Course unit 12

Assignment D: access control

1 Introduction

In course unit 11 we introduced the Java security model. In the early Java versions, the distinction between trusted and untrusted Java code is based on the location of the code. Local applications are treated as trusted code, while applets downloaded from the network are treated as untrusted code and their permissions are restricted by the Java sandbox. Code signing was added in JDK 1.1. In addition to the location of the code, the signer of the code can now also be included in the decision whether the code should be treated as trusted or untrusted. An applet signed by a trusted party can be executed as trusted code. However, access control still follows an all-or-nothing approach: code runs either within the sandbox with strictly limited permissions or outside the sandbox with unlimited permissions.

Java 2 offers more fine-grained access control. The identity of the code is still determined by the location and signature of the code, but policies can specify which permissions are assigned to the code. The permissions indicate which actions can be performed on which resources. The AccessController checks the policies during code execution and applies stack inspection to determine if an action is allowed based on the permissions.

Access control in Java 2 is code based, since the policy is directly related to the identity of the code base. This differs from traditional access control in operating systems, in which access control is based on users: the identity of the user executing the code determines with which rights the code is executed. User-based access control in a Java application is not possible. This can be problematic for applications in which different roles and users are required. An example is a servlet on a web server that different types of users and administrators use (see for example the web auction in assignment A). To solve this problem, JAAS (Java Authentication and Authorization Service) was introduced, an extension of Java 2 that allows user-based access control. With the help of JAAS and Java 2, both code-based and user-based access control are possible.

In this assignment you will apply JAAS to add user-based access control to a Java application. In the remainder of this document, Section 2 elaborates on how JAAS works. Sections 3 and 4 contain tutorials on how to use JAAS for user authentication and user authorization. Section 5 presents the Java application that you will be extending and the tasks you should perform. Section 6 describes what to submit.

2 JAAS

Weblinks

The paper 'User authentication and authorization in the Java platform' by Charlie Lai et al. is available on the course site in yOUlearn. It presents the basics of JAAS. Study this paper.

For more detailed information on JAAS, see the JAAS Reference Guide. This reference guide refers to a number of tutorials, of which some are included in the Sections 3 and 4 of this assignment.

3 User authentication in JAAS

Weblinks

The JAAS Authentication Tutorial explains how users are authenticated in JAAS. The tutorial takes a simple application as an example. By the end of the tutorial, you will have an application in which a principal testUser with password testPassword can be authenticated. For simplicity, the username and password are hard-coded in the source code. Study this tutorial and run the corresponding code.

4 User authorization in JAAS

Weblinks

The JAAS Authorization Tutorial explains how users are authorized in JAAS. This tutorial extends the authentication tutorial from Section 3. By the end of the tutorial, you will have an application in which testUser can access `java.home`, `user.home` and the file `foo.txt`. Study this tutorial and run the corresponding code.

5 Tasks and questions

In this assignment we consider a print server in a small company. This print server can perform the operations shown in Table 1. All employees in the company can use the print server to print a file (by the print operation), and view the status of the printer and the print queue (by the operations status and queue). The other operations can only be performed by certain employees who are authorized to do so. We consider the following employees:

- Alice is the system administrator. She manages the print server and is authorized to perform all operations.
- Bart is concierge who performs basic maintenance work on the printer and the printer server. He can start and stop the print server (by the operations start, stop and reset) and also read and adjust the configuration parameters (by the operations readConfig and setConfig).
- Cecile is power user. She can manipulate the print queue (by the operation topQueue) and can also restart the print server when needed (by the operation reset).
- Dirk and Erica are normal users who are only allowed to print files and view the status of the printer and the print queue (by the operations print, queue and status).

Operation	Description
print(String filename)	print a file
queue()	show the queue of print jobs (job numbers and file names)
topQueue(int job)	put the print job at the top of the print queue
start()	start the print server
stop()	stop the print server
reset()	stop the print server, erase the print queue, and restart the print server
status()	show the status of the printer
readConfig(String parameter)	show the value of a parameter in the printer configuration
setConfig(String parameter, String value)	set the value of a parameter in the printer configuration

Table 1 Operations of the print server

Your task is to make a rudimentary implementation of the print server that supports the operations from Table 1 and provides access control as described above. Use the following guidelines:

– Take the code from the JAAS Authorization Tutorial in Section 4 as a starting point.
– The print server does not have to be fully functional. Create the class `PrintServer` containing a separate method for each print server operation. When calling such a method, it is sufficient to show only the name of the method and the principal concerned, and a message whether or not the operation can be performed.
– You can add the users hard-coded in the code (in `SampleLoginModule.java`) and use the password testPassword for all users.
– A user who is logged in, corresponds to one principal.
– Create a separate class for each operation of the print server, similar to the `SampleAction` class in `SampleAction.Java`. This class is an implementation of `java.security.PrivilegedAction` and contains the method run to execute code on behalf of the subject. In this method the print server operation is called. (See Appendix A for the print operation as an example.)
– Add `PrintServerPermission` as an extension of the `BasicPermission` class (see Appendix B).
– When executing the code, each user must invoke all operations. A message should appear on the screen for each operation indicating whether or not the operation is allowed to be performed.

In addition, answer the following questions:

1. The usernames and passwords are hard-coded. A better implementation would be to store these in a file.
   – What security measures should be taken to secure this file?
   – How should the sample application be adjusted to use this file and the security measures?

2. The print server application has different users in different roles, and hence RBAC (Role Based Access Control) can be applied. How is RBAC supported in JAAS?
3. The roles in the print server application are ordered in a hierarchy. How does this hierarchy look like, and how could this be implemented in JAAS? (You don’t have to implement this. A description is sufficient.)

6 Submitting your answers

You should submit the following:

a. the source code of the print server
b. a brief explanation of the adjustments you made in the sample application to implement the print server
c. the answers to the questions in Section 5.

Submit the source code (a) in a zip-file (including scripts to compile and execute the code). Submit the explanation and answers (b and c) in a single pdf-file.

Submit your answers through the course site in yOUlearn.

Appendix A: Print.java

```java
package ...;

import java.security.PrivilegedAction;
import java.security.Permission;
import printServer.*;

public class Print implements PrivilegedAction {
    private String filename;

    public Object run() {
        Permission p = new PrintServerPermission("print");
        SecurityManager s = System.getSecurityManager();
        if (s != null)
          s.checkPermission(p);
        PrintServer.print(filename);
        return null;
    }

    public void setFilename(String filename) {
        this.filename = filename;
    }
}
```

Appendix B: PrintServerPermission

```
import java.io.Serializable;
import java.security.BasicPermission;
import java.security.Permission;

public class PrintServerPermission extends BasicPermission
implements Serializable {
  String id = null;

  public PrintServerPermission(String id) {
    super(id);
    this.id = id;
  }

  public boolean implies(Permission permission) {
    PrintServerPermission bp = (PrintServerPermission) permission;
    return id.equals(bp.id);
  }

  public String getActions() {
    return "";
  }

  public int hashCode() {
    return id.hashCode();
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof PrintServerPermission)) {
      return false;
    }
    PrintServerPermission bp = (PrintServerPermission) obj;
    return id.equals(bp.id);
  }
}
```