Web Application Security Assessment Report

Target Application: XSS(cross site scripting) in Acunetix testphp.vulnweb.com Application

Assessment Type: Manual and Automated using Burp Suite, and browser inspection

Tester: Sagar

Date: 21 October 2025

Tools Used:

• Burp Suite

• Browser DevTools

Vulnerabilities Identified

🔁 Reflected XSS

Location: [http://testphp.vulnweb.com/search.php?test=<script>alert(“Hey,This](http://testphp.vulnweb.com/search.php?test=%3cscript%3ealert(“Hey,This) is Sagar”)</script>

Description:

User input is reflected directly in the response without proper encoding or sanitization. This allows attackers to inject malicious JavaScript that executes in the victim’s browser.

Impact:

• Session hijacking

• Redirecting users to malicious sites

• Defacing content

• Stealing cookies or credentials

Proof of Concept:

[<script>alert(“Hey,This](http://testphp.vulnweb.com/search.php?test=%3cscript%3ealert(“Hey,This) is Sagar”)</script>

💾 Stored XSS

Location : <http://testphp.vulnweb.com/search.php?s>ignup.php

Description:

Malicious scripts entered in form fields (e.g., name, email, message) are stored in the backend and rendered when the page is viewed by any user.

Impact:

• Persistent browser hijacking

• Malware delivery

• Credential theft

• Exploiting admin panels

Payload Used:

<script>alert(“Hey,This is Sagar”)</script>

2️⃣ Exploitation Steps

Reflected XSS:

1. Navigate to vulnerable endpoint.

2. Inject payload in query string.

3. Observe script execution in browser.

Stored XSS:

1. Submit form with script payload.

2. Reload page or view stored content.

3. Script executes for all viewers.

3️⃣ Root Cause Analysis

• Lack of input validation and output encoding.

• No Content Security Policy (CSP).

• Forms and query parameters accept raw HTML/JavaScript.

• Server-side logic does not sanitize user input before storing or rendering.

4️⃣ Mitigation Plan

✅ Input Validation

• Use allowlists for expected input types (e.g., only alphanumeric for names).

• Reject or sanitize unexpected characters.

✅ Output Encoding

• Encode all user-generated content before rendering in HTML.

• Use libraries like OWASP Java Encoder or PHP’s .

✅ Content Security Policy (CSP)

• Implement CSP headers to restrict script execution:

http:

Content-Security-Policy: default-src 'self'; script-src 'self'

✅ Form Field Sanitization

• Strip or escape HTML tags in form submissions.

• Use server-side validation in addition to client-side.

5️⃣ Fix Implementation

If you have access to the codebase or are working with the developer:

• PHP Fix Example:

$safe\_input = htmlspecialchars($\_POST['name'], ENT\_QUOTES, 'UTF-8');

• JavaScript Fix Example:

const safeText = document.createTextNode(userInput);

element.appendChild(safeText);

• Database Layer:

Ensure stored data is sanitized before display, not just before storage.

Vulnerability Report: Price Manipulation in Acunetix testphp.vulnweb.com Application

🔍 Overview

The application displays product details including Product ID, Name, Artist, Category, and Price. The price field is client-side rendered and editable via the browser (e.g., through developer tools or intercepting requests). This opens the door to price manipulation attacks, where a malicious user can change the price of an item before checkout.

⚠️ Vulnerability Type

Client-side Price Manipulation

This occurs when the price of a product is stored or validated only on the client side (browser), allowing attackers to modify it before submitting the order.

🧪 Exploitation Steps

Scenario:

1. User adds a product to the cart.

2. The price is displayed as $800 after it will show $1 in the second screenshot.

3. Using browser developer tools or intercepting the request with Burp Suite, the attacker modifies the price field to a lower value.

4. The manipulated price is submitted to the server.

5. If the server does not validate the price against its database, the attacker is charged the altered amount.

Tools Used:

• Burp Suite (as shown in your screenshot)

🔓 Impact

• Financial Loss: Attackers can purchase items at reduced or zero cost.

• Inventory Fraud: Manipulated transactions may affect stock and accounting.

• Reputation Damage: Exploits like this can erode customer trust.

