Computer project

Secure software development

Absil R.

October 25, 2018

This document details the features required for the computer project to implement for the secure software development course. You will find here details about the requirements of your applications, along with the constraints to respect and the submission procedure. The deadline is set to 20/01/2019, 23:59.

Contents

1 Introduction **1**

2 System characteristics **2**

3 Features **3**

4 Deadline and submission **5**

1 Introduction

The goal of this project is to implement a *secure* client / server file system storage, such as the ones availlable in NAS systems. A lot of freedom is left to your discretion regarding the

security policy and actual data storage. Considering the main aspect of this project is security, apprioriate techniques have to be used, wether they have been covered in class or not.

Hence, altough you are free to choose protocols and languages that you find appropriate, you are responsible of these choices. That is, should you favour some technique over another, and if it figures the choice you made is not relevent regarding security, you will be penalised.

2 System characteristics

The architecture to use for your system is "client / server". Consequently, there are only two types of actors : a server, and a set of clients.

From a general point of view, the server allows

- new clients to register to the system;
- clients to log in the system;
- authentified clients to store files where there are allowed to;
- authentified clients to download files if they are allowed to;
- authentified clients to delete files if they are allowed to.

Information described in the next part of the document precising theses features is deliberately high-level: it is your job to detect the key points to secure in your project, and how to do it. For that purpose, you are allowed to deviate from what is written here in order to strengthen security.

From a network perspective, your project is only required to work behind a NAT: you can consider that the clients and server are on the same (virtual) local network. Note that you cannot make security hypothesis regarding this topology[1].

The server

The server is, at first, *not* a trusted entity : you do not know its set of public keys, consequently, you will have to provide a mechanism to "securely" transfer it and checking its ownership.

[1]For instance, you cannot state "I'm in a local network, so I'm in a trusted network."

Additionaly, it is your job to decide what security to implement regarding the transmission of datas to the server, as well as regarding the storage of datas on the server.

Clients and users

The client allows users registered in the server to log in order to use the features described above. Consequently, a user has the following attributes :

- some authentification material (passwords, cryptographic keys, and/or whatever you find fitting);
- a set of informations mandatory for its secure communication with the server, such as keys. You are free to handle the generation of these informations when a new client register in any way you find satisfactory.

Files

By file, we mean a file in the unix file system. Hence, a file can, in particular, be a directory. Each file has a unique owner, and associated rights regarding other users. Only users with the proper credential can then download / upload / delete / write files.

For instance, if the considered file is a directory, you can restrict listing its content by the user only. You can prevent a directory to be deleted only if it is empty[2].

Note that, both the content of a file along with its name is considered sensitive information. It has the to be protected accordingly.

3 Features

In each of the following protocols, data exchange must be secured at best (according to the relevence of the provided protection). The same remark can be applied to the storage of data resulting from an exchange. Obviously, if files are stored ciphered, when a user downloads them, the system has to decipher the considered files.

[2]These are examples. You are free to handle rights the way you want.

The type and level of security to use is left to your discretion. Consequently, it is recommended to implement more measures than those dedicated to integrity, confidentiality and authenticity, such as denial of service, dictionnary attacks, injections, etc.

Again, note that you are allowed to deviate from the protocols describes here, as long as these changes are motivated by security. However, you are responsible for these choices: should you change a protocol by another less secured, you will be penalised.

User registration, authentification and revocation

When a new user wants to register to the server, after the authenticity of the server has been verified, credentials are to be generated for the user. The form of these credentials (passwords, keys, etc.) is left to your discretion.

A user that has made a registration request is not able to log in until a system administrator has manually accepted the request.

After this step, the user can log in the system, by giving its credentials. It is expected that the system provides a 2-factor authentification protocol to log in, for instance with the help of a software token.

An administrator can also deactivate / delete a user.

Uploading a file to the server

A user can, in a directory he owns or have write rights

- create an empty directory;
- submit a file to the server, that will be uploaded from its computer. If the file is a directory, the upload has to be made recursively in order to submit the integrality of the content of the directory to the server.

Deleting a file from the server

A user can, in a directory he owns or have write rights, delete a file. If the file is a directory, the deletion has to be made recursively in order to remove the integrality of the content of the

directory to the server.

Downloading a file from the server

A user can, in a directory he owns or have read rights, download a file. If the file is a directory, the download has to be made recursively in order to remove the integrality of the content of the directory to the server.

Group files

A set of users can decide, together, to upload a file to the server. By using the principles of threshold cryptography, the file cannot be recovered / deleted unless a sufficient amount of users cooperate to perform this operation.

4 Deadline and submission

It is mandatory that your project meets the following requirements.

1. The project must be submitted by email, in one of the following ways :
   - a compressed archive under an open-source and portable format, such as `.zip`, `.7z`, `.tar.gz`, etc.[3] You probably want to `clean` your project before compression, if you do not want the mail server to reject your attachment.
   - a github or gitlab link to your repository. Note that, in this case, it is mandatory that I have write rights to your repository, so I can revert your project to the last version before the deadline.
2. The deadline is set to 20/01/2019, 23:59. Late project will not be graded. Server clocks (mail / gitlab / github) decide what is late.
3. You must submit a report (approx. 10 pages long) explaining
   (a) how to build / use your project,
   (b) the security concerns you addressed, and how you did it.
4. The project dependencies must be in updated debian packages (your report explains what to install and how to configure it should you use third party librairies).

[3] This excludes `.rar` archives.

5. Any project that does not compile will not be graded.

You are free to use any language or library you want, providing

- they respect the above requirements,
- they are open-source,
- they are certified regarding security.

It is not expected that you implement a graphical user interface (but you can do it if you want). It is expected that you implement your project according to good pratices regarding modelisation, implementation and software evolution : it must be documented, easy to maintain, with low coupling, etc.