**End-to-End Implementation Process for OAuth Token Authentication**

**Step 1: Define Token Issuance Endpoint (/oauth/token)**

Create an endpoint that will issue tokens. In this case, we'll simulate the OAuth token flow, where the client sends credentials to the /oauth/token endpoint and receives an access token.

In your camel-context.xml, define the route for token issuance:

xml

Copy code

<route id="authTokenRoute">

<from uri="jetty:https://0.0.0.0:9080/oauth/token?sslContextParameters=#sslContextParameters"/>

<log message="Received token request from ${header.CamelHttpRemoteHost}"/>

<!-- Authentication processor to validate credentials and issue token -->

<process ref="authTokenProcessor"/>

<!-- Return the token as a response -->

<setBody>

<simple>{"access\_token": "${header.authToken}", "token\_type": "Bearer", "expires\_in": 3600}</simple>

</setBody>

</route>

**Step 2: Implement Token Processor**

This processor generates the token and authenticates the client. For now, it generates a UUID as the token and stores it in the authToken header.

java

Copy code

public class AuthTokenProcessor implements Processor {

@Override

public void process(Exchange exchange) throws Exception {

// Placeholder for client credentials validation (username/password)

String clientId = exchange.getIn().getHeader("client\_id", String.class);

String clientSecret = exchange.getIn().getHeader("client\_secret", String.class);

// Simple validation (in real scenarios, use a DB or other mechanisms)

if ("validClientId".equals(clientId) && "validClientSecret".equals(clientSecret)) {

// Generate a token (using UUID for simplicity)

String authToken = UUID.randomUUID().toString();

// Set token in header for the response

exchange.getIn().setHeader("authToken", authToken);

} else {

throw new Exception("Invalid client credentials");

}

}

}

In your ol-security.xml or other Spring beans file, configure the bean for the processor:

xml

Copy code

<bean id="authTokenProcessor" class="com.yourcompany.security.AuthTokenProcessor"/>

**Step 3: Secure /financial and /nonfinancial Endpoints**

Modify your existing endpoints so they require a valid token for access. The token will be passed in the Authorization header, and you'll need to validate it before processing the request.

Update your camel-context.xml:

xml

Copy code

<route id="financialRoute">

<from uri="jetty:https://0.0.0.0:8443/financial?sslContextParameters=#sslContextParameters&amp;needClientAuth=true"/>

<!-- Token validation -->

<process ref="authTokenValidator"/>

<log message="Processing financial transaction for ${header.CamelHttpRemoteHost}: ${body}"/>

</route>

<route id="nonFinancialRoute">

<from uri="jetty:https://0.0.0.0:8443/nonfinancial?sslContextParameters=#sslContextParameters&amp;needClientAuth=true"/>

<!-- Token validation -->

<process ref="authTokenValidator"/>

<log message="Processing non-financial transaction for ${header.CamelHttpRemoteHost}: ${body}"/>

</route>

**Step 4: Implement Token Validator**

The token validator will check if the token provided in the Authorization header is valid.

java

Copy code

public class AuthTokenValidator implements Processor {

@Override

public void process(Exchange exchange) throws Exception {

// Get the Authorization header

String authHeader = exchange.getIn().getHeader("Authorization", String.class);

if (authHeader != null && authHeader.startsWith("Bearer ")) {

String token = authHeader.substring(7);

// Here, validate the token (in a real scenario, you'd check a DB or cache)

if (isValidToken(token)) {

// Token is valid, proceed

return;

}

}

// If invalid, throw an exception

throw new Exception("Invalid or missing token");

}

private boolean isValidToken(String token) {

// For demonstration, accept any token that isn't null/empty

return token != null && !token.isEmpty();

}

}

Register the AuthTokenValidator bean in ol-security.xml or another Spring config file:

xml

Copy code

<bean id="authTokenValidator" class="com.yourcompany.security.AuthTokenValidator"/>

**Step 5: Testing the Endpoints**

**Step 5.1: Get a Token**

To get a token, you will need to call the /oauth/token endpoint and provide the client credentials.

Example cURL command to get a token:

bash

Copy code

curl -X POST https://localhost:9080/oauth/token \

-H "Content-Type: application/x-www-form-urlencoded" \

-d "client\_id=validClientId&client\_secret=validClientSecret" \

--cert /certificates/client-cert.pem \

--key /certificates/client-key.pem \

--cacert /certificates/ca-cert.pem

This should return a response like:

json

Copy code

{

"access\_token": "your\_generated\_token",

"token\_type": "Bearer",

"expires\_in": 3600

}

**Step 5.2: Access Secured Endpoints**

Once you have the token, you can use it to call the secured endpoints (/financial and /nonfinancial).

Example cURL command to access /financial:

bash

Copy code

curl -X POST https://localhost:8443/financial \

-d '{"key": "value"}' \

-H "Content-Type: application/json" \

-H "Authorization: Bearer your\_generated\_token" \

--cert /certificates/client-cert.pem \

--key /certificates/client-key.pem \

--cacert /certificates/ca-cert.pem

**Summary of the Steps**

1. **OAuth Token Endpoint** (/oauth/token): Create a new route to issue tokens upon client authentication.
2. **Token Processor**: Validate client credentials and issue a token.
3. **Securing Existing Endpoints**: Modify the /financial and /nonfinancial routes to check for a valid token in the Authorization header.
4. **Token Validator**: Check the token's validity before processing the request.
5. **Testing**: Use cURL to test the token issuance and access secured endpoints.

This setup uses both mTLS for mutual authentication and OAuth-like token authentication to secure your API endpoints.