Distributed Systems – Labs

Securing Java

Learning Outcomes:

Upon completion of this lab, the student should be able to

1. Use Java's digital signing tools to provide authentication in networked Java applications.

Tasks

Signing Code and Granting It Permissions

1. In this scenario, Susan wishes to send code to Ray. Ray wants to ensure that when the code is received, it has not been tampered with along the way - for instance someone could have intercepted the code exchange e-mail and replaced the code with a virus.

Create two directories named susan and ray. Extract the program T1\Count.java and save it in `susan`. Create a file named data.txt and save it in the susan folder. Run the Count program, which will count the number of characters in the file:

```
:/> java Count data.txt
```

2. Because Susan wants to send her data with authentication, she must create a public/private key pair. The java keytool allows users to do this:

:/> keytool -genkey -alias signFiles -keypass kpi135 -keystore susanstore -storepass ab987c

Enter all the required information for Susan. The `keytool` will create a certificate that contains Susan's public key and all her details.

This certificate will be valid for 90 days, the default validity period if you don't specify a - validity option.

The certificate is associated with the private key in a keystore entry referred to by the alias signFiles. The private key is assigned the password kpi135.

3.   Susan now wants to digitally sign the code to send it to Ray. The first step is to put the code into a JAR file:

```
:/> jar cvf Count.jar Count.class
```

Then the `jarsigner` tool can be used to sign the JAR:

```
:/> jarsigner -keystore susanstore -signedjar sCount.jar
Count.jar signFiles
```

You will be prompted for the store password (ab987c) and the private key password (kpi135).

4.   Susan can now export a copy of her certificate and send this with the signed JAR to Ray:

```
:/> keytool -export -keystore susanstore -alias signFiles –file
SusanJones.cer
```

The certificate will be in the file SusanJones.cer - this contains her public key which Ray can use to authenticate the origin of the file he received.

5.   Copy (simulated e-mail) the file SusanJones.cer and the signed JAR sCount.jar to the ray folder. Create or copy a file named data.txt to put in Ray's folder. Try to execute the code with the security manager in place:

```
:/> java -Djava.security.manager -classpath sCount.jar Count data.txt
```

Note the AccessControlException - you are not permitted access the disk with the Security Manager in operation.

6.   Ray will now create his own keystore (use any password you want), into which he will import Susan's details:

```
:/> keytool -import -alias susan -file SusanJones.cer -keystore raystore
```


7.   Ray can now verify that the code, sCount.jar was signed by Susan:

```
:/> jarsigner -verify -verbose -keystore raystore sCount.jar
```


Note:
Ray can also give any code signed by Susan permission to access certain files or perform operations it would not otherwise be permitted to do, by creating the following policy file (raypolicy.policy) (using policytool or TextPad):

```
keystore "raystore";

grant signedBy "susan" {

permission java.io.FilePermission "data.txt", "read";
```

```
};
```

and using this when running the code:

```
:/> java -Djava.security.manager -Djava.security.policy=raypolicy.policy -
classpath sCount.jar Count data.txt
```

In the next steps we will see how the Policy file can be setup using **Policy Tool**.

Set Up a Policy File to Grant the Required Permission

You will use the Policy Tool to create a policy file named **exampleraypolicy** and in it grant a permission to code from a signed JAR file.

The JAR file must have been signed using the private key corresponding to the public key imported into Ray's keystore (`raystore`) in the previous step. The certificate containing the public key is aliased by `susan` in the keystore. We will grant such code permission to read any file in the `C:\TestData\` directory.

