Professional Vulnerability Report

Title Page

- Report Title: Vuln-bank Vulnerability Assessment
- Author: Jaseel K
- Date: 2025-10-28

Executive Summary

A security test was conducted against the "Vulnerable Bank" web application. The test revealed five (5) critical vulnerabilities, consisting of several high-severity issues. These vulnerabilities put the application and users at high risk for unauthorized access, data theft, and money laundering.

Remediation should occur immediately, prioritizing the repair of the compromised access controls and server-side validation logic.

Severity Definitions

- Critical: Enables full system takeover or blatant, large-scale financial theft.
- High: Enables an intruder to read or change any user's information, or assume control of another user's account.
- Medium: Reveals sensitive information or application vulnerabilities that may be exploited in a larger attack.
- Low: Low-impact issues that have little impact on the business (e.g., "best practice" suggestions).

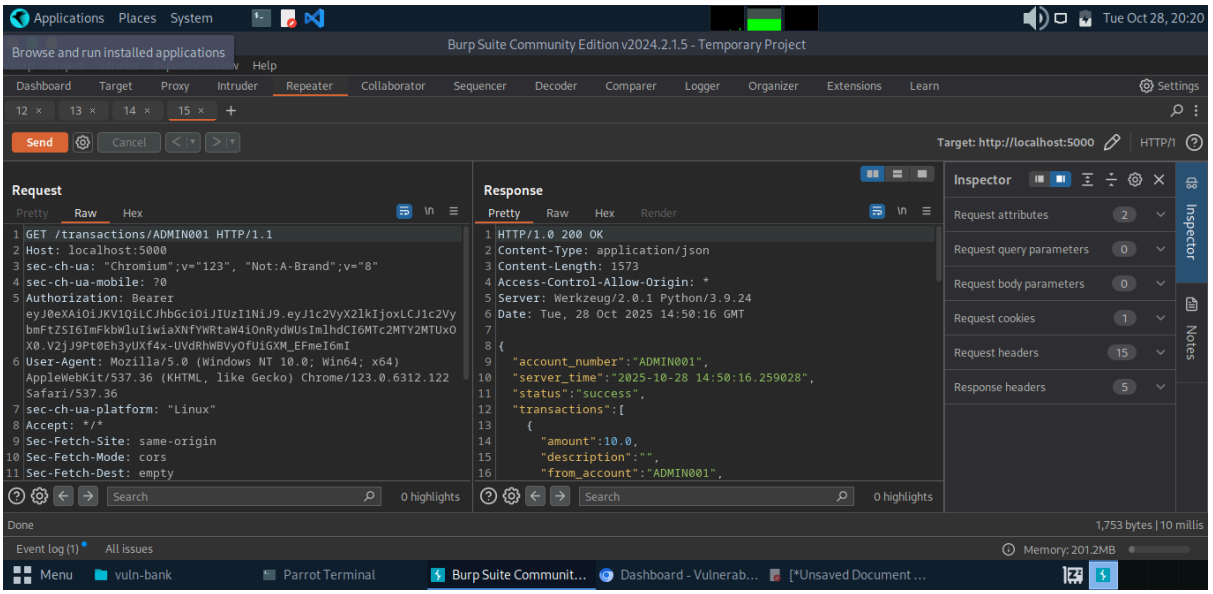
Vulnerability Findings

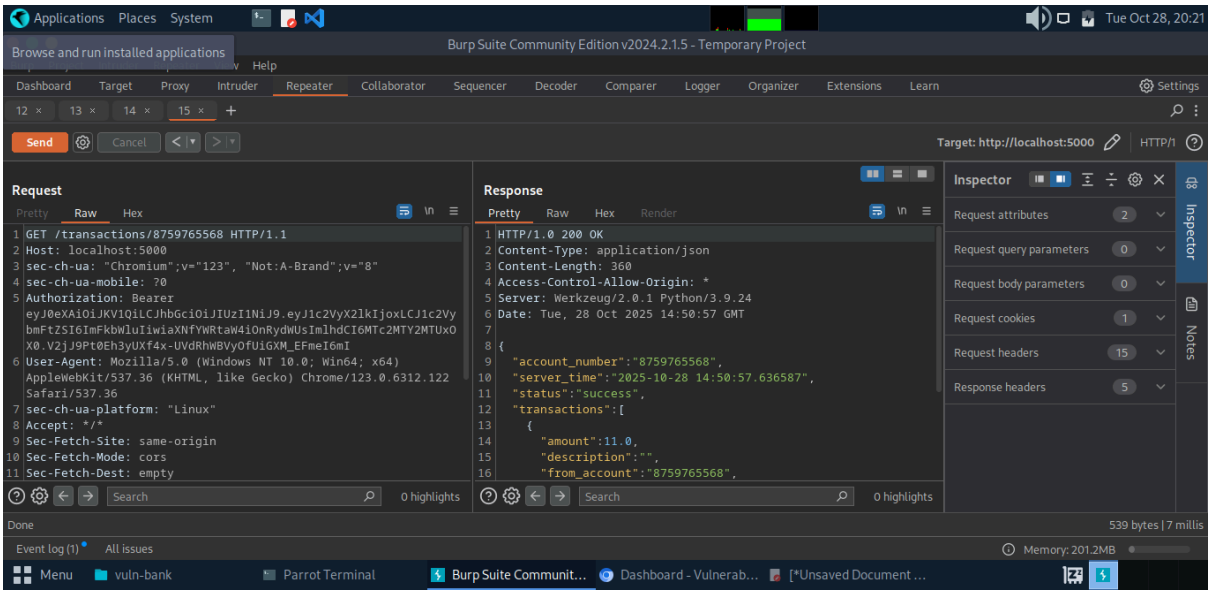
Here are the detailed findings from the assessment.

