Project report

> First project of Cyber Physical System and IoT Security (course of Cybersecurity) - *Master degree in Computer Science*

Replication of first paper - IDS for CAN: A Practical Intrusion Detection System for CAN Bus Security

- link to the paper: https://ieeexplore.ieee.org/document/10001536
- link to the dataset: https://ocslab.hksecurity.net/Dataset/CAN-intrusion-dataset (section 1.3)

Authors:

- Davide Bassan
- Alberto Morini (mat. 2107783)

Date: 02 December 2023

ALL SOURCE CODE IS PUBLIC ON GITHUB (in various branch):
https://github.com/albertomorini/CyberphisicalSystem
For storage reasons we do not provide the datasets of the paper in this repository

Table of contents

Abstract

We have to replicate what done in the paper linked before, so, we can divide the job in two side:

1. **Intrusion detection**: which analyze the datasets and identify potential threat
2. **Mobile app**: that via notifications alert the user of the threat detected before

Architecture

We designed a simple architecture, made by backend (server with detection algorithm) and frontend (webapp).

In short, the backend analyze the datasets and after identify the threats, make them available to clients through an HTTP server.

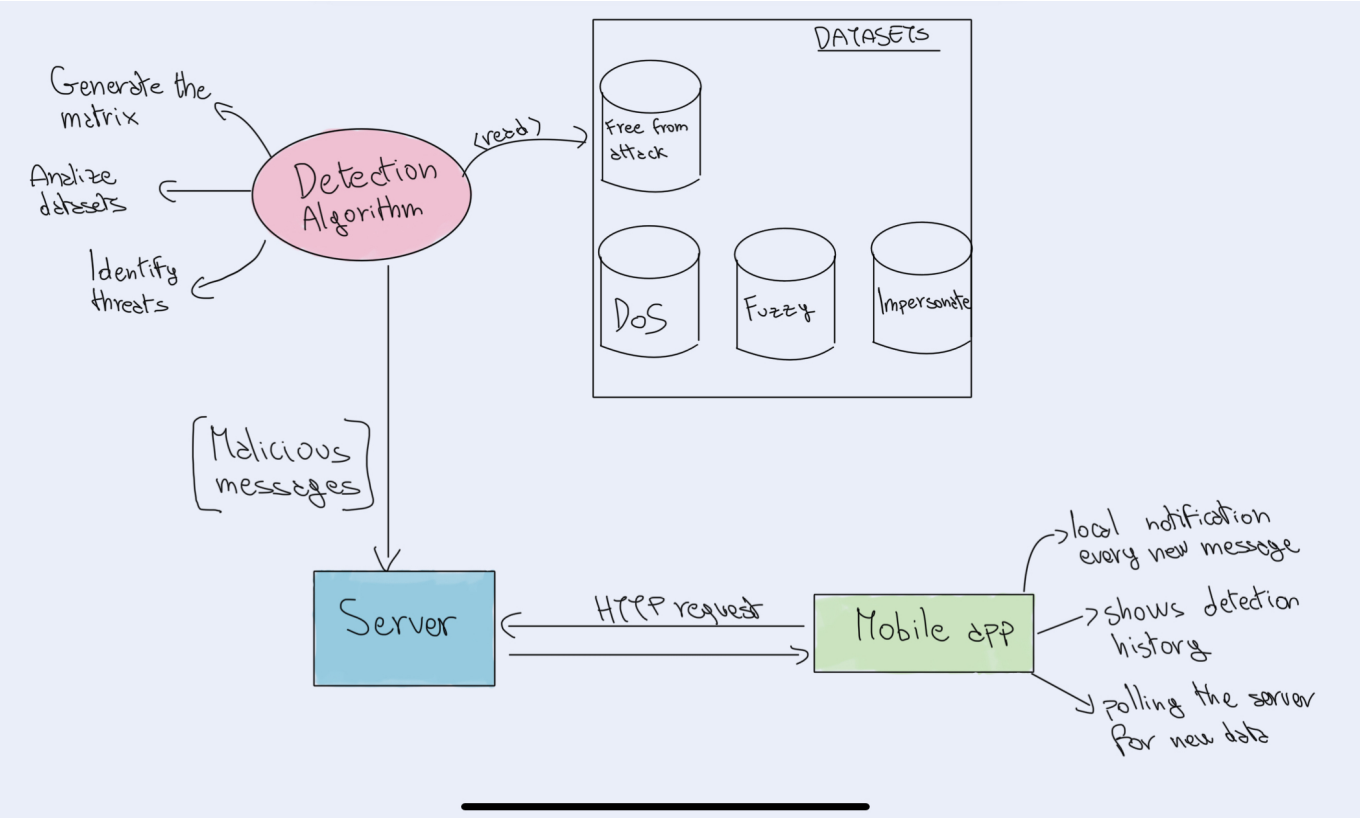

Fig1: flowchart of designed architecture

Technical specifications

The **backend** is made entirely in Python

- *The building and testing has been made with Python 3.11.4*

The **frontend** is a web application made with Ionic framework and compiled for Android

- *Building and testing has been made with Ionic 7.1.1*
- *Android 14 (5 sept 2023) and API34 emulated in Android Studio Giraffe 2022.3.1 Patch 2*

Intrusion detection side

Creation of the matrix

In this phase, we have replicated the algorithm proposed in the paper.

Specifically, from the file `Attack_free_dataset.txt`, we used a regular expression to extract all the IDs of the frames.

```
def extract_id(file_path):
    """
    Extracts values from the 'ID' column in a text file.
    :file_path (str): The path to the text file.
    @returns List[str]: A list of strings containing the values from the 'ID'
column.
    """
    with open(file_path, 'r') as file:
        file_content = file.read()
    ids = re.findall(r'ID:\s+(\w+)', file_content)
