

CMPU4021 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.
2. Create a Policy File

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 kpil35 -  
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.

Note: As we used the preceding `keystore` command, you will be prompted for your distinguished-name information. Following are the prompts; the bold indicates what you should type.

```
What is your first and last name?  
[Unknown]: Susan Jones  
What is the name of your organizational unit?  
[Unknown]: Purchasing  
What is the name of your organization?  
[Unknown]: ExampleCompany
```

```
What is the name of your City or Locality?  
[Unknown]: Cupertino  
What is the name of your State or Province?  
[Unknown]: CA  
What is the two-letter country code for this unit?  
[Unknown]: US  
Is <CN=Susan Jones, OU=Purchasing, O=ExampleCompany,  
L=Cupertino, ST=CA, C=US> correct?  
[no]: y
```

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.

4. 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
```

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

5. 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
```

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

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

7. 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.

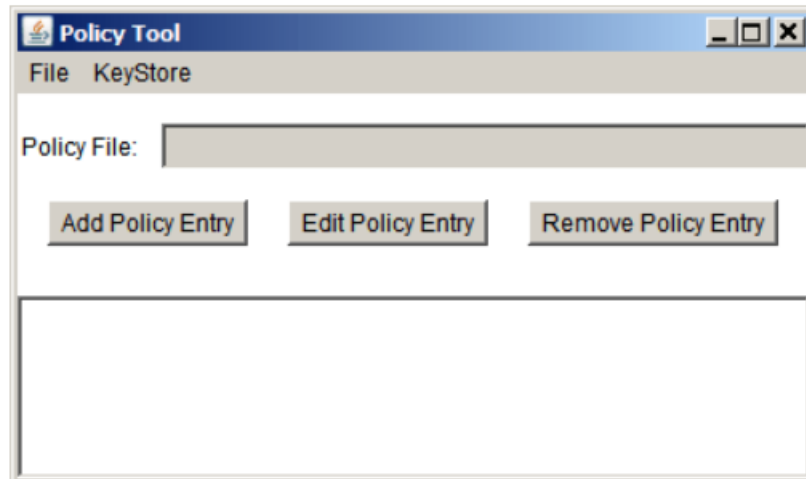
Note: Alternatively, You can follow the Lesson: Creating a Policy File, at

<https://docs.oracle.com/javase/tutorial/security/tour1/index.html>

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

You will grant all code in JAR files signed by the alias `susan` read access to all files in the `C:\TestData\` directory. You need to

1. Specify the keystore containing the certificate information aliased by `susan`
2. Create the policy entry granting the permission

The keystore is the one named `exampleraystore` created earlier.

To specify the keystore, choose the **Change Keystore** command in the **Edit** 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.

