CMPU4021 Distributed Systems – Labs

Securing Java

Key management

Creating a keystore

A keystore stores cryptographic keys and certificates and is frequently used in conjunction with servers and clients. A keystore is usually a file, but it can be a hardware device.

Java supports the following types of keystore entries:

- PrivateKey: This is used in asymmetric cryptography
- Certificate: This contains a public key
- SecretKey: This is used in symmetric cryptography

There are different types of keystores that are supported by Java (8): JKS, JCEKS, PKCS12, PKCS11, and DKS:

- JKS - Java KeyStore that usually has an extension of jks.
- JCEKS - Java Cryptography Extension KeyStore (JCE). It can store all three keystore entity types, provides stronger protection for keys, and uses a jceks extension.
- PKCS12: this keystore can be used with other languages. It can store all three keystore entity types, and it uses an extension of p12 or pfx.
- PKCS11: This is a hardware keystore type.
- DKS: This is the Domain KeyStore (DKS) that holds a collection of other keystores.

The default keystore type in Java is JKS. Keystores can be created and maintained using the keytool command line tool or with Java code.

Creating and maintaining a keystore with keytool

At the command prompt, enter the following command. This will start the process of creating a keystore in a file named keystore.jks. The alias is another name that you can use to reference the keystore:

- `keytool -genkey -alias mykeystore –keystore keystore.jks`

You will then be prompted for several pieces of information as follows. Respond to the prompts as appropriate. The passwords that you enter will not be displayed.

Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: some name
What is the name of your organizational unit?
[Unknown]: development
What is the name of your organization?
[Unknown]: mycom.com
What is the name of your City or Locality?
[Unknown]: some city
What is the name of your State or Province?
[Unknown]: some state
What is the two-letter country code for this unit?
[Unknown]: jv

You will then be prompted to confirm the input as follows. Respond with yes if the values are correct:
Is CN=some name, OU=development, O=mycom.com, L=some city, ST=some state, C=jv correct?
[no]: yes

You can assign a separate password for the key, as shown next:

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

The keystore is then created. The contents of a keystore can be displayed using the –listargument, as shown next. The –v option produces verbose output:

- `keytool -list -v -keystore keystore.jks -alias mykeystore`

The keystore password needs to be entered along with the alias name.

Keytool command-line arguments

Entering the information for a keystore can be tedious. One way of simplifying this process is to use command line arguments. The following command will create the previous keystore:

```
> keytool -genkeypair -alias mykeystore -keystore
keystore.jks -keypass
password -storepass password -dname "cn=some name,
ou=development,
o=mycom.com, l=some city, st=some state c=jv
```

Matching double quote at the end of the command line is not needed.

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.

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.

Note: You will get warning about th emigration to PKCS12. Ignore it – as we are showing the principles of Signing and Verification.

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 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 -cp sCount.jar Count data.txt
```

Note the AccessControlException: access denied - 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
```

Permissions and Security Policy

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 a text editor):

```
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
```

You can read more info about Policy File Format in Java 19 at:

https://docs.oracle.com/en/java/javase/19/security/java-se-platform-security-architecture.html#GUID-B1EE1A73-CF14-40CB-A524-F76600BC9DF0

A more common way of creating secure clients and servers uses the `SSLServerSocket` Class (from JSSE). This performs the automatic encryption and decryption of data based on a secret key found in a `keystore`.

Java Secure Socket Extension (JSSE) - Optional

Read about Java Secure Socket Extension (JSSE)

https://docs.oracle.com/javase/10/security/java-secure-socket-extension-jsse-reference-guide.htm#JSSEC-GUID-93DEEE16-0B70-40E5-BBE7-55C3FD432345

Try to run the sample code from T2 folder. The description of the code can be found at:

https://docs.oracle.com/javase/10/security/sample-code-illustrating-secure-socket-connection-client-and-server.htm#JSSEC-GUID-A4D59ABB-62AF-4FC0-900E-A795FDC84E41

References and Further Reading

https://docs.oracle.com/en/java/javase/19/security/java-security-overview1.html#GUID-F8BE6C49-3506-4D1A-8E4E-053CA439D1E2

https://docs.oracle.com/en/java/javase/19/security/java-secure-socket-extension-jsse-reference-guide.html#GUID-93DEEE16-0B70-40E5-BBE7-55C3FD432345

Learning Network Programming with Java, Reese, R.M., 2015 Packt Publishing