    return ids
```

Next, we initialized a matrix of size `N*N`, where `N` is the number of unique IDs that appear in the `Attack_free_dataset.txt` dataset, with all values set to `False`.

Then, we populated the matrix, setting those IDs to True that appear one after the other in the dataset. This because we assume that the traffic on the CAN bus is highly recurrent.

```
def generate_matrix(ids):
    """
    Generation of the matrix
    :ids (List[str]): a list of strings containing the values from the 'ID'
column.
    @returns List[List[Boolean]]: a matrix of all transaction appeared on the
dataset
    """
    auxiliary_ids = list(dict.fromkeys(ids))

    matrix_size = len(auxiliary_ids)
    boolean_matrix = [[False] * matrix_size for _ in range(matrix_size)]

    for i in range(len(ids)-1):
        row_index = auxiliary_ids.index(ids[i])
        column_index = auxiliary_ids.index(ids[i+1])
        boolean_matrix[row_index][column_index] = True

    return boolean_matrix, auxiliary_ids
```

Threat detection

We have evaluated the algorithm on three different subset of frames

- `DoS_attack_dataset.txt`
- `Fuzzy_attack_dataset.txt`
- `Impersonation_attack_dataset.txt`

To achieve this, we checked if subsequent IDs appeared in one of these subsets, which is also present in the `Attack_free_dataset.txt` dataset. In other words, in the matrix built previously, if the value is `True` at position `id, next_id`, then the ID is considered to have appeared after the other.

If the ID does not exist in the matrix, or if the value in the matrix is `False`, it is added to a list of potential dangerous messages.

```
    [...]
    if actual_id not in ids or next_id not in ids:
        threat_len += 1
        batch_threat.append(dataset[i])
        if actual_id not in ids:
            id_to_be_saved.append(actual_id)
        if next_id not in ids:
            id_to_be_saved.append(next_id)
    else:
        row = ids.index(actual_id)
        column = ids.index(next_id)

        if not matrix[row][column]:
            threat_len += 1
            batch_threat.append(dataset[i])

    counter += 1
    [...]
```

We have divided the evaluation in batch, in order to calculate a BATCH_RATIO of threat, we have also fixed a threshold, in order to ignore false positives.

