🔍 ThreatScope Security Analysis Report

Date: 5/16/2025, 8:02:47 PM

📋 SECTION 0: Executive Summary

- Potential exposure of sensitive user data (email, password) if HTTPS is not enforced.
- Lack of CSRF protection can lead to unauthorized actions performed by users.
- Insufficient client-side validation can result in backend processing of invalid data.

Overall Security Posture: Needs Improvement

Threat Impact: Moderate - User data compromise and potential account takeover.

Business Risk: User data exposure, damage to reputation, potential regulatory fines.

Suggested Next Action: Conduct a security re-architecture focusing on input validation, authentication, and authorization mechanisms.

📝 SECTION 1: Summarized Screen Overview

Attribute	Value
Form Action URL	/login
HTTP Method	POST
Input Fields	email (type: email), password (type: password)
Hidden Fields	None visible in the provided HTML

🔐 SECTION 2: Security Design Analysis

- **HTTPS Usage:** Critical - Must be enforced to protect data in transit.
- **HTTP Method:** POST is appropriate for login.
- **Security Attributes:** Recommended - `autocomplete="off"` for password field to prevent password caching. Missing from provided HTML.
- **Hidden Field Usage:** Not applicable based on the provided HTML. Consider adding CSRF token.

- **CSRF Token:** Missing. This is a critical vulnerability.
- **Client-Side Validation:** Minimal. Should implement more robust validation for email format and password complexity.
- **Information Leakage:** Low. Data is sent via POST, minimizing exposure in URLs.

🧠 SECTION 3: Threat Modeling (STRIDE-based)

Spoofing

- **Threat:** Attacker spoofs legitimate user by guessing/obtaining credentials.
- **Attacker Goal:** Gain unauthorized access to user accounts.
- **Scenario:** Attacker uses leaked credentials from a data breach to log in.
- **Likelihood:** Medium | **Impact:** High

Tampering

- **Threat:** Attacker manipulates login request (e.g., modifies email or password).
- **Attacker Goal:** Bypass authentication.
- **Scenario:** While unlikely due to POST, manipulating the request outside the UI could cause unintended behavior if not sanitized server-side.
- **Likelihood:** Low | **Impact:** Medium

Repudiation

- **Threat:** User denies performing login action.
- **Attacker Goal:** Avoid responsibility for actions performed within the application.
- **Scenario:** Difficult to prove user logged in without sufficient logging and auditing mechanisms.
- **Likelihood:** Low | **Impact:** Low

Information Disclosure

- **Threat:** Attacker intercepts or obtains sensitive information (e.g., password).
- **Attacker Goal:** Steal credentials, personal data.
- **Scenario:** Without HTTPS, attacker intercepts login credentials over an insecure network. Weak server-side logging of passwords.
- **Likelihood:** Medium | **Impact:** High

Denial of Service

- **Threat:** Attacker floods the login endpoint with requests.
- **Attacker Goal:** Make the login functionality unavailable to legitimate users.
- **Scenario:** Botnet sends a large number of login requests, exhausting server resources.
- **Likelihood:** Low | **Impact:** Medium

Elevation of Privilege

- **Threat:** Attacker gains higher-level access using login form vulnerabilities.
- **Attacker Goal:** Gain administrative or super-user privileges.
- **Scenario:** Unlikely, but a successful SQL injection could allow an attacker to modify user roles if not properly mitigated.
- **Likelihood:** Low | **Impact:** High

🛡️ SECTION 4: Actionable Security Recommendations

- **Enforce HTTPS:** Redirect all HTTP traffic to HTTPS to protect data in transit.
- **Implement CSRF Protection:** Add a CSRF token to the login form and validate it on the server-side.
- **Add `autocomplete="off"`:** Include this attribute in the password input field.
- **Implement Strong Client-Side Validation:** Validate email format and password complexity using JavaScript. However, always validate on the server-side.
- **Implement Rate Limiting:** Protect against brute-force attacks by limiting the number of login attempts from a single IP address.
- **Use Secure Password Storage:** Use bcrypt or Argon2 for password hashing on the server-side.
- **Implement Account Lockout:** Lock accounts after multiple failed login attempts.
- **Use secure HTTP headers:** Implement secure headers such as HSTS, X-Frame-Options, X-Content-Type-Options, and Content-Security-Policy.

✅ SECTION 5: Positive Security Observations

- Use of POST method for submitting credentials.

📊 SECTION 6: Security Score (0–10)

Security Score: **4 / 10**

Justification: The login form uses POST, which is good, but lacks critical security measures such as HTTPS enforcement and CSRF protection, and secure headers, leading to a moderate security risk.

🧪 SECTION 7: Manual Security Checklist for Reviewers

- ☐ Does the form use HTTPS?
- ☐ Are sensitive fields masked or obfuscated?
- ☐ Do hidden fields include secure tokens?
- ☐ Are inputs validated and sanitized client-side?
- ☐ Is CSRF protection implemented and tested?
- ☐ Are secure HTTP headers configured?

- [ ] Can input lead to XSS or data injection?
- [ ] Is data stored in cookies/localStorage secure?

Report generated by ThreatScope Security Extension