• Legal Risk: If exploited in production, it may lead to lawsuits or regulatory penalties.

🛡️ Mitigation Strategies

Server-Side Validation

• Always validate product price and details on the server using the product ID.

• Never trust client-side data for sensitive fields like price, discount, or quantity.

Use Secure Session Tokens

• Tie cart items to session tokens and validate them against server-side records.

• Prevent tampering by using HMAC or signed tokens for cart data.

Recalculate Price on Server

• During checkout, fetch the latest price from the database.

• Recalculate totals server-side before finalizing the transaction.

Disable Client-Side Price Editing

• Avoid rendering editable price fields in HTML or JavaScript.

• Use read-only fields and disable form manipulation via JavaScript.

Implement Integrity Checks

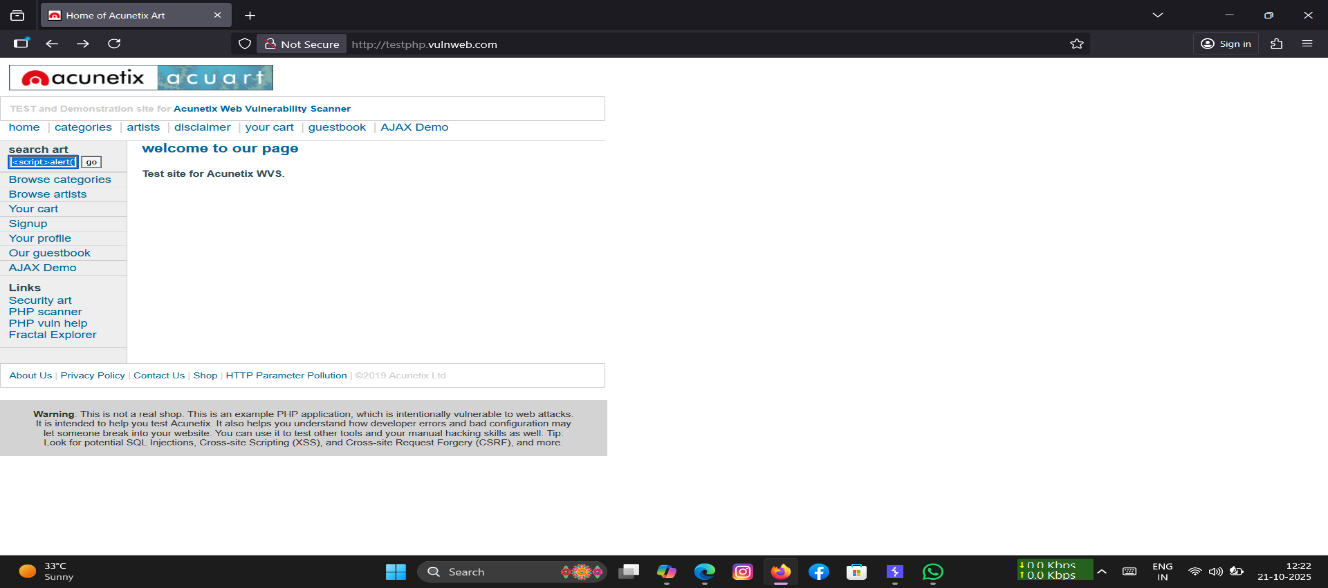
• Use hash-based integrity checks for cart data.