In fact, if a batch have a BATCH_RATIO lower than a certain threshold we have considered those messages as false positives, and instead of notify the users of potentially unknown messages, we have update the matrix setting to true those false positives.

```
def update_matrix(matrix, ids, dataset, adding_ids):
    ids += adding_ids

    for _ in range(len(adding_ids)):
        matrix.append([False] * len(matrix[0]))

    for _ in range(len(adding_ids)):
        matrix[_].extend([False] * len(adding_ids))

    for i in range(0, len(dataset)-1, 2):
        row = ids.index(dataset[i][1])
        column = ids.index(dataset[i+1][1])
        matrix[row][column] = True

    return matrix
```

Mobile app

Otherwise as the paper, we opted to create a web-application which communicate to server via HTTP packets.

We took this decision for many aspects:

- **portability**: web application can be easily build for both mobile OS major(iOS and Android); representing a huge benefit in the maintainability and extendability of the code
- **real life scenario**: we imagined the backend detection system installed in the cars, maybe integrated with Android Auto or CarPlay, which systems can communicate via Wi-Fi.

Architecture

Server ''sends'' the client the message retrieved by the Detection Algorithm, adding the property of which dataset the detection is from.
The Content-type of packets is in "application/json" from.

Packet structure

```
{
    "timestamp": integer,
    "msg_length": int,
    "msg": "VARCHAR",
    "id": id_CANComponent,
    "kind": "Sospicious/Unknown",
    "dataset": "DoS/Fuzzy/Impersonate/..."
}
```

DISCLAIMER: to simulate a real life scenario, server sends the client a new detection at every request received.
In an effective scenario, server will makes available detections in real time as soon as they are identified

So, we let the client define the polling time to check server for new data. *Actually every 5 seconds*

Server HTTP

Made in Python and default libraries:

```
import json #to parse/stringify JSON
from http.server import BaseHTTPRequestHandler, HTTPServer # HTTP server
import sys # to get command line params
import DetectionAlgorithm # the algorithm explained before
```

For simplicity, server as soon as it started, launch the detection algorithm on every dataset provided; then, save the detections in a local variable.

Server provides to clients a limited number of detections, indicated at the startup in the command line by user. We made this decision in way to provide detections of every dataset in a reduced time.
If none parameter is provided, the default number of message for every dataset is 10.

For example: `$ python3 server.py 5` --> will send 15 packet in total to the client (5 detection for each dataset)

