RWTH Aachen University
Software Engineering Group

Workshop DSA
Theft Detection

Written report

Kölsch, Alexander

Richter, Christoph

Vasiljevic, Stefan

Sommerhoff, Peter

Strepkov, Ievgen

Supervisor: Klaus Müller

The present work was submitted to the Chair of Software Engineering

Aachen, February 16, 2017

Abstract

This written workshop report describes a theft detection method for Vehicle Connectivity Gateway (VCG). After a short introduction this report describes the used technologies for this approach. Then a theft detection algorithm will be presented and evaluated. The algorithm uses a decision tree to detect wether a car or trailer is stolen or not.

Table of contents

Chapter 1

Introduction

In the context of the lecture *Applied Software Engineering within the life cycle of Automotive Electronics* hold by *Dr. Ansgar Schleicher* at *RWTH Aachen University* we took part at a workshop organized by *DSA - Daten- und Systemtechnik GmbH* [DSA17]. The workshop was devided into two seperate topics: Extending the web *portal BetoTrack* and the development of new functions for *Vehicle Connectivity Gateway (VCG)* devices [Mü17]. Within the VCG area we were allowed to choose between 3 different use cases:

1. Collision detection
2. Theft detection
3. Dead reckoning

At the workshop, we worked in the area of VCG development. We have decided to implement "Theft detection" use case because in our opinion it's the most interesting and useful case. In Germany, every 30 minutes had been stolen one car in 2014.

Dead reckoning was a interesting topic we wanted to work on first. The downside was that many dead reckoning algorithms rely on sensors which are not available in the VCG (see section 2.1). Dead reckoning often uses sensors like a compass and gyroscope to estimate the direction of movement [Kao91]. Because of this we decided not to work at this topic.

The goal was to implement a C++ program which detects an attempted theft and reports it. We have used the preconfigured VirtualBox virtual machine with Eclipse IDE and the partially implemented project. We had to implement additional functional for the project, depending on the chosen use case. The main device which we used to receive data for our program was VCG device. It provides functions like:

1. Location-Based functions : location and tracking of vehicles
2. Functions and status monitoring: monitoring a status of vehicle, onboard diagnostic
3. Update services : update some firmware
4. Entertainment and communication services : access to the data in the cloud and communication services

5. Safety functions : check of safety-relevant vehicle functions or crash-detection

6. Predictive Maintenance : continuous recording of vehicle behaviour and sending data to central service for analysis and initiation of maintenance activity.

For our purpose, we have used the acceleration sensor of the VCG, the GPS sensor, and information from FMS. We used WIFI network of VCG device to send compiled a program to VCG and receive information from it.
Our team consisted of 5 people and for higher efficiency, we decided to split the team into two sub-teams. First one worked on an algorithm of stealing detection and second team tried to get necessary data from the device. We applied pair-programming technic to achieve the better quality of the program.

Chapter 2

Used technology

This chapter briefly describes the used technologies to implement a theft detection algorithm for vehicles. It introduces the Vehicle Connectivity Gateway and its web application Portal BetoTrack. After this the FMS standard is summarized.

2.1 Vehicle Connectivity Gateway

The *Vehicle Connectivity Gateway* (VCG) is a telematics and diagnostics unit. It connects a vehicle with different web portals and enables remote status monitoring of the vehicle as well as remote control actions. This unit can be build in cars, vehicles, trucks or agricultural machines [VCG17].

Figure 2.1: VCG device [DSA17]

The VCG provides multiple communication mechanisms to transport data from the vehicle to the web:

- GSM/UMTS Internet connection to the portal
- GPS/GLONASS satellite position detection
- WLAN for local diagnostics and additional functions

- One-Wire, RS232, digital and analog input for communication with sensors/transmitters
- CAN and Ethernet for ECU communication and diagnostics

Additionally the device has an build in accelerometer and build in battery to be independent of a vehicles power supply. We use the VCG to detect a theft of vehicles.

2.2 Portal BetoTrack

Portal BetoTrack is a tracking web application which allows to track vehicles equipped with VCGs. The app can be displayed in a browser an shows different states of a vehicle [Bet17]. In context of this workshop the theft detection results should be displayed in this portal. The implementation of this was done by other groups of the workshop.

