Vulnerability Assessment Report
Bank Web Application (Docker Deployment)

Maheshwar Anup

October 27, 2025

1 Executive Summary

A security assessment was conducted on the vulnerable banking web application deployed locally via Docker. The objective was to identify and report critical security vulnerabilities. The application exhibited several high-severity security deficiencies, including Cross-Site Scripting (XSS), Cross-Origin Resource Sharing (CORS) misconfiguration, and a lack of a comprehensive Content Security Policy (CSP). Immediate remediation steps are required to mitigate the risk of unauthorized data access, session compromise, and client-side attacks.

2 Scope and Methodology

2.1 Scope

The assessment focused on the `vuln-bank` application, which was deployed locally in a Docker container environment.

2.2 Deployment Methodology

The target application was deployed using the following procedure:

1. Environment setup was confirmed with Docker installed.
2. The application repository was cloned:

```
git clone https://github.com/Commando-X/vuln-bank
```

3. The container was built and started in detached mode:

```
docker compose up --build -d
```

4. The application was accessed via `http://localhost:8080` (or `http://localhost:5000` as noted in some documentation) and authenticated using default credentials to access all functionalities for testing.

2.3 Assessment Tools and Techniques

The vulnerability assessment employed a combination of automated scanning tools and manual inspection techniques. Tools utilized included:

- OWASP ZAP (Proxy and Scanner)
- Burp Suite
- Nikto
- SQLMap

Manual testing focused on common web vulnerabilities, specifically SQL injection, Cross-Site Scripting (XSS), and configuration weaknesses.

3 Findings

The following vulnerabilities were identified during the security assessment.

3.1 CORS Misconfiguration

CORS Misconfiguration (High Severity)

Description: The application's Cross-Origin Resource Sharing (CORS) policy is configured improperly, allowing access to its resources from any origin (`*`). This was identified during proxy analysis.

Impact: This critical misconfiguration allows an attacker to launch malicious web pages on any domain that can make authenticated requests to the vulnerable application. This can lead to unauthorized access to sensitive data and the execution of sensitive actions (e.g., fund transfers) from malicious websites (CSRF-like attacks).

Recommendation: Restrict CORS Policies: The server must be configured to only allow origins explicitly trusted by the application. The use of a wildcard (`*`) should be removed from the `Access-Control-Allow-Origin` header.

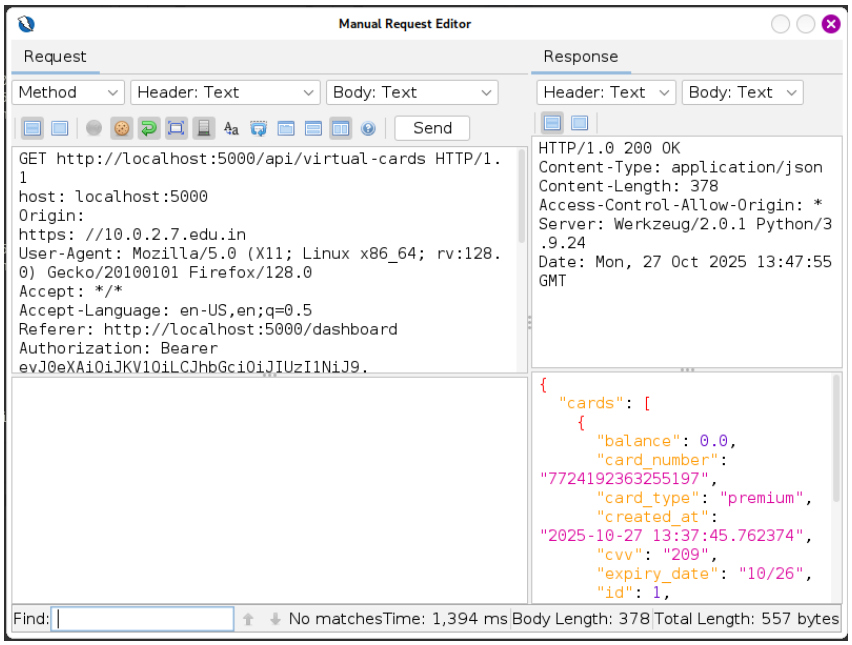

Figure 1: OWASP ZAP identifying the CORS Misconfiguration

3.2 Content Security Policy Deficiency

Missing Content Security Policy (CSP) on Dashboard (Medium Severity)

Description: The application's dashboard page is missing a robust Content Security Policy (CSP) header. This security layer is essential for mitigating client-side injection attacks.