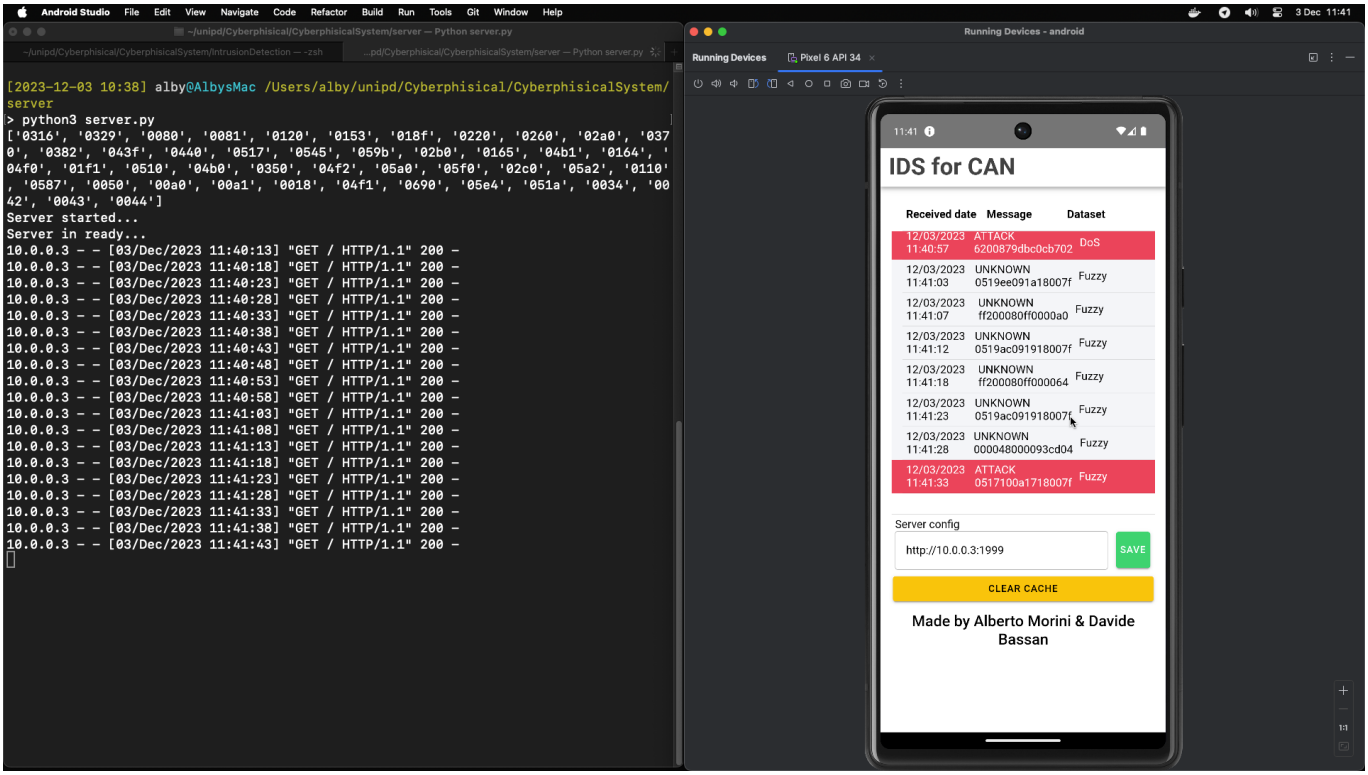

Fig2: Entire system running

Adding the dataset

Server also add the current dataset into the detections:

```
detections = {
    "DoS" : DetectionAlgorithm.analyze_traffic(matrix,aux_ids,
datasetExtractions["DoS"]),
    "Fuzzy" : DetectionAlgorithm.analyze_traffic(matrix,aux_ids,
datasetExtractions["Fuzzy"]),
    "Impersonate" : DetectionAlgorithm.analyze_traffic(matrix,aux_ids,
datasetExtractions["Impersonate"]),
}

for i in detections:    ## for each attack
        for j in range(limitDetections):
            detections[i][j]["dataset"]= i
            AllDetections.append(detections[i][j])

    return AllDetections
```

Client application

As said before, the client application is made with Ionic framework (in React) and then compiled as Android APK app.
For this test, iOS app hasn't been made.

We added some libraries:

- `@ionic/storage` to store into app's cache the detection history
- `@capacitor/local-notifications` in way to trigger a native mobile's notification
- `moment.js` to have an easily manipulation of datetime format

Life cycle

App starts asking the permission, then start polling the server which socket needs to be configured by user.

