Medical Test Records

Segurança Informática em Redes e Sistemas

Taguspark G02

Autores Rafaela Jorge Timóteo, nº 90773
Ana Marta Sousa Contente, nº 98643
Alexandru Pena, nº 98742

november 9, 2020

1 Problem

The Medical test Records project approaches two main issues that hospital information systems face.

The first issue, is controlling access to a patient profile. A patient profile has associated with it a group of information with different levels of sensitivity, such as age, name, blood type, diseases, family records and others. A patient's name is not extremely sensitive so most employees can access it. However, for instance,the patient's family records do not need to be accessed nor should be accessible to most of the hospital staff, for example to volunteers.

Given this scenario, the access control mechanism must be able to correctly determine, and enforce the correct policies so that different parts of a patient's profile are only available to whom actually requires them to fulfill one's obligations to the patient, regarding the performed role and in the correct context.

The second problem comes from medical tests being analyzed in different facilities, like partner labs, that are not part of the hospital infrastructure, but require that both are interconnected in a way that enables all sides to identify each other and communicate in a secure way.

1.1 Requisites

Given the previous problems, the implementation of this project requires the following security guarantees:

- A fine grained access control and secure authentication mechanism.
- Authenticity of all communications with the information systems.
- Confidentiality of all communications with the information systems.
- Integrity of all communications with the information systems.
- Non-repudiation of the communications with the partner labs.

1.2 Trust assumptions

There are three main entities in this project:

- The Hospital System

  The system that will store all the patient data and tests results received from the labs.
- The Partner Labs System

  The partner lab can be internal, or external to the Hospital network. The internal lab is in the same network as the Hospital, but as the external lab, they will have different infrastructures from the hospital, which means they will have their own certificates and machines.
- The users of those systems, the staff.
- The certificate authority.

In case of the staff there is a full trust assumption that they won't have their credentials stolen, shared or compromised in any way, and as such we will not provide any mechanism to rapidly revoke those credentials.

Regarding the certificate authority, there is a full trust assumption that the private key used to issue/sign certificates won't be stolen and as such, we won't have a mechanism to revoke certificates.

Concerning the access control components which will be explained further (PDP,PEP...), there is a full trust assumption that those components will be correctly and logically well implemented, and won't allow any type of unauthorized access.

All communications with the partner labs (internal or external) will use HTTP requests with TLS, and will be considered secure and resistant against attacks like replay, tampering and eavesdropping.

There is also a full trust assumption, that all the libraries and frameworks used are error free and won't give any type of access to an attacker.

2 Proposed solution

2.1 Overview

The implementation will be separated in two main components, an hospital REST API[**?**] and a laboratory REST API, which will act as client or server, depending on the action being done. When further mentioned API in this text, both the hospital's API and lab's API's are being referred to, as they will be pretty similar in terms of security and access control. The only difference between those two entities (hospital and labs) API's, are the possible actions and data stored.

The project client and API infrastructure will be developed using the Java programming language.

All requests received by the API need to possess a proof of identity to be further processed (accept or refuse action), in this case will be a token provided by the respective entity API, after authentication using the correct set of credentials (username:password). The token is transported in the "Authorization" header of the HTTP request. If the token is not present the request will be refused with status code 401 UNAUTHENTICATED.

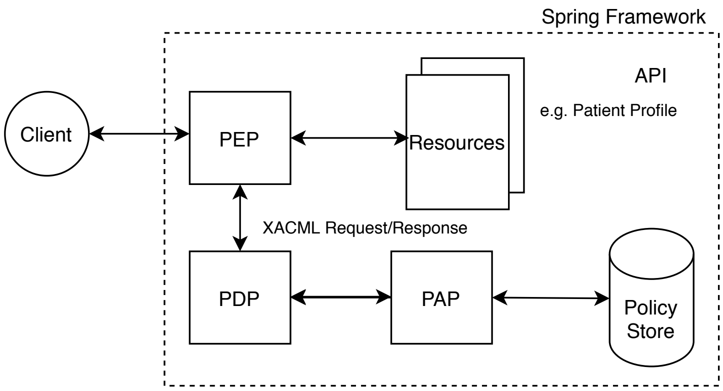
The database will store the hash version of the password using the Bcrypt[**?**] algorithm, which already salts the passwords. The API will validate the received credentials by hashing them and comparing to the value already stored in the database. Further steps can be taken to protect against brute-force attacks like increasing the processing time or limiting the number of login attempts, which will only be implemented in the advanced version of the project if there is time.

If the request contains all necessary information (access token), it will be forwarded to the access control mechanism.

