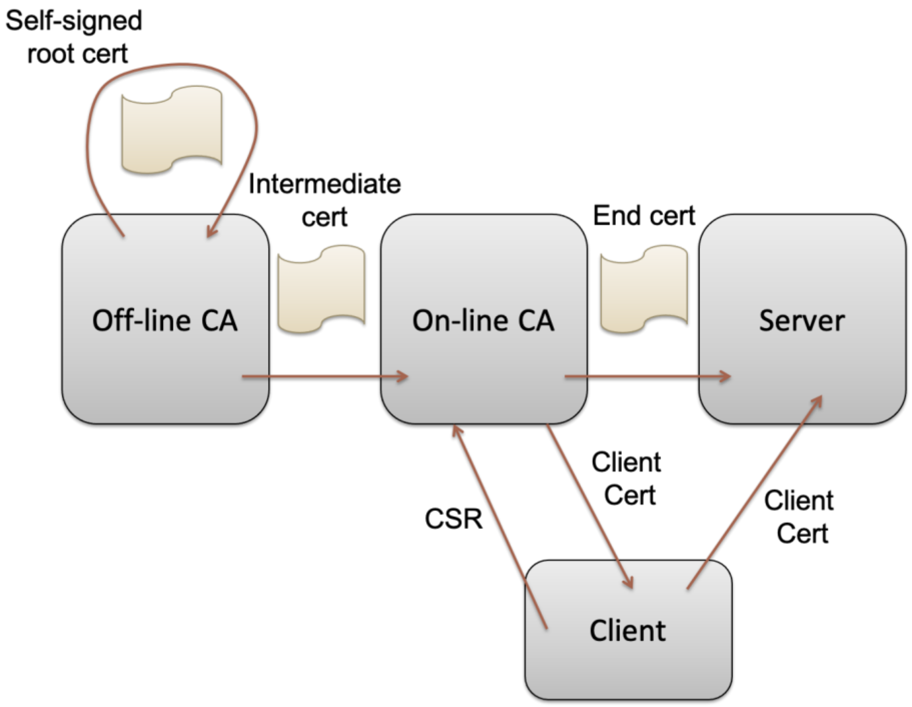
CS 594—Spring 2021
Enterprise and Cloud Security
Assignment Four—Secure REST Web Service

In this assignment, you will build on some of the infrastructure that you developed for the previous assignment, to implement a secure REST Web Service that uses client certificates for authentication and authorization of Web service clients. **The client only needs to support the operation of posting a message.**

We are now going to complete the scenario for the previous assignment:



You are provided with these additional Maven projects:

1. `ChatWebApp:` This is the same application as in Assignment 2. The file structure has been modified so you can compile it just using Maven (`Run As…|Maven Install`), and you should copy your solution from Assignment 2 (modified Java files and web.xml) into this project. It also includes `ChatService` and `ChatInit` as dependences, so that Maven generates a standalone WAR archive `chat-web.war` for deployment. We no longer use `ChatApp`.
2. `ChatPkiWebService`: A JAX/RS-based Web service that provides a simple certificate authority for issuing client certificates. Web clients authenticate using BASIC authentication. Maven will generate a WAR archive `chat-pki.war` that you can deploy in Payara Server.
3. `ChatPkiServiceClient`, `ChatPkiService`: Services for the CA Web service.

4. `ChatRestWebService`: A JAX/RS-based Web service that provides access to the functionality of the chat application that you completed and secured for Assignment 2. This only supports the operation of posting a new message. Web clients authenticate using client certificates issued by the online CA. Maven will generate a WAR archive `chat-rest.war` that you can deploy in Payara Server.
5. `ChatRestServiceClient`, `ChatRestService`: Services for the REST Web service.
6. `ChatRestWebClient`: A JAX/RS-based Web service client that provides access to the Web services via a CLI. You will use this to generate a self-signed client certificate, send a CSR to the online CA, and authenticate to the messaging Web service using the client certificate from the online CA. You can view the results in the chat Web app from Assignment Two.
7. `ChatCerts`: Provides a CDI bean CertsService that is used by both the Web services and the Web client to implement some basic certificate functionality (e.g. loading and saving keystores). This incorporates a lot of the code from the certificate manager from Assignment 2.
8. `ChatRoot`: An updated Maven parent module. You should always run Maven from this project.