Finding 1: Broken Access Control (IDOR)

- Severity: Critical
- Description: The application fails to validate that a logged-in user is authorized to access the data they are requesting. An authenticated attacker can view the private transaction history of any other user on the platform by simply changing the account number in the API request.
- Impact: This allows any user to steal the complete financial history of any other user, including the admin, leading to a massive data breach and loss of customer trust.
- Steps to Reproduce:
    - Log in as ADMIN001.
    - Capture the request for transaction history: GET /transactions/ADMIN001

    - Change the request to GET /transactions/8759765568 (a different user's account).
    - The server responds with the private transaction data for account 8759765568.
- Remediation: Implement Server-Side Authorization. On the server, before fetching any data, check if the logged-in user's session_account_number is equal to the requested_account_number. If they do not match, return a 403 Forbidden error.



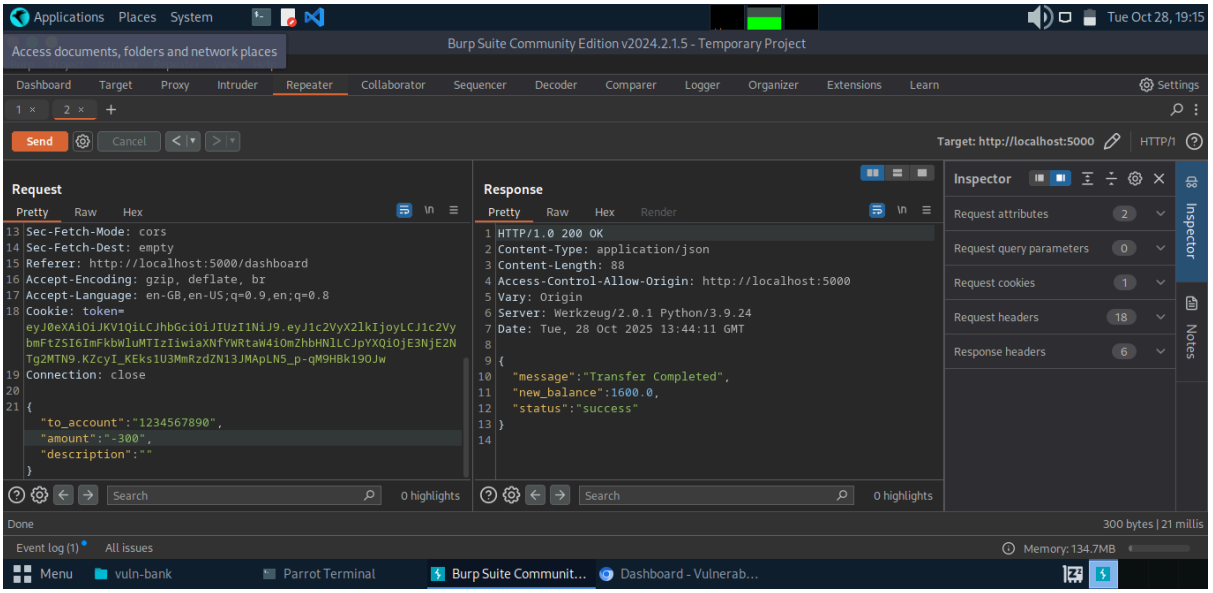


Finding 2: Business Logic Flaw (Negative Fund Transfer)

