Secure software development and web security
Computer project

R. Absil

January 2025

This document details the features required for the computer project to implement for the security course. You will find here details about the requirements of your applications, along with the constraints to respect and the submission procedure. Deadlines are given in Section 4.

Contents

1 Introduction

The goal of this project is to implement a *secure* client / server streaming dashcam system.

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, appropriate techniques have to be used, whether they have been covered in class or not.

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

🅭🅯🄏🄎

Moreover, between to equivalently secure solutions, you have to favour the most efficient one, even if it complexifies the system.

Finally, the features labelled “Master note” and ended by the “◀” symbol only apply to students in a Master’s cursus.

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 driven by users.

From a general point of view, the server allows:

- new users to register to the system;
- users to log in the system;
- authenticated users to perform various tasks, when the context is appropriate.

Information described in the next part of the document specifying these 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.

The server

The server is a *not* trusted entity. In particular, you do not know its set of public keys. Consequently, you have to provide a mechanism to “securely” transfer it and checking its ownership.

Furthermore, the server should not be trusted with *any* kind of sensitive information, such as confidential data and critical cryptographic material. Hence, you are expected to decide what security to implement regarding the transmission of data to the server, as well as regarding the storage of data on the server.

Clients and users

The client allows users registered in the server to authenticate themselves in order to use the features described above. As with any authenticated system, a user has the following attributes:

- some authentication material (passwords, cryptographic keys, and/or whatever you find fitting);

- a set of information mandatory for its secure communication with the server, such as keys. You are free to handle the generation of that information when a new client registers in any way you find satisfactory.

Furthermore, there are two types of users:

- regular users (drivers),
- trusted users (members of insurance companies, law enforcement executives, etc.).

No regular user is a trusted user, and vice-versa.

Trusted users are not meant to upload video streams, instead, they are users the streams can be shared with. Moreover, trusted users have the following attributes

- first and last name,
- organisation,
- country.

Stream

By stream, we really do mean a *stream*, and not a file. As such, you cannot wait for a video stream to be over to actually upload it.

Moreover, your system must be highly resistant to faults: should the video stream be interrupted abruptly (for instance, because the dashcam is destroyed or the connection is lost), you should minimise at best the amount of data that is actually lost.

For the sake of simplicity, use the audio-video stream of your webcam and microphone as a substitute for a proper dashcam.

In this context, a sensitive information is an information that should be only be accessible to those users who are explicitly granted access (see the details under section 3).

Furthermore, note that in no way you may consider that the server is uncompromised regarding storage. That is, if the server is compromised[1], the confidentiality of any sensitive information stored to the server must never be put in jeopardy.

Master note: Note that, for each “pack of data” or request sent to the server, an associated set of metadata is also sent to the server. This metadata contain at least

- the time the actual data or request was sent,

[1] For instance, if an administrator maliciously updates the server so that he recovers every password sent to it, or steals any cryptographic key stored on the server.

- the size of the data or request,
- the privileges needed for the request (if relevant),
- the tree depth of the required resource (if relevant),

and any other metadata you see fit meant to be used by the server for anomaly detection purposes[2]. Note that this metadata cannot in any way depend on the content of actual data or request, which is highly sensitive.

◀

3 Features

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

The type and level of security to use are 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, dictionary attacks and injections.

Master note: It is expected that a robust system of logs records user's activity, and that there is sufficient monitoring against attacks, for instance with a form of input sanitisation and automated log analysis. ◀

Again, note that you are allowed to deviate from the protocols described 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. Additionally, out of two equivalently secure solutions to a problem, you have to favour the most efficient one[3].

Finally, remember that *only* security is graded here: you won't be penalised if you decide not to follow the guidelines and good practices of web-development (as long as it has no negative impact on security and efficiency).

User registration, authentication and revocation

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

After this step, the user can log in the system, by giving its credentials.

[2] A machine learning model is not expected here, since you won't have enough data to train it.

[3] Obviously, simplicity is to be favoured over complexity, but if a simpler solution hinders the performances of a system, you have to coplexify your application to make it more efficient.

Master note: Any user can, at any point, change his credentials and any piece of information he uses to securely communicate with the server, or store information. Furthermore, it must be possible for users to log in from different devices. ◀

Master note: Considering the additional attributes of trusted users, it is expected that some form of integrity, accountability and non repudiation is provided regarding these attributes. As such, we suggest that,

- at registration, the to-be-trusted user crafts a CSR to be signed by a CA (which could be the server),
- the CA only signs CSRs of users that can actually be trusted[4],
- at authentication, the uses the CA-signed certificate to authenticate.

The CSR as well as the CA-signed certificate are assumed to contain at least the attributes of the user.

CA-signed certificates are assumed to be handled in a decentralised way, meaning you will have to setup a PKI and a chain of trust for the server to actually check the validity of certificates that it didn't sign. ◀

Uploading streams

As soon as a user authenticates, the client streams and uploads to the server.

The format of the persistent storage is left to your discretion. You can, for example, split the stream into files to store it persistently.

Remember that your system must be highly resistant to faults: should the video stream be interrupted abruptly (for instance, because the dashcam is destroyed or the connection is lost), you should minimise at best the amount of data that is actually lost.

Downloading streams

Any user can download his uploaded streams. Trusted users can download streams that have been shared with them.

Deleting streams

Users can only delete their own streams. In particular, trusted users cannot delete streams shared with them).

[4] For development purposes, a simple interface on server-side can be used to approve CSRs.

Sharing streams

Users can share their own streams to trusted users only.

4 Submission

Projects can be implemented in groups of arbitrary size, and submitted with the help of a gitlab[5] repository[6]. For that purpose, send us an email on ?? at the latest with the ssh URL[7] to your repository[8], and the name and matricule of your group members.

You have to submit your work on ?? at the latest. The minimal requirements for submitted projects are as follows:

- projects have to be submitted on time,
- projects have to provide a `README` file
  - mentioning the name and matricule of your group members,
  - explaining how to build[9] your project on a x64 ubuntu 22.04 distribution or a x64 Windows 10 machine (we recommend here to either provide a Makefile, or a shell script to install missing dependencies, compile the project and run relevant scripts),
  - explaining how to use your project (for example, “to launch the project, type the following command in a shell”).

Projects failing to meet these requirements will not be graded (that is, they will get 0/20). In particular, projects that do not compile according to your *exact* instructions will not be graded. Furthermore, note that we shall in *no way* build or run your projects in an IDE.

As usual, you must provide a modular code, easy to maintain, etc. Moreover,

- any submitted code must be duly documented and provide needed configuration files in order to produce the developer documentation.
- *only* security features are graded for this project.

Master note: Furthermore, any submission must include a report under PDF format detailing your choices regarding security. You are strongly advised to follow the guidelines presented by H. Mélot [1] for your redaction, and to answer all questions listed in the check-list attached to this document. ◀

[5] The only two allowed instances are `gitlab.com` and `git.esi-bru.be`.

[6] Create the repository yourself, add your teachers (`rabsil`) as maintainers.

[7] A gitlab ssh URL looks like `git@instance.com:username/projectname.git` .

[8] The automatic email notification is *not* enough.

[9] It is expected that your script installs missing dependencies in addition to compiling your code.

References

[1] H. Mélot. Éléments de rédaction scientifique en informatique. http://informatique.umons.ac.be/algo/redacSci.pdf - Consulté le November 15, 2024., 2011.