Truthful Emergencies

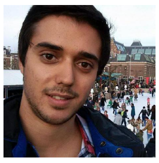
Proposal

Network and Computer Security

Alameda

Group 11

Miguel Pinto 79060


Bernardo Casaleiro 87827

João Godinho 87830

October 26, 2016

1 Problem

First response emergency systems are a limited resource. Their usage makes a 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 punished while real requests should trigger an appropriate response.

2 Requirements

2.1 Application Requirements

1. Send a request of an emergency.
2. Consult the expected time for the request to be answered.

2.2 Server Requirements

1. Insert a request in a queue depending on the user rating.
2. Log the requests.
3. Send the expected time of response to the user
4. Send the confirmation that an ambulance is en route.
5. Rate the user accordingly at the end of the request.
6. 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. Auditing mechanisms.
5. Secure channel between user and Dispatch Central.

2.4 Non-functional Requirements

Our main goal is to build a secure application, not forgetting that in real life, our service couldn't be slow due to the fact that time in an emergency situation is crucial.

3 Proposed solution

What's the cost of a human life?

There's no perfect solution and everything breaks eventually. Our goal is to make it harder to break with the time and money we have at our disposal. So with this in mind we design this solution.

3.1 Basic solution

In this solution we assume that the communications between the User and the Dispatch Central are secure. 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. Everytime a request is received it is registered in a log for the auditing personal be able to review. Each entry will have a timestamp and the contents of the message (ID, Location, Situation).

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.

The user will be registered into a database with a ranking associated that will be used for the placement in queue. Incorrect usage will lead to a bad ranking and consequently worse response time when makind a real request.

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. This will give a structure to implement non-repudiation taking the assumption that the private keys are never stolen by some sort of malware or someone accessing the machine physically.

Each message sent by the user will be signed with his own private key. With the previous assumption we can guarantee that the user sending the message was really him.

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.

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 building a firewall responsible for detecting bad or duplicate requests, redirecting connections to secure ports and analyzing packet contents.

4 Tool references

1. [**Well-Tested**] Java
2. [**Tested**] Docker
3. [**Tested**] nginx

5 Work Plan

To facilitate, we will divide the project according to the requirements.

Example: b-2.2.2 means Basic job on Server Requirement(2.2) - "Log the requests"(1).

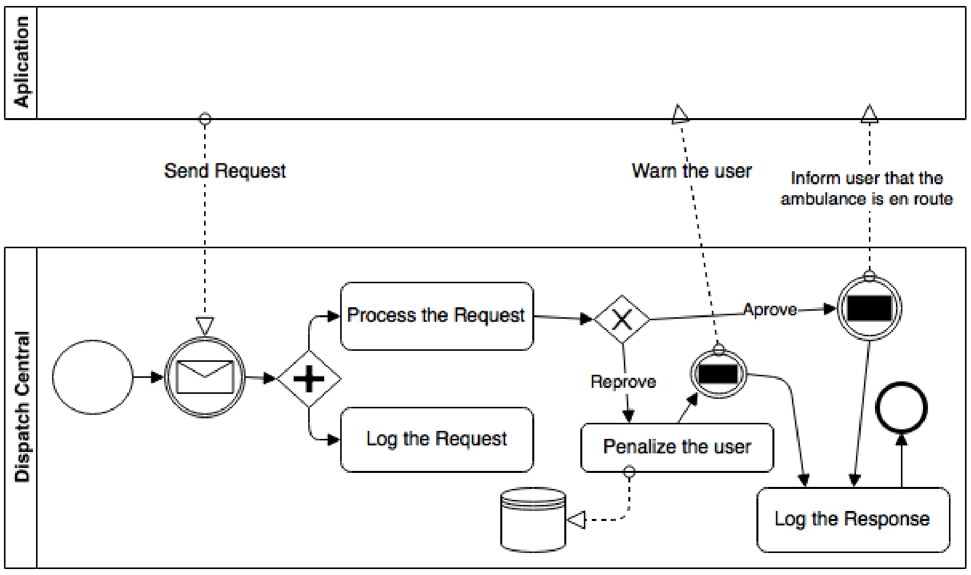
The jobs might be basic(b), intermediate(i) or advanced(a).

[1]self-signed

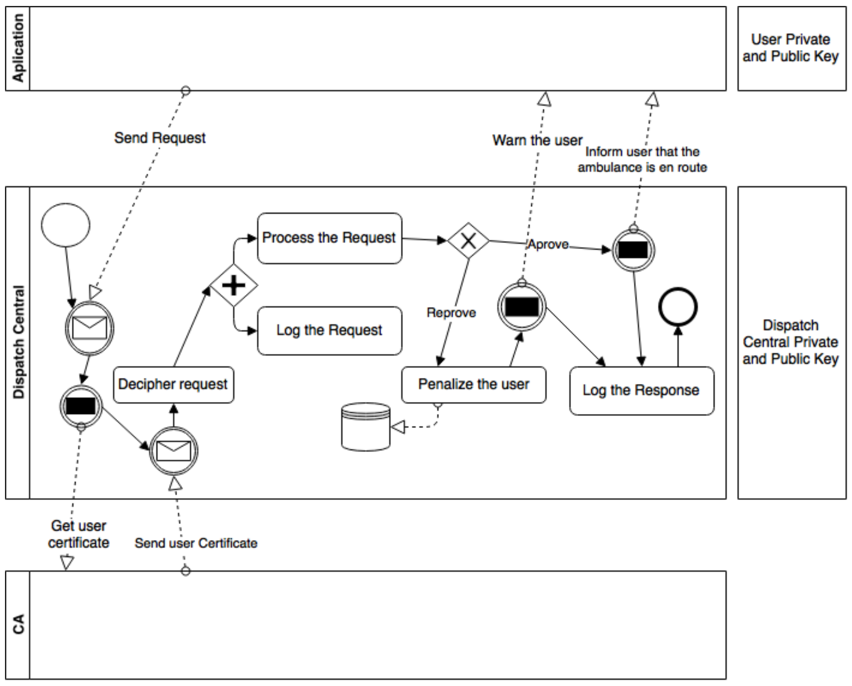
Week Start	Miguel	Bernardo	João
31 Out	b-2.1.1	b-2.2.1	b-2.2.2
7 Nov	b-2.2.4	b-2.2.5	b-2.3.4
14 Nov	i-2.1.2 i-2.1.5	i-2.2.3	i-2.3.1
21 Nov	a-2.2.6	a-2.3.2	a-2.3.3
28 Nov	a-2.2.6	a-2.3.2	a-2.3.3
5 Dez	Test and Report	Test and Report	Test and Report