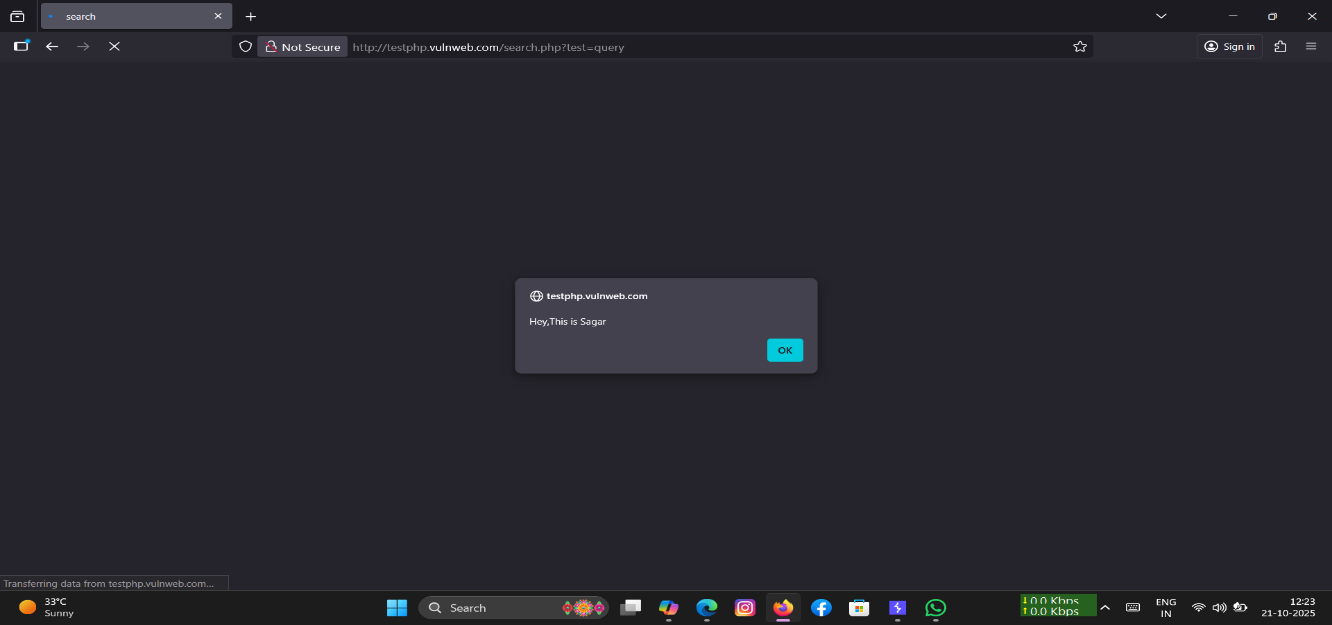
• Compare submitted hash with server-generated hash to detect tampering.

Logging and Monitoring

• Log all cart modifications and checkout attempts.

• Flag suspicious behavior like price changes or unusually low totals.