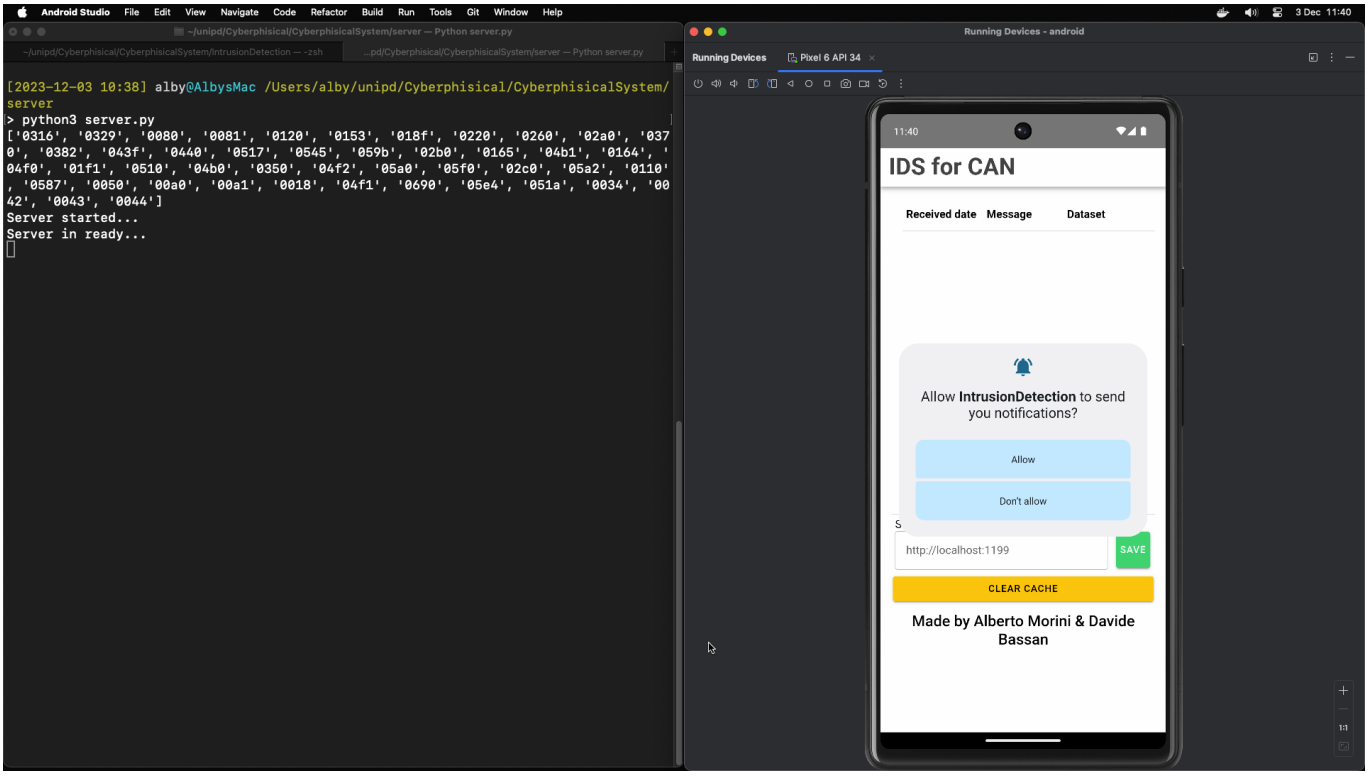

Fig3: App asking for permission to local notification As soon IP address and port of the HTTP server are correct, client retrieve data and store into the cache, after that, via a "React State" automatically update the UI.

