SSO & RBAC with KeyCloak

CIA Final Project - Ahmed Nouralla

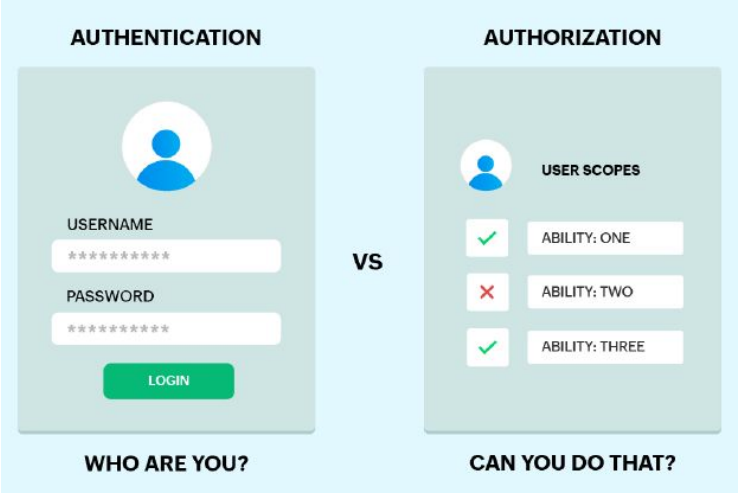
Content

- **Introduction**
  - Identity and Access Management (IAM)
  - Access Control & Role-Based Access Control (RBAC)
  - Single-Sign On (SSO) & KeyCloak
- **Methods**
  - Keycloak Deployment
  - Web Server Deployment
  - Login Process
  - Enabling TLS
- **Results**
- **Discussion**
- **References**

Intro: Identity and Access Management (IAM)

Core concepts for IAM:

- **Identification:** user claiming an identity
  - E.g., by specifying their name/email
- **Authentication:** user proves their identity
  - E.g., by means of a password/access key
- **Authorization:** user gets access to certain system resources/functionalities based on their proven identity
  - E.g., access to the admin portal is only available for system administrators).



Authentication vs. Authorization

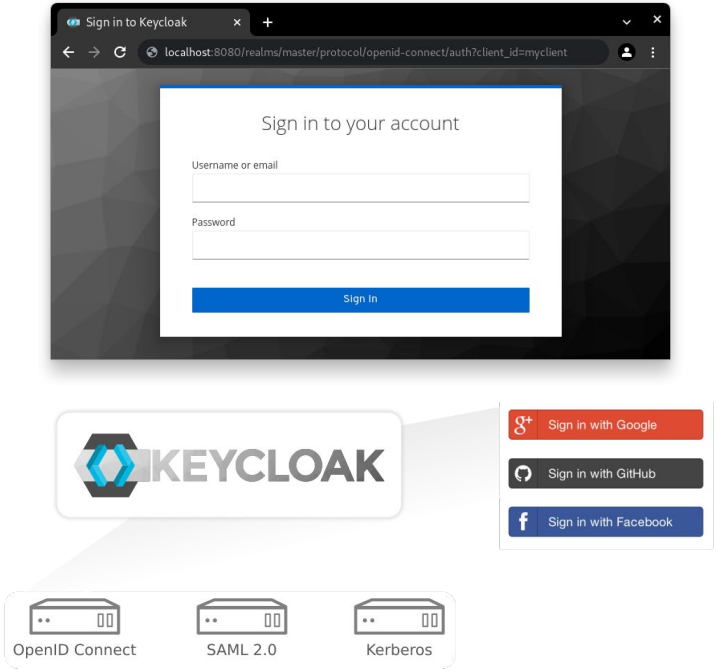
Intro: Access Control and RBAC

- Access Control defines who should have access to what and how to enforce such access
  - The general idea is that **users** are given **access** to **resources** based on their proven identities.
  - Policies define access rules (permissions) and mechanisms are used to enforce them
  - In Role-Based Access Control (RBAC), policies concern roles rather than individuals

- As systems get more complex, access control can get very tricky to implement correctly and securely
  - It’s the reason why "Broken Access Control" is the most common security issue according to OWASP TOP 10 (2021).