Impact: The absence of a proper CSP significantly increases the risk of successful Cross-Site Scripting (XSS) attacks, as it provides no mechanism to control the sources of content (scripts, styles, images) that the browser is permitted to load.

Recommendation: Implement CSP: A strict CSP should be implemented across all pages, particularly the dashboard. This policy should restrict script execution to trusted domains and disallow inline scripts and `eval()` functions.

3.3 Stored Cross-Site Scripting (XSS)

Stored Cross-Site Scripting (XSS) in Payment Form (High Severity)

Description: The bill payment form and associated backend processing do not properly sanitize user-supplied input before it is rendered on the page, allowing for the injection and persistence of mali-

Figure 2: OWASP ZAP finding related to CSP vulnerability

cious scripts. The vulnerability was successfully exploited by submitting a JavaScript payload (e.g., `<script>alert('XSS')</script>`) in the payment description field.

Impact: An attacker can exploit this vulnerability to steal session cookies, hijack user sessions, deface the website, or redirect users to malicious external sites. Since the payload is stored, any user viewing the affected page will be compromised.

Recommendation: Implement Input Validation and Output Encoding:

- All user input must be validated on the server-side to ensure it conforms to expected formats and content.
- Output encoding (context-aware escaping) must be applied to all user-controlled data before it is rendered in the HTML page to prevent the browser from interpreting the data as executable code.

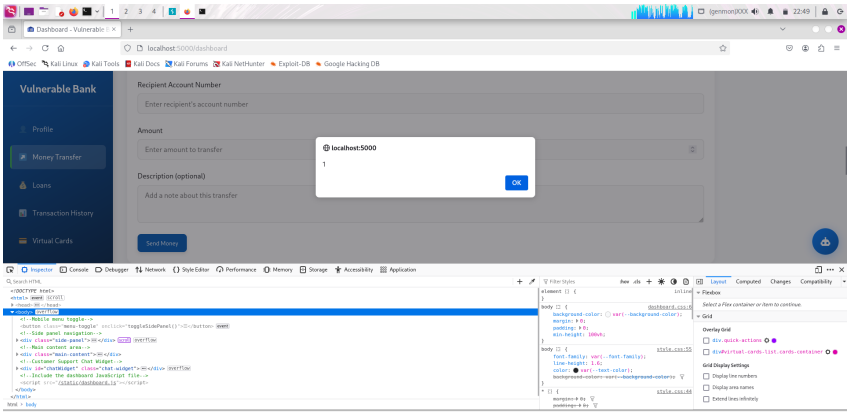

Figure 3: Injection of XSS payload in the payment form

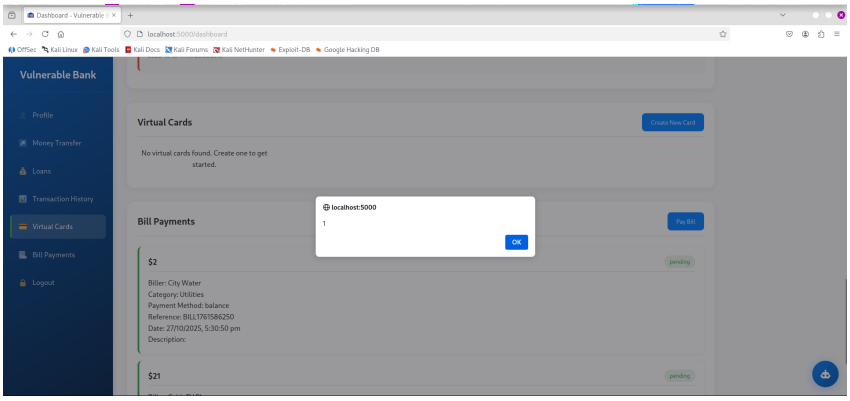

Figure 4: Successful exploitation of the XSS vulnerability (e.g., on the bill payment confirmation page)

4 Conclusion and Remediation Summary

The assessment confirms the existence of several critical and high-severity vulnerabilities within the banking application, notably XSS and significant configuration flaws (CORS and CSP). These issues expose the application and its users to a high risk of data theft and unauthorized activity.

4.1 Remediation Priorities

The following actions should be prioritized for immediate remediation:

1. Implement strict output encoding on all user-supplied input to resolve the Cross-Site Scripting (XSS) vulnerability.
2. Configure the CORS policy to allow only explicitly trusted origins, eliminating the use of the wildcard character (*).
3. Deploy a comprehensive and strict Content Security Policy (CSP) header across the entire application to provide an effective defense-in-depth layer against injection attacks.

5 Post-Assessment Cleanup Note

For resource management, the Docker container used for this assessment must be stopped and removed by running the following command:

```
docker compose down
```