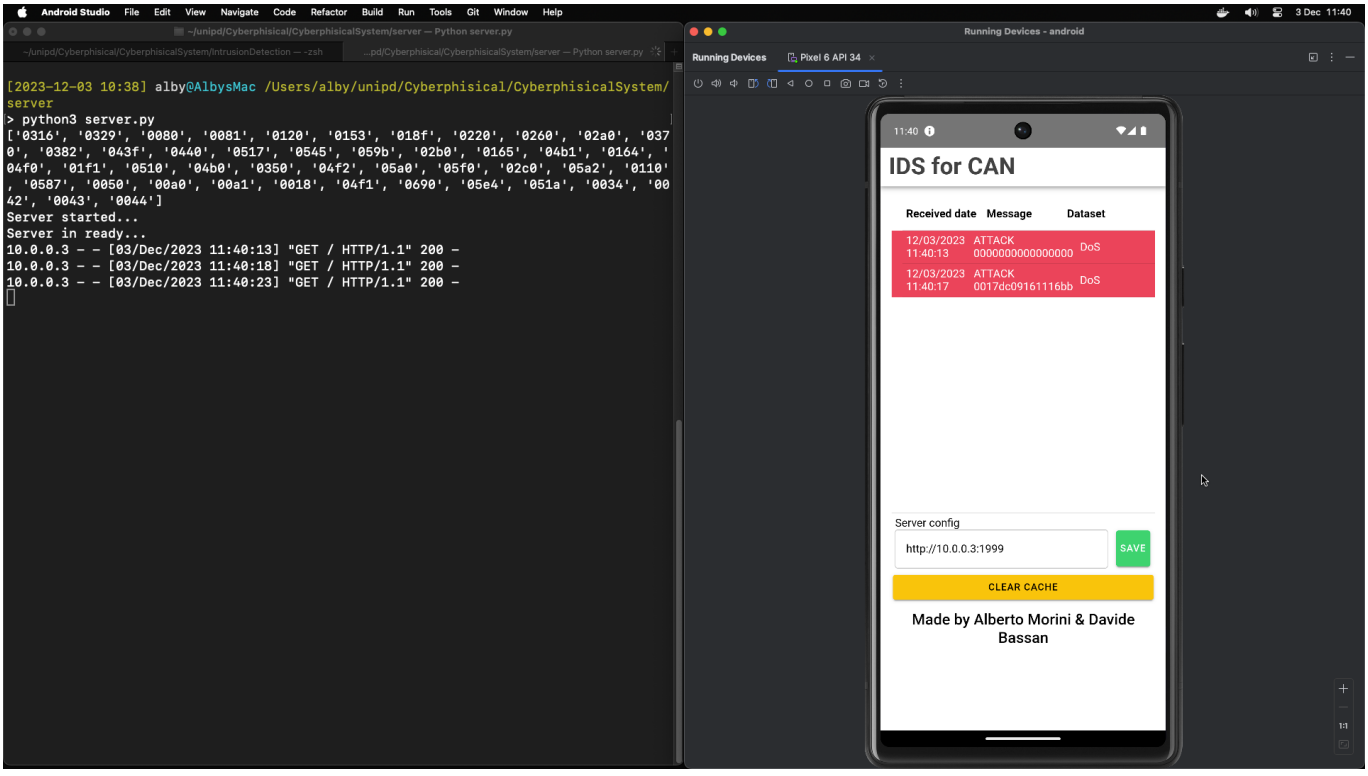

Fig4: Mobile app polling the server and retrieving data

Also at every new data, apps push int the notification center of Android the new alert.