Intro: Single Sign-On (SSO) & KeyCloak

- Single Sign-On (SSO) allows users to authenticate to multiple independent services using a single ID.
- Keycloak is an open-source IAM solution that helps software developers "add authentication to applications and secure services with minimum effort."
    - It provides a feature-packed management console to quickly deploy functional login forms supporting SSO, social login, integration with existing directory servers, RBAC, and more.
- Keycloak is based on and supports standard protocols: OAuth 2.0, OpenID Connect, and SAML



KeyCloak Login Form and Features

Methods

- This work implements a minimal authentication/authorization scenario with three main components
  - **Keycloak server:** pre-configured to authenticate web users through OpenID Connect.
  - **Web server (Relying Party):** Python (Flask) application outsourcing authentication functionalities to Keycloak.
  - **Web client:** a pure HTML/CSS/JS web page to communicate with the server.

Methods: KeyCloak Deployment

- Created a compose file to quickly run a keycloak development instance with volumes for persisting data
    - Containers are the industry standard for reproducibility and easy migration to the cloud.
- Configure KeyCloak from the Admin Console
    - **Created a realm:** `myrealm`
    - **Created an OIDC Client:** `demo`
    - **Created test users:** `user1`, `user2`, `user3`
    - **Created roles:** `admin`, `editor`, `viewer`
    - Bound roles to users respectively

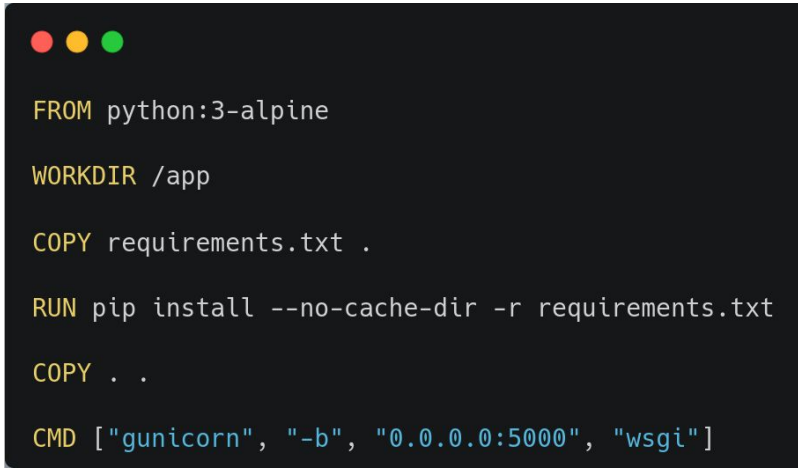
```yaml
name: demo

services:
  keycloak:
    container_name: keycloak
    ports:
        - "8080:8080"
    environment:
        - KC_BOOTSTRAP_ADMIN_USERNAME=admin
        - KC_BOOTSTRAP_ADMIN_PASSWORD=admin
    image: quay.io/keycloak/keycloak:26.0
    volumes:
      - keycloak_data:/opt/keycloak/data
    command: start-dev
```

Sample Compose File to Run KeyCloak Development Container

Methods: WebServer Deployment

- Created Python Virtual Environment with needed dependencies: Flask, PyJWT, Requests, Gunicorn, and Dotenv
- Created a Dockerfile for the application for reproducibility and easier migration to the cloud
- Wrote application logic to handle endpoints
    - `/`: returns HTML content of application homepage
    - `/login`: called when user clicks login
    - `/callback`: called after a successful login
    - `/logout`: sign the logged-in user out



Minimal Dockerfile for the server

Methods: Login Process

The following diagram shows the exact HTTP requests and responses being exchanged for SSO to work.

Brief explanation of the process:

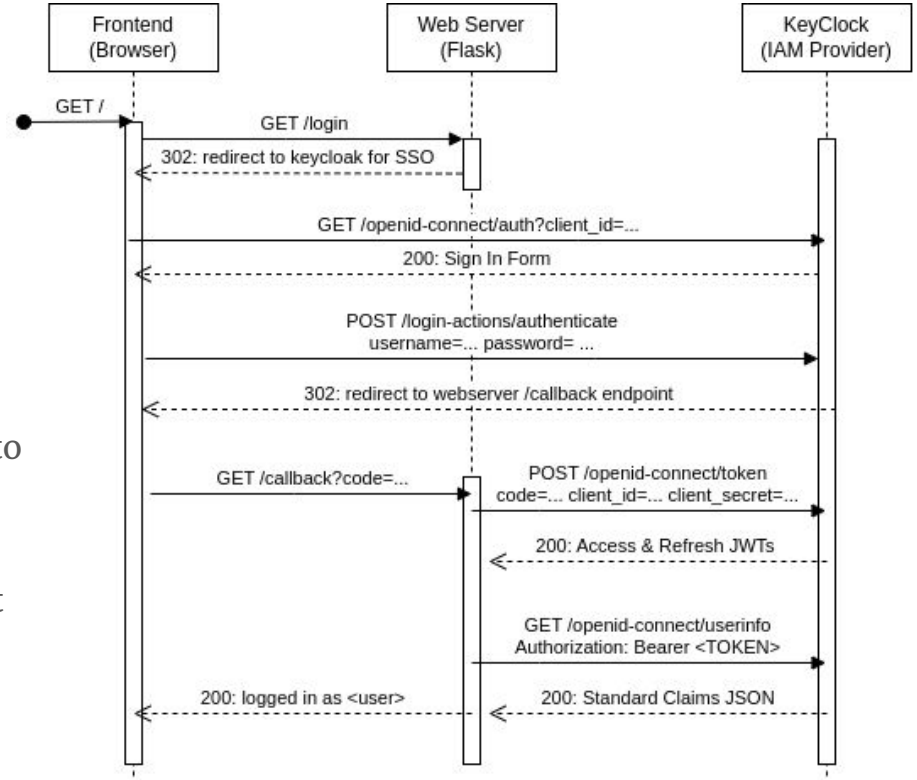
1. User accesses the home page of the application

2. User clicks "Log In" button and gets redirected to keycloak login form.

3. User logs in with their credentials and gets redirected to the `/callback` endpoint (with authorization code)

4. Browser informs the web server of the code, which in turn contacts keycloak with the code, client id, and client secret to obtain the access token (JWT in our case).

