Introduction

This document details the implementation of Single Sign-On (SSO) using Authelia, LDAP, NGINX, Grafana, and Gitea. The environment was set up locally using Docker and Docker Compose. Terraform was not used in this setup as it was a local environment focused on manual configuration and setup.

The goal was to integrate LDAP-based authentication via Authelia, using NGINX as a reverse proxy to enable seamless authentication across Grafana and Gitea. Authelia acts as the authentication gateway, and authenticated sessions are passed to the services using trusted headers.

LDAP + Authelia + NGINX + Grafana + Gitea Setup

1. LDAP server was used as the user directory backend.
2. Authelia configured to connect to the LDAP server for authentication.
3. NGINX reverse proxy was used in front of Grafana and Gitea to enforce authentication via Authelia.
4. Authelia session cookies and headers were used to forward authenticated sessions.
5. Grafana and Gitea were configured to trust headers coming from NGINX only.

How to deploy.

Directory structure.

```
project-root/
├── docker-compose.yml
├── nginx/
│   ├── conf.d/
│   │   ├── authelia.conf
│   │   ├── grafana.conf
│   │   └── gitea.conf
│   └── certs/
│       ├── authelia.local.crt
│       ├── authelia.local.key
│       ├── grafana.local.crt
│       ├── grafana.local.key
│       ├── gitea.local.crt
│       └── gitea.local.key
├── authelia/
│   ├── configuration.yml
│   └── users_database.yml
```

Clone the repository.

https://github.com/Adityakulkarni08/sre-assignment.git

cd sre-assignment

Prerequisites

- Docker & Docker Compose installed
- Local host entries

```
127.0.0.1 grafana.local.test
127.0.0.1 gitea.local.test
127.0.0.1 authelia.local.test
```

Generate the required certs with generate-certs script.

./generate-certs.sh

Generate and update the secret_value, encryption_value, jwt_secret_value in authelia/configuration.yaml.

./generate-key-values.sh

```
session:
  name: authelia_session
  secret: secrete_value
  expiration: 3600
  inactivity: 300
  same_site: lax
  cookies:
    - domain: ".local.test"
      authelia_url: https://authelia.local.test
      default_redirection_url: https://gitea.local.test

storage:
  local:
    path: /config/db.sqlite3
  encryption_key: encryptionkey_value

notifier:
  filesystem:
    filename: /config/notification.txt

identity_validation:
  reset_password:
    jwt_secret: jwt_value
```

Generate the password for user-database.yaml
Run the below command form cli

docker run -i authelia/authelia:latest authelia hash-password <<< "YourSecurePassword"
Update the generated password in authelia/users_database.yaml

Start the stack up.
docker-compose up –build -d

Update nginx-container-ip in docker-compose.yaml under grafana environment variable.
docker inspect nginx | grep "IPAddress"

Problem Faced

After successful authentication with Authelia, user was redirected to the Grafana login page instead of being

logged in directly. This indicated that Grafana was not receiving or trusting the authenticated user information from the reverse proxy.

Root Cause Analysis

- Mismatched Authentication Header:
NGINX was sending 'X-Forwarded-User', but Grafana was configured to expect 'X-WEBAUTH-USER'.

- Missing Whitelist:

Grafana's authentication proxy needed the NGINX container's IP to be whitelisted using GF_AUTH_PROXY_WHITELIST.

Fixes Applied

1. Updated NGINX to send 'X-WEBAUTH-USER' header:

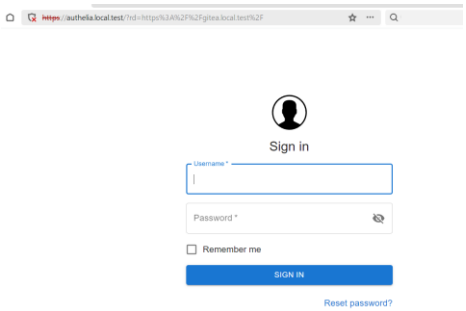
auth_request_set $user $upstream_http_remote_user;

proxy_set_header X-WEBAUTH-USER $user;

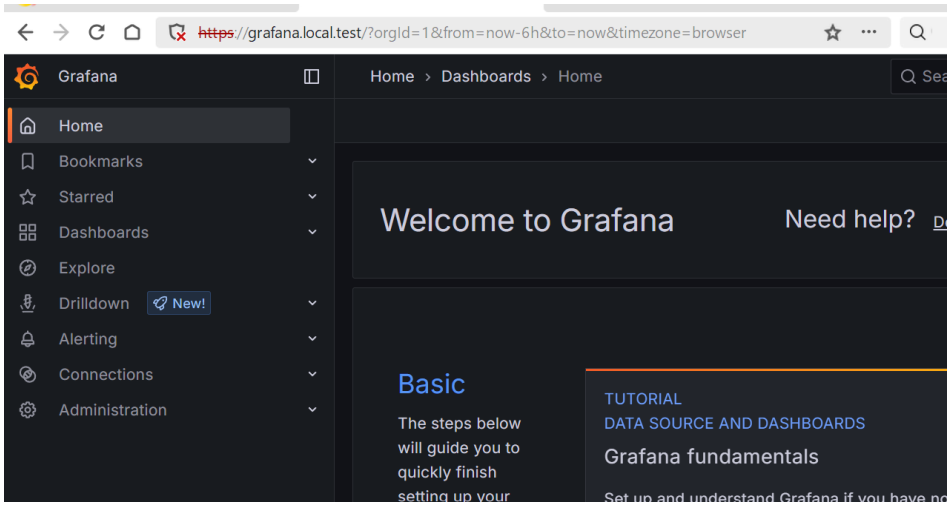
2. Set 'GF_AUTH_PROXY_HEADER_NAME=X-WEBAUTH-USER' in Grafana.

3. Added the IP of the NGINX container to 'GF_AUTH_PROXY_WHITELIST'.

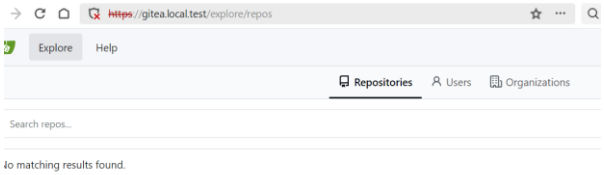
Snap-shots



Authelia login page for gitea/grafana



Grafana direct login



Gitea for managing repos

Conclusion

The issue was caused by inconsistent header configuration and missing proxy trust settings. By aligning the header names, whitelisting the reverse proxy IP, and correcting configuration, seamless SSO using Authelia and LDAP was achieved. This solution was implemented locally using Docker Compose for demonstration and testing purposes.