Start the Policy Tool

1. To start Policy Tool, simply type the following at the command line:

```
policytool
```

This brings up the Policy Tool window. Whenever Policy Tool is started, it tries to fill in this window with policy information from what is sometimes referred to as the "user policy file," which by default is a file named `.java.policy` in your home directory. If Policy Tool cannot find the user policy file, it reports the situation and displays a blank Policy Tool window (that is, a window with headings and buttons but no data in it, as shown in the following figure.
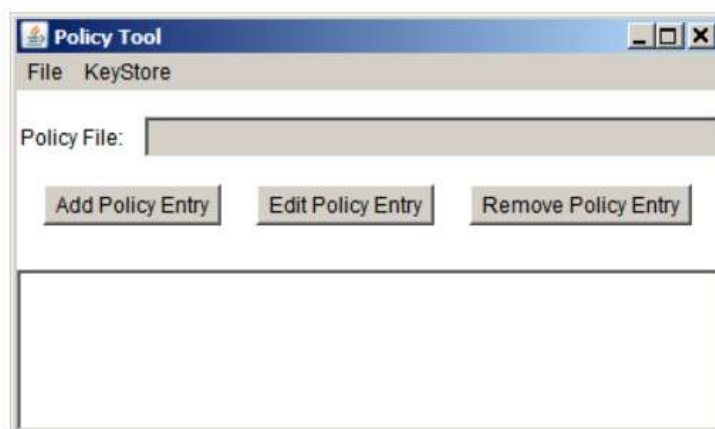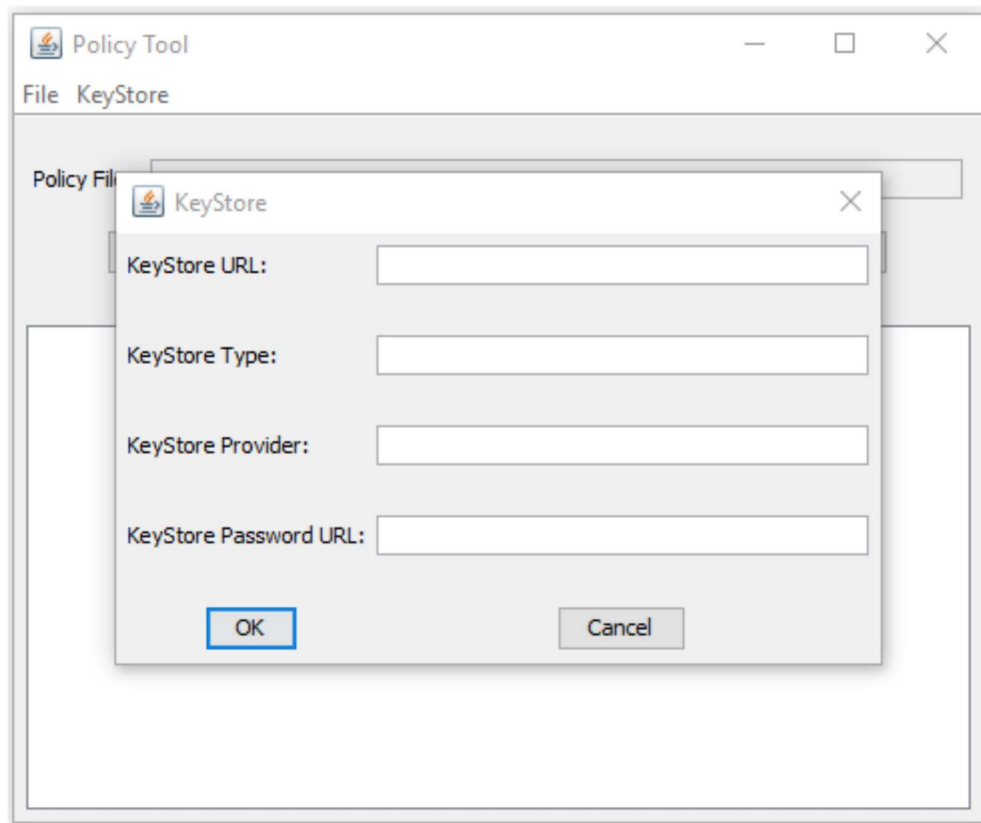

Specify the Keystore

First, you need to specify the keystore containing the certificate information aliased by susan.

2. Choose the **Edit** command from the **Keystore** menu of the main Policy Tool window.

    This brings up a dialog box in which you can specify the keystore URL and the keystore type.

3. Type a file URL into the text box labelled **KeyStore URL** to identify the location of the *exampleraystore* keystore file.