Figure 2.2: Portal BetoTrack [Bet17]

2.3 Fleet Management System (FMS)

Fleet Management System (FMS) is a standardized interface for vehicle data. The FMS standard is supported by many big truck manufactures like Daimler, MAN Truck & Bus, Scania, DAF Trucks, IVECO, Volvo Trucks and Renault Trucks [FMS02]. The communication over the vehicle's CAN bus is standardized by the SAE J1939 network protocol. FMS includes the following sensor values [FMS12]:

- Vehicle improvement (all round)
- Vehicle speed (wheel based)
- Vehicle speed (from tachograph)
- Clutch switch (on/off)
- Brake switch (on/off)
- Cruise control (on/off)

- PTO (Status/Mode)
- Accelerator pedal position (0–100
- Total fuel used (litres since lifetime)
- Fuel level (0–100
- Engine speed
- Axle weight (kg)
- Total engine hours (h)
- FMS-Standard software version (supported modes)
- Vehicle identification number (ASCII)
- Tachograph information
- High-resolution vehicle distance
- Service distance
- Engine coolant temperature

Additional in version 2.0 the following values are also supported:

- Environment temperature
- Driver ID
- Current fuel consumption

We use the FMS to have standardised values we can work with in our algorithm. In section 3.3.3 we describe which values we are using in our algorithm.

Chapter 3

Theft Detection

3.1 Use Cases

We want to implement an algorithm which detects if a vehicle is stolen. Therefore we shortly explain the use cases we thought of. The user of the system wants to check if a theft alert is triggered via Portal (see section 2.2). Figure 3.1 shows an overview of the use cases.

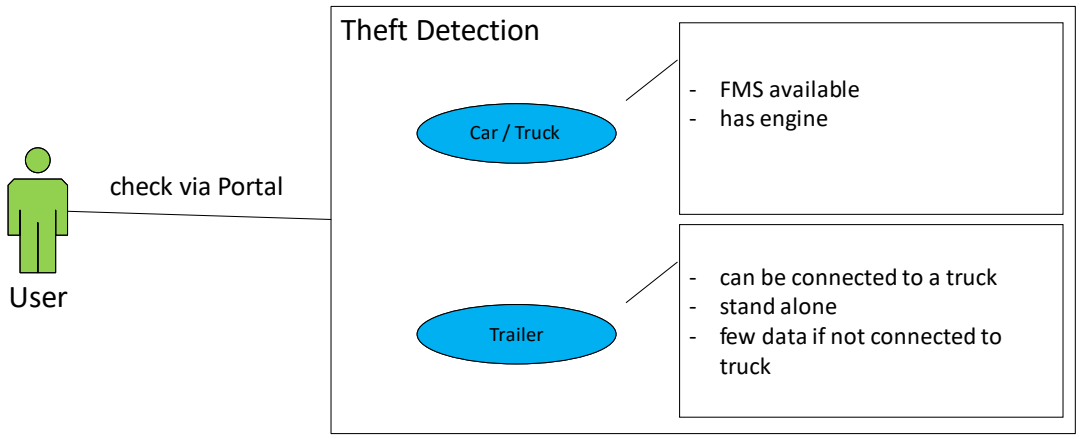

Figure 3.1: Use Cases

We distinct between teft detection for a car or truck and a trailer only because VCGs are often build in trailers. Depending on those configurations we have different data available which have to be considered in the implementation.

3.2 Algorithm

The algorithm works on a principle of the decision tree (see figure 3.2) and it can be used for detecting a theft of either a car/truck or a trailer.

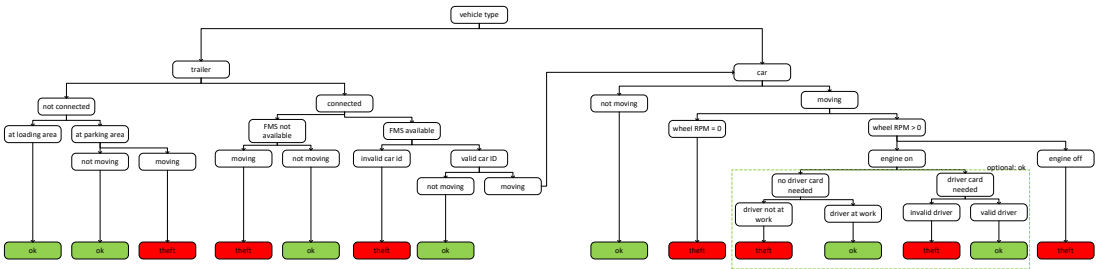

Figure 3.2: Decision tree

Based on the information whether the specific data is available, the different path in the tree is selected and consequently, the decision is made. Since the use cases for trailer and truck are different, the algorithm will be explained for both cases independently (see figure 3.3).

