# CA 2E / CA Plex Worldwide Developers Conference 2023

## Securing CA Plex Web Apps

Andrew Leggett & Lily Taharudin – CM First

## Exercise One

startKeycloak.bat

Text

Description automatically generated

Running on port 8090

### Create Keycloak Administrator

Graphical user interface, application

Description automatically generated

First time in, create administrator

User: kcadmin

Password: kcadmin

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Click on “Administration Console >”

Enter user/password

Graphical user interface, application

Description automatically generated

### Create “SecurityWorkshop2023” Realm

In Keycloak, a realm is a container for a set of users, applications, and security configurations. A realm defines a boundary for authentication and authorization of its users and clients, providing a secure environment for the components within it.

When a user logs in to Keycloak, they log in to a specific realm. Each realm has its own set of users, roles, clients, and other security components. By creating multiple realms, it's possible to separate different applications or user groups from each other, each with its own authentication and authorization rules.

By default, Keycloak provides a ‘master’ realm. We’ll create a new realm by clicking on the dropdown next to ‘master’, then click on the ‘Create Realm’ button.

Graphical user interface, application

Description automatically generated

Name the domain “SecurityWorkshop2023” and press Create.

Graphical user interface, text, application, email

Description automatically generated

Logo, company name

Description automatically generated

### Create Client for Web Store

In the context of the Keycloak identity and access management system, a client is a representation of a web application or service that wants to use Keycloak for authentication and authorization.

A Keycloak client defines a set of configuration options and settings that allow the application to interact with Keycloak in a secure way. These options include things like the client ID, client secret, redirect URIs, authentication flow, and security settings.

Once a client is registered in Keycloak, it can obtain access tokens and refresh tokens that can be used to authenticate and authorize requests to protected resources. The client can also interact with Keycloak's APIs to perform various tasks, such as registering new users, managing roles and permissions, and revoking tokens.

We are going to create a client named ‘webstore’ to control access to our web application.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Set ClientID and Name to WebStore. Click on ‘?’ to learn what any setting is.

Click ‘Next’

Leave every other option as its default value.

Graphical user interface, text, application, chat or text message

Description automatically generated

Click ‘Next’

Graphical user interface, text, application, email

Description automatically generated

In Keycloak, redirect URLs are used to define where the user should be redirected after authentication or logout. These URLs can be set at the client level or globally for a realm.

When a user tries to access a protected resource on a client application that requires authentication, the application will redirect the user to the Keycloak server login page. After the user logs in successfully, Keycloak will redirect the user back to the original URL requested by the application.

Redirect URLs are also used for logout. When a user logs out of a client application, the application will redirect the user to the Keycloak server logout page. After the user is successfully logged out, Keycloak will redirect the user back to the URL specified by the application.

Redirect URLs can be configured to include query parameters and other information, allowing for a more customized user experience.

We have 2 applications to secure, ‘storeweb’ and ‘storeAPI’.

Add the following URLs to the ‘Valid redirect URIs’ value’

* [http://localhost:8080/storeweb/\*](http://localhost:8080/storeweb/*)
* [http://localhost:8080/storeAPI/\*](http://localhost:8080/storeAPI/*)

Web origins are URLs from which requests to the Keycloak server can be made. When a client application redirects a user to the Keycloak server for authentication or authorization, Keycloak will only redirect the user back to URLs that are listed as valid redirect URLs or web origins for that client. This is an important security measure to prevent unauthorized access to client applications.

Set the web origin to [http://localhost:8080/\*](http://localhost:8080/*)

Click ‘Save’.

### Create Roles

We want to control access to various resources within our web applications. For example, if we have a web application that allows someone to edit an employee’s salary, we would only want certain members of the HR Department to access that. We can control that by assigning ‘Roles’, in this case we could create a role named ‘HR Administrator’ and assign it to certain users. We can then configure our web application to only allow users with specific roles to access specific resources.

In Keycloak, there are two types of roles: realm roles and client roles. Realm roles are global and can be assigned to users across all client applications within the realm. Client roles, on the other hand, are specific to a particular client application within a realm.

Roles can be managed and assigned through the Keycloak Admin Console or through the Keycloak REST API. Once a user or a client application is assigned a role, they inherit the permissions associated with that role. This allows for fine-grained access control and enables administrators to easily manage permissions and access levels for users and client applications.

For this exercise, we’ll manage our roles from the ‘Realm roles’ menu option.