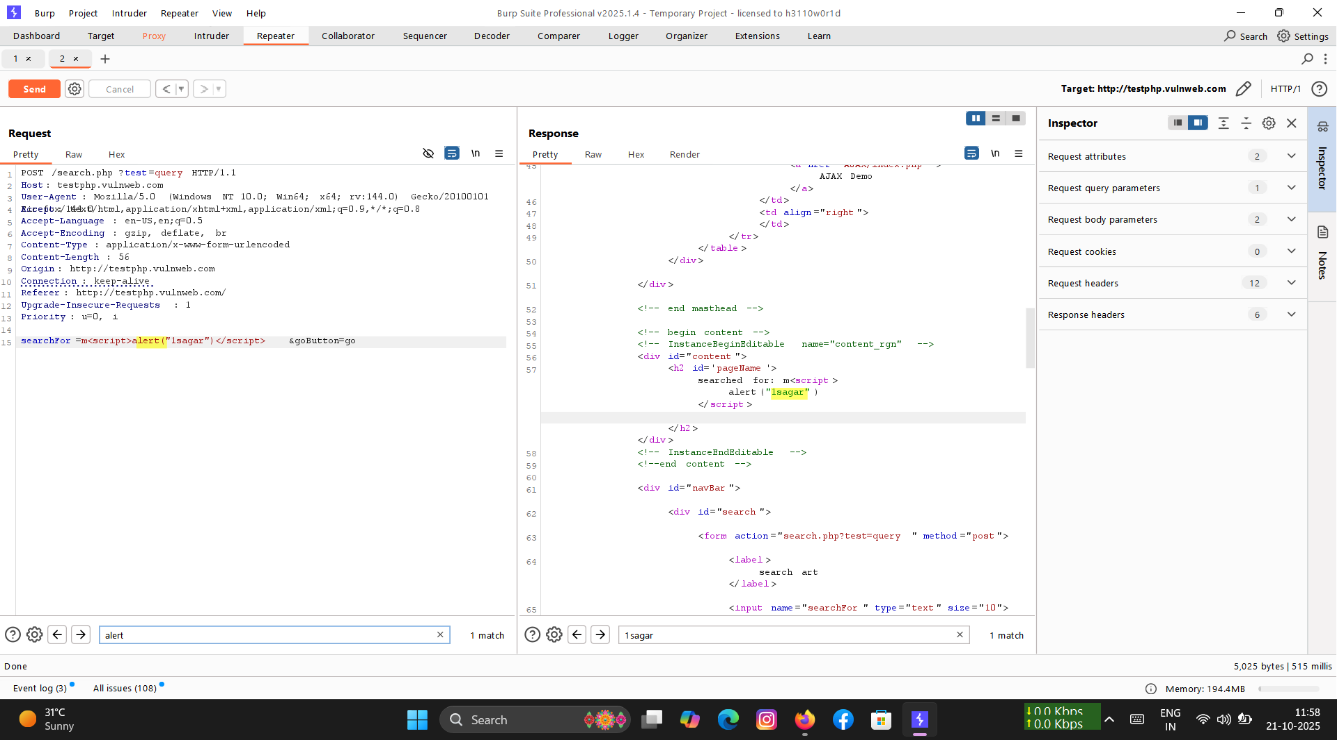
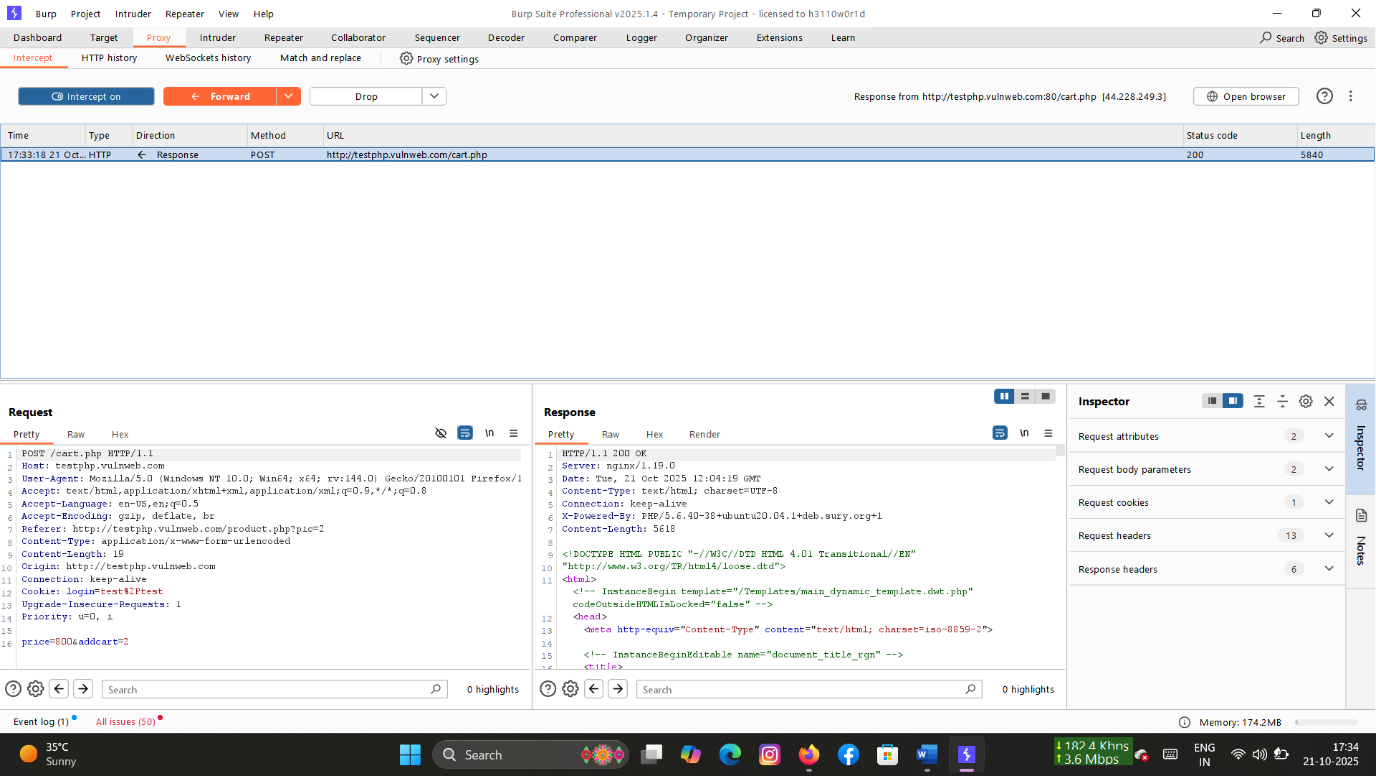