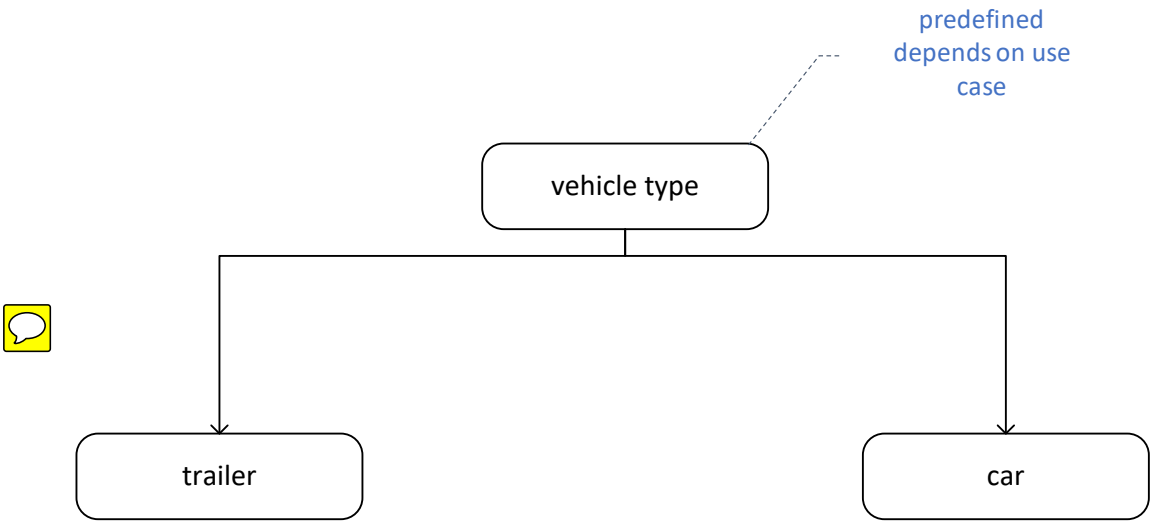

Figure 3.3: Decision tree top

When the trailer case is considered, there exist specific properties that have to be taken into account in order to distinguish correctly between the possible outcomes. In other words, it has to be determined whether the trailer is connected to a vehicle, whether there is available FMS data, but also the current position of the trailer has to be acquired. The first decision is made according to the information of the power connection. If it is the external one, the trailer is not connected, which triggers a query on the current position. Under the assumption that the company that owns the trailer geofenced the loading and parking areas, further decisions are made. On the one side, while the trailer is located within the loading area, no potential theft is detected, regardless of whether

the trailer is in motion or not. However, if the trailer is at a parking area, it is assumed that any kind of movement could represent the potential theft and the algorithm will end its path in the "theft detected" final node. The data needed for answering the previously described queries is acquired using the GPS and accelerometer sensors. The GPS is used for collecting the trailer location and checking whether the trailer is in the predefined position. It should be stated that our algorithm can easily be applied to the case when the trailer is being transported by a ship, where the geo-fence can be dynamically generated based on the boat position. When it comes to gathering the movement information, both the GPS and the accelerometer sensors are used.
In case that the trailer is connected to the vehicle, one additional resource is used for the algorithm analysis - the FMS data. If this data is not available and the trailer is moving, the theft is obviously detected, because the trailer is attached to an unrecognized vehicle. Again the movement information is acquired from the GPS/accelerometer sensors. However, if the FMS is available, that does not necessarily mean this is an acceptable case because a unique car ID is still to be checked. If it is an invalid id the stealing is detected, otherwise, the information of the vehicle movement is considered. At this point, the algorithm generates queries regarding the vehicle information only, therefore the theft of the car/truck analysis is to be described next and it follows below.