The access control mechanism is composed by 4 major components:

- PEP - Policy Enforcement Point
- PDP - Policy Decision Point
- PAP - Policy Administration Point
- Policy Store

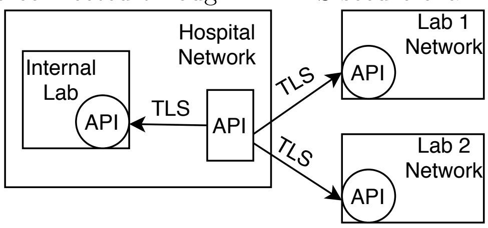
The PEP will send a XACML[**?**] request to the PDP, the PDP will then based on the policies defined by the system return a XACML response which will dictate the action (ACCEPT or REFUSE) that PEP will enforce. If the result is ACCEPT, the request will continue and read or write the requested resource, otherwise the request will be dropped and replied with a 403 UNAUTHORIZED.



All those components will belong to the API but properly separated so that a flaw in one of them doesn't put in danger the whole system.

2.2 Deployment

The following figure includes the distinct machines of this system, and as mentioned they will be interconnected through HTTPS secure channels.



For simplicity, a lab will contain one machine with an API and in the context of this project there will be only one external lab. The partner lab staff will send HTTP requests to their API which in turn will communicate with the Hospital API using the custom protocol, to send and receive any data related to the tests data.

The deployment scripts will use Vagrant[**?**] to manage and deploy the VM's with the correct network configurations and all the necessary software to run the applications.

2.3 Secure Channels

All communications between the end-users (staff) and respective API's will run with TLS to provide confidentiality, integrity and freshness. TLS, between those entities will be provided by the Spring framework[**?**] (Java)[**?**], which is the framework and language that will be used to develop the project.

The communications between the Hospital API and partner lab API will be under a custom made protocol which will provide the same properties as the TLS channel (confidentiality, integrity and freshness).

TLS will require the definition of a certificate for each entity, we will consider that the certificates are installed manually by an administrator on each respective machine. We will have a made-up CA (we will create it with OpenSSL[**?**]) , which we will use to issue and sign the hospital's certificate and each lab's certificate (including labs inside the hospital). Those certificates alongside with the secure communication channel, will be used to digitally sign the sent tests data to guarantee its authenticity at any time. In order to validate the certificate received by the server during the TLS Handshake, each machine will have the root CA certificate already installed, manually, by an administrator.

2.4 Secure Protocol to Develop

3 Plan

3.1 Basic

The basic version of the project must have the following components in place:

- API.

  The following endpoints will exist in the API:

  - GET /patient/id/name, returns the name of the patient with id id.

  - GET /patient/id/diseases, returns the diseases the patient has (e.g. tuberculosis...)

  - GET /patient/id/treatment, the necessary treatment (e.g. what medicine the patient needs, 500mg of Vicodin).

  - GET /patient/id/testsresults, the results of the tests.

  - POST /patient/id/testresults, used by the partner labs to write the tests results.

- TLS for secure communication channels.
- Access Control.

  The following roles will exist: Doctor, Nurse, Janitor and Lab Employee. The doctor role will give read access to any information about the patient, the nurse role will give read access to any information regarding the necessary treatment (e.g. food, medicine...), the janitor role will only give read access to superficial information like the patient name, finally the lab employee won't have access to any patient profile information, they can only see in their entity respective API if there is any requests from the hospital to process tests data and writing the results of the tests data.

- Custom communications protocol. (Hospital - Partner lab's).

All of the read and write operations will use objects following the JSON format.
In an advanced version of the system the id of the patient shouldn't be used to write the tests data, but another id generated to make it difficult to associate some tests data with a given patient.
It is expected that this version will take around 2 weeks to complete (worst case scenario).

3.2 Intermediate

The intermediate version will expand what was built in the first 2 weeks adding the following:

- The necessary mechanism to check tests data authenticity at any time. (Digital Signature)
- Sensitive information like passwords stored in a safe way using Bcrypt (already salts).
- Finishing the implementation of the custom communications protocol

This phase should take around 2 week's.

3.3 Advanced

By this point the necessary mechanisms are all implemented, possible bugs should be corrected now, if there are none, if there is time the project can take a step further and do the following to increment the security or optimize the system:

- Encryption of all the confidential information in the database, to avoid leaks in case of a breach.
- Policies and authentication information cache. (Will avoid constant reads to the database everytime there is a request).
- Defining rule combining algorithms in the access control for more complex decisions.
- Firewall implementation.

3.4 Effort Commitment

	Alexandru	Ana Marta	Rafaela
Week 1	API	Deployment Scripts	Database
Week 2	Access Control	Custom Protocol	Tests data authenticity mechanism
Week 3	Access Control	Custom Protocol	Tests data authenticity mechanism
Week 4	Firewall	Rule Combining Algorithms	Database encryption

Bibliography