Figure 1,2,3: Reflected XSS payload execution



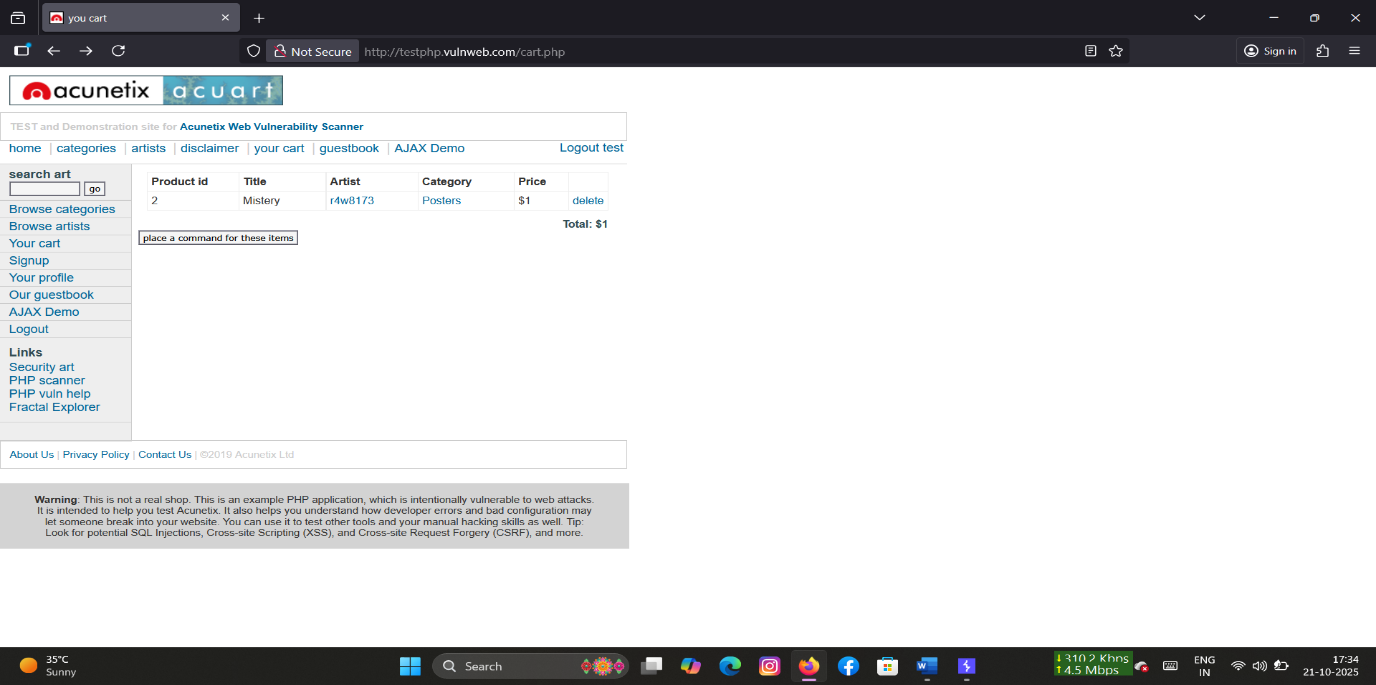
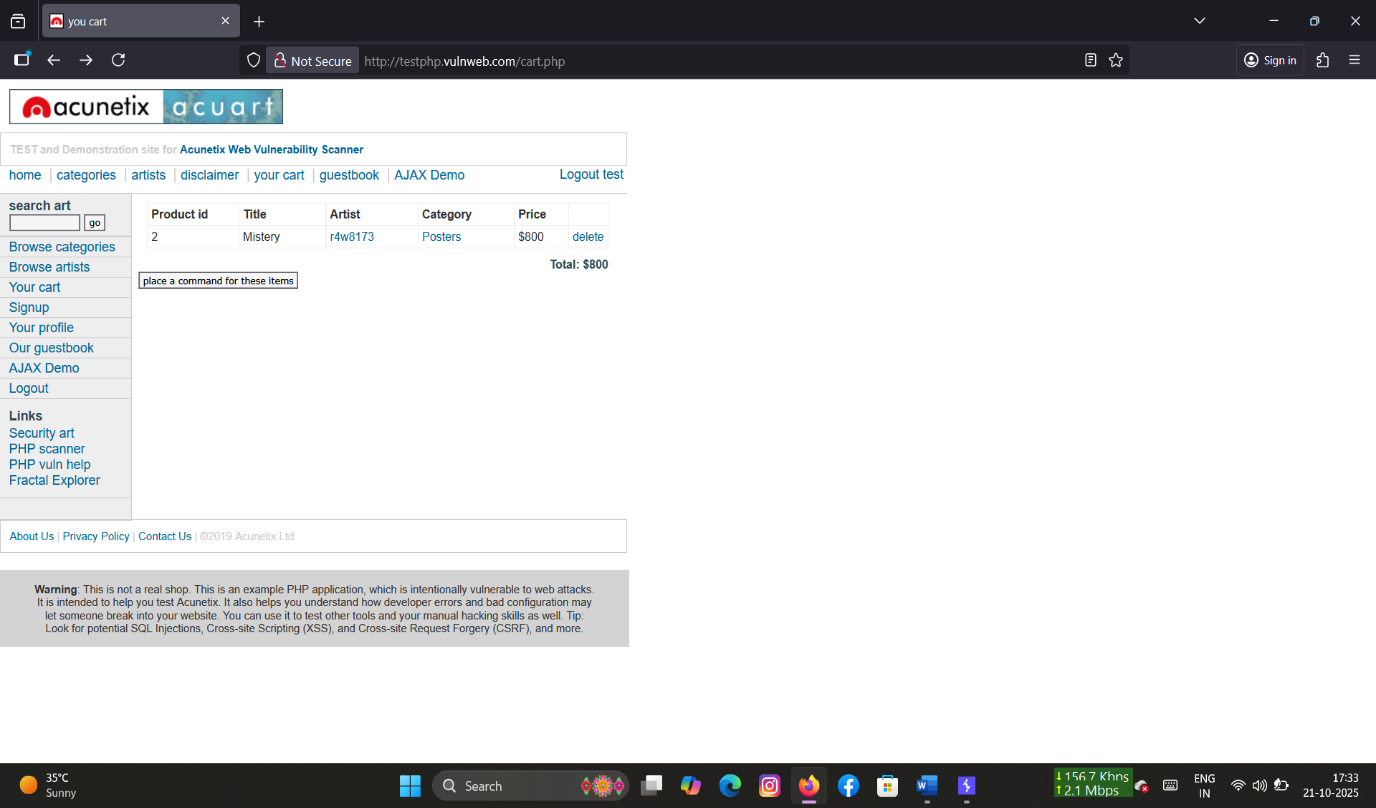


