# **2Security Defenses Implementation Report**

## **Overview**

This report summarizes the security defenses integrated into the Flask-based IoT Dashboard application. Each defense layer addresses specific threat vectors, ensuring a robust, multi-layered security posture.

## **1. Brute-Force Protection**

**Objective:** Prevent automated, repeated login attempts that aim to guess valid credentials.

* **Failed-Attempt Tracking**: Records each unsuccessful login by {username}:{IP} with a timestamp.
* **Attempt Threshold**: Allows up to **3** failed attempts within a **2-hour** window (lockout\_duration).
* **Account Lockout**: If the threshold is exceeded, the account is locked for the remainder of the 2-hour window.
* **Remaining Attempts Indicator**: Calculates and displays how many attempts remain before lockout occurs.

**Benefit:** Mitigates credential stuffing and dictionary attacks by slowing down attackers and alerting users.

## **2. CSRF Protection**

**Objective:** Prevent attackers from forging state-changing requests on behalf of authenticated users.

* **Token Generation**: Generates a 32-byte random token per session and embeds it in forms (csrf\_token).
* **Token Verification**: On each POST (after login), the @csrf\_protection decorator compares the submitted token against the session-stored token using a constant-time HMAC check.
* **Failure Handling**: Redirects to the login page with an error message if the token is missing or invalid.

**Benefit:** Stops cross-site request forgery attacks by ensuring form submissions originate from genuine user sessions.

## **3. JWT-Based Session Management**

**Objective:** Securely manage user sessions with cryptographically signed tokens.

* **JWT Payload**: Includes username, issuance time (iat), expiration (exp), and a unique session\_id.
* **Cookie Storage**: Stored as an HTTP-only session\_token cookie with SameSite enforcement.
* **Server-Side Tracking**: Maintains an in-memory map of active session\_ids with last-activity timestamps.
* **Session Validation**: @require\_secure\_session decorator verifies signature, expiration, and presence in the active sessions map, then refreshes the timestamp.

**Benefit:** Combines stateless token validation with server-side control, preventing replay of expired tokens and session fixation.

## **4. Multi-Factor Authentication (MFA)**

**Objective:** Add a second factor to significantly increase login security.

* **TOTP Secret Generation**: Uses pyotp to generate a base32 secret per user.
* **QR Code Provisioning**: Renders a provisioning URI as a scannable QR code via the qrcode library.
* **Backup Codes**: Generates eight one-time-use hex codes for recovery if the authenticator device is unavailable.
* **Enforcement Decorator**: @require\_mfa ensures protected routes redirect to MFA verification until the user confirms a valid TOTP code.
* **Verification**: Validates TOTP codes with pyotp and consumes backup codes securely.

**Benefit:** Strongly reduces the risk of account compromise even if the primary password is stolen.

## **5. Session Cleanup**

**Objective:** Remove stale session entries to minimize memory footprint and close inactive sessions.

* **Expiration Policy**: Defines a **1-hour** maximum inactivity interval (session\_duration).
* **Cleanup Routine**: Periodically invoked to purge any session IDs older than the inactivity threshold.

**Benefit:** Prevents accumulation of expired sessions and enforces timely re-authentication.

## **Conclusion**

By integrating these layered defenses—brute-force throttling, CSRF tokens, JWT session management, MFA, and session cleanup—the IoT Dashboard achieves a comprehensive security model. Each mechanism complements the others, ensuring resilience against common web-based attacks and unauthorized access.