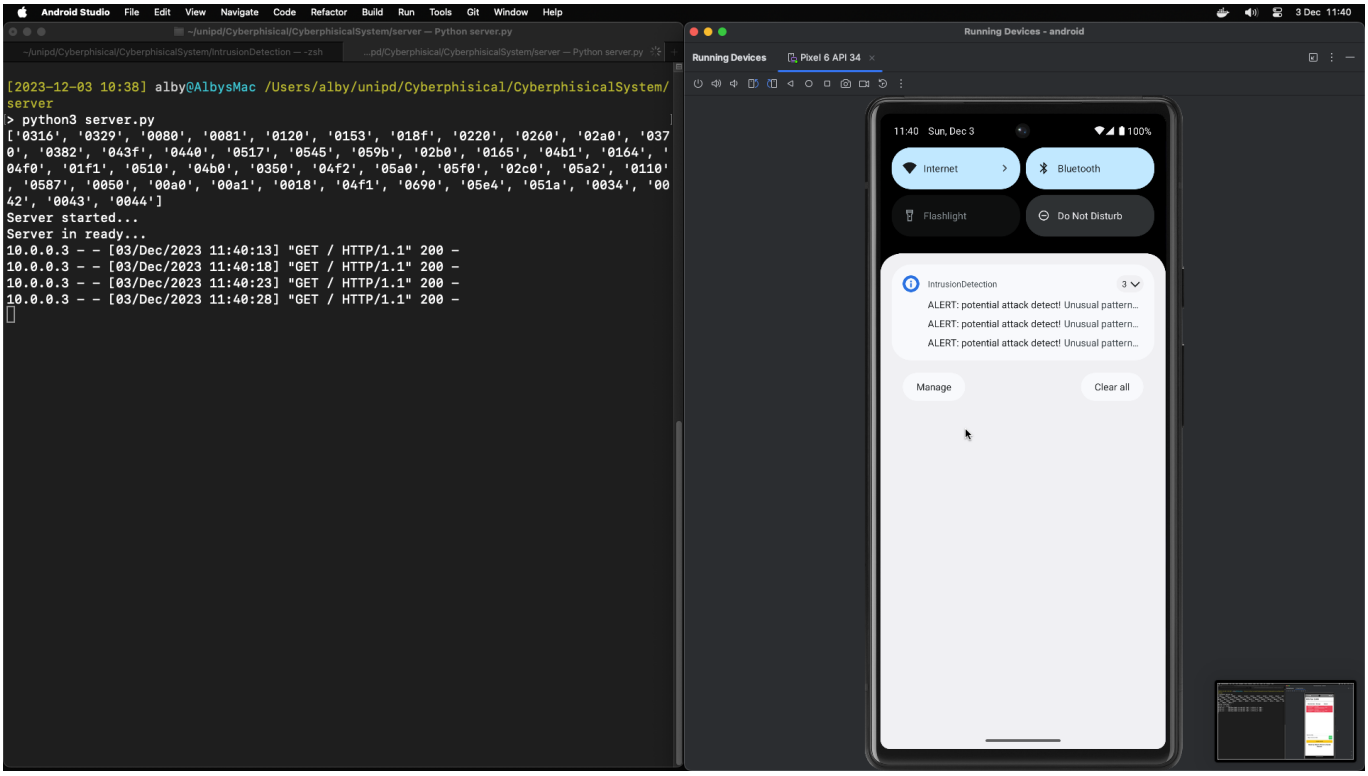

Fig5: The notification message in the notification bar

Messages

At every detection received, client baptize the message with a timestamp called "receptionTS". This information can be very useful in future thus to measure potential delay/latency of the system.

```
res.data.forEach(message => { //res.data is an array of message, forEach one show
a notification
    showLocalNotification(message.id, message.kind, message.msg); //trigger the
android notification

    message.receptionTS = moment().format("DD/MM/YYYY HH:mm:ss"); //baptize the
reception timestamp
})
```

Local notifications (as the detections history) are divided in two group/kind, as the paper did: "attacks" and "unusual traffic"
This information is retrieved by the Detection Algorithm and carried by the server in the client, which basically shows the respective message.

```
async function showLocalNotification(id, kindMessage = "NODATA", CANMessage) {
        let tmpBody;
        if (kindMessage == "ATTACK") {
            tmpBody = "Invalid messages have been detected. This may indicate a
bus error or an attack."
        } else if (kindMessage == "UNUSUAL") {
            tmpBody = "Unusual patterns of messages have been detected. This may
be the result of unusual activity, or it may indicate an attack."
        }

        let options = {
            notifications: [{
                id: id,
                title: "ALERT: potential attack detect!",
                body: tmpBody + "\n CAN MESSAGE: " + CANMessage
            }]
        }

        try {
            await LocalNotifications.schedule(options); //SHOW NOTIFICATION
        } catch (ex) {
            console.log(ex);
        }
    }
```

Detections have a red background in case of "Attack" and light grey if are identified the threat as "Unusual traffic". (*See it in Fig2*)

Conclusions

In conclusion, some critical thought about the system created:

- The only negative observation that we need to make is about energy consumption of Wi-Fi, Bluetooth is lighter and probably available on more vehicles.
- HTTP is an unsecure protocol, we adopt it just for an easily approach, but we strongly suggests to use a SSL certificate (can be a self-signed one) in way to use HTTPS.

Data of detection

With the data available, the algorithm we replicated, have detected many malicious messages, in numbers:

- 605'379 over 656'579 messages for **DoS attack** (numerically in line with a Denial of Service attack)
- 21890 over 591'900 messages for **Fuzzy attack**
- 18101 over 995'427 messages for **Impersonate attack**