Figure 4,5: Price Manipulation



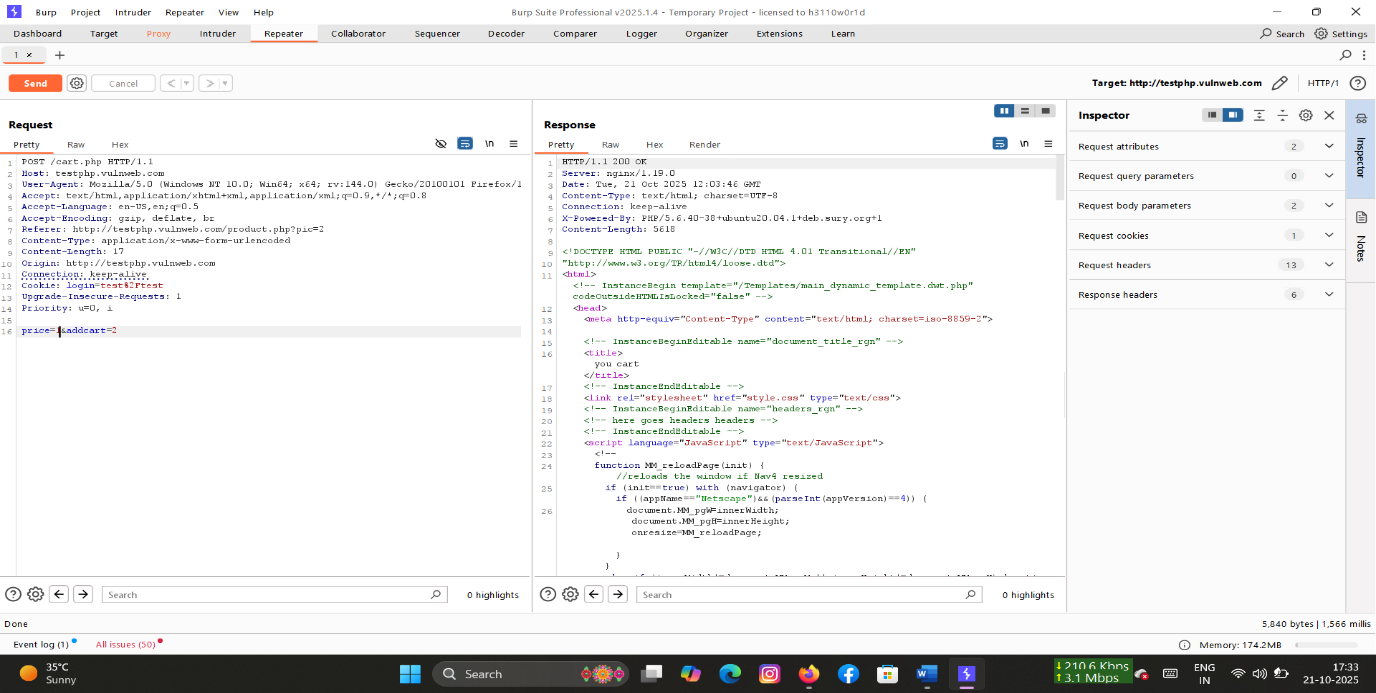
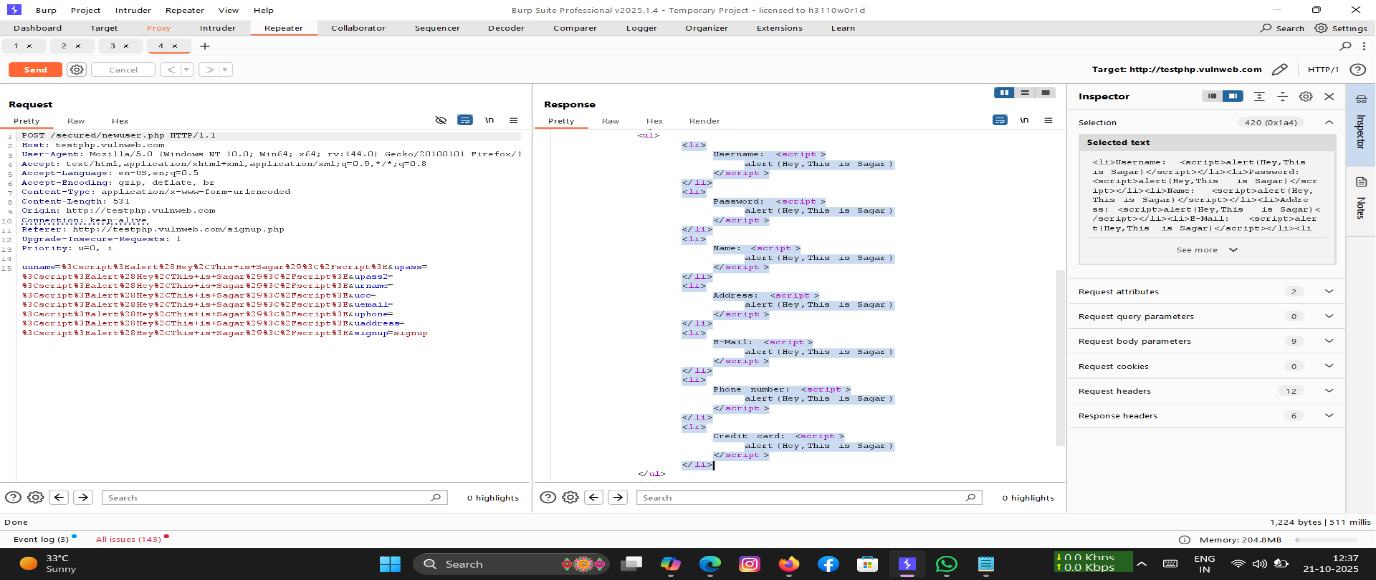


