Lab 09_01: Securing an Application

Performance Checklist

Lab Overview:

In this exercise, you will configure the **example.war** application to require a user to login before accessing the application.

Lab Resources/Configuration:

Lab Files Location:	*LABS/Lab09_01*
Application URL:	`http://192.168.0.xx:8080/example2/`

Success Criteria: You can successfully login to the example2 application using credentials stored in a database.

Outcome: A secure application that uses a database for authentication.

Lab Outline:

1. Configure the Security Domain
2. Configure the Application Security
3. Configure the Security Domain
4. Package the Application
5. Deploy the Application
6. Verify the Security Settings

☐ 1. Configure the Security Domain

- ☐ 1.1. Go to the **Profiles** page of the Management Console and select the **ha** profile.
- ☐ 1.2. Click on the **Security Domains** link in the **Security** section.
- ☐ 1.3. Click the **Add** button and add a new security domain named **JBTravel**. (You are going to use this security domain later in the course for an application named **JBTravel**.)

☐ 1.4. Click the **View** link next to **JBTravel** in the list of **Security Domains**.

☐ 1.5. On the **Authentication** page (which you should be on right now), click the **Add** button to add a new login module. Enter **Database** for the code, and leave the **Flag** as required.

☐ 1.6. Click the Save button and a **Database** login module should now appear in the list of **Login Modules**.

☐ 1.7. Click on the **Module Options** tab for the **Database** module. Add three module options:

```
name="dsJndiName" value="java:jboss/JBTravelDatasource"
name="principalsQuery" value="select password from JBTRAVEL.USER where
 username=?"
name="rolesQuery" value="select null,'Roles' from JBTRAVEL.USER where
 username=?"
```

Note the JBTravel application does not use roles. Any valid user can access the application.

☐ 1.8. Verify your settings by entering the running the following CLI command.

```
/profile=ha/subsystem=security/security-domain=JBTravel:read-
resource(recursive=true)
```

The response should look like:

```
{
    "outcome" => "success",
```

```
    "result" => {
        "acl" => undefined,
        "audit" => undefined,
        "authorization" => undefined,
        "cache-type" => "default",
        "identity-trust" => undefined,
        "jsse" => undefined,
        "mapping" => undefined,
        "authentication" => {"classic" => {"login-modules" => [{
            "code" => "Database",
            "flag" => "required",
            "module-options" => [
                ("dsJndiName" => "java:jboss/JBTravelDatasource"),
                ("principalsQuery" => "select password from JBTRAVEL.USER
where username=?"),
                ("rolesQuery" => "select null, 'Roles' from JBTRAVEL.USER
where username=?")
            ]
        }]}}
    }
}
```

☐ 1.9. Check the **Messages** in the Management Console. If necessary, restart **`production-server-A`** and **`production-server-B`**.

☐ 2. Configure the Application Security

To secure an application deployed on EAP, the first step is to modify the application so that it requires credentials to access it.

☐ 2.1. Using a text editor, open the **`web.xml`** file of in the ***`LABS`*`/Lab09_01/example/WEB-INF`** folder.

☐ 2.2. After the **`<welcome-file-list>`** section (i.e., after the **`</welcome-file-list>`** tag), add the following XML, which defines a security constraint on all URLs to the application and requires a user to be authenticated. You can copy-and-paste this XML from the file ***`LABS`*`/Lab09_01/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>JBTravel</realm-name>
</login-config>
```

☐ 2.3. Save your changes to **`web.xml`**.

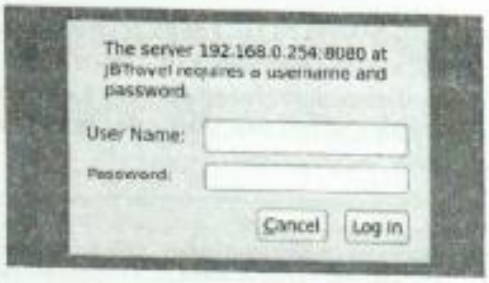
- ☐ 3. Configure the Security Domain in the Application
  - ☐ 3.1. Using a text editor, open the **jboss-web.xml** file in the ***LABS/Lab09_01/example/WEB-INF*** folder.
  - ☐ 3.2. Within the **<jboss-web>** tag, enter the following **<security-domain>** tag:

```
<security-domain>java:/jaas/JBTravel</security-domain>
```

  - ☐ 3.3. Save your changes to **jboss-web.xml**.
- ☐ 4. Package the Application
  - ☐ 4.1. Open a terminal window and change directories to ***LABS/Lab09_01/example***.
  - ☐ 4.2. Enter the following command to create a WAR file of the example application (don't forget the dot (.) at the end of the command, which specifies all files in the current folder:

```
jar -cvf example2.war .
```

- ☐ 5. Deploy the Application
  - ☐ 5.1. Using the Management Console or CLI, deploy your newly-created **example2.war** onto the **production-group** Server Group.
- ☐ 6. Verify the Security Settings
  - ☐ 6.1. Point your browser to `http://192.168.0.xx:38080/example2/`. You should be prompted to login:



  - ☐ 6.2. Enter "**mobius**" for the username and "**jboss**" for the password (which is a username/password combination from the JBTravel database). You should see the "Welcome to EAP 6" page once you are logged in successfully.

Lab 09_02: LDAP Login Module

Performance Checklist

Lab Overview: There is an LDAP server running on the instructor's machine on port 389. In this Lab, you will configure your EAP domain to use this LDAP for authenticating users attempting to access an application.

Lab Resources/Configuration:

Lab Files Location:	***LABS*/Lab09_02**
Application URL:	`http://192.168.0.xx:38080/guessLDAP`

Success Criteria: You can login to the guessLDAP application.

Outcome: A deployed application that authenticates using the LDAP.

Lab Outline:

1. The guessLDAP Application
2. Configure the LDAP Security Domain
3. Test the LDAP Security Domain

☐ 1. The guessLDAP Application

The application you are going to use in this exercise is already configured to require authentication.

☐ 1.1. In your ***LABS*/Lab09_02** folder is a web application named **guessLDAP**. In the ***LABS*/Lab09_02/guessLDAP/WEB-INF** folder, open the file **web.xml**.

☐ 1.2. What is the realm name used by this application?

_____

☐ 1.3. Open the **guessLDAP/WEB-INF/jboss-web.xml** file. What must be the name of a **<security-domain>** defined in **domain.xml** (or **standalone.xml**)?

_____

☐ 2. Configure the LDAP Security Domain

☐ 2.1. The CLI command to define the LDAP security domain is provided for you. Run the following command from the ***EAP_HOME*/bin** folder on RHEL:

```
./jboss-cli.sh -c --controller=192.168.0.xx:9999 --file=/home/student/
JB248/labs/Lab09_02/jb248_ldap-security-domain.cli
```

On Windows the command is:

```
jboss-cli.bat -c --controller=192.168.0.xx:9999 --file=c:\JB248\labs
\Lab09_02\jb248_ldap-security-domain.cli
```

☐ 2.2. Start the CLI.

☐ 2.3. Enter the following command in the CLI to verify the security domain settings of **jb248_ldap**:

```
/profile=ha/subsystem=security/security-domain=jb248_ldap:read-
resource(recursive=true)
```

You should see the following result:

```
{
    "outcome" => "success",
    "result" => {
        "acl" => undefined,
        "audit" => undefined,
        "authorization" => undefined,
        "cache-type" => "default",
        "identity-trust" => undefined,
        "jsse" => undefined,
        "mapping" => undefined,
        "authentication" => {"classic" => {"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")
            ]
        }]}}
    }
}
```

☐ 2.4. Go to the **Server Instances** page of the Management Console, then stop and restart **production-server-A** and **production-server-B**.

☐ 3. Test the LDAP Security Domain

☐ 3.1. Using the Management Console or CLI, deploy the **guessLDAP.war** file located in your ***LABS/Lab09_02*** folder onto the **production-group** server group.

☐ 3.2. Point your browser to `http://192.168.0.xx:38080/guessLDAP`. You should be prompted to login.

☐ 3.3. Enter "**bt1**" for the username and "**ldap1**" for the password, and you should be logged in successfully. Other valid login credentials include **admin/admin** and **bt2/ldap2**.

Lab 09_03: Encrypting a Password

Performance Checklist

Lab Overview:

In this exercise, you will encrypt the password for the **JBTravelDatasource**, which is currently stored as plain text in **domain.xml**.

Lab Resources/Configuration:

Lab Files Location:	*LABS/Lab09_03*
Application URL:	http://192.168.0.*xx*:38080/dstest/

Success Criteria: The password for the **JBTravel** datasource is no longer stored in **domain.xml** as plain text.

Outcome: The http://192.168.0.*xx*:38080/dstest/ page still works for the **JBTravel** datasource.

Lab Outline:

1. Create a Java Keystore
2. Run the Vault Script to Encrypt a Password
3. Configure the Vault
4. Configure the Datasource
5. Verify the Datasource

☐ 1. Create a Java Keystore

☐ 1.1. Open a terminal window and change directories to **/home/student**. (On Windows, go to **c:\JB248**.)

☐ 1.2. Enter the following command to create a keystore file named **vault.keystore**:

```
keytool -genkey -alias vault -keyalg RSA -keysize 1024  -keystore
 vault.keystore
```

Notice the value of **alias** is **vault** - you will need this information later.

☐ 1.3. The first prompt asks for a password. Simply use **"password"** as the password for your vault.

☐ 1.4. You are now prompted for company information. Feel free to enter your own information, or you can use the following for demonstration purposes:

```
What is your first and last name?
  [Unknown]:  EAP Vault
What is the name of your organizational unit?
  [Unknown]:  Red Hat
What is the name of your organization?
  [Unknown]:  Red Hat
```

```
What is the name of your City or Locality?
  [Unknown]:  Raleigh
What is the name of your State or Province?
  [Unknown]:  NC
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=EAP Vault, OU=Red Hat, O=Red Hat, L=Raleigh, ST=NC, C=US correct?
  [no]:  yes

Enter key password for <vault>
 (RETURN if same as keystore password):
```

Important

Be sure you use the same password as for the keystore! At this time, there is no method to support a different vault and keystore password.

- ☐ 1.5. Verify you now have a file named **vault.keystore** in your *STUDENT_HOME* folder.

☐ 2. Run the Vault Script to Encrypt a Password

- ☐ 2.1. Change directories to your ***EAP_HOME*/bin** folder.
- ☐ 2.2. Enter the following command to run the vault tool script - used for adding passwords to a vault. On RHEL:

```
./vault.sh
```

On Windows:

```
vault.bat
```

- ☐ 2.3. At the first prompt, enter 0 to select **Start Interactive Session**.
- ☐ 2.4. When prompted for the directory to store encrypted files, RHEL users enter **/home/student/**. (Note the trailing slash is required.) For Windows users, enter **C:\JB248**.
- ☐ 2.5. The **Keystore URL** is the path to the **vault.keystore** file you just created in the previous step. On RHEL, enter:

```
/home/student/vault.keystore
```

On Windows:

```
c:\JB248\vault.keystore
```

- ☐ 2.6. The password for your **Keystore** is **password**.
- ☐ 2.7. For the **8 character salt**, simply enter **12345678**.

- ☐ 2.8. Enter 50 for the **Iteration count**.
- ☐ 2.9. Make a note of the **Masked Password**. This is the encrypted value of your keystore's password, and it needs to appear in **`host.xml`**.
- ☐ 2.10. Enter **`vault`** for the **Keystore Alias**.
- ☐ 2.11. The vault tool has now connected to your vault. Enter 0 to **Store a password**.
- ☐ 2.12. At the prompt to enter attribute value, enter **`postgres`**, which is the password to connect to the PostgreSQL database running on your machine. You will have to enter **`postgres`** twice to verify.
- ☐ 2.13. At the prompt to enter a **Vault Block**, enter **`JBTravel`**.
- ☐ 2.14. At the **Enter Attribute Name** prompt, enter **`password`**.
- ☐ 2.15. Make a note of the resulting information, as prompted.
- ☐ 2.16. The password for the PostgreSQL database is now in **`vault.keystore`**, so enter 2 to **Exit** the vault tool.

☐ 3. Configure the Vault

- ☐ 3.1. Shutdown the **`host2`** and **`host3`** EAP instance by entering **`Ctrl+c`** in the terminal window.
- ☐ 3.2. Using a text editor, open **`host-slave.xml`** in **`machine2/domain/configuration`**.
- ☐ 3.3. The vault of an EAP instance is configured in the **`<vault>`** section of the Host Controller configuration file, which appears between the **`<paths>`** and **`<management>`** entries. Add the following **`<vault>`** section immediately before the **`<management>`** section:

```
    <vault>
 <vault-option name="KEYSTORE_URL" value="/home/student/vault.keystore" />
 <vault-option name="KEYSTORE_PASSWORD" value="MASK-31x/z0Xn83M4JaL0h5eK/
N" />
 <vault-option name="KEYSTORE_ALIAS" value="vault" />
 <vault-option name="SALT" value="12346670" />
 <vault-option name="ITERATION_COUNT" value="50" />
 <vault-option name="ENC_FILE_DIR" value="/home/student/* />
   </vault>
```

You can copy and paste this XML from the ***`LABS/Lab09_03/vault.xml`*** file. Windows users need to specify **`c:\JB248\`** for the **`ENC_FILE_DIR`** value.

- ☐ 3.4. Save your changes to **`host-slave.xml`**.
- ☐ 3.5. Start the **`host2`** EAP instance back up again. Within the first few lines of the log output, you should see log events showing a Security Vault successfully initialized and ready.
- ☐ 3.6. Repeat this for **`host3`**.

- ☐ 4. Configure the Datasource
  - ☐ 4.1. Go the **Profiles** page of the Management Console and select the **ha** profile.
  - ☐ 4.2. Click on the **Datasources** link in the **Connector** section.
  - ☐ 4.3. Click on the **JBTravel** datasource in the list of **Available Datasources**.
  - ☐ 4.4. Click the **Disable** button to disable the **JBTravel** datasource. (You can not modify the security settings of a datasource that is currently in use.)
  - ☐ 4.5. Select the **Security** tab, then click the **Edit** button.
  - ☐ 4.6. In the **Password** text field, remove the current password value.
  - ☐ 4.7. Within a **${}** notation, copy and paste the configuration entry displayed at the end of the vault tool script. The **Password** field should look like:

```
${VAULT::JBTravel::password::
MDJjMTI4OGUtOTU0YS00NzhlLWE4YmItZmE4YTYzZjcyYzk5TElORV9CUkVBS3ZhdWx0}
```

Note

Unfortunately, you will not be able to see your typing in the browser window.

  - ☐ 4.8. Click the **Save** button to save your changes.
  - ☐ 4.9. Re-enable the **JBTravel** datasource by clicking the **Enable** button.
- ☐ 5. Verify the Datasource
  - ☐ 5.1. Open the **domain.xml** in **machine1/domain/configuration**. Verify the **<password>** entry for the **JBTravel** is enclosed in **${}** and contains the text you copy-and-pasted from the previous step.
  - ☐ 5.2. Point your browser to `http://192.168.0.xx:38080/dstest/`.
  - ☐ 5.3. Enter **java:jboss/JBTravelDatasource** for the JNDI name and **jbtravel.airport** for the table name.
  - ☐ 5.4. Click the **Submit** button and verify that the database was connected successfully and the list of airports is displayed. If the airports are displayed, then you have successfully encrypted the database password of the PostgreSQL database!