- Severity: Critical
- Description: The fund transfer endpoint (/api/v1/transfer) does not properly validate the amount parameter. The server accepts a negative number, processing it as a valid transaction.
- Impact: An attacker can "transfer" a negative amount from their account to another, effectively pulling money from the victim and adding it to their own balance. This allows for direct theft of funds.
- Steps to Reproduce:

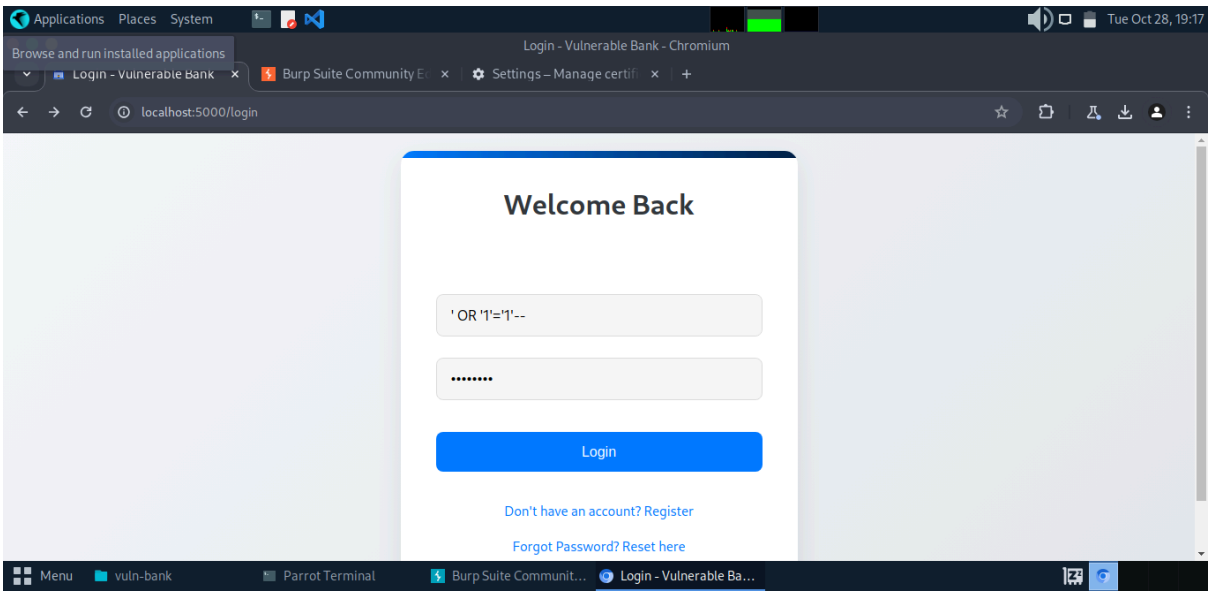
- Log in and capture a valid transfer request.
- Send the request to Burp Repeater.
- Change the JSON body to use a negative value: {"to_account": "...", "amount": "-300"}
- Send the request. The server responds with 200 OK and "Transfer Completed."

