CS 3326: Networks Security

Spring 2016

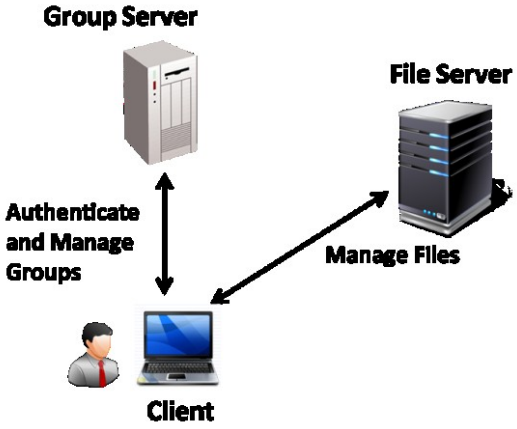
Course Project Description, Phase II

Assigned: March 11, 2016

Secure File Sharing Application

1. Background

Over the course of this semester, you will experiment with and apply the security concepts that are covered in the classroom by developing a group based file sharing application that is secure against a number of different types of security threats. At a high level, the system will consist of three main components: a single group server, a file server, and some number of clients.



The *group server* manages the users in the system and keeps track of the groups to which each user belongs. The group server will provide each legitimate user with an authentication token that answers the question: "*Who you are, and what are you permitted to do*?". Users within the system make use of a networked client application to log in to the system and manage their groups (via the group server), as well as view files stored in the system (via the file servers).

2. Part 1

The above description of the system is deliberately underspecified so that you have the intellectual freedom to consider many different possibilities for how such a system should work. Work with your fellow group members to brainstorm a list of *security requirements* that you feel should be respected by a

group-based file sharing application like that described above. You may assume that the system will support the following types of operations:

- Create/delete user
- Create/delete group
- Add/remove user *u* to/from group *g*
- Upload file *f* to be shared with members of group *g*
- Download file *f*
- Delete file *f*

Given this high-level description of the system's functionality, your group should develop a list of properties that a secure group-based file sharing application must respect. For each property, come up with (i) a name for the property, (ii) a definition of what this property entails, (iii) a short description of why this property is important, and (iv) any assumptions upon which this property depends. As an example, consider the following:

> **Property 1: *Correctness***. Correctness implies that if file *f* is shared with members of group *g*, only members of group *g* should be able to access *f*. The notion of "access" entails the creation, modification, and deletion of *f*, as well as the ability to see that *f* even exists. Without this requirement, any user could access any file, which is contrary to the notion of group-based file sharing.

The goal of this exercise is to get your group thinking about some the challenges involved with building secure distributed systems. We neither assume that you are file sharing experts, nor that you have prior experience developing secure applications. To begin with, spend some time thinking about what would make *you* trust the security of such a system, and use this intuition to formulate your requirements.

3. Part 2

In this phase of the project, you will implement a bare-bones system that provides the necessary core functionality without any security features whatsoever. The implementation of security features will be deferred to the later phase of the project. The reason for this is having a fully-functional system in hand will simplify the later phase of the project, as you will only be debugging security features, not security features and core functionality.

Here is a brief description for each component of this system:

- **The Group Server**

  The primary purpose of the group server is to manage user-to-group bindings. In particular, the group server is a central point at which (i) an administrator can create users; (ii) any user can create groups, and add other users to the groups that they create; and (iii) a user can obtain a *token* that can be used to prove that they are members of certain groups. Once a user obtains a token from the group server, they can log into the file server to view files.

  Here is a brief description of the functionality that your group server and client need to implement:

  - **boolean connect(String server, int port)**

    Your implementation of this method is responsible for establishing a connection to the group server. No other method provided by your group client application should be able to work until this connection is established!

  - **void disconnect()**

    This method is responsible for cleanly tearing down the connection to the group server. It should be invoked when exiting the client application, or any other time that the user wishes to disconnect from a particular group server.