You should install these projects in the same directory as your other projects, and then import the new projects as Maven projects into your Eclipse workspace.

Setup

To set up your certificate manager, you should first copy the application server keystore, `keystore.jks`, from the `domains/domain1/config` subdirectory of the Payara installation directory, to the `certs-online` subfolder of your base directory:

```
docker cp \
   cid:/opt/payara/appserver/glassfish/domains/domain1/config/keystore.jks \
   certs-online
```

Do this before you create the server certificate or the online CA certificate. Then use the certificate manager to create the root CA certificate, the server SSL (application server) certificate and the online CA certificate. Note that the application server keystore, `keystore.jks`, and truststore, `cacerts.jks`, are encrypted (for integrity) with the master password, with a default value of "changeit". These commands in your Dockerfile will change the master password and save the new password in the server container[1]:

```
RUN echo \
'AS_ADMIN_MASTERPASSWORD=changeit\nAS_ADMIN_NEWMASTERPASSWORD=newpassword' \
  >> /opt/masterpwdfile
```

[1] You are not expected to do this for the assignment.

```
RUN ${PAYARA_PATH}/bin/asadmin change-master-password \
   --passwordfile=/opt/masterpwdfile ${PAYARA_DOMAIN}
```

Obviously hardcoding passwords in your Dockerfile is not a best practice. A better option would be to put the password in a text file that is made available in the container by Docker Secrets.

Creating the root CA certificate with your certificate manager will save the credential in `caroot.p12`, a PKCS12 keystore in the `certs-offline` subdirectory.

Creating the server SSL certificate will save the credential in `keystore.jks` (in the `certs-online` subdirectory) and will save the root CA certificate in the application truststore, `cacerts.jks`. Clients will import the root CA certificate into their truststore, so that they will trust SSL certificates provided by the server and issued by the root CA. Inserting the root CA certificate in `cacerts.jks` ensures that the administration console for Payara trusts the server it is administering. Any clients wishing to access the server safely would include the root CA certificate in their truststore, so that they can authenticate the server[2].

Creating the online CA certificate will save the credential in the online `keystore.p12`, that will be used by the *application* rather than the *application server.* The certificate will be saved in `cacerts.jks`, the application server truststore, so that the application server will accept client certificates issued by the online CA.

To deploy the certificates to your service, you should create another folder, for building a custom Payara docker image. In addition to the JDBC driver from Assignment 1, copy the app keystore, app server keystore and app server truststore (`keystore.p12, keystore.jks` and `cacerts.jks`, respectively) to this folder. The application server uses the master password to decrypt its keystore (`keystore.jks`) and truststore (`cacerts.jks`). The online CA will need keystore and key passwords to decrypt its credentials in its own keystore, `keystore.p12`, and these should have the same password definitions as used by the credential manager. Copy the file with these passwords, default name `passwords.properties`, to the directory that will contain your Dockerfile. to the docker container[3]. Now create a Dockerfile in the folder, with these contents:

[2] Once you follow these steps, you will have generated the credentials for the server *administrator*, signed by your root CA, and stored in the server keystore. You should still need to generate the credentials for the server *instance*. Use the same server truststore and keystore files. Using the sequence described above, you could generate the server instance credentials signed by your root CA, and store them in the server keystore with key alias `glassfish-instance` (This alias is assumed by Payara). However the admin GUI still works after disabling and re-enabling secure administration, presumably because the server instance does not authenticate to the admin GUI.