5. The web server uses the token to obtain information about the end-user (e.g., username and email).

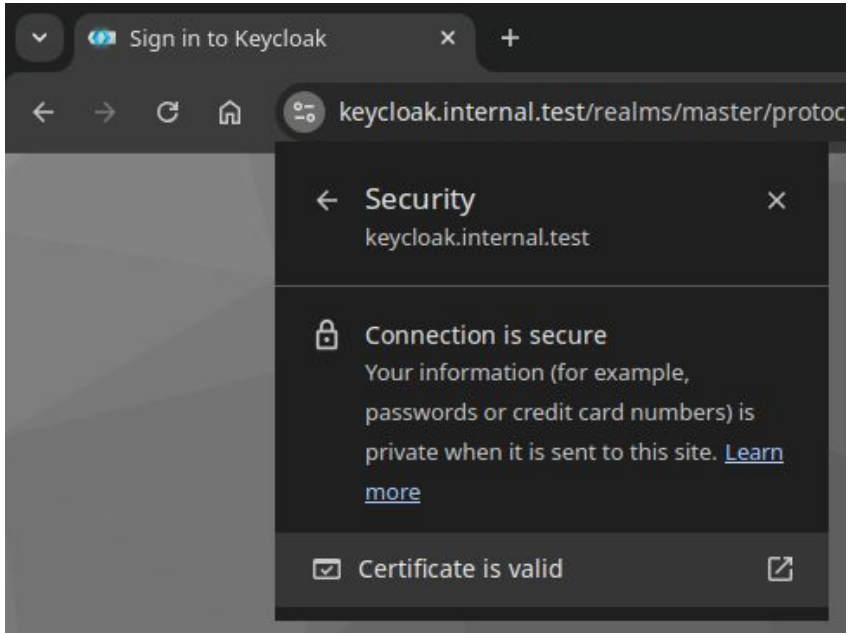


Sequence Diagram Illustrating the Login Process

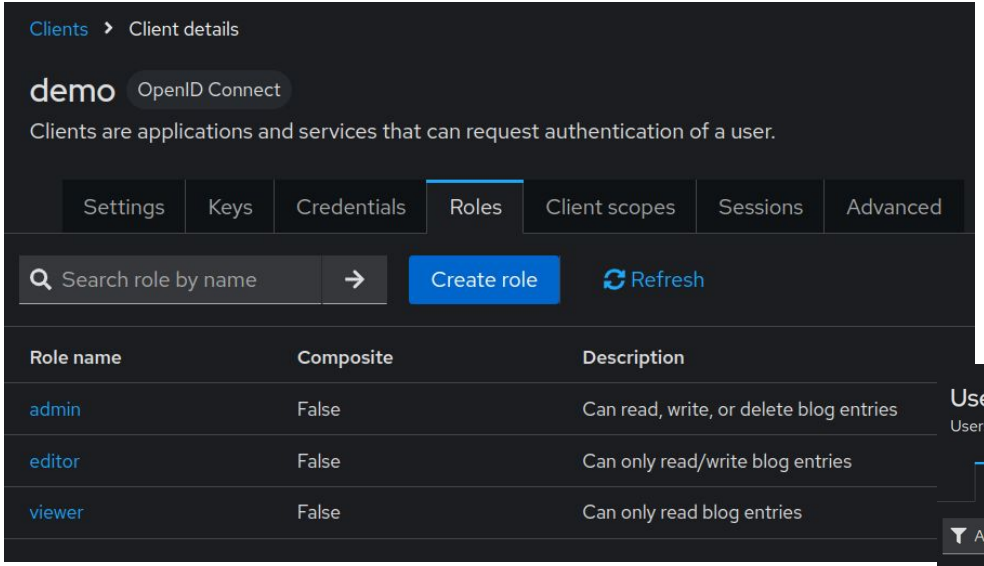
Methods: Enabling TLS

To enable TLS, the following steps were taken:

1. Generated certificates using mkcert for testing
2. Configured local hostnames at `/etc/hosts` for `app.internal.test` and `keycloak.internal.test`
3. Mounted certs directory into keycloak and app containers
4. Configured services to read cert files and use it.

