LEI: Segurança Informática e nas Organizações 2020-2021

Projects 2+3: Digital Rights Management

Deadline: December 30, 23:59

1 Description

This work aims to explore concepts associated with a secure media player, which enables clients to consume media content from a catalog, while enforcing strong restrictions through the use of cryptographic primitives. The overall project is split into two parts, which should be executed in sequence. The first part will consider the establishment of a secure session between the player and the server, while the second part will deal with authentication, access control and confinement.

The assignment should be implemented by a group of two students. The use of external resources besides python libraries providing base functionality (e.g. `Cryptography.io`), or public domain media files is strictly forbidden. Not complying with this requirement will imply that the work will not be considered for grading.

2 Setup

Students should consider the code provided as a base for the implementation of the required components. The code contains some of the basic mechanisms, without any security assumptions. Methods can be changed/added/removed as required for the correct execution of the features described later in this guide. The base project is provided with a example client and server. The `Twisted` python framework was used to implement the server, while the player makes use of the `requests` package. In order to actually play media, the client relies on the `ffplay` tool from the `ffmpeg` project, and its capability of playing music provided from the `stdin`. The client will fetch data from the server in chunks and will write each chunk to `ffplay`, avoiding any persistence in the hard disk.

If you develop you project in Linux, the tool should be available through the package manager. If you are using Microsoft Windows, you can download `ffplay` at `https://ffmpeg.org/download.html`.

It is recommended the creation of a distinct folder for the client and server, with a virtual environment to store python packages. This can be achieved by downloading the compressed file provided, and executing the following commands:

```
$ apt install virtualenv
$ virtualenv -p python3 venv
$ bash
$ source ./venv/bin/activate
$ pip3 install -r client/requirements.txt
$ pip3 install -r server/requirements.txt
```

3 Work description

The project consists in the design (**Diagrams and text**), and implementation (**python code**) of a system allowing secure communication between the client and the server, for the purpose of securely distributing media files. The server provides a list of media titles, which can be listed, and rented for a specific period (seconds, minutes or days). The rental results in the creation of a cryptographically secure license token, allowing the user to access a media file during some time, or for a limited amount of visualizations. The architecture considers a content distributor, which ensures the authenticity of the content, a python client, a python server, and a user.

The client will be able to list and rent titles. While the client possesses a valid rental license, it may obtain the media files to be reproduced in the embedded player. It is assumed that users need to acquire viewing licenses before the media content can be accessed. Licenses should be cryptographic objects, created by the server, and used for providing access to files. Clients will not store any data in a permanent storage, and the contents will be provided directly to a media player.

Moreover, it is assumed that files are transmitted in chunks, with individual keys used for each chunk. A key derivation mechanism is to be selected by the students, ranging from a hash chain based scheme, counters, or asymmetric cryptography (to name a few). The result is that intercepting data will not allow an attacker to reconstruct the file as everything is encrypted with temporary keys. Even a Man-in-the-middle attack should not be possible, either to access media files, or to inject alternative chunks in an attempt to disturb the viewing experience.

The server will provide an API through which the client interacts in order

to setup the cryptographic processes, or obtain media. HTTP is suggested as the underlying transport protocol. Encryption mechanisms should be supported on top of HTTP and HTTPS should not be used [1].

The user is an individual which uses the client to view a media file. He has a real identity, supported by a hardware token such as a Digital Identity Card.

The system should be capable of the following features:

Confidentiality and Integrity (proj 2)

1. Negotiate a cipher suite between the client and server (at least 2 ciphers, 2 digests, 2 cipher modes)
2. Negotiate ephemeral keys between the client and server (valid only for a single session)
3. Encrypt all communications
4. Validate the integrity of all messages and chunks
5. Manage cryptographic viewing licenses, based on time or number of views
6. Provide the means for chunk based key rotation

Authentication and Isolation (proj 3)

7. Mutually authenticate the client and server supported by a custom PKI
8. Authenticate the user viewing the media content
9. Authenticate the content viewed so that the client can verify the content authenticity
10. Integrate hardware tokens to authenticate users
11. Protect the media content at rest in the server

User registration, media upload, and the PKI required for client/server of the content distributor can be done off-line, and should not result in functionality added to the client or server.

The following attacks should not be trivially possible:

1. An attacker being able to access any content exchanged in clear text.

[1] This is not a good practice as we should rely on tested and widely adopted solutions. This approach will be followed in order to explore the application of base cryptographic methods.

2. An attacker being able to inject, omit, replay or modify any message, or media chunk without detection.
3. An attacker being able to access the media content after gaining access to a storage volume (e.g., a hard disk when for repair).
4. An attacker being able to extract media files from the storage volume of the system where the client is executed.
5. An attacker being able to pretend to be the server.
6. An attacker being able to downgrade the encryption methods.
7. An attacker being able to decrypt a session captured at instant T0, after accessing cryptographic material used for another session at later instant T1.
8. An attacker being able to know which content is being viewed.

The assignment should be returned with all code, documentation and keys required for full operation. A report and should describe the protocols in detail (entities, messages, keys, processes, flows), the mechanisms, and demonstrate the correct operation of the features implemented. The report is mandatory and will have a substantial contribution to the grading.