Keycloak實現網頁Single Sign-On的登入機制

Using Keycloak as User federation to implement Single Sign-On login flow

Wei-shao Lu,
Chun-Feng Liao

Abstract

Challenges:

- Traditional id/password suffers from security vulnerabilities: If an attacker manages to crack the password, the entire system could be compromised.
- Handling system downtime and maintenance becomes a challenge(e.g. Database Maintenance)

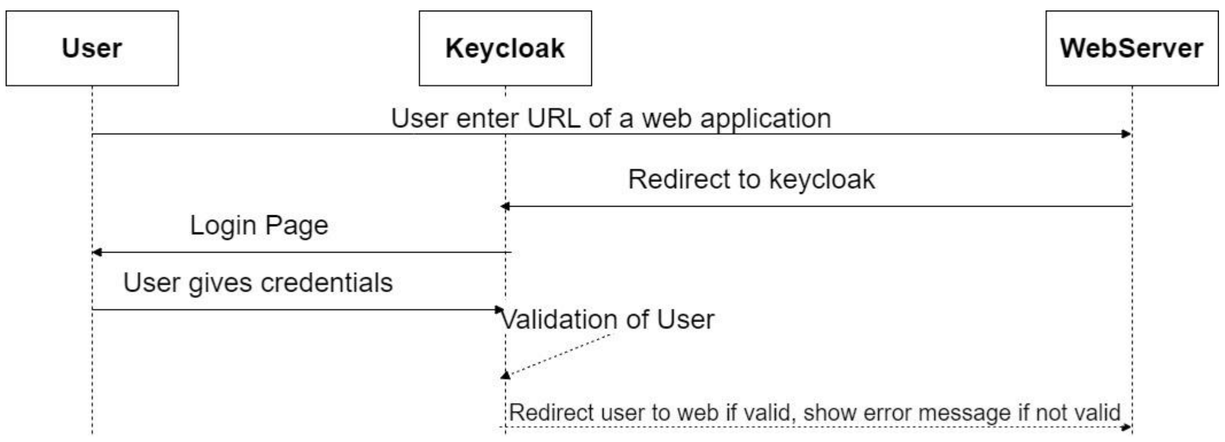
Approach:

- Introduce an Authentication and Authorization services (e.g., Keycloak)
  - Authentication: using an LDAP (Lightweight Directory Access Protocol) server
  - Authorization: using Keycloak as the identity and access management solution
- Benefits
  - Keycloak, being an open-source identity provider, offers features such as single sign-on, multi-factor authentication, and centralized user management.
  - User credentials and relevant information are securely stored. This not only improves security but also simplifies user administration tasks.

Results:

- Quicker response in the event security incidents
- Provide additional layer of security
- Makes overall A/A more scalable and maintainable

Methodology



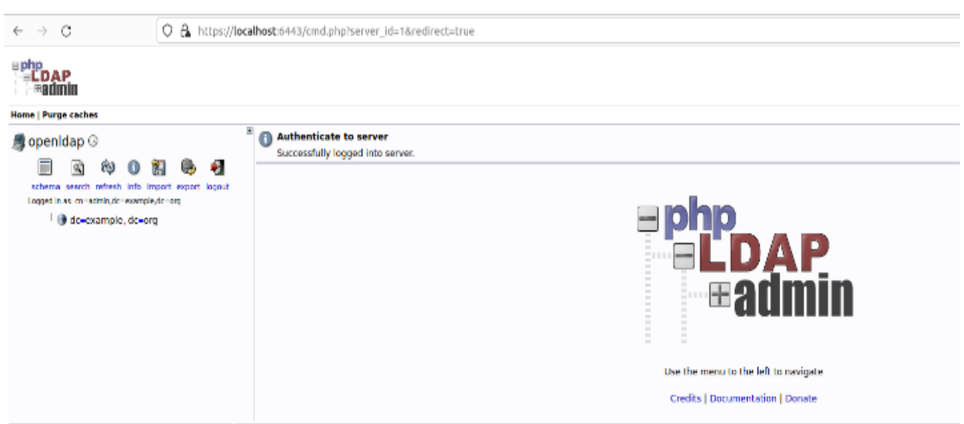
Python aspect:
In a Flask project, importing flask_oidc and configuring it by loading the settings from a JSON file allows you to implement Keycloak-based authentication. By adding the oidc.require_login decorator to your login function, you can ensure that users are redirected to the Keycloak login page for authentication.

```python
@app.route("/login/redirect=<direct>", methods=["GET", "POST"])
@oidc.require_login
def login(direct):
    direct = direct.replace("&", "/")
    if oidc.user_loggedin:
        userName = oidc.user_getfield("preferred_username")
        session["userName"] = userName
        message("1", f'USER: {session["oidc_auth_profile"]} ALREADY LOGGED IN')
        return redirect("/")

    form = loginForm(request.form)
    if request.method == 'POST' and form.validate():
        userName = oidc.user_getfield("preferred_username")
        #message("1", f"{userName}")
        message("1", f"{session['oidc_auth_profile']}")
        # Validate user credentials using OIDC

        # Example: Check if the user is in the database
        # Replace this with your OIDC user validation logic
        connection = sqlite3.connect("db/users.db")
        cursor = connection.cursor()
        cursor.execute(
            f'select * from users where lower(userName) = "{userName.lower()}"'
        )
        user = cursor.fetchone()
        if not user:
            message("1", f'USER: "{userName}" NOT FOUND')
            flash("User not found", "error")
        else:
            # Authenticate the user using OIDC
            # You may need to customize this based on your OIDC provider
            if authenticate_user_with_oidc(userName, password):
                session["userName"] = userName
                addPoints(1, userName)
                message("2", f'USER: "{userName}" LOGGED IN')
                flash(f"Welcome {userName}", "success")
                return redirect("/")
            else:
                message("1", "WRONG PASSWORD")
                flash("Wrong password", "error")

    return render_template("login.html", form=form, hideLogin=True)
```