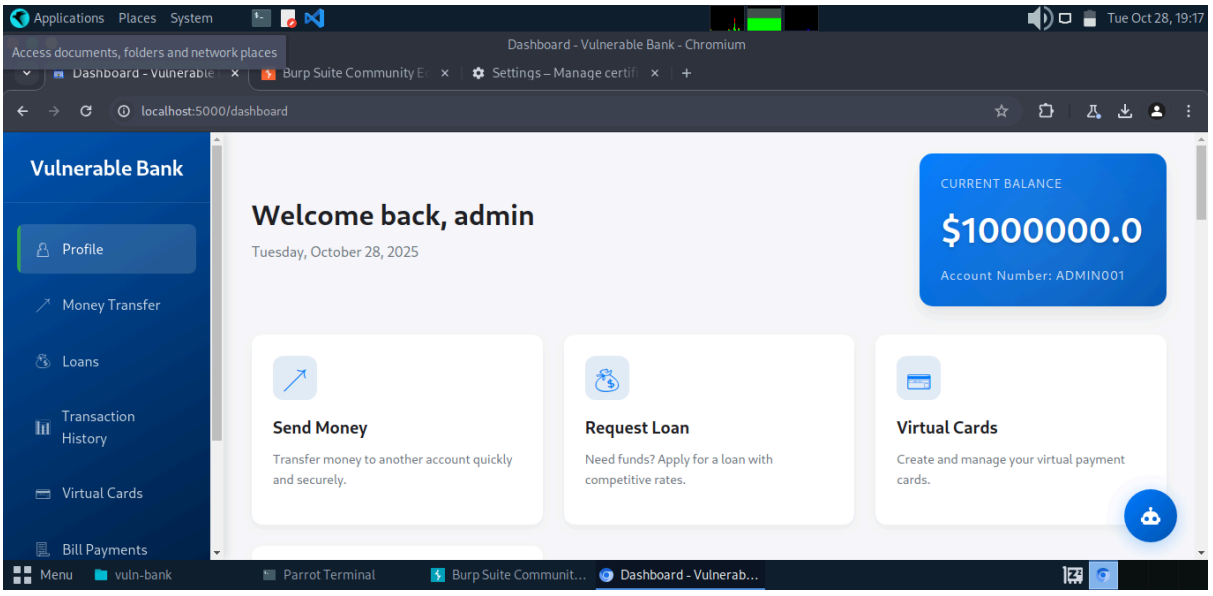
- Remediation: Implement Strict Server-Side Validation. The server must reject any transaction where the amount parameter is less than or equal to zero.



Finding 3: SQL Injection (Authentication Bypass)

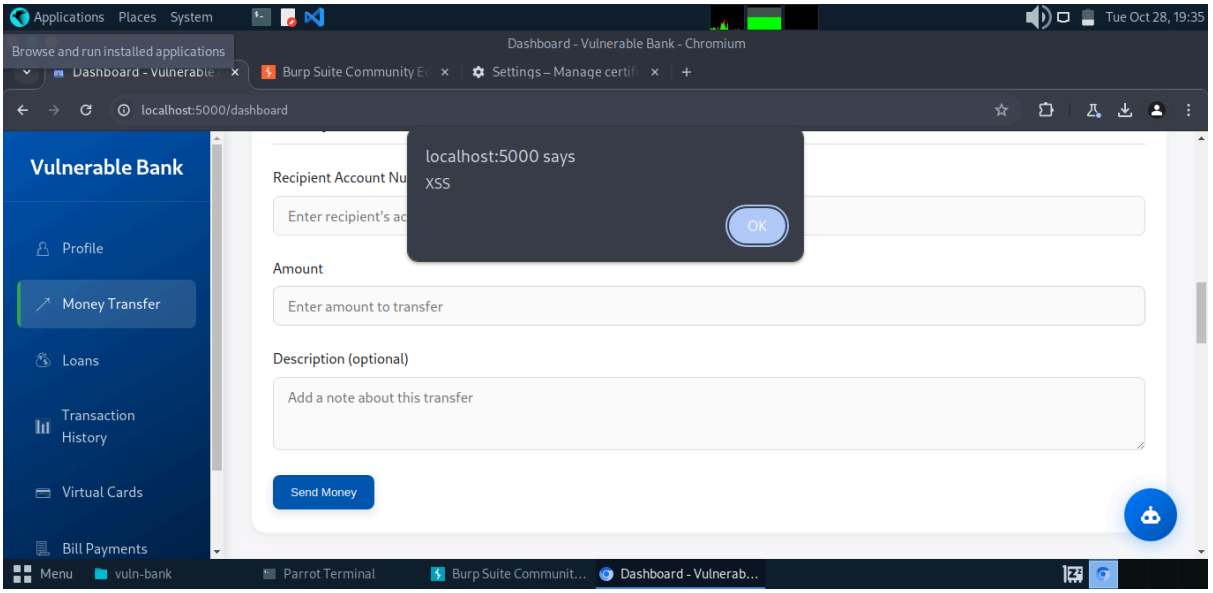
- Severity: Critical
- Description: The login form (/login) is vulnerable to a classic SQL Injection attack. An attacker can use a malicious string in the username field to bypass the password check.
- Impact: This allows an attacker to gain unauthorized access to any user's account, including the administrator account, giving them full control over the application.
- Steps to Reproduce:
  - Go to the login page.
  - In the username field, enter: ' OR '1'='1'--
  - In the password field, enter any text.
  - The user is successfully logged in as the "admin" account.
- Remediation: Use Parameterized Queries (Prepared Statements). Never concatenate user input directly into a database query. Use parameterized queries, which separate the SQL command from the user data, making this attack impossible.





