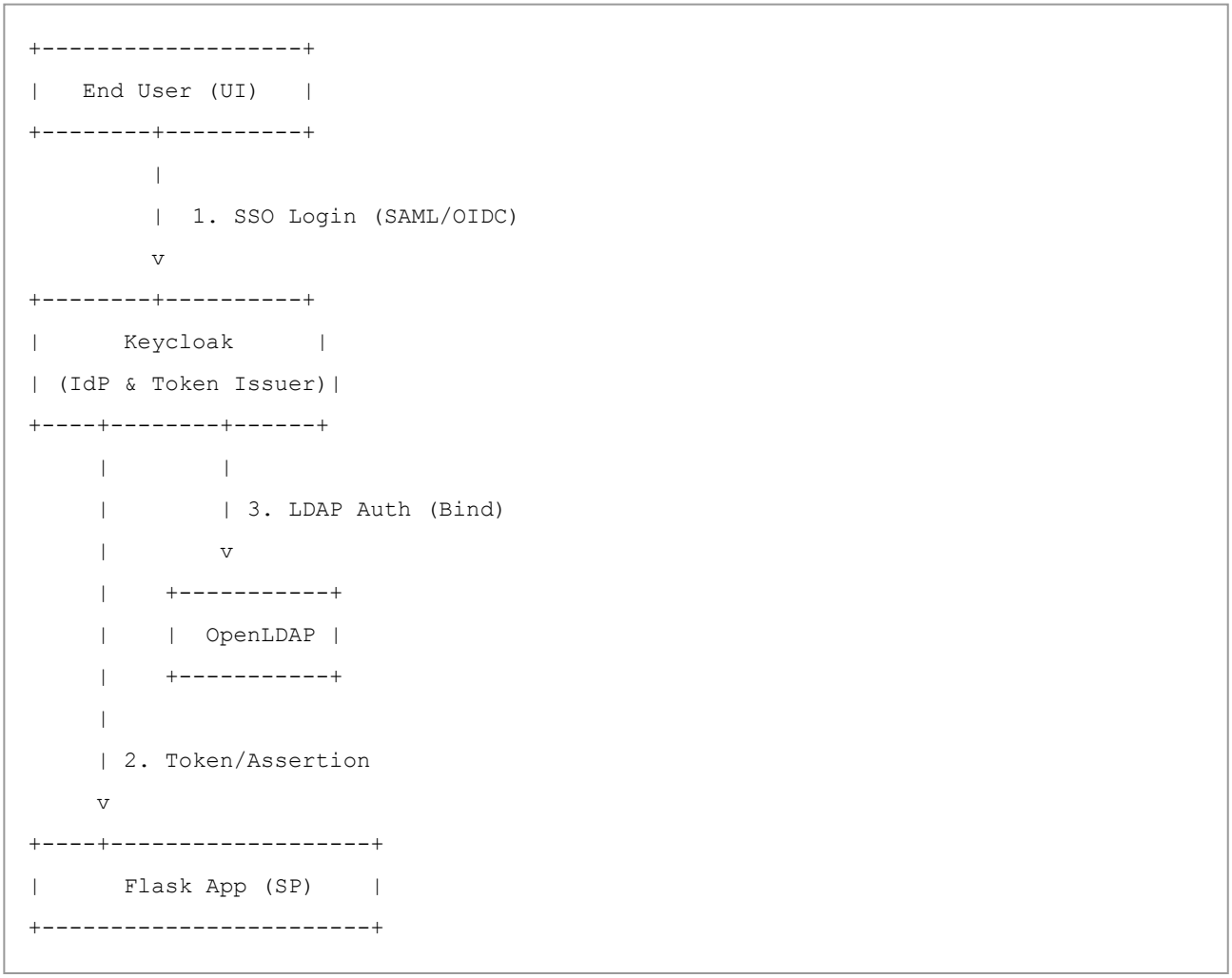
Identity and Access Management (IAM) Architecture Report

---

1. Architecture Diagram Overview

The IAM system is composed of the following components:

- **Keycloak** (Identity Provider): Carries out the authentication process via five standardized protocols for authorization including OAuth 2.0, OIDC, and SAML 2.0..
- **Flask Application** (Service Provider): Confirms the identity of the user by validating the assertions or access tokens given.
- **OpenLDAP**: Is the muscle behind the organizational user directory for the single-leader policy.
- **Docker Compose**: Keycloak, Flask, and LDAP services are managed by Docker Compose.



---

2. OAuth 2.0 and OIDC Flow Explanation

- **OAuth 2.0 Flow (Password Grant)**:

    1. User gives their username and password to a client (e.g., Postman or test script).
    2. The Flask app communicates with Keycloak '/token' endpoint to verify the user's identity.
    3. Keycloak is responsible for checking the data and sending back the access token.
    4. The Flask API endpoint is available after the Bearer token is passed.

- **OpenID Connect (OIDC)**:
    - OpenID Connect 2.0 is a protocol that enhances OAuth 2.0 by incorporating an ID Token that carries user claims.
    - The most important claims are: `sub`, `iss`, `aud`, `exp` and `email`.
    - Used in authentication; ID token passes through the Flask or browser frontend.
- **SAML 2.0 Flow ()**:
    1. User enters `/sso/login` URL in Flask app and visits the page.
    2. The user is sent to Keycloak for logging in.
    3. After the success, Keycloak sends a POST request with a signed assertion to `/sso/acs`.
    4. Flask validates the assertion and establishes a session.

---

3. Security Analysis: STRIDE Model

Threat	Description	Mitigation
Spoofing	Fake tokens or credentials sent to Flask app	Validate token signature & issuer; use TLS
Tampering	Token contents modified (e.g., change `aud`)	Use signed JWTs (RS256); verify audience and expiration
Repudiation	Users deny actions	Log `sub`, `iss`, and timestamp in secure audit logs
Information Disclosure	Unencrypted tokens intercepted	Use HTTPS and short-lived tokens
Denial of Service	Excessive login or token requests	Rate-limit auth endpoints; CAPTCHA or MFA
Elevation of Privilege	Attacker uses SAML replay or JWT forgery	Bind tokens to session/IP; enforce short expiry; validate `nbf`, `exp`

---

4. Reflection on the Okta Case Study

The 2023 Okta breach, which exploited a support engineer's session and accessed HAR files with session tokens, shaped the design in the following ways:

- **Session Security**: All tokens are short-lived; no long-lived refresh tokens used.
- **Token Hygiene**: No tokens stored or exposed in support logs or scripts.
- **TLS Enforcement**: All communication (Flask ↔ Keycloak, client ↔ Flask) requires HTTPS.
- **Least Privilege**: Admin credentials are isolated from apps and automated scripts.
- **Audit Readiness**: All authentication and access requests are logged with user identity, timestamps, and IP address.

By applying these lessons, this IAM architecture adheres to modern Zero Trust principles and minimizes the blast radius of compromised credentials.

End of Report