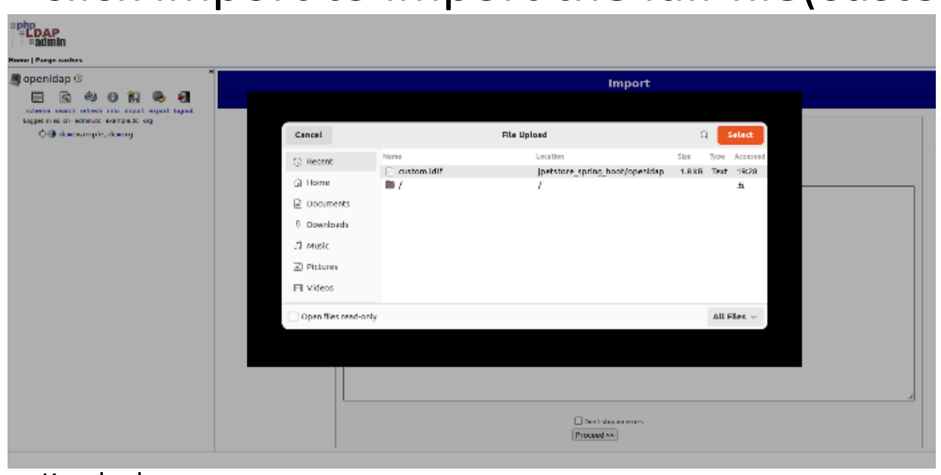
OpenLDAP:

use phpldapadmin to manipulate openldap

login into the openldap



click import to import the ldif file(custom.ldif)



Keycloak:

Create a Realm:

Log in to the Keycloak Administration Console.

On the left sidebar, click on "Add Realm" to create a new realm.

Set a unique name for your realm and click "Create."

Create a Client:

Inside the newly created realm, go to the "Clients" tab and click on "Create."

Set a unique name for your client (this is used by your application).

Click "Save" to create the client.

Define Client Permissions:

Within the client settings, you can define various configurations, including redirect URIs, access types, and client roles.

Assign appropriate roles to the client (e.g., "user" or "admin") based on your application requirements.

Configure User Federation with LDAP:

In the realm settings, navigate to the "User Federation" tab.

Choose "Add Provider" and select "LDAP" from the drop-down menu.

Configure the LDAP settings, including connection details, base DN, and user login attributes.

After saving the LDAP configuration, you can synchronize users from the LDAP server into Keycloak.

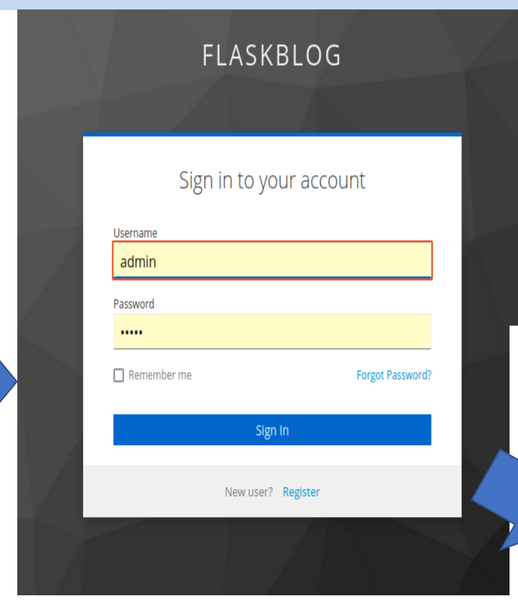
Assign Roles to Users:

Once user federation is set up, go to the "Users" tab and select a specific user.

In the user's details, go to the "Role Mappings" tab.

Assign the appropriate roles (defined in your client) to the user.

Results and Discussions



Redirect to Keycloak Login Page

Successful login

Conclusion

- Traditional login systems that depend on username and password authentication have intrinsic security weaknesses.
- To address these issues, a web-based system utilizing LDAP/Keycloak was implemented successfully.
- The integration of Keycloak, as an open-source identity provider, with LDAP for the secure storage of login data, offers substantial security improvements.
- This combination establishes a strong and reliable authentication mechanism.