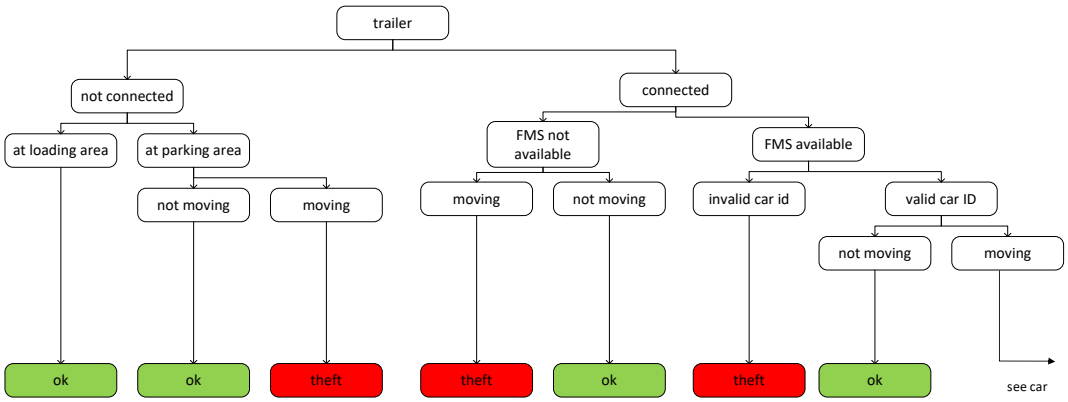

Figure 3.4: Decision tree trailer

The GPS/accelerometer sensors answer the first important decision question in this part of the analysis - whether the car is moving or not. It is assumed that no motion represents a valid situation, so the only interesting case is recognizing the theft if there is a movement detected. Therefore, it is henceforth assumed that the vehicle is moving and the rest of the data that determines the path through the tree originates from the FMS. The wheel RPM is the first thing to check in this kind of situation because if the RPM is zero, the car is probably lifted and is being transported illegally. Otherwise, if the wheels are spinning, it should be asserted that the engine is on, which represents a valid scenario. Contrarily, if the engine is off with the RPM greater than zero, the theft is detected.
Moreover, if the driver cards are used in a company, the case when the engine is on can be extended in order to provide a more precise outcome. If a driver card is not required, it is checked whether the driver is on a job. If so, it is expected that the driver is in a vehicle, otherwise, the theft is identified. On the other hand, if the driver card is required, information on whether it is an invalid or a valid driver distinguishes between a theft and a regular situation.

In section 2.3 it will be explained thoroughly how the main data sources are used.

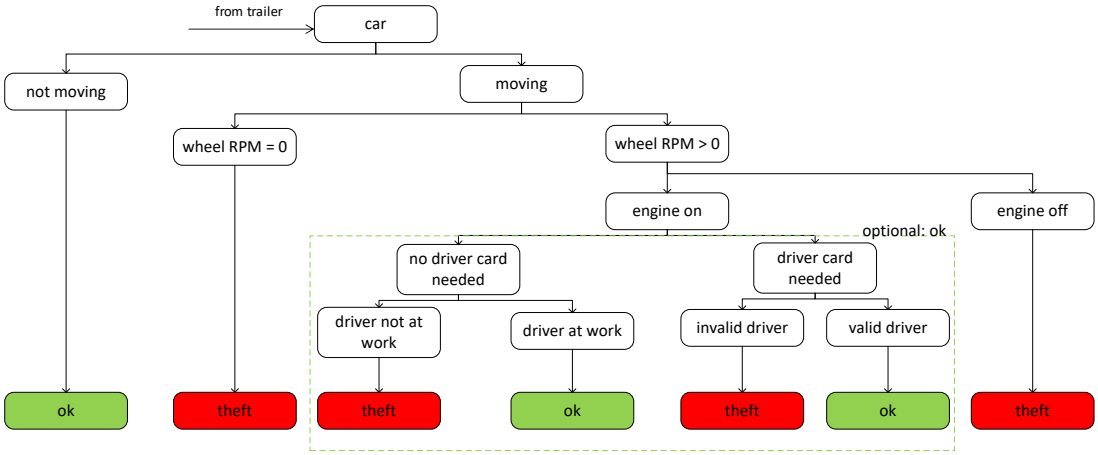

Figure 3.5: Decision tree car

3.3 Data Sources

Based on our algorithm idea represented by the decision tree, we knew which data we need to implement the theft detection. In this chapter, we discuss in more detail how we extracted these data from the vehicle and encapsulated it in our code.