Graphical user interface, text, application, email

Description automatically generated

We’ll create the following roles:

* SiteAdmin – Access all
* Shopper – Access Products
* Registrant – Can register customers and view products.

Graphical user interface, text, application, email

Description automatically generated

Enter the details above and click Save.

Graphical user interface, text, application, email

Description automatically generated

Enter the details above and click Save.

Graphical user interface, application

Description automatically generated

Enter the details above and click Save.

### Create Users

Keycloak users can be created manually or imported from external sources, such as LDAP or Active Directory. We’ll create some users manually.

Select ‘Users’ from the menu and select ‘Create new user’.

Graphical user interface, text, application

Description automatically generated

Create the user ‘dave’ and enter the values below.

Graphical user interface, text, application, email

Description automatically generated

You will see the details confirmed. Note that keycloak has assigned an internal ID for the user.

Graphical user interface, text, application, email

Description automatically generated

To be able to sign-on, the user needs credentials. We’ll assign a password.

Graphical user interface, text, application, email, website

Description automatically generated

Graphical user interface, application

Description automatically generated

To keep things simple for the workshop, set the password to ‘dave’ as well. (This is *never* recommended in the real world)

Set Temporary to Off to prevent asking to change when signing on.

Graphical user interface, text, application, email

Description automatically generated

Confirm and save.

Add user ‘Emma Einstein’ with password ‘emma’

Add user ‘Fred Fandango’ with password ‘fred’.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

### Assign User Roles

From the user list, click on ‘dave’, then select ‘Role mapping’ then ‘Assign role’.

Graphical user interface, application

Description automatically generated

Assign role ‘Registrant’ and ‘Shopper’.

Graphical user interface, text, application, email

Description automatically generated

Set Emma to ‘SiteAdmin’ role,

Set Fred to ‘Shopper’

## Exercise Two

### Configure the ‘storeweb’ application to be secured by Keycloak.

This approach is being deprecated, but alternative is not available yet.

<https://www.keycloak.org/docs/latest/securing_apps/#_tomcat_adapter>

On VM keycloak tomcat adapter has already been installed, so just configure Tomcat to use it.

Configure storeweb application

Create storeweb\src\main\webapp\META-INF\context.xml

A picture containing timeline

Description automatically generated

Set contents to the following.

<?xml version="1.0" encoding="UTF-8"?>

<Context>

<Valve className="org.keycloak.adapters.tomcat.KeycloakAuthenticatorValve"/>

</Context>

We need information about our keycloak client, so we can authenticate against the users and roles that we have set up. This can be downloaded from Keycloak in a file named keycloak.json

Navigate to your WebStore client in keycloak, you may need to sign in as ‘kcadmin’ again if the session has expired.

Graphical user interface, application, chat or text message

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Click ‘Download’, and save in C:\Conference 2023 Workshops\Web Technology\Workspace\storeweb\src\main\webapp\WEB-INF

In Eclipse, press F5 to refresh, and the file should show up there.

A picture containing chart

Description automatically generated

We’ve now configured Tomcat to use the Keycloak adapter and use our SecurityWorkshop2023 settings.

Let’s see if our site has been secured yet. Restart our Tomcat server,

Graphical user interface, text, application, chat or text message

Description automatically generated

When Tomcat is showing as ‘Started’, visit the site <http://localhost:8080/storeweb/wc>

Graphical user interface

Description automatically generated

The site loads, just as before. The reason for this is that we haven’t specified which resources are secured, and who has access to those resources yet.

### Set site Security Constraints

We can specify this in the web application’s WEB-INF/web.xml file.

A picture containing timeline

Description automatically generated

Double-click on the web.xml file to open it in the text editor, and paste the following section between the </welcome-file-list> and </web-app> tags near the end of the file.

<security-constraint>

<web-resource-collection>

<web-resource-name>storeweb</web-resource-name>

<url-pattern>/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>SiteAdmin</role-name>

</auth-constraint>

</security-constraint>

<security-role>

<role-name>SiteAdmin</role-name>

</security-role>

Your file should look like this. Save the file with Ctrl+S.

Graphical user interface, text, application

Description automatically generated

This states that the whole ‘storeweb’ web application is secured, and only available to users with the ‘SiteAdmin’ role.

Restart Tomcat in Eclipse.

***We are going to test the security setting by signing in as a number of different users. Normally, the login information is remembered in a browser cookie, making it more difficult to sign in as someone else. To get around this, we can access the site in ‘Incognito Mode’.***

Right-click on the Chrome icon in the task bar and select ‘New Incognito window’, then visit the site <http://localhost:8080/storeweb/wc> again.

