Truthful Emergencies

Report

Network and Computer Security

Alameda

Group 11

Miguel Pinto 79060

Bernardo Casaleiro 87827

João Godinho 87830

December 9, 2016

1 Problem

First response emergency systems are a limited resource. Their usage makes difference in life-or-death situations! Requesting an ambulance is a serious matter. It should be simple, secure, fast and first of all always available. For that the Dispatch Center must be fault tolerant.

It should separate the real requests from the false ones and provide a response accordingly. Improper usage must be detected to avoid misallocation of resources while real requests should trigger an appropriate response.

2 Requirements

2.1 Application Requirements

1. Send an emergency request.
2. Inform the estimated time for the user's request to be answered.
3. ~~Confirm or deny if there is an emergency nearby.~~

2.2 Server Requirements

1. Receive a request
2. Insert a request in a queue depending on the user rating.
3. ~~Request confirmation from users nearby the emergency.~~
4. Log the requests.
5. Send the expected response time.
6. Send the confirmation that an ambulance is on route.
7. Rate the user accordingly at the end of the request.
8. Temporarily block a user for abusive usage.

2.3 Security Requirements

1. Non-repudiation.
2. ~~Fault tolerance to DoS.~~
3. ~~Firewall, responsible for filtering the requests.~~
4. Filter, responsible for filtering the requests.
5. Auditing mechanisms.
6. Secure channel to communicate.

Label

Requirement that we were not able to implement ~~Requirement~~

New requirement implemented Requirement

3 Solution

3.1 Basic solution

The main goal of this sprint is to build the foundation and the architecture to facilitate the implementation of the security parts. There are no keys or certificates involved in this step so non-repudiation will not be our goal here.

We will have a centralized Dispatch Central that will receive all the requests from the Users.

Auditing mechanisms will be considered in this phase and for that we will use logs on the Central side.

After processing the request, and approving it, the Dispatch sends the confirmation message that an ambulance is on the way. Otherwise the user will be accounted for the incorrect usage and warned of the possible consequences of further reckless behavior.

After attending the emergency the team sent provides feedback regarding the truthfulness of the emergency.

The user will be registered into a database with a ranking associated that will be used for the placement in queue.

3.2 Intermediate solution

In this phase, we will create manually all the keys and certificates[1].

Each entity will have his own set of keys, but the certificates will have to be requested to the CA to obtain the public key.

Each message sent by the user will be signed with his own private key.

The Dispatch Central will retrieve the user public key from the certificate that was requested after receiving the message and then deciphers it. The procedure is the same for the message flow from the Dispatch to the user.

In this phase we will also build a network level firewall, with some basic rules to discard duplicate packets and redirect connections to secure ports (HTTPS).

3.3 Advanced solution

On this last phase of the project, we will focus on fault tolerance on the primary server (DoS, crash, etc.). We will also focus in updating the firewall to an intrusion detection system. This IDS will receive the data from the sensors of the phone and analize its content to verify the veracity of the message.