3.3.1 Accelerometer

The first data source we used was the vehicle accelerometer which emits the current acceleration in X, Y and Z direction every 100ms by default. We chose to extract these data first because it can be used to implement a simple but effective theft detection already. Every illegal movement of the device (and thus vehicle) will be reflected in the accelerometer data.

In order to get access to the accelerometer data, we used a C++ library dedicated to the BMA255 accelerometer[1] built into the test devices. The basic procedure to read data from it is as follows:

1. Open connection via `open("/dev/bma255", O_RDWR)`
2. Use the `select()` and `FD_ISSET()` functions to check if data can be read from the file descriptor
3. Read the available data via `int numberRead = read(accelerationHandle, buffer, length - 1);` with a buffer of the right size

We can parse the resulting data to retrieve the X, Y and Z acceleration. Naturally, reading has to occur continuously to read the current values. We decided to encapsulate the basic data access as above into a class `AccelerometerSensor` and add another proxy layer on top which persists the 5 most recent data points and calculates whether the vehicle is moving based on these. We adjusted the threshold until we were satisfied with the behavior but future work remains in this area.

3.3.2 GPS

In order to implement further layers of theft protection, we need to incorporate GPS data. This way, we can detect a theft if the vehicle is moving but the wheels are not spinning for example. To extract GPS data from the device, we used the GPSd daemon[2] for C/C++. The procedure to access the data uses file descriptors again and is thus similar to the accelerometer.

GPSd provided various values of which we only need a small subset. Thus, we encapsulated the values we need into a `gps_data` struct to constrain the thousands of values provided by FMS to those we need and increase performance. Also, we again added another proxy layer which calculates the distance between two data points from GPS and decides whether the vehicle is moving.

[1] `https://www.bosch-sensortec.com/bst/products/all_products/bma255`
[2] `http://www.catb.org/gpsd/`

3.3.3 FMS

For commercial vehicles, additional data from the Fleet Management System (FMS) is available (see section 2.3). This can be used e.g. to check whether a valid driver card has been inserted (indicating that the designated driver is present) or not (indicating that the vehicle is moving without its driver). All relevant data for our algorithm are:

- **The FMS status** to check whether FMS is currently not connected or online
- **The battery ECU power state** to check if a trailer is connected to its truck or not
- **The vehicle ID** to verify with a whitelist
- **The engine speed** to see whether the engine is running or not
- **The wheel-based speed** to decide whether the wheels are spinning or not
- **The driver card state** to check whether a valid driver card is present or not
- **The driver status** to check whether the driver is currently not at work, working, on a break, etc.

Again, we encapsulated these values we need into a struct called `fms_data` to constrain the thousands of values provided by FMS to those we need. This also acts as documentation for which values are used and transmitted across the classes. For FMS, we do not need a proxy class because we only need the actual values coming directly from FMS.

For all data access, we made sure to open the connection only once, then continuously read data, and then close the connection in the destructors of the appropriate classes. Also, we refactored our architecture to use singletons for the accelerometer and GPS sensor classes because we really only need one instance of both and this can prevent accidental use of copy constructors.

3.4 Architecture

As described before, the three main data sources that the algorithm uses are FMS bus, accelerometer sensor and GPS sensor. Since the overall available data is much larger than the data required by the algorithm, the corresponding API is provided for each data source. In addition to the encapsulation role these interfaces have, they can be extended for the future use as well, in the case of additional modifications or improvements. Moreover, a layer is added above the accelerometer and GPS API in order to maintain the acquired data and provide further actions.

