Intrusion Detection System (IDS) on automotive CAN bus

Nieutin Vincent

September 2, 2020

Apprenticeship paper with IoT.bzh.

1 Abstract

The main objective of this paper is to implement an Intrusion Detection System (IDS) over the Controller Area Network (CAN) data buses. In order to achieve this, we will first study the architecture of CAN buses in the automotive industry and then develop a detection method plan in order to treat various attacks.

Keywords IDS, Intrusion Detection System, CAN, Controller Area Network, data bus, automotive

Contents

2 Problem Statement

Modern cars have a wide variety of networks that interconnect components called "Electronic Control Units" (ECUs). One of the oldest and most famous automotive bus is called the Controller Area Network (CAN). A majority of the attacks performed nowadays are done on this data bus. The data transmitted on it has the particularity of not being authenticated nor confidential.

An ECU is an embedded system that controls physical devices. It can act according to data from sensors traveling over the data bus on which they are connected. As ECU, we can for example mention ABS (Anti-lock Braking System), ESP (Electronic Stability Program) or RKE (Remote Keyless Entry).

Since an ECU controls physical devices on a vehicle, corruption of the ECU by an attacker may allow the attacker to take control of these physical devices. The RKE system is a good example of this. With a team of researchers, Flavio D. Garcia exposed vulnerabilities in the RKE system that allow an attacker to control the opening and closing of a vehicle [1].

An ECU controlled by an attacker can be used as an entry point on the CAN bus to attack other ECUs. An attacker could also connect directly to the CAN bus electrically or pretending to be an ECU.

This paper focuses on physical attacks that can occur on the CAN bus network. With the addition of more and more autonomous features such as vehicle steering, it is important to focus on data bus security in order to limit the impact and propagation of such attacks.

Some examples of possible attacks:

- falsifying the data frames of airbag activation in order to deactivate them.
- in an "intelligent" vehicle, corrupting the autonomous steering frames in order to deviate the vehicle's trajectory.
- triggering an emergency braking of the vehicle (functionality used as

[1] see ref. 7 in references part

an anti-collision if an obstacle is in front of the vehicle) when another vehicle is just behind (detected by the back-up radar).

Attack detection tools already exist (e.g. Snort). In the embedded world, problems such as computing power, energy consumption or signature centralization occur.

3 Background

This part presents the core concepts we will need as we move through the rest of this paper.

3.1 CAN bus

3.1.1 Assumptions

Architecture The number of nodes on a CAN bus is theoretically unlimited. There is no relationship between the number of connectable nodes and the number of identifiers. However, there are physical limitations that limit the number of nodes on a bus (propagation, electrical charge) depending on the medium used. We know that the number of different ECUs used in the automotive sector is increasing. In order to reduce the load and increase security on a bus, car manufacturers have decided to multiply the number of CAN buses.

We will see different architecture types. Note that, in practice, we will certainly meet a mixture of these architectures.

- Segmentation

  A first way to see the picture may not be to connect the different buses together.

  In terms of security, if an ECU in a network is corrupted, the attack surface will only be the bus containing that ECU. However, an ECU may need to be on several separate buses at the same time. This would weaken bus segmentation since the attacker could now use this ECU as a gateway.

- Interconnection

  A second way to picture the architecture of the buses would be to connect them together via a single gateway. No more segmentation.

  There is no longer the need to connect an ECU to several buses. However, the expansion of the attack surface will only take into account the robustness of the gateway.

It is still possible to filter the packets going through the gateway. We can therefore virtually reproduce the segmentation seen above. However, it remains less viable than the first one for obvious reasons.

3.1.2 Mechanisms

We are looking here at some CAN bus data link layer mechanisms that will be useful for us later.

- Cyclic Redundancy Code

  CRC is used to check the integrity of the transmitted frames. Any changes to the Start Of Frame, arbitration (data identifier), control (data length) and data during transmission will therefore be visible.

- Transmission priority

  During a simultaneous data transmission from several nodes, each node checks that its data identifier is not overwritten by another message. If so, the node waits before resending the frame. The lower the data identifier is, the higher the priority of the message.

- Error counters

  There is a whole transmission and reception error detection system. It checks that the CAN bus specifications are met.

  If this is not the case, error counters are incremented. There are 2 per node: in transmission (TEC) and in reception (REC).

  Depending on specific thresholds, the node will change its error mode to active, passive or "bus off". The last one is particularly significant since it tells the node to disconnect from the bus. Once disconnected, only the reception of 128 particular frames will allow the node to get out of this mode.