    ```
    file:/C:/Test/exampleraystore
    ```

4. You can leave the text box labelled **KeyStore Type** blank if the keystore type is the default one, as specified in the security properties file.

    Your keystore will be the default type, so leave the text box blank.

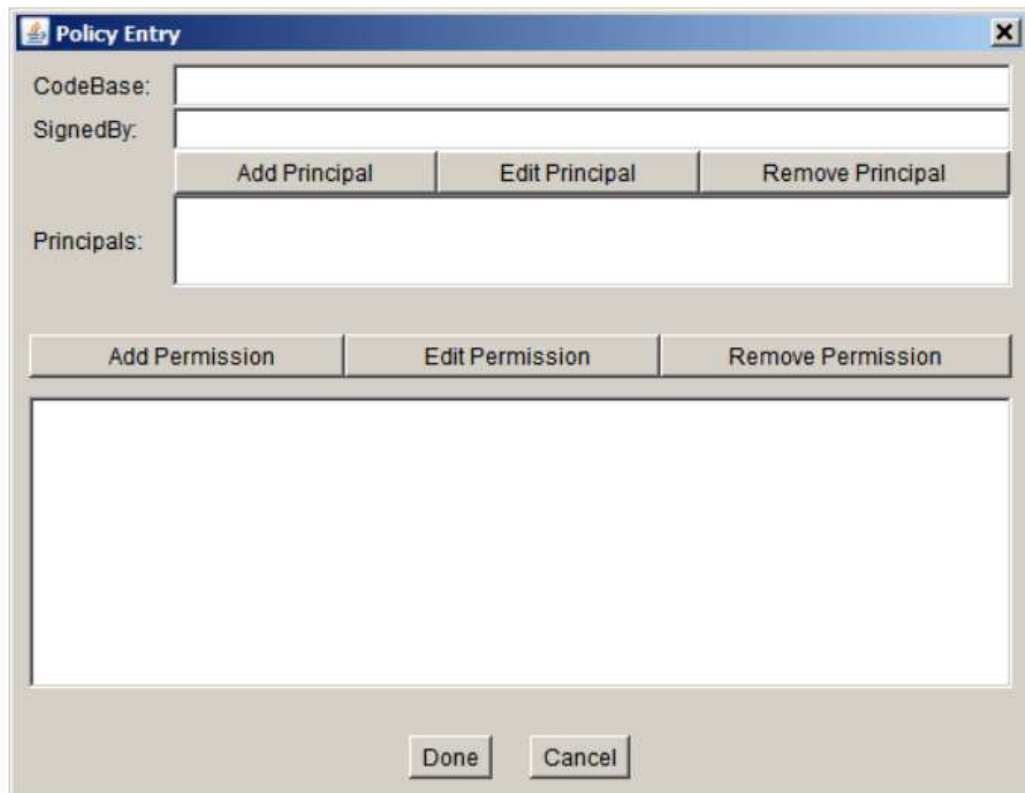5. When you are done specifying the Keystore URL, choose **OK**.

Specify the Policy Entry
Next, you need to create a policy entry granting all code in JAR files signed by the alias susan permission to read any and all files in your directory.

Note that "Code signed by susan" is an abbreviated way of saying "Code in a class file

contained in a JAR file, where the JAR file was signed using the private key corresponding to the public key that appears in a keystore certificate in an entry aliased by `susan`."

6. Choose the **Add Policy Entry** button in the main Policy Tool window. This brings up the Policy Entry dialog box.

`codebase` refers to the original location of the code i.e. where it was downloaded from.

`Signed By` is used when we digitally sign code so we can authenticate that it was written by a certain person.

If you have both a codebase and a Signed By entry, the permission(s) will be granted only to code that is both from the specified location and signed by the named alias.

7. Using this dialog box, type the following alias into the **SignedBy** text box:

   susan

8. Leave the **CodeBase** text box blank, to grant *all* code signed by `susan` the permission, no matter where it comes from.

   If you wanted to restrict the permission to just code signed by susan that comes from the C:\Test\ directory, you would type the relevant file URL into the CodeBase text box:

```
file:/C:/Test/*
```

9. To add the permission, choose the **Add Permission** button.

    This brings up the Permissions dialog box.
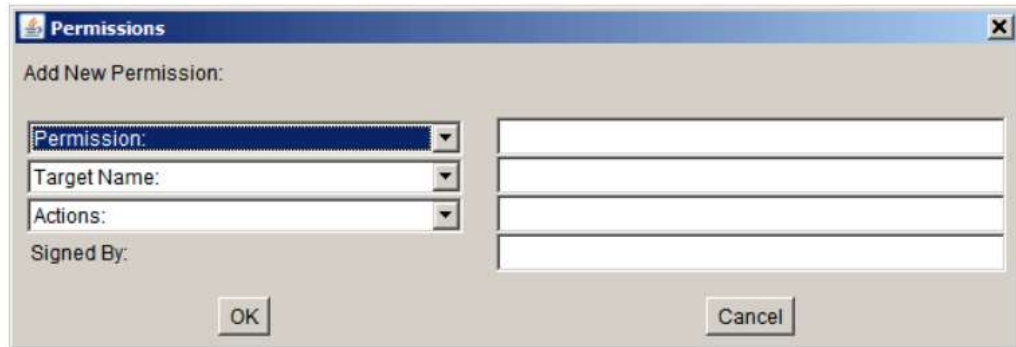
    Do the following.

    a. Choose **File Permission** from the Permission drop-down list. The complete permission type name (`java.io.FilePermission`) now appears in the text box to the right of the drop-down list.

    b. Type the following in the text box to the right of the list labeled Target Name to specify all files in the directory where the Count.java file is, e.g. (C:\TestData*).

    c. Specify read access by choosing the **read** option from the Actions drop-down list.

        Now the Permissions dialog box looks like the following.
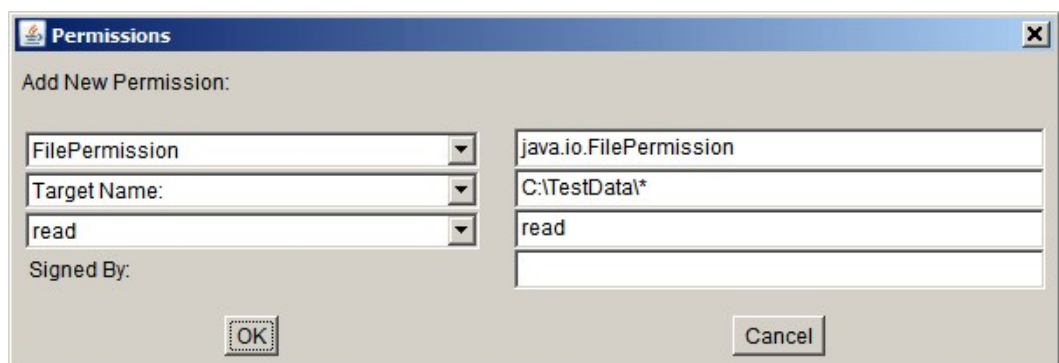
    d. Choose the **OK** button.

        The new permission appears in a line in the Policy Entry dialog.

    e. Click **Done**.

Save the Policy File

10. To save the new policy file you've been creating, choose the **Save As** command from the **File** menu.

    This brings up the **Save As** dialog box.

11. Navigate the directory structure to get to the directory in which to save the policy file: the `Test` directory on the `C:` drive.

12. Type the file name

    exampleraypolicy

13. Then choose the **Save** button.

    The policy file is now saved, and its name and path are shown in the text box labelled Policy File.

14. Exit the Policy Tool by selecting the **Exit** command from the **File** menu.

See the Policy File Effects
You should now be able to successfully execute the `Count` program to read and to count the characters in a file from the specified directory, even when you run the application with a security manager.

15. Type this when running the code:

    ```
    :/> java -Djava.security.manager -
    Djava.security.policy=raypolicy.policy -classpath sCount.jar Count
    data.txt
    ```

    If it still reports an error, something is wrong in the policy file.

Further Reading

Java tutorials:
https://docs.oracle.com/javase/tutorial/security/
https://docs.oracle.com/javase/tutorial/security/toolsign/index.html