Figure 6,7 : Price Manipulation



A screenshot of a computer

AI-generated content may be incorrect.

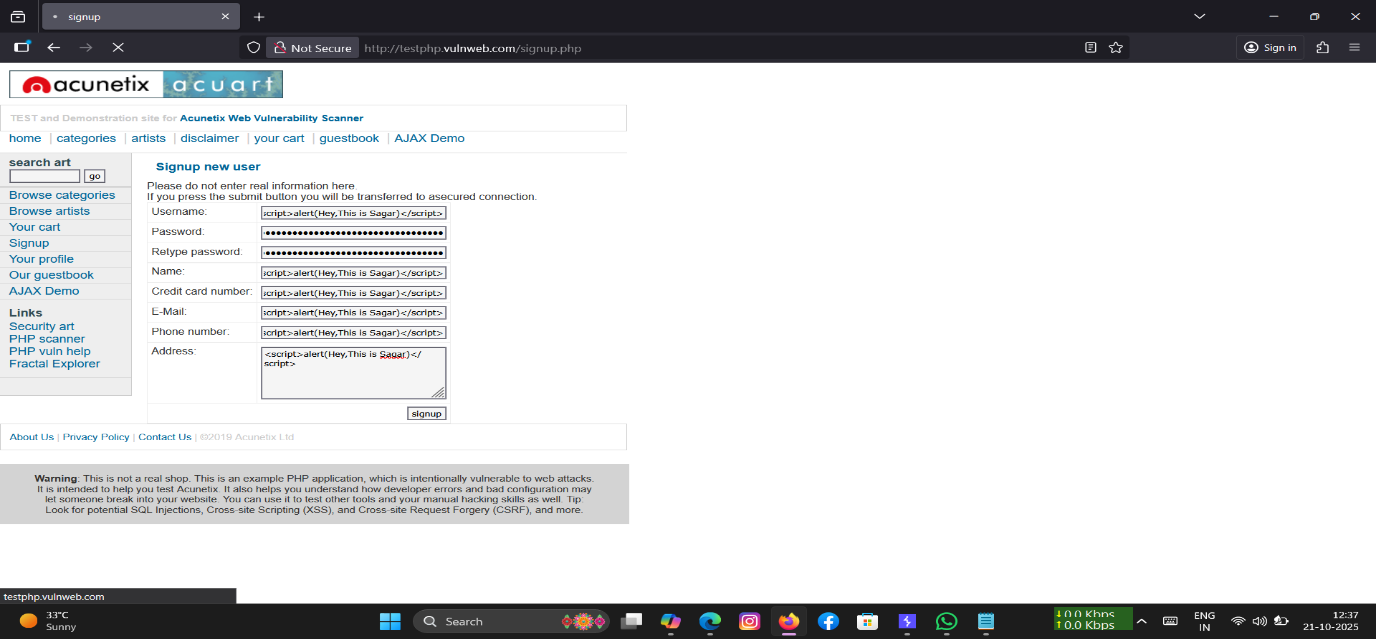


Figure 8,9,10: Stored XSS in new user details form