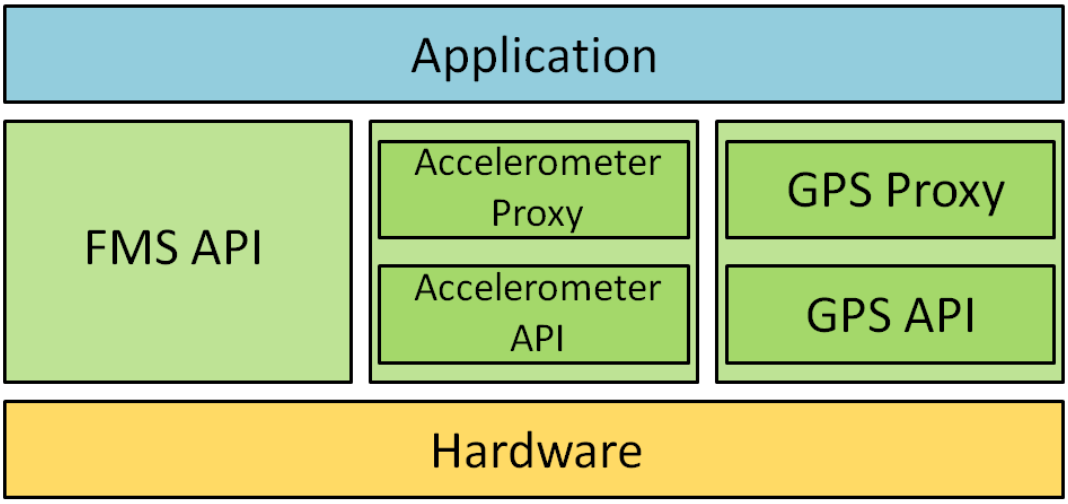

Figure 3.6: Architecture

As it is shown in the Figure 2.1, the architecture is layer based. The higher layers abstract from lower layers, while the lower layers do not know about higher layers. Nevertheless, not all the communication links are permitted. Allowed ones are only the module based communication links into the directly lower level. That being said, neither the horizontal communication nor bypassing the layers is permitted. Currently, the only present communication is top-down by function calls, but the possible one could also be bottom-up by triggering some event. This structure improves the maintainability and scalability on the one hand, but in contrast, it could increase a communication load significantly with including more functions to the algorithm.

Chapter 4

Evaluation

At the end of the workshop, we were able to have our algorithm idea implemented. Due to the timeframe of the workshop, we were only able to manually test the essential functions, i.e. theft detection via the accelerometer and suspicious GPS movements. As soon as the device is moved (e.g. lifted up from a table), the accelerometer-based detection correctly fired. Additionally, we field-tested the GPS-based detection by running down the street to simulate a noticeable change in location.

However, in the scope of the workshop, much of the FMS data was not available. Thus, not all branches from our decision tree could be tested. This could have been achieved by simulating the FMS data. However, we did not have the necessary time left towards the end of the workshop.

Theft detection is a challenging problem. Even though all the different branches in our decision cover many possible ways to steal a vehicle, there are cases which are not covered – and cases that are hard to detect in general. This includes the case where a thief has gained access to the vehicle keys, thus leaving no way to detect a theft in non-commercial vehicles (which do not require a driver card).

Chapter 5

Future Work

In our program, basic functions for theft detection are implemented, which are necessary for the correct operation of the program. Still, some weak points remain which can be improved in the future:

- Use binary mode to read from acceleration sensor to avoid parsing the data manually as a string
- Calibrate acceleration sensor threshold based on in-the-field testing (e.g. beside a highway)
- Test all branches of the algorithm, ideally with real vehicle data

Bibliography

[Bet17] Professional telemetric solution betotrack http://www.dsa.de/en/solutions/products/ February 2017.

[DSA17] DSA - Daten- und Systemtechnik GmbH website http://www.dsa.de/de/, February 2017.

[FMS02] Information about the fms-standard http://www.bus-fms-standard.com/truck/index January 2002.

[FMS12] Fms-standard description version 03, September 2012.

[Kao91] Wei-Wen Kao. Integration of gps and dead-reckoning navigation systems. In *Vehicle Navigation and Information Systems Conference, 1991*, volume 2, pages 635–643. IEEE, 1991.

[Mü17] Klaus Müller. Workshop: Software engineering for connected vehicle platforms. 2017.

[VCG17] Vehicle connectivity gateway vcg http://www.dsa.de/en/loesungen/produkte/vehicl February 2017.