University of Maryland
CMSC414 — Computer and Network Security
Professor Jonathan Katz

Problem Set 2

Due by Oct. 18, 11:59 PM

The goals of this assignment are: (1) to learn about the Java 2 (a.k.a. JDK 1.2) stack-introspection-based security mechanism: how it works and what kinds of problems it has; and (2) to explore various methods of implementing access control, and to learn about their relative advantages and disadvantages. This homework requires a fair amount of background reading, but the programming component should not be very time intensive once you have understood the material. *Begin early!*

Please **check the HW2 FAQ** (to be linked from the course webpage) before working on this homework. This page will include helpful links, homework clarifications and possible hints, format requirements for the various submitted files, and submission instructions.

Finally, you will be using your code for this assignment again in a later assignment. This should be an incentive to do things right this time, so your job will be easier next time!

Distributed applications often have a *server* who receives requests from *clients*. Here, you will implement some rudimentary access control within such a system. In particular, you will work with an application called *GradeStore* which involves a server who maintains grades for a set of student projects. You will add access control mechanisms to this system which will control who can access the grades, and to what extent.

In *GradeStore*, the server manages a database of grades. The data is stored in a file (on disk) which contains student IDs, project names, and the corresponding grades on each project. The server is the only one who directly accesses this file. A *GradeStore* client can retrieve certain information from the *GradeStore* server.

The system as a whole consists of six types of applications: `Client`, `ClientStub`, `ClientComm`, `ServerComm`, `ServerStub`, and `Server`. The source code for `Client` and `Server` is not available to you and, therefore, may not be modified. On the client side, the `Client` application communicates with `ClientStub` while `ClientStub` in turn communicates with `ClientComm`. Similarly, on the server side `Server` communicates with `ServerStub` which in turn communicates with `ServerComm`. The `ClientComm` and `ServerComm` applications communicate with each other over a network.

The following operations are defined:

- `int getProjectCount()` Get number of projects
- `int getStudentCount()` Get number of students
- `String getProjectName(a,x)` Get name of project number x
- `String getProjectAvg(a,x)` Get average grade for project x
- `String getStudentID(a,y)` Get ID for student y

- `String getStudentAvg(a,y)` Get average grade for student y
- `String getGrade(a,x,y)` Get project x grade for student y
- `String setGrade(a,x,y,g)` Set project x grade for student y to g

Some things to note about these operations:

- The "a" parameter — where present — will store an authentication string. This parameter should equal the ID of the user on whose behalf `Client` is being executed and should be obtained from the Login dialog box already provided in `ClientStub`.
- `Client` hides (rather than displays) any fields storing `null`. This will prove useful if you represent the value of a field to which access has been denied as null.

Security Policy. Specifically, *GradeStore* users are grouped into three categories: professor, teaching assistant, and student. The different categories of users are permitted to perform different operations. The following defines what operations are allowed by each category of user; any other operation is considered prohibited.

Professor:

- A professor is allowed to view all grades and averages for all projects.
- A professor is allowed to change the value of any grade to any integer.

Teaching assistant:

- A teaching assistant is allowed to view all grades and averages for projects graded by that teaching assistant.
- A teaching assistant is allowed to reduce or increase the value of any grade for a project graded by that teaching assistant, provided the new grade is an integer between 0 and 100 inclusive.

Student:

- A student is allowed to view his/her own grades and grade average.
- A student is allowed to view the project average for any project.
- A student is allowed to **reduce** his/her own grade for any project, provided the new grade is an integer between 0 and 100.

Obviously, this is but one of many plausible security policies for this application. The rationale behind this particular policy is to protect student privacy and to ensure that grades are not corrupted. In particular, a teaching assistant who graded a project is in a position to know whether there was a grading error, so that teaching assistant is allowed to change such a grade. Also, a student who is pleasantly surprised by an unrealistically high grade for material that has not been mastered is in a position to let his/her conscience be

a guide and to reduce the grade accordingly. Finally, the professor is in a position to flag exceptional performance (good or bad) by assigning grades outside of the normal range.

You are asked to design and implement modifications to *GradeStore* that will enforce the security policy defined above.

1. (**Java stack-introspection-based security mechanism.**) Using the JDK 1.2 stack-introspection-based security mechanism, create a suitable directory structure and make modifications to `ClientStub`, `ClientComm`, `ServerComm`, and `ServerStub` to ensure that:

   - `ClientStub` can only be used by `Client`.
   - `ClientComm` can only be used by `ClientStub`.
   - `ServerStub` can only be used by `ServerComm`.
   - `Server` can only be used by `ServerStub`.

   Thus, you should create a Java 2 security policy file granting permissions to components based on their codebase, ensuring that each component is in the right codebase. `Client` already performs all `ClientStub` operations in a `doPrivileged` block, and for every operation `Server` performs the check:

   ```
   checkPermission(new RuntimePermission("gradesheet.server"))
   ```

2. (**Access Control.**) Add a suitable access control mechanism to *GradeStore* to implement the security policy discussed above. Implement either access control lists (ACLs) or capabilities. In an accompanying document (ASCII text or PDF file) explain why you chose ACLs or capabilities. Be sure to address the following questions:

   (a) How easily does your approach handle the addition of new students or projects?

   (b) How efficient is your mechanism with regards to system performance?

   (c) How easy would it be to add either one of the following to the policy being enforced (assume they are not both added together)?

      i. Student "A" may raise a project grade for student "B" as long as student "B" has a lower grade than student "A".

      ii. A student may transfer points to another student; i.e., student "A" may raise a project grade for student "B" by n points by lowering his own grade by n points, provided both new grades are between 0 and 100.

   Also discuss why it would be awkward to implement the above security policy using the Java 2 stack-introspection-based security mechanism. To be compelling, your discussion should explain how the Java 2 stack-introspection-based security mechanism could be used to enforce the policy.

   In your document, you should also describe any security issues you find with the stack-introspection-based security mechanism you implemented in the previous part.

Files to download and logistics. A tar file containing a folder with both the Java 2 source you need to modify and Java Archives (JAR) with compiled class files for *GradeStore* can be downloaded from the class web page. The following files are included:

> `server.jar` and `client.jar` are Java Archives (JAR) containing the compiled Java 2 class files necessary for the server and client, respectively.
>
> `server/ServerComm.java` and `server/ServerStub.java` contain the Java 2 source files for the `ServerComm` and `ServerStub` components of *GradeStore*, respectively.
>
> `client/ClientComm.java` and `client/ClientStub.java` contain the Java 2 source files for the `ClientComm` and `ClientStub` components of *GradeStore*, respectively.
>
> `example.data` is a sample input file for the *GradeStore* server.

Before you attempt to execute *GradeStore* you must compile the Java 2 source files included, which can be done by executing the following command while in the directory where you unzip the files:

```
javac -classpath .:./client.jar -g client/*.java
javac -classpath .:./server.jar -g server/*.java
```

Before you can successfully execute *GradeStore*, however, you must create a policy file (partially completing Part I of this assignment). Assuming you name that file `mypolicy`, you can then execute the *GradeStore* server and client with the following commands:

```
java -cp .:./server.jar -Djava.security.manager -Djava.security.policy=mypolicy
server.Main example.data [portnumber]
```

```
java -cp .:./client.jar -Djava.security.manager -Djava.security.policy=mypolicy
client.Main [portnumber]
```

You need to specify port number at the end. You should pick a "random" number after 1024. If you receive the error "java.net.BindException: Address already in use. . . ", someone else is using that port number and you must try again with a different port number.