3.1.3 Linux implementation

- SocketCAN

  kernel.org - can

SocketCAN is a set of drivers and a network layer that allows you to interface with CAN data buses. SocketCAN introduce a new protocol family named $\mathbf{PF_{CAN}}$ like $\mathbf{PF_{INET}}$ for the Internet Protocol.

  - Kernel modules
    * *can* : CAN $\mathbf{PF_{CAN}}$ core
    * can_{raw} : raw CAN sockets
    * can_{bcm} : broadcast manager
    * *vcan* : offers a virtual local CAN interface

- can-utils

  github.com - can-utils

  can-utils offer userspace SocketCAN utilities. Those that will be useful will be presented later.

  - *candump* : display, filter and log CAN data to files
  - *cansniffer* : display CAN data content differences (only 11-bit CAN IDs are supported)
  - *canplayer* : replay CAN logfiles
  - *cansend* : send a single frame
  - *cangen* : generate (random) CAN traffic
  - *cangw* : CAN gateway userpace tool for netlink configuration

3.2 Intrusion Detection/Prevention System

An Intrusion Detection System (IDS) is a mechanism to identify abnormal or suspicious activities on the target being analyzed (a network or host). It thus allows one to have knowledge of the successful and failed attempts of intrusions.

In our case, we will use and be inspired by network IDS (NIDS). For example, we can observe how an IDS over USB works.

There are also Intrusion Prevention System (IPS).

An IPS is a tool similar to IDS, allowing measures to be taken to reduce the impact of an attack. It is an active IDS, it can for example block ports

automatically. Like IDS, they are not 100% reliable and may even block legitimate traffic in case of a false positive.

In the automotive industry, the disadvantages that an IPS could cause are :

- False positive : They block anything that seems infectious to them, but IDSs are not 100% reliable, so they can inadvertently block legitimate applications or traffic. This is not an option in some cases in the automotive field, like blocking the communication of the front distance sensors of a car. If the car is equipped with it, the automatic braking could not be activated when an obstacle approaches dangerously.
- False negative : They sometimes let some attacks go by without detecting them.
- They are not very discreet and can be discovered during an attack. If compromised, an attacker has access to IPS permissions which can be numerous. He will then be able to deflect it from its original use.

4 Attack types

Can be seen here the different types of attacks which may be encountered on the CAN bus. These types will be characterized as active (packet injection, writing) or passive (packet interception, reading).

They are due to the following security principles :

- confidentiality : non-authenticated party cannot examine the data.
- authenticity : a receiver or a sender is who it claims to be.
- non-repudation : a sender ensures that a receiver has received the message, and a receiver is sure about the identity of a sender.
- integrity : data cannot been changed or generated by an unauthorized node.

The following attacks are mainly due to the lack of authentication and confidentiality.

4.1 Sniffing / Man In The Middle (MITM) (passive)

Each node of a CAN bus sends its messages in broadcast mode. So all nodes of a bus receive the packets of the others. By pretending to be an ECU, an attacker can therefore see all the packets traveling on a bus. Since there is no encryption of CAN messages, the attacker will be able to view all the content.

4.2 Spoofing (active)

There is no information about the sender in a CAN frame. In order to simulate an ECU, it is only necessary to simulate its behaviour.

4.2.1 Replay

A replay attack would consist of recording one or more frames of an ECU and then injecting them back into the bus at the desired time. For example, we can imagine recording frames that disable airbags. Then injecting them into a multitude of similar vehicles. Notice that this attack is not very effective on ECUs injecting frames periodically even if the data has not changed. Indeed, the injected information will very quickly be overwritten by a new frame.

4.3 Fuzzing (active)

By injecting random messages (ID, payload) and then observing the reactions of the bus and the car in general, it is possible to improve your understanding of the structure and content of the frames.

This method is obviously not very quiet and can lead to the disruption of vehicle functions.

4.4 Low level (active)

Instead of injecting frames according to the CAN protocol, there is nothing to prevent dominant and recessive bits from being injected directly into the bus. It is therefore possible to alter a frame already transmitted by another ECU.

One of the difficulties lies in the correct timing of the injection. A majority of ECUs send their frames periodically. It is then possible to detect this period and synchronize with it. It is also possible to simply send a multitude of bits at a time.

4.4.1 Bus off

The CAN protocol provides a mechanism for ECU confinement when the frames it sends have too high error rates (fault confinement). By altering the frames sent by an ECU with random bits, it is possible to trigger the

containment of this ECU. It will therefore no longer be able to communicate on the bus.

This attack can be used as a precursor to spoofing attacks. An IDS could detect an abnormal change between transmission times of the same frame. By confining the ECU off the bus, we can then spoof frames.