4 Results

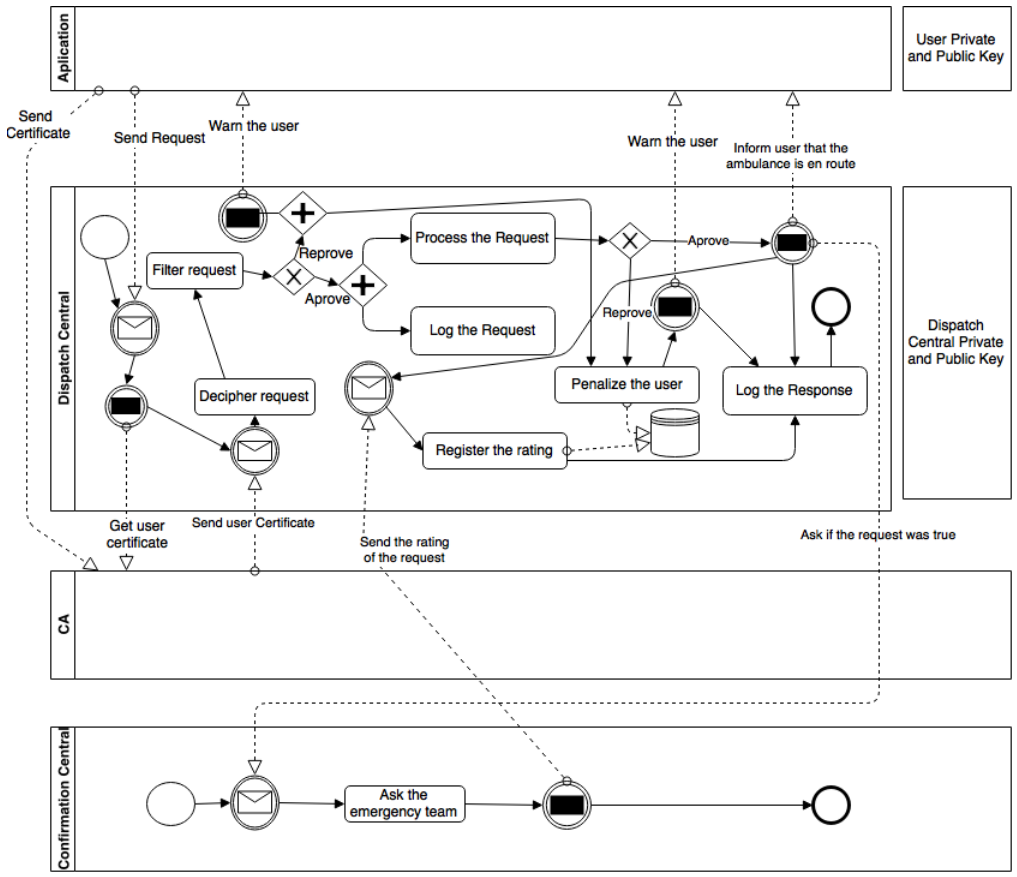
Our solution has two main components: the dispatch central and the user aplication. In order to meet the security requirements we also implemented a certificate authority and a confirmation central.

The user aplication allows the user to send emergency requests. When starting the application it first sends the user certificate to the certificate authority. Then the user is able to send requests to the dispatch central.

When receiving a request the dispatch central asks the CA for the user certificate in order to decipher the request and forwards it through a series of filters to exclude false ones, only processing the accepted ones. The dispatch central then notifies the user either that help is on the way or that the request sent was false and reminds the user of the consequences. After an emergency team is dispatched the server asks the confirmation central to rate the request depending on the authenticity of the emergency, awarding or penalizing the user.

[1] self-signed

All this comunications between the components are made using Java SSL Sockets in order to guarantee a secure channel and always cyphered using the proper certificates provided by the certificate authority.



5 Evaluation

With our implementation we reached the majority of the security requirements proposed. As well as the majority of the functional requirements.

We began by aproaching the **Auditing Mechanisms** point using the Log4j API to keep record of every request received and action taken. Moving then to the **Secure Channel** point where we upgraded from Java Sockets to Java SSL Sockets, and to the **Non-Repudiation** point using the certificates and a certificate authority to distribute them.

Our initial goal in order to filter the requests received was to build a firewall but due to the lack of time we opted for a simpler solution and implemented a filter. This filter allow us to somehow filter the invalid requests, such as blank packets, duplications or abusive behaviour, in the moment the server receives and deciphers them.

Another point that we had to adapt was the **Fault Tolerance to DoS**. Initially this was a really important point for us and we hoped to distribute our system and use a load balance to optimize resource use and avoid overload of a single resource. But with the project's development and the scarce time we decided that it was just not possible.

We also hoped to move from a terminal that simulates a user application to an actual Android application where we would have access to the user location and implement new features like *confimation of a nearby request based on the user location.* Although our expectations it had to be put aside.

6 Conclusion

With the completion of this project, we can conclude that most of the requirements established in the proposal have been completed. Time management and planning in a project are extremely important, and even though we have passed several times in these phases there were unmet requirements, which in this case were the most ambitious in the proposal (port for Android and build an IDS).

We built an application that provided a minimum of security for the users and the Dispatch Central, so we assume that we reached and completed our main goal from the beginning.

7 References

7.1 Tool References

Apache Maven Software project management and comprehension tool.

PostgreSQL Database to save the users.

javax.net.ssl.SSLServerSocket Establish a secure connection.

PostgreSQL JDBC Driver to use PostgreSQL with Java.

Log4j Logging system.