**A screenshot of a computer

Description automatically generated with medium confidence**

This time we will see a sign-on window. Try to sign in as ‘fred’, with password ‘fred’.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Fred hasn’t been assigned the ‘SiteAdmin’ role, so he doesn’t have access to the web application.

Close the browser and restart it in Incognito mode again. Access the site <http://localhost:8080/storeweb/wc>, this time sign on as ‘emma’ with password ‘emma’. This time she has access to the whole site as the administrator.

Graphical user interface

Description automatically generated

### Set API Security constraints

In the workspace, we also have an API generated by HSync. This allows a user to access products or customers with the following URLs.

<http://localhost:8080/storeAPI/api/v1/products>

<http://localhost:8080/storeAPI/api/v1/customers>

We can configure the API in a similar way to only allow ‘Shopper’ roles to view products, and ‘Registrant’ roles to view Customers. We also want users with the SiteAdmin role to view both.

We can copy a couple of the config files we set up in this exercise to the storeAPI project.

Copy file: storeweb\src\main\webapp\META-INF\context.xml

To folder: storeAPI\WebContent\META-INF

Copy file: storeweb\src\main\webapp\WEB-INF\keycloak.json

To folder: storeAPI\WebContent\WEB-INF

Graphical user interface

Description automatically generated with medium confidence

To specify the security constraints, we can add a web.xml file.

Right-click on storeAPI\WebContent\WEB-INF, then select New -> File. Enter the name “web.xml”

Graphical user interface, text, application

Description automatically generated

Double-click on the new file to edit it, then paste the following XML. This constrains the Products API to users with role ‘Shopper’ or ‘SiteAdmin’, and Customers API to users with role ‘Registrant’ or ‘SiteAdmin’.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app id=*"WebApp\_ID"* version=*"2.4"* xmlns=*"http://java.sun.com/xml/ns/j2ee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"*>

<display-name>storeAPI</display-name>

<security-constraint>

<web-resource-collection>

<web-resource-name>storeAPI</web-resource-name>

<url-pattern>/api/v1/products</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>Shopper</role-name>

<role-name>SiteAdmin</role-name>

</auth-constraint>

</security-constraint>

<security-constraint>

<web-resource-collection>

<web-resource-name>storeAPI</web-resource-name>

<url-pattern>/api/v1/customers</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>Registrant</role-name>

<role-name>SiteAdmin</role-name>

</auth-constraint>

</security-constraint>

<security-role>

<role-name>SiteAdmin</role-name>

<role-name>Registrant</role-name>

<role-name>Shopper</role-name>

</security-role>

</web-app>

Try accessing the APIs as different users.

<http://localhost:8080/storeAPI/api/v1/products>

<http://localhost:8080/storeAPI/api/v1/customers>

The expected results are:

* “dave” can access both as he is both a “Shopper” and a “Registrant”.
* “emma” can access both as she is a “SiteAdmin”
* “fred” can only access customers, as he is just a “Registrant”.

## Exercise Three

Control access from Plex application

Menu, enable buttons for user

Get attributes.

### Starting Point

Restore “Security-PlexStart.7z” – REDO THIS TO INCLUDE KC JARS

### Set up Plex function

Open store.mdl

Open the StoreMenu function in the Action Diagram Editor

Graphical user interface, text, application

Description automatically generated

Add a new Subroutine “Sub Get User Information”

In the Object Browser, look for an Entity named ‘ConferenceKeyCloak’. This contains several source code objects we can use to work with our KeyCloak setup.

Graphical user interface, text, application, email

Description automatically generated

First, we’ll check our user roles and enable menu options based on these.

Source code object ConferenceKeyCloak.GetKeycloakUserHasRole allows us to check if the User is assigned a specific role.

Add the statement API Call Source code: ConferenceKeyCloak.GetKeycloakUserHasRole

Map the following parameters:

Input<YesNo> SecurityL<YesNo>

Input< WebStoreRole > <WebStoreRole.Shopper>

A picture containing background pattern

Description automatically generated

Conditionally hide the Products button.

If SecurityL<YesNo> == <YesNo.No>

Set State Protected, Event: Products

Now do the same for the Registrant Role to enable/disable the Customers button.