4.5 DOS (active)

A Denial Of Service (DOS) attack is used to block the communication of ECU(s) on the bus. This could put the vehicle in an unstable state.

4.5.1 High priority

The CAN protocol allows one to prioritize frames over others in the case of a collision. This is done through the message ID field. The lower the value, the higher the priority. One way to flood the bus is to send a multitude of frames at the highest prority (0x000). All frames that collide with these malicious frames will be dropped.

4.5.2 Error flags

As we said earlier (bus off attack), it is possible to confine an ECU by altering the frames it sends. Instead of blocking a single ECU, we can alter each frame passing through the CAN bus. This would have the effect of confining as many ECUs as possible.

4.6 Gateway corruption

As described in the architecture part, a gateway can be used as a central node between several CAN buses. Let's assume that we have succeeded in taking control of this gateway. We therefore have the possibility to read and modify all the frames that are supposed to pass through this gateway. It is therefore possible to perform sniffing, spoofing and DOS attacks.

5 Attack detection

We assume that we receive a large majority (or even all) of the CAN bus frames. This means that we were able to connect to all CAN buses (for example via the outputs of a gateway).

We will also consider the working costs of these detection methods, as well as the impacts on the hardware.

Let's study the different methods of attack detection.

5.1 Signature

The idea is to identify patterns. We can either authorize or refuse a list of patterns (white/black list). This method costs few resources, but it can need a CAN frame history. The following signature proposals are non-exhaustive.

5.1.1 On specifications

The CAN bus already has a low level frame non-conformity detection mechanism according to specifications (error counters). In order to supply the IDS with information, we can read the error frames sent on the bus.

We can add higher level detections.

Constructor If we have access to the information structure in the CAN frames, we can simply set limits on what can be done according to the manufacturer's specifications.

- ID

  If all the frame identifiers passing on a bus are known, we can simply whitelist them. All other frames will be detected. This method would prevent any ID fuzzing, low level ID hitting or DOS priority attacks.

- Frequency

As we said earlier (low level attack), some ECUs send their frames periodically. We can then detect abnormal behaviour if this period is broken.

- Data

  In the same way as filtering identifiers, if the behavior of frames of the same identifier are known, we can detect any unwanted behavior. This method thus makes it possible to prevent any uncontrolled spoofing, fuzzing or low level attacks on the data.

5.1.2 On attacks

Patterns can be identified within the active attacks described above (attack types).

Data frame fuzzing We only focus on data frame fuzzing and we do not know the manufacturers' specifications. It is therefore necessary to memorize a set of frames in order to visualize typical inconsistencies of a fuzzing attack. Usually, the higher the number of frames observed, the lower the false positive rate and the longer the analysis time will be.

- Overall identifiers

  Over the last N frames, if the number of different identifiers reaches a threshold, we can conclude a fuzzing attack on identifiers. This is possible because we know that not all CAN identifiers are actively used, only a part of the identifiers are used.

- Different sending frequencies for the same identifier

  Over the last N frames with the same identifier, if the number of different distances between each frame timestamp reaches a threshold, we can conclude that there is a fuzzing attack on this identifier. This is possible because we know that many ECUs send frames on a regular frequency. A difference breaking the regularity can therefore be interpreted as an attack.

- Different identifiers with same data

Over the last N frames with the same data, if the number of different identifiers reaches a threshold, we can conclude a fuzzing attack on identifiers. This is possible because we know that the data can be very different depending on the identifiers and that an attacker could send the same data with many different identifiers.

DOS In a similar way to fuzzing attacks, a major characteristic of a denial of service attack is the frequency with which frames are sent over the bus. Other approaches based on high priority frames and CAN error counters can also be studied.

- Overall frequency

  Overs the last N frames, if the number of frames in a period of time reaches a threshold, we can conclude that there is a DOS or fuzzing attack. We know that the general sending frequency over a long period of time is not going to vary much, not as much as in a denial of service attack that suddenly increases the general sending frequency on the bus. However, this detection requires having the number of frames sent in memory over a relatively long period of time in order to reduce false positives.

5.2 Anomaly

Here, we will observe the global behavior of CAN frames. First of all, it is necessary to record and collect legitimate and normal frame behaviour.

During monitoring, any deviations from normal behaviour will be identified as suspicious.

Deep learning can be used to identify deviations from a normal behavioural model. The advantage of creating a normal behavioural model with deep learning is that you do not need to know the meaning of the frames created by car manufacturers. No more reverse engineering.

