing an error. When the site responded unexpectedly, it indicated the input wasn't sanitized, confirming the vulnerability. This test is part of learning to identify security weaknesses responsibly.

**A screenshot of a computer

Description automatically generated**

**Requirement 5. Exploit a Broken Access Control vulnerability.**

I exploited a Broken Access Control vulnerability in Juice Shop by utilizing a SQL injection technique. By inputting `'1 or 1=1--` into a vulnerable input field, I manipulated the SQL query executed by the server. This specific payload forces the SQL query to return true for all rows by adding an unconditional ( something' OR 1=1 --) clause, effectively bypassing any intended access control checks. The double dashes `--` comment out the remainder of the query, preventing syntax errors. The server's response confirmed that access restrictions were bypassed, indicating that the input was not properly sanitized and thereby exposing a critical security flaw. This exercise was conducted to understand and identify security vulnerabilities in web applications as part of ethical hacking practices.

A screenshot of a computer

Description automatically generated

**Requirement 6. Exploit an Authentication Bypass vulnerability.**

I exploited an Authentication Bypass in Juice Shop using Burp Suite. By turning on the intercept feature, I captured the traffic during a password reset request for an email. After turning off intercept, I manipulated the intercepted data in Burp Suite to obtain the password reset token. With this token, I accessed and reset the user's password without authorization, bypassing the authentication mechanism. This method highlighted vulnerabilities in the password reset process.

**A screenshot of a computer

Description automatically generated**

**Requirement 7. Exploit an Improper Input Validation vulnerability.**

I exploited an Improper Input Validation vulnerability on the "Pho Wall" website. By right clicking an image on the left and selecting "Inspect Element," I located its hyperlink. I modified the URL in the hyperlink by replacing "#" with "%23" on both sides of the link. This manipulation leveraged an encoding issue, breaking the site's normal functionality and allowing me to complete the challenge, demonstrating the critical importance of proper input validation in web applications. **A screenshot of a computer

Description automatically generated**

**Requirement 8. Exploit a Sensitive Data Exposure vulnerability.**

I exploited a Sensitive Data Exposure vulnerability on the Prometheus.io website. By researching the site, I discovered the `/metrics` endpoint, which was not properly secured. By appending `/metrics` to the localhost URL in my browser, I accessed sensitive information exposed by this endpoint. This access allowed me to gather data that was crucial for solving a challenge, highlighting the risk associated with exposing sensitive data through unsecured endpoints.

**A screenshot of a computer

Description automatically generated**

**References:**[OWASP-Juice-Shop-Write-Up/juice-shop-writeup.md at master · apox64/OWASP-Juice-Shop-Write-Up (github.com)](https://github.com/apox64/OWASP-Juice-Shop-Write-Up/blob/master/juice-shop-writeup.md)

[Exposed Metrics - Sensitive Data Exposure - OWASP Juice Shop - Walkthrough - Solution (youtube.com)](https://www.youtube.com/watch?v=PZzMh14IEJk)

[Exploiting SQL Injection: The Step-by-Step Guide to OWASP Juice Shop Admin Challenge (youtube.com)](https://www.youtube.com/watch?v=S-KF_YC-Gfc)

[Missing Encoding - Improper Input Validation - OWASP Juice Shop - Walkthrough - Solution (youtube.com)](https://www.youtube.com/watch?v=cEPMmfLSgsg)

**OWASP Juice Shop Exploration and Vulnerability Exploitation Summary**

**Requirement 1: OWASP Juice Shop Exploration**

**Requirement 1: Setup of Virtual Machine and Kali Linux**

**Activity:** Downloaded and set up a virtual machine, then installed Kali Linux to create a secure and isolated environment for conducting web application security testing. This setup provided the necessary tools and platform for exploring and testing the OWASP Juice Shop's security vulnerabilities.

**Requirement 2:** **OWASP Juice Shop Functionality Testing**

**- Features and Functions Tested:** Product browsing, searching, account login, shopping cart management, checkout process, security features, order tracking, feedback options, and company overview.

**- Inputs and Outputs**: Included user details like delivery address and outputs like product listings, login status, and order updates.

**- Unexpected Behaviours and Technical Issues:** Focused on identifying functionality mistakes and security vulnerabilities.

**Requirement 3: Exploit a Cross-Site Scripting Vulnerability**

- Method: Injected a malicious `<iframe>` tag into a URL parameter during order tracking, triggering a JavaScript alert to confirm the XSS attack.

**Requirement 4: Exploit a SQL Injection Vulnerability**

- Method: Identified a SQL injection vulnerability by inserting a malformed input, which led to unexpected site responses.

**Requirement 5: Exploit a Broken Access Control Vulnerability**

- Method: Utilized SQL injection to manipulate the query, bypassing access control and exposing a critical security flaw.

**Requirement 6: Exploit an Authentication Bypass Vulnerability**

- Method: Intercepted password reset traffic with Burp Suite to obtain and manipulate a reset token, bypassing authentication.

**Requirement 7: Exploit an Improper Input Validation Vulnerability**

- Method: On the "Pho Wall" website, manipulated a URL by changing "#" to "%23," exploiting an encoding issue to break site functionality and solve the challenge.

**Requirement 8: Exploit a Sensitive Data Exposure Vulnerability**

- Method: Accessed an unsecured `/metrics` endpoint on Prometheus.io, revealing sensitive data and highlighting the need for endpoint security.

**Overall Insights**

This comprehensive review covered an initial exploration of the OWASP Juice Shop, leading to a deeper understanding of its functionality and potential security vulnerabilities. The hands-on experience with exploiting vulnerabilities emphasized the importance of secure coding practices, thorough input validation, and robust security measures. Utilizing tools like Burp Suite and resources from YouTube and GitHub enhanced the understanding of ethical hacking techniques, underscoring the significance of identifying and mitigating vulnerabilities to enhance web application security.