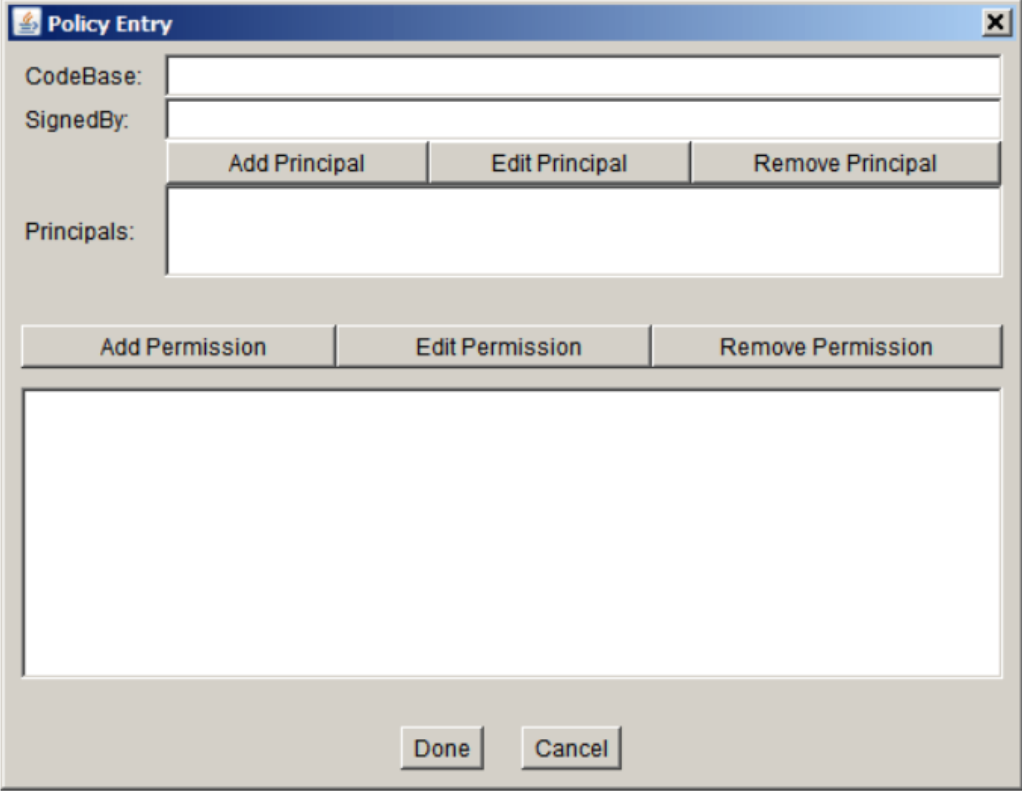
To specify the keystore named `exampleraystore` in the `Test` directory on the `C:` drive, type the following `file` URL into the text box labeled **New KeyStore URL**

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

You can leave the text box labeled **New 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.

Add a Policy Entry with a SignedBy Alias

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

The image shows a Java Policy Entry dialog box. It has a title bar with a close button. The dialog contains two text input fields: 'CodeBase:' and 'SignedBy:'. Below the 'SignedBy:' field is a horizontal bar with three buttons: 'Add Principal', 'Edit Principal', and 'Remove Principal'. Below this bar is a list box labeled 'Principals:'. Below the list box is another horizontal bar with three buttons: 'Add Permission', 'Edit Permission', and 'Remove Permission'. At the bottom of the dialog are two buttons: 'Done' and 'Cancel'.

To grant code signed by `susan` permission to read any files in your directory, you need to create a policy entry granting this permission. 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`."

`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.

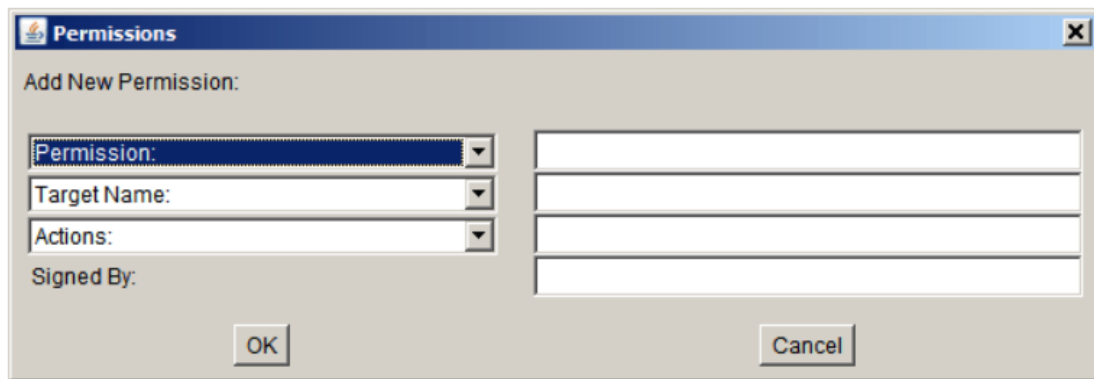
If you have both a `codebase` and a `Signed By` entry, the permission(s) will be granted only to code that is both from the specified location and signed by the named alias