API Call Source code: ConferenceKeyCloak.GetKeycloakUserHasRole

Map the following parameters:

Input<YesNo> SecurityL<YesNo>

Input< WebStoreRole > <WebStoreRole.Registrant>

Graphical user interface

Description automatically generated with low confidence

If SecurityL<YesNo> == <YesNo.No>

Set State Protected, Event: Customers

Now we can get the user attributes from KeyCloak for the Welcome message.

We’ll use the Source code object *ConferenceKeyCloak.GetKeycloakAttribute* for this.

Add the statement:

API Call Source code: ConferenceKeyCloak.GetKeycloakAttribute

And map the following parameters:

Input<Attribute> <WebStoreAttribute.WelcomeMessage>

Input<Attribute Value> SecurityL<Attribute Value>

Table

Description automatically generated

If SecurityL<Attribute Value> != <LongDescription.\*Blank>

Cast Environment<\*Message text>, SecurityL<Attribute Value>

Go Sub Send message

Finally, add the following line to the End Initialize Post Point

Go Sub Get User Information

Graphical user interface, text

Description automatically generated

Save your model, close the action diagram and generate the StoreMenu function:

Graphical user interface

Description automatically generated with medium confidence

Open the *C:\Conference 2023 Workshops\Web Technology\Workspace* workspace in Eclipse.

Press F5 to refresh the workspace with your latest changes. The WebClient generator will build your function.

>>> Starting WebClient Build for storeJava

> Using CM WebClient v1.8.8-pre13284

> Licensed to CMFirst Technologies

> Valid until Mon Jan 01 00:00:00 CST 2024

Generating Web templates.

>>> WebClient Build complete.

1 panels processed. 0 panels not processed.

In the previous exercise we restricted access to our web application by assigning access to the ‘SiteAdmin’ role. Let’s change this to allow access to all users.

Graphical user interface, text, application

Description automatically generated

Change the ‘SiteAdmin’ values to ‘\*’ for all access.

Graphical user interface, text, application, chat or text message

Description automatically generated

Save the web.xml file.

Restart the Tomcat Server on the Servers tab.

Graphical user interface, text, application, chat or text message

Description automatically generated

Make sure that Keycloak is running and access the web application again.

<http://localhost:8080/storeweb/wc>

Sign-on as ‘fred’ with password ‘fred’.

A screenshot of a computer

Description automatically generated with medium confidence

The ‘Customers’ menu option is disabled, as Fred does not have the ‘Registrant’ role assigned.

We don’t see a welcome message as we haven’t defined that yet. We’ll do that in the next step.

### Create OpenID Connect Mappers

When we set up Keycloak, we defined it (by default) to use the OpenID Connect (OIDC) protocol. This is an extension to the OAuth 2.0 protocol that allows it to attach user identity data to an access token, e.g. an email address or nationality. This information is known as a “claim”.

To allow these attributes to be attached to the token, we first need to set up mappers for these attributes. There are many types of mappers, but we’ll just deal with a User Attribute mapper for this exercise.

Log on to the Keycloak administration console at <http://localhost:8090/> as ‘kcadmin’ with password ‘kcadmin’ and select the “SecurityWorkshop2023” Realm.

Graphical user interface, application

Description automatically generated

Select Client, then click on our WebStore client.

Graphical user interface, text, application, email

Description automatically generated

Click on the Client Scopes tab. In Keycloak, a client scope is a set of predefined permissions or attributes that can be associated with a client application. Client scopes provide a way to manage and share sets of permissions or attributes across multiple client applications in a standardized way.

Graphical user interface, text, application

Description automatically generated

Click on the “WebStore-dedicated” link.

Graphical user interface, text, application, chat or text message

Description automatically generated

Click on “Add mapper”. The first time in you’ll see the “No mappers” message.

Graphical user interface, text, application

Description automatically generated

Click on the “Configure a new mapper” button. There are several mapper types listed. Select “User Attribute”

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

On the “Add Mapper” panel, set the values as above and click “Save”.

Now that we have the mapper, we can add an individual welcome message to each user.

Select ‘Users’ from the menu, and click on ‘emma’.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Click on the Attributes tab and enter the key “WelcomeMessage”, then enter a message in the Value column. Click Save.

In Eclipse, start the Tomcat server if it’s not already running, and visit the web application in a new browser in Incognito mode at <http://localhost:8080/storeweb/wc>

Sign in as “Emma” with password “Emma”.

The message should now show up.

Graphical user interface, text, application, email

Description automatically generated

Add WelcomeMessage attributes to users Fred and Dave. Test out those users.