[3] Again, a better practice would be to use Docker Secrets to make these passwords available to the server application.

```
FROM payara/server-full
COPY --chown=payara postgresql.jar \
   ${PAYARA_DIR}/glassfish/domains/${DOMAIN_NAME}/lib/
COPY --chown=payara cacerts.jks \
   ${PAYARA_DIR}/glassfish/domains/${DOMAIN_NAME}/config/cacerts-new.jks
COPY --chown=payara keystore.jks \
   ${PAYARA_DIR}/glassfish/domains/${DOMAIN_NAME}/config/keystore-new.jks
COPY --chown=payara keystore.p12 \
   ${PAYARA_DIR}/glassfish/domains/${DOMAIN_NAME}/config/keystore.p12
COPY --chown=payara  passwords.properties \
   ${PAYARA_DIR}/glassfish/domains/${DOMAIN_NAME}/config/passwords.properties
```

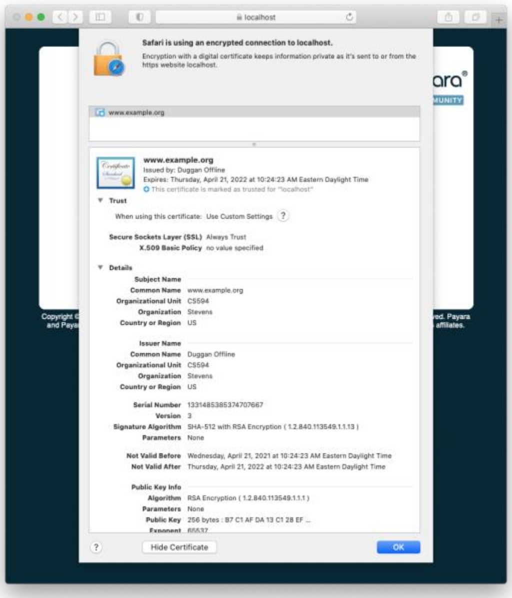
Build the custom image and run the container, as in previous assignments. Now connect to the running container and replace the existing keystore and truststore with the updated versions from the certificate manager:

```
$ docker exec -it container-id bash
# cd /opt/payara/appserver/glassfish/domains/domain1/config
# mv cacerts-new.jks cacerts.jks
# mv keystore-new.jks keystore.jks
# asadmin disable-secure-admin
# asadmin restart-domain
```

This last command will stop the container. Restart the container, connect to it again, and re-enable secure administration:

```
# asadmin enable-secure-admin
# asadmin restart-domain
```

You will have to start the container again after this last command. If you access the application server with a Web browser, you should see that the Web server provides a certificate signed by your root CA:



Safari is using an encrypted connection to localhost.

Encryption with a digital certificate keeps information private as it's sent to or from the https website localhost.

www.example.org

www.example.org
Issued by: Duggan Offline
Expires: Thursday, April 21, 2022 at 10:24:23 AM Eastern Daylight Time
This certificate is marked as trusted for "localhost"

Trust

When using this certificate: Use Custom Settings

Secure Sockets Layer (SSL)	Always Trust
X.509 Basic Policy	no value specified

Details

Subject Name	
Common Name	www.example.org
Organizational Unit	CS594
Organization	Stevens
Country or Region	US
Issuer Name	
Common Name	Duggan Offline
Organizational Unit	CS594
Organization	Stevens
Country or Region	US
Serial Number	1331485385374707667
Version	3
Signature Algorithm	SHA-512 with RSA Encryption (1.2.840.113549.1.1.13)
Parameters	None
Not Valid Before	Wednesday, April 21, 2021 at 10:24:23 AM Eastern Daylight Time
Not Valid After	Thursday, April 21, 2022 at 10:24:23 AM Eastern Daylight Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes : B7 C1 AF DA 13 C1 28 EF ...
Exponent	65537

Running the Client

Run the client in a different working directory from the certificate manager. The client will by default manage a keystore and truststore, `clientKeystore.jks` and `clientTruststore.jks`, in the directory where you run the app. It will also expect a properties file `passwords.properties`, *different from the properties file for the server*, that stores the passwords for accessing client keys and certificates:

```
client.keystore.password=...
client.key.password=...
client.truststore.password=...
```

The client CLI provides these operations:

1. `help`: Self-explanatory.
2. `import`: Import the PEM file (`--cert`) with the root CA certificate into the client truststore. This is used to authenticate the server to the client.
3. `init`: Generate a self-signed certificate in the client keystore. You will need to specify a distinguished name (`--dn`) whose common name should be a user name in the chat app.
4. `register`: Send a CSR to the online CA (`--caUri http://...:8080/chat-pki`). The client will need to specify a username (`--sender`) and password (`--password`) to authenticate against the online CA. This can only be done after importing the root CA certificate and generating a self-signed credential.
5. `post`: Post a new chat message on the chat server (`--serverUri http://...:8080/chat-rest`). This can only be done after obtaining a credential signature from the online CA, since the client authenticates using its client certificate.
6. `show`: Show the certificates in the keystore and truststore.

Modifying the Apps

You will have to finish parts of the code. You will have to complete missing parts in:

```
ChatCerts:
  CertsService
  CAUtils
ChatPkiService:
  PkiService
ChatRestWebClient:
  ClientCerts
  ChatClient
```

Most of the code changes will involve copying and pasting code from your solution to Assignment 3. The changes to `ChatClient` involve setting up authentication on the client side. You will need to add annotations to these server classes to enable authentication:

1. In `WSConfiguration` in `ChatPkiWebService`, annotate the class to enable BASIC authentication for HTTP requests, where the user database from Assignment 2 is used to authenticate user credentials.
2. In `WSConfiguration` in `ChatRestWebService`, annotate the class to enable client certificate authentication for HTTP requests (relying on the server truststore to verify client certificates). Assign parties that authenticate to the "`posters`" group (role).

You can continue to use your solution for Assignment 2, but follow the file structure of `ChatWebApp` with your solution, to make recompiling code easier:

1. Copy your Java code to `res/main/java` in the provided `ChatWebApp`.
2. Copy your `web.xml` (in `WebContent/WEB_INF`) to `res/main/webapp/WEB_INF` in the provided `ChatWebApp`.

Compiling and Testing

There are four applications for this assignment: three Web applications and a CLI application:

1. `chat-web`: This is the chat Web app you finished in Assignment 2.
2. `chat-pki`: This is an online CA (generated from ChatPkiWebService), that issues client certificates.
3. `chat-rest`: This is an online Web service that allows clients to post messages to the chat server. Clients must authenticate with client certificates issued by the online CA.
4. `chatclient`: This is the CLI for the Web service client. The use of this CLI uses the `init` command to create an initial self-signed certificate, then uses `register` to get this credential signed by the CA, and thereafter posts messages using this client certificate to authenticate.

Use Eclipse Maven (`Run As...|Maven Install`) from the `ChatRoot` project to compile all projects at the same time. The Maven POM files will leave WAR files for the Web applications, and a JAR file for the CLI, in a cs594 subfolder of your home directory. You should deploy the WAR files in Payara (`chat-web.war` first, because it initializes the user database), and then run the CLI.

In your testing, demonstrate the following with the CLI:

1. Show certificates after importing the root CA certificate.
2. Show client certificate after initializing (generating a self-signed certificate).

3. Show BASIC authentication to the server failing with the wrong credentials, then succeeding with the correct credentials. Show the client certificate after success, and show the server log. The register command will be of the form:
```
register --caUri http://...:8080/chat-pki
               --sender username --password password
```
4. Show client certificate authentication failing (trying to post a message) before registering with the server, then succeeding after getting a CA-signed client certificate. Show via the chat Web application the messages successfully posted, and show the server log. Make sure to post to port 8181 via https for this command:
```
post --serverUri https://...:8181/chat-rest --sender username
```

Submission

You should record mpeg videos of a demonstration of your assignment working. Make sure that your name appears at the beginning of the video. *Do not provide private information such as your email or cwid in the video*.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have the Eclipse projects for all the applications (the four Web applications and the chat client CLI). You should also provide a completed rubric for your assignment, as well as videos demonstrating the working of your assignment. Finally the root folder should contain the jar file for your compiled client, as well as the war files for your server applications.

It is important that you provide a completed rubric that documents your submission, included as a PDF document in your submission root folder. As part of your submission, export your Eclipse projects to your file system, and then include these folders as part of your archive file.