  - **UserToken getToken(String username)**

    This method is used to obtain the token describing the group memberships of a particular user. For simplicity, the client does not need to “log in" to obtain this token: simply providing the user name is good enough for this phase of the project.

  - **boolean createUser(String username, UserToken token)**

    This method creates a new user at the group server. The token parameter is used to identify the user who is requesting the creation of the new account. This should only succeed if the requesting user is a member of the administrative group “\ADMIN". That is, not all users should be allowed to create other user accounts.

- **boolean createGroup(String groupname, UserToken token)**

  This method allows the owner of token to create a new group. The owner of token should be flagged as the owner of the new group. Any user can create a group.

- **boolean addUserToGroup(String user, String group, UserToken token)**

  This method enables the owner of token to add the user **user** to the group **group**. This operation requires that the owner of token is also the owner of group.

- **boolean deleteUserFromGroup(String user, String group, UserToken token)**

  This method enables the owner of token to remove the user **user** from the group **group**. This operation requires that the owner of token is also the owner of group.

- **List<String> listMembers(String group, UserToken token)**

  Provided that the owner of token is also the owner of group, this method will return a list of all users that are currently members of group.

The **UserToken** interface/class plays an important role in most of the above methods, as it represents the binding between a user and the groups that he or she belongs to. Given the centrality of this interface, it would be prudent for your project group to design your implementation of this interface before tackling any other coding tasks. The interface code is attached to the project description in the file UserToken.java

- **The file Server**

After obtaining a token from the group server, a user can make use of the file server to manage his or her files. To implement the required functionality, you will need to develop a client application that implements the functionalities described below.

- **boolean connect(String server, int port)**

  Your implementation of this method is responsible for establishing a connection to the file server. No other methods provided by your file client application should be able to work until this connection is established!

- **void disconnect()**

  This method is responsible for cleanly tearing down the connection to the currently connected file server. It should be invoked when exiting the client application, or any other time that the user wishes to disconnect from the file server.

- **List<String> listFiles(UserToken token)**

  This method generates a list of all files that can be accessed by members of the groups specified in token. The user should not be able to see any files that are shared with groups to which he or she does not belong.

- **boolean upload(String sourceFile, String destFile, String group, UserToken token)**

  This method allows the owner of token to upload a file to be shared with members of group, provided that he or she is also a member of group. The sourceFile parameter points to a file on the local file system, while destFile is simply a name used to identify the file on the server.

- **boolean download(String sourceFile, String destFile, UserToken token)**

  This method allows to owner of token to download the specified file, provided that he or she is a member of the group with which this file is shared.

- **Client functionalities**

  You need to develop a client application that allows a human user to access this functionality. It is totally up to you to design the way the client application interacts with the user. One group can just implement a simple numeric command list to execute the different functions. Another group can implement a simple GUI. In all cases, keep it as simple as possible, there is no need to spend too much time on this part of the project, the core functionalities are more important.

3. Implementation guidelines

Provided with this document is a zip file named "server_sample.zip". This is a sample code for a chat program that allows machines to connect and send string messages between them. The code is well commented for first time sockets programmers. The code provided is in Java. To run this code (and your project) you do not need to physically connect two or more machines, you can just run two or more

separate applications and have them communicate using sockets (mimicking an actual over the cable connection). The only difference in the code will be the IP address of the machines, if you are running all your applications (client and server) on the same machine then you can use “localhost” as your IP address. The other zip file is a “file_server_code.zip” that is a sample code for a file server that downloads a socket connection to a client.

4. Project Deliverables

Both Parts 1 and 2 of this phase are due: Friday, April 1st, midnight

For this phase each group should submit the complete source code of the project along with a README file with any instructions for compiling and running your code. Make sure you comment your source code well to make it easier to look at your functionalities and evaluate them.
Besides the source code, you should submit a document with the writeup of the security requirements you specified in Part 1 of this phase.

Bundle all of these files in a single file and upload it to the turnin of phase 2 of the project. Note that: Late submission will not be accepted.