6 Appendix

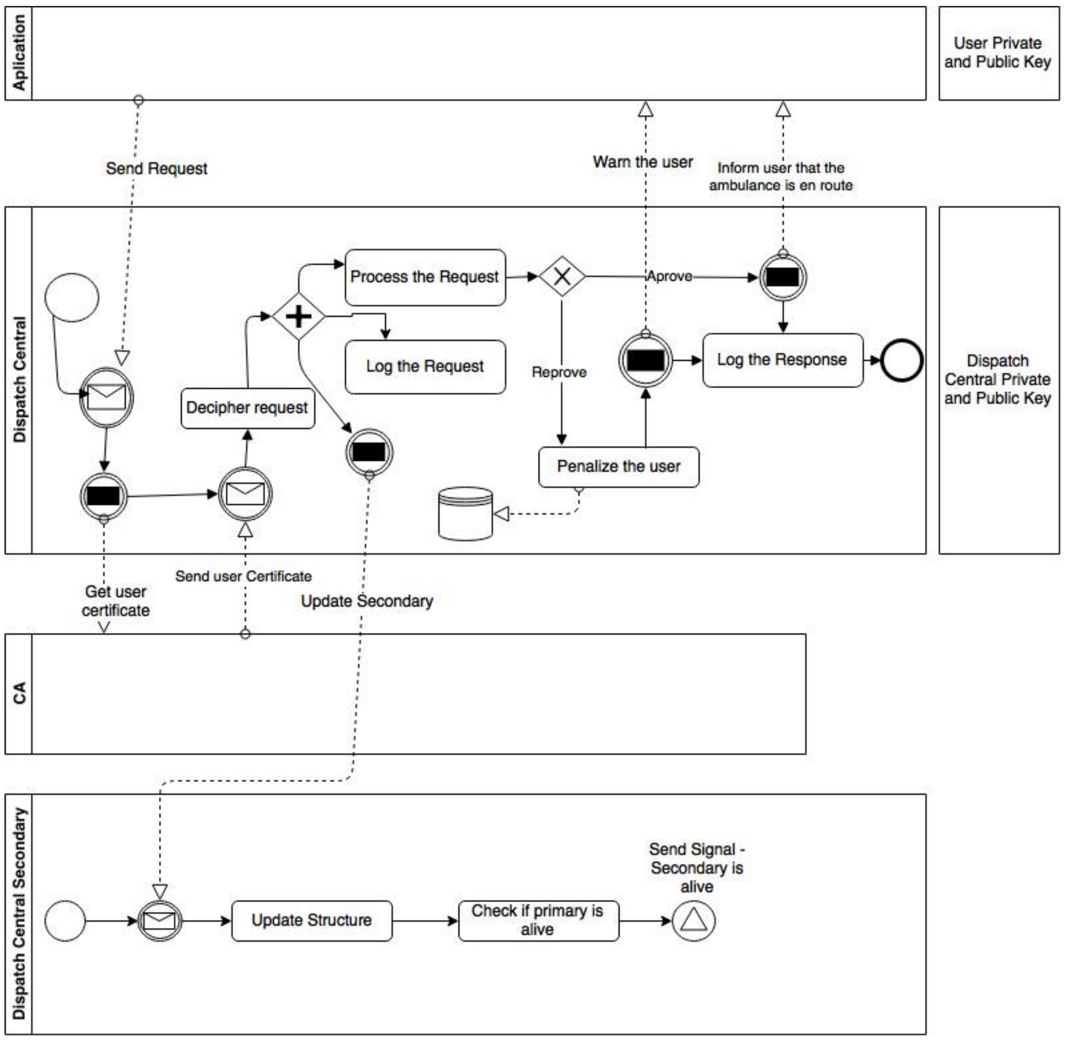
6.1 Basic solution



6.2 Intermediate solution



6.3 Advanced solution


Aplication
User Private and Public Key
Send Request
Warn the user
Inform user that the ambulance is en route
Dispatch Central
Process the Request
Aprove
Log the Request
Reprove
Log the Response
Decipher request
Penalize the user
Dispatch Central Private and Public Key
Send user Certificate
Get user certificate
Update Secondary
CA
Dispatch Central Secondary
Send Signal - Secondary is alive
Update Structure
Check if primary is alive