**Unico Coding Challenge**

1. **My Solution & Design**

For securing this application I created a basic Role-based access control (R-back) module to authenticate and authorize access to the web services.

Then, I created annotation using CDI interceptors and used them at service layer to declaratively check the user role before calling method.

For the Rest service, I go for apache Shiro and use its own @Requires\* annotation to check access to my rest endpoint. For the sake of this exercise, both methods in this service grant access to users with “Admin” role. For testing this service, use http basic authentication (Authorization Header) to put your username and password.

This application will create 20 users at startup in database for facilitating the test of services for you. There are user1/pass1 to user20/pass20. I used BCypt hashing algorithm to store passwords in database.

For implementing a secure Soap service, I choose WS-Security standard and implement it using the combination of JBossWS and Apache CXF. This demo service, requires UsernameToken in its header as a WS-Policy. I created custom wss4j validator to validate given username and password with my R-back module. Also, I developed a CDI based interceptor (CheckUserRole class) for declaratively checking user’s role against the R-back module. For the sake of this exercise, all methods in this service grant access to users with “Admin” role. For testing this service, use WS-Security extension with UsernameToken to put your username and password.

1. **Attacks**

Both services are robust against XSS attack using proper validation at service layer. Also, the Soap endpoint is robust against replay attack if and only if you put nonce in soap header. You can test this ability easily using SoapUI tool.

1. **Recommendation**

For ease of testing this services I didn’t force to use SSL (1-way or 2-way) at the service layer. However, it’s recommended to setup WildFly SSL with valid certificates in production and force using SSL in the code.

1. **Assumptions**

The use of basic authentication in this application is only for demo purpose. In production, we can use Token-based protocols like OAuth 2.0 or SAML assertion tokens in order to authenticate the users. This application has been written in a way that the replacement of Authentication protocol is quite easy in the future.

1. **JMS and Server Outage Resistance**

To be able to be reliable against server crashes, I used “durable” property for my JMS queue in order to persist this object in Application server data store. Also for putting data in JMS queue as well as database in one transaction, I used application server JTA (2-phase commit protocol) transactions. This code also applies new JMS 2 API through using of JmsContext object.

1. **Technology stack**

The following technologies/libraries have been used in this project:

* Java EE 7
* Junit & Arquillian as testing frameworks
* Apache Maven as dependency management & build tool
* JMS 2.0 API
* Apache Shiro 1.4
* Apache CXF
* WS-Security
* JBossWS
* CDI 1.1
* EJB 3.2
* JPA 2.1
* JMS 2.0
* Bean Validation 1.1
* WildFly 10