Training a model with deep learning requires a lot of resources. The cost of on-board equipment would be too high.

We can list two solutions available to the car manufacturer:

- deport the computing power to manufacturer and maintain an embedded model.
- use low embedded computing power and develop an embedded model common to the vehicle fleet.

6 CAN bus simulations

We will create a total of 3 virtual CAN buses.

bus name	comment	speed (kbits/s)
vcanHSS	Inter-Systems	500
vcanLSEXT	Car's exterior	125
vcanLSINT	Car's interior	125

The 3 buses are connected between each other via a gateway.

6.1 Implemented ECUs

6.1.1 Inter-Systems

name	frame id(s)	comment
ABS	0x37B, 0x52C	Anti-lock Braking System
GB	0x479	Gearbox

6.1.2 Exterior

name	frame id(s)	comment
ALARM	0x123	Alarm system

6.1.3 Interior

name	frame id(s)	comment
WL	0x2C3, 0x345	Window Lifter

7 IDS implementation

7.1 Signature

We listen to each bus in parallel and store the last frames sent in python buffers. There will be no inter-bus correlation in this part. The detection will start with each new frame sent on a bus and will determine whether this frame is malicious or not.

For performance considerations, we use several buffers arranged in the following structures :

- The N last frames (buffer)
- The N last frames with the same identifier (buffer dictionary)
- The N last frames with the same data (buffer dictionary)

7.1.1 On specifications

A simple whitelist is used to implement manufacturers' specifications.

- ID

  We just check that the last frame sent is listed in our constructor whitelist.

- Frequency

  We only check the distance between the last two frames with the same identifier. A detection rate allows us to apply a margin.

7.1.2 On attacks

We are looking here for abnormal behaviour that could be blacklisted, unlike the previous section.

Data frame fuzzing All of the cases cited in attack detection part have been implemented. A detection rate is applied to all cases. It can be adjusted during normal bus behaviour.

- Overall identifiers

  Once the N last frames buffer is filled, we count the number of different identifiers. The detection rate applies to the ratio between the number of unique identifiers over the buffer size.

- Different sending frequencies for the same identifier

  For all N last frames with the same identifier, we calculate the sending time between each frame rounded to the nearest hundredth. For frames considered periodic, a different number of delays (using the detection rate) greater than 1 raises an alert.

- Different identifiers with same data

  Once the N last frames with the same data buffer size exceeds the detection rate multiplied by the buffer size, the number of unique identifiers in the buffer is counted. The detection rate applies also to the ratio between the number of unique identifiers over the buffer size.

7.2 Anomaly

7.2.1 Outlier detection

We used the Python Outlier Detection (PyOD) [2] toolkit in order to identify CAN frames that would be outside their normal behavior.

Note that only legitimate data training has been done, the detection scenarios in the following section does not include anomaly detection.

Here the example of the driver window frames (0x2C3) will be taken. The meaning of the frames content will not be used but only its value converted into integer. The dump used simulates a complete opening and closing of the driver's window.

[2] see ref. 4 in references part

The code will not be described here but only the results of the different detection methods. If you want to reproduce these tests yourself [3].

Despite the fact that the data are raw, we can see on the next distribution diagram the different phases of the window's position.

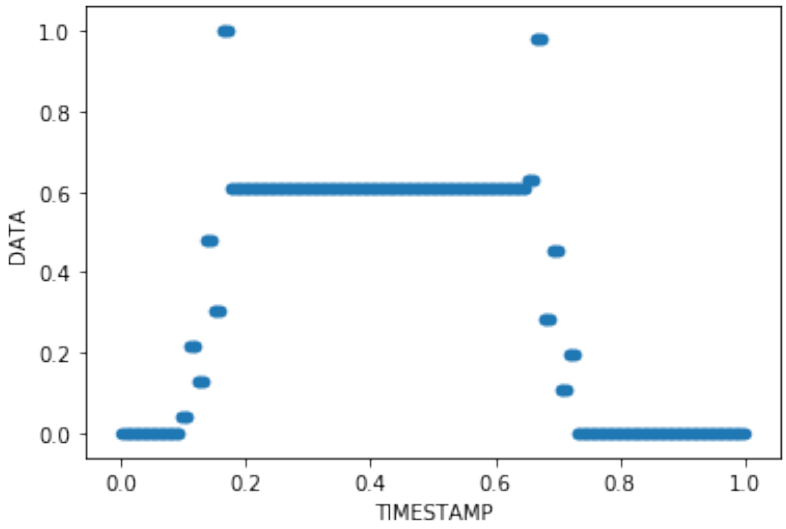