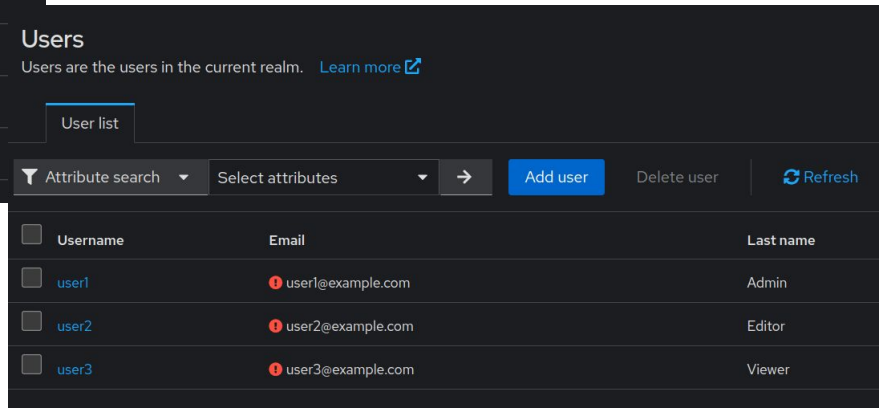


Results

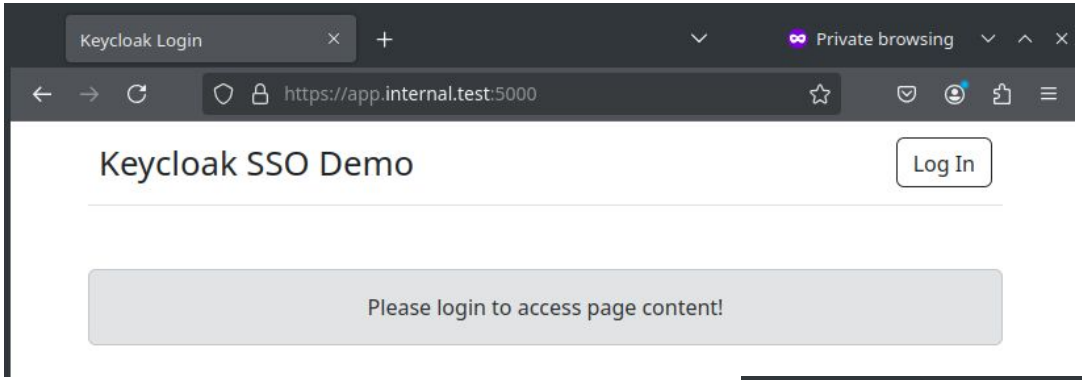


Created Roles

Created Users

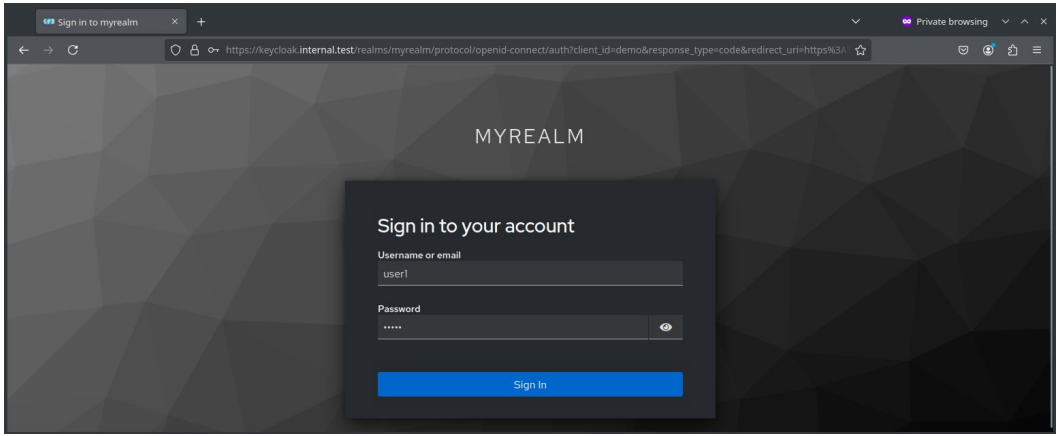


Results

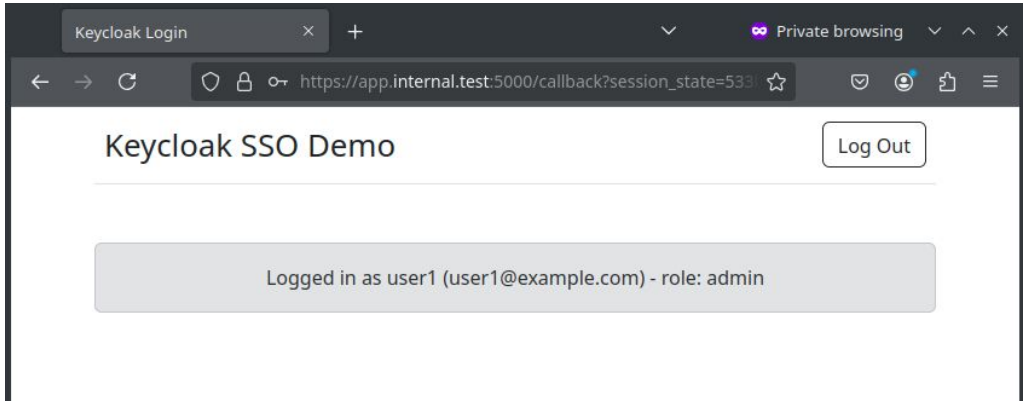


Webpage before login

Redirection to KeyCloak login form



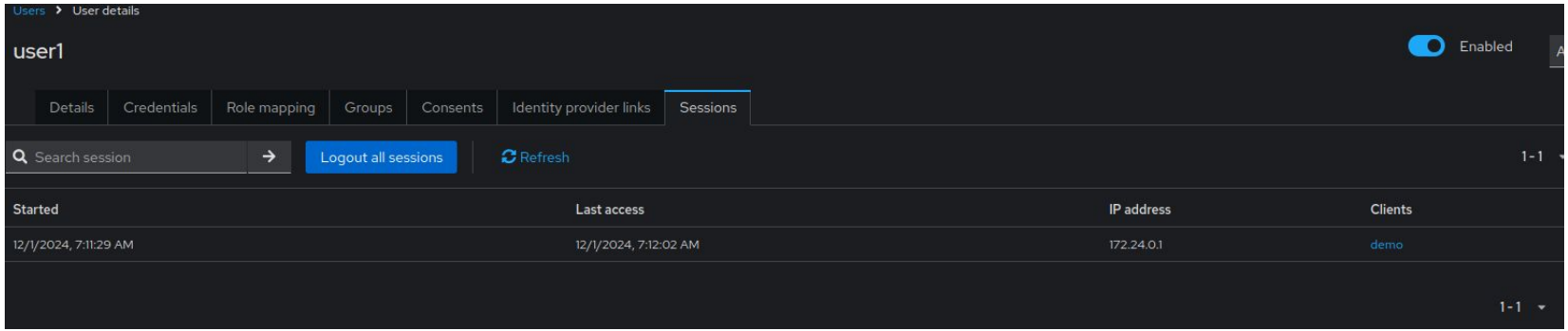
Results



Successful Login

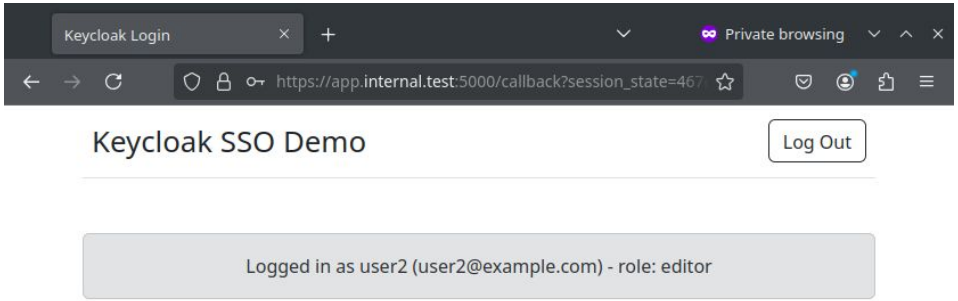
&

Session recorded in server

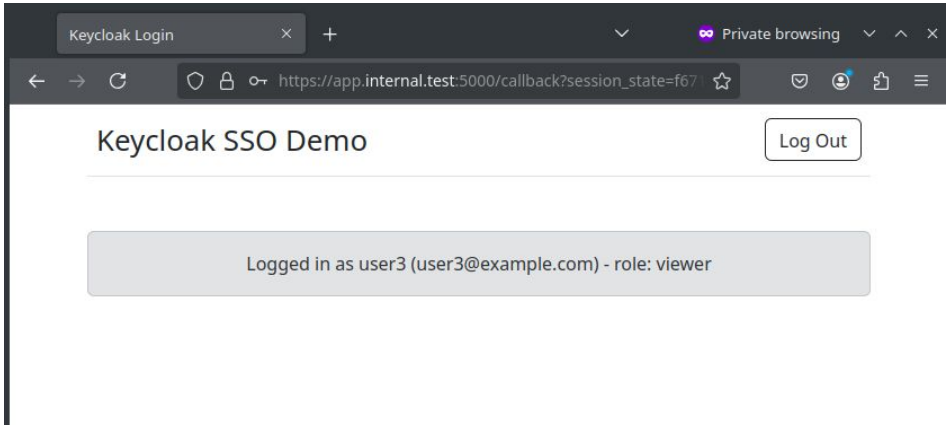


Results

Login status for an editor:



Login status for a viewer:



Discussion

- Even for a small app, implementing login correctly can take time and be error-prone.
- Keycloak starts to prove more useful as the number of clients (services) increase
- Support for additional features (e.g., social login, OTP/email verification, password resets, user registration, requesting additional user info, etc.) can also be configured faster through Keycloak.
- Additional measures should be taken into account when deploying this infrastructure in production
  - Using TLS for secure communication.
  - Configuring UI and admin endpoints under different hostnames
  - The use of reverse proxy in distributed environments.
  - Limiting the number of queued requests.
  - Replacing the default `dev-file` with a production-grade database (e.g., PostgreSQL or MySQL).
  - Enable observability (e.g., Prometheus metrics and alerts for monitoring).

References

- https://www.keycloak.org/securing-apps/oidc-layers
- https://www.keycloak.org/server/configuration-production
- https://openid.net/developers/how-connect-works/
- https://openid.net/specs/openid-connect-basic-1_0.html