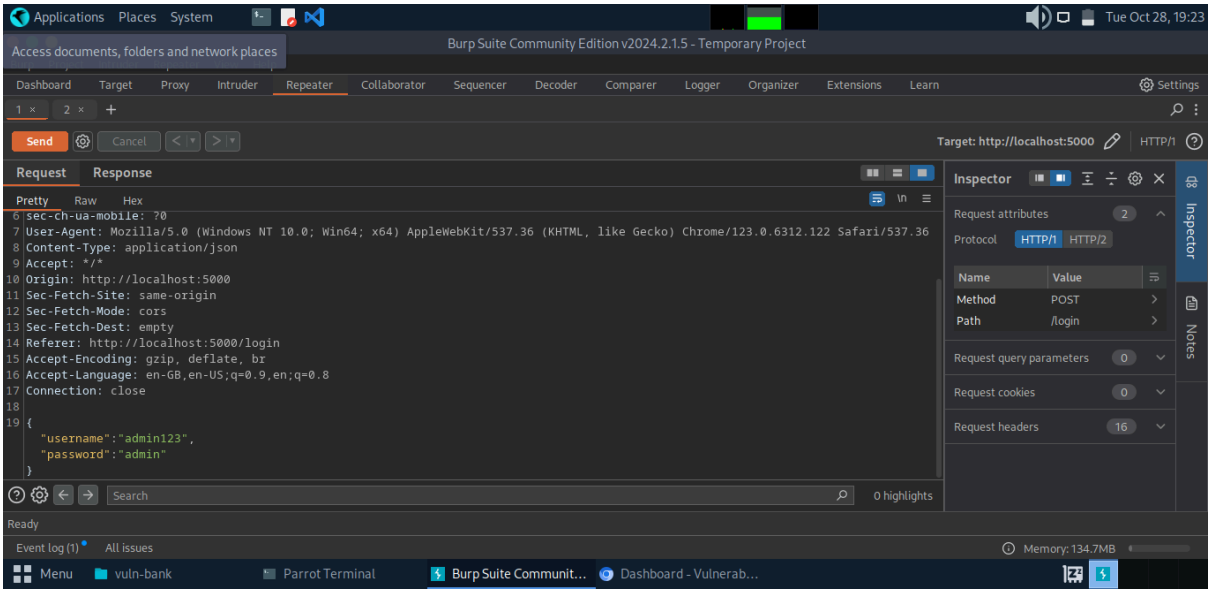
Finding 4: Stored Cross-Site Scripting (XSS)

- Severity: High
- Description: The application fails to sanitize user input before storing it in the database and rendering it back to the user. An attacker can inject malicious HTML and JavaScript (e.g., in a "description" or "payee" field).
- Impact: An attacker could use this flaw to steal other users' session cookies, take over their accounts, redirect them to malicious websites, or capture their keystrokes.
- Steps to Reproduce:
    - Go to a field where text is saved and displayed (e.g., "Money Transfer" description).
    - Enter the payload: <img src=x onerror=alert('XSS')>
    - Save/submit the form.
    - When the page reloads or is viewed, the JavaScript executes, and an alert box appears.
- Remediation: Implement Output Encoding. Before rendering any user-supplied data in an HTML page, encode it so the browser treats it as plain text, not as executable code. (e.g., < becomes <).



Finding 5: Sensitive Data Exposure (Password in Plain Text)

- Severity: Medium
- Description: The login form submits user credentials to the server in plain text over HTTP, without encryption.
- Impact: An attacker on the same network (e.g., public Wi-Fi) can easily "sniff" the network traffic and steal the usernames and passwords of all users.
- Steps to Reproduce:
    - Intercept the login request (POST /login) in Burp Suite.
    - Observe that the request is sent via http:// (not https-://).
    - The request body clearly shows the password in plain text: "password": "admin".
- Remediation: Implement HTTPS Sitewide. The entire application must be served over HTTPS (SSL/TLS) to encrypt all traffic between the user and the server.



Conclusion

The "Vulnerable Bank" application is in a highly insecure state. The high number of critical-severity vulnerabilities demonstrates a lack of fundamental security controls, particularly in input validation and authorization.

We strongly recommend the development team prioritize these findings for immediate remediation to prevent financial loss and data compromise.