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

susan

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

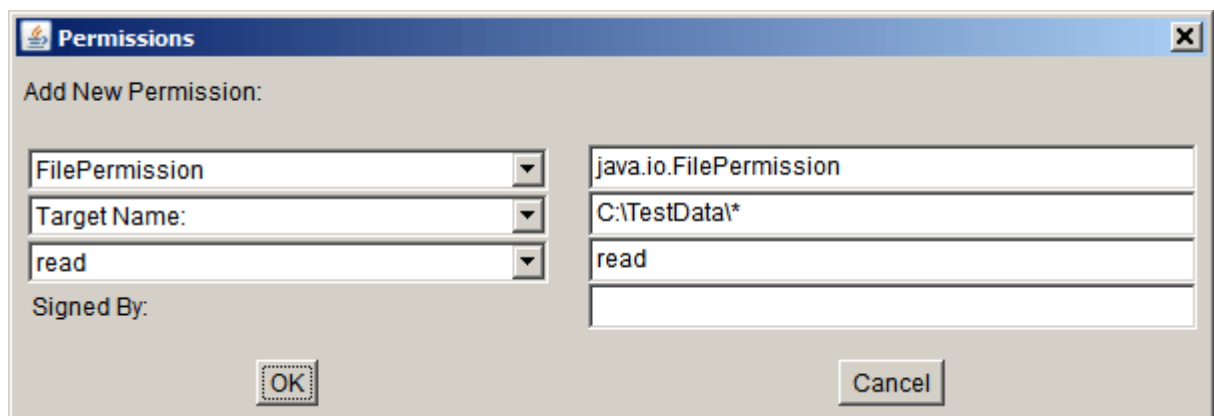
To add the permission, choose the **Add Permission** button. This brings up the Permissions dialog box.



Do the following.

1. 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.
2. 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*`)
3. Specify read access by choosing the **read** option from the Actions drop-down list.

Now the Permissions dialog box looks like the following.



Choose the **OK** button. The new permission appears in a line in the Policy Entry dialog, and click Done.

Save the Policy File

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.

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

```
expleraypolicy
```

Then choose the **Save** button. The policy file is now saved, and its name and path are shown in the text box labeled Policy File.

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

See the Policy File Effects

In the previous steps you created an entry in the `expleraypolicy` policy file granting code signed by `susan` permission to read files from the `C:\TestData\` directory (or the `testdata` directory in your home directory if you're working on UNIX). Now you should 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.

Observe Application Freedom

A security manager is *not* automatically installed when an *application* is running. We will see how to apply the same security policy to an application found on the local file system as to downloaded sandbox applets. But first, let's demonstrate that a security manager is by default not installed for an application, and thus the application has full access to resources.

Extract the program `T2\Count.java`

As you can see if you examine the source file, this program tries to get (read) the property values, whose names are `"os.name"`, `"java.version"`, `"user.home"`, and `"java.home"`.

Now compile and run `GetProps.java`. You should see output which shows that the application was allowed to access all the property values, for example:

```
C:\DIT\19_20\DS\Labs\Week 5\T2>java GetProps
About to get os.name property value
  The name of your operating system is: Windows 7
About to get java.version property value
  The version of the JUM you are running is: 1.8.0_221
About to get user.home property value
  Your user home directory is: C:\Users\Edina.Hatunicwebster
About to get java.home property value
  Your JRE installation directory is: C:\Program Files\Java\jdk1.8.0_221\jre
```



You can further follow the Lesson “See How to Restrict Applications” at

<https://docs.oracle.com/javase/tutorial/security/tour2/step2.html>

This lesson shows how to use a security manager to grant or deny access to system resources for applications. The lesson also shows how resource accesses, such as reading or writing a file, are not permitted for applications that are run with a security manager unless explicitly allowed by a permission entry in a policy file.

Further Reading

Java tutorials:

<https://docs.oracle.com/javase/tutorial/security/>

<https://docs.oracle.com/javase/tutorial/security/toolsign/index.html>

<https://docs.oracle.com/javase/tutorial/security/tour2>