Guided Exercise: Securing an Application

In this lab work, you will use a database security scheme to secure the Sample application.

Resources	
Files:	/home/student/JB248/labs/security-dbrealm
App URL:	http://127.0.0.1:8080/example

Results

You should be able to configure an application to verify username and password, using a security module backed by the database.

before you start

Before beginning the guided exercise, run the following command to verify that EAP is installed in /opt/jboss-eap-7.0, that no EAP instance is running, and that the MySQL driver is installed:

```
[student@workstation ~]$ lab securing-dbrealm setup
```

1. Start the standalone instance of EAP by running the following command with the base directory located at /home/student/JB248/labs/standalone:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

2. Connect to the security data source.

In this step, you will configure the non-XA data source, which connects to the database, storing credentials used by the sample application. The data source will be used later by the security subsystem to obtain the username and password.

The database is a MySQL database running on localhost and named bksecurity. The credentials to access the database are bkadmin/redhat. To access it, use the JNDI name java:jboss/ datasource/bksecurity-ds.

2.1. Access the management console by browsing to 127.0.0.1:9990. Go to the Settings page.

use

The administrator username is jbossadm and the password is JBoss@RedHat123.

2.2. Navigate to the database subsystem by clicking Subsystems and then Data Sources.

23. Select the Non-XA data source type and click Add.

2.4. In the first window, select MySQL Data Source and click Next.

2.5. Use the following parameters to fill out the form:

- **Nombre: bksecurity-ds**
- **Nombre de JNDI: java:jboss/datasources/bksecurity-ds**

Click Next.

2.6. In step 2 of the selection menu, click Detected Driver and select the driver named mysql. This is the driver that was installed in the previous exercise.

Click Next.

2.7. Use the following information for the database connection:

- **Connection URL: jdbc:mysql://localhost:3306/bksecurity**
- **Username: bkadmin**
- **Password: redhat**

Click Next and then Finish, and the bksecurity data source will appear in the fourth column.

3. Configure the security domain.

Create a security domain that will use the data source added in the previous step to get the credentials used by the sample application from a database.
The security domain must have a default cache type, which will be used for authentication purposes.

3.1. Go back to the Subsystems column.

3.2. Click Security and Add in the Security Domain column to add a new security domain named bksecurity.

3.3. Enter the security domain name as bksecurity, set the Cache Type to default, and then click Save.

3.4. To configure the authentication modules, start the EAP CLI on a new terminal and connect to the standalone server. Enter jbossadm as the username and JBoss@RedHat123 as the password.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect
```

3.5. Use the following command to create an Authentication Module.

```
[standalone@localhost:9990] /subsystem=security/security-domain\ =bksecurity/
authentication\
=classic:add
```

use

The output will indicate that the server should be reloaded. However, this message should be ignored for now, as the server will be reloaded after the login module configuration is complete.

3.6. Create a login module that will connect to the data source created in the previous step. You should query the users and roles table to identify the credentials. The password is encrypted with a SHA-256 hash algorithm, and the hash uses a base64 encoding.

Use the following values when creating the login module:

- **Name: database**
- **Code: Database**
- **Marker: required**
- **These module options:**
  ÿ ***dsJndiName:*** **java:jboss/datasources/bksecurity-ds**

  ÿ ***principalsQuery:*** **select password from users where username=?**

  ÿ ***rolesQuery:*** **select role, "Roles" from roles where username=?**

  ÿ ***hashAlgorithm:SHA-256***

  ÿ ***hashEncoding: base64***

Use the following CLI command to create the login module:

```
[standalone@localhost:9990] /subsystem=security/security-domain=bksecurity/\
  authentication=classic/login-module=database:add( \
  code=Database, \
  flag=required, \
  module-options=[ \
   ("dsJndiName"=>"java:jboss/datasources/bksecurity-ds"), \
   ("principalsQuery"=>"select password from users where username=?"), \
   ("rolesQuery"=>"select role, 'Roles' from roles where username=?"), \
   ("hashAlgorithm"=>"SHA-256"), \
   ("hashEncoding"=>"base64") \ ])
```

use

This command can be copied and pasted from /home/student/JB248/labs/ security-dbrealm/login-module.

3.7. Reload the server to allow the changes to take effect:

```
[standalone@localhost:9990] :reload
```

3.8. Check the values closely by running the following command on the CLI:

```
[standalone@localhost:9990] /subsystem=security/security-domain=bksecurity\ /
authentication=classic/login-module=database:read-resource
```

The response should appear as follows:

```
{
        "outcome" => "success",
        "result" =>
            { "code" => "Database",
            "flag" => "required", "module"
            => undefined, "module-
            options" => { ("rolesQuery"
                  => "select role, "Roles" from roles where username=?"), ("principalsQuery" => "select password
                  from users where username=?"), ("dsJndiName" => "java:jboss/datasources/bksecurity-ds"),
                  ("hashAlgorithm" => "SHA-256"), ("hashEncoding" => "base64"), }




        }
}
```

use

If an error is found in module-options, each module option can be updated using the map-put command in the EAP CLI. For example, if there is an error with the dsJndiName value, the following command may update the module option:

```
[standalone@localhost:9990] /subsystem=security/security-domain\ =bksecurity/
authentication=classic/login-module=database\ :map-put(name=module-
 options,key="dsJndiName",\ value="java:jboss/datasources/
bksecurity-ds")
```

4. Set up application security.

In order to secure an application deployed in EAP, the first step is to modify the application so that it requires credentials to access it.

4.1. Use a text editor to open the web.xml file in the /home/ folder student/JB248/labs/security-dbrealm/example/src/main/webapp/WEB INF.

4.2. After the <welcome-file-list> section in the web.xml file, add the following XML, which places a security constraint on all URLs in the application and requires user authentication. You can copy and paste this XML

del archivo /home/student/JB248/labs/security-dbrealm/security constraint.xml.

```
<security-constraint> <web-
        resource-collection> <web-resource-
                  name>All resources</web-resource-name> <url-pattern>/*</url-pattern>
                  </web-resource-collection> <auth-
        constraint>

                  <role-name>*</role-name>
        </auth-constraint>
</security-constraint> <security-
role> <role-name>*</
        role-name> </security-role>
<login-config>

        <auth-method>BASIC</auth-method>
        <realm-name>bksecurity</realm-name> </login-
config>
```

4.3. Save the changes to web.xml.

5. Configure the security domain in the application.

5.1. Use a text editor to open the jboss-web.xml file in the /home/student/JB248/labs/security-dbrealm/example/src/main/webapp/WEB INF folder.

5.2. Inside the <jboss-web> tag, enter the following <security tag domain>:

```
<security-domain>bksecurity</security-domain>
```

bksecurity refers to the previously created security domain.

5.3. Save the changes to jboss-web.xml.

6. Package the application.

Open a new terminal and enter the following command to create a WAR file of the sample application:

```
[student@workstation ~]$ cd /home/student/JB248/labs/security-dbrealm/\ example/src/main/
webapp [student@workstation
webapp]$ jar -cvf example.war .
```

7. Deploy the application.

Using the administrative console or the CLI, deploy the newly created example.war file in /home/student/JB248/labs/security dbrealm/example/src/main/webapp/example.war to the standalone server.

```
[standalone@localhost:9990 /] deploy \ /home/student/JB248/labs/security-dbrealm/ example/src/main/
webapp/example.war
```

Monitor the server log for errors in the deployment. The application should be deployed without errors.

8. Check the security settings.

   8.1. Go to http://127.0.0.1:8080/example/. You will be prompted to sign in.

   8.2. Enter "admin" as the username and "admin" as the password. You should see the "Welcome to EAP 7" page once you have successfully logged in.

use

If authentication fails with those credentials, verify that the security domain has been configured with the correct values for each module option.

   8.3. Uninstall the Sample application from the standalone server using the CLI:

```
[standalone@localhost:9990] undeploy example.war
```

   8.4. Exit the CLI and stop the running EAP instance.

This concludes the guided exercise.

Configuring an LDAP-based security domain

Goals

After completing this section, students should be able to do the following:

- Configure a security domain based on the LDAP login module.

Defining a security domain based on LDAP

A security domain backed by an LDAP server, which stores users and role assignments for an application, can be defined using the Ldap login module. It integrates with an LDAP server and authenticates users against data stored in the LDAP Directory Information Tree (DIT). Users and roles are stored in Organizational Units (OUs) in the LDAP tree with the username used as the Distinguished Name (DN) to uniquely identify a user.

An LDAP-based security domain can be created with the EAP CLI, using the following command:

```
[standalone@localhost:9990] /subsystem=security/security-domain= \
    ldap-domain:add(cache-type=default)
```

```
[standalone@localhost:9990] /subsystem=security/security-domain=ldap-domain \
    /authentication=classic:add \
    (login-modules=[{"code"=>"Ldap","flag"=>"required", \ "module-
    options"=>[("java.naming.factory.initial"=> \
    "com.sun.jndi.ldap.LdapCtxFactory"), \
    ("java.naming.provider.url"=>"ldap://instructor:389"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("principalDNPrefix"=>"uid="),("principalDNSuffix"=>", \ ou=people,
    dc=redhat, dc=com"),("rolesCtxDN"=>"ou=Roles,dc=redhat,dc=com"), \ ("uidAttributeID"=>"member"),
    ("matchOnUserDN"=>"true"), \ ("roleAttributeID"=>"cn"), \
    ("roleAttributeIsDN"=>"false")]}])
```

The corresponding LDAP-based security domain XML definition is as follows:

```
<security-domain name="ldap-domain">
    <authentication>
        <login-module code="Ldap" flag="required">
            <module-option name="java.naming.factory.initial"
  value="com.sun.jndi.ldap.LdapCtxFactory"/>
            <module-option name="java.naming.provider.url" value="ldap://instructor:389"/> <module-option
            name="java.naming.security.authentication" value="simple"/> <module-option name="principalDNPrefix"
            value="uid="/> <module-option name="principalDNSuffix" value=",
            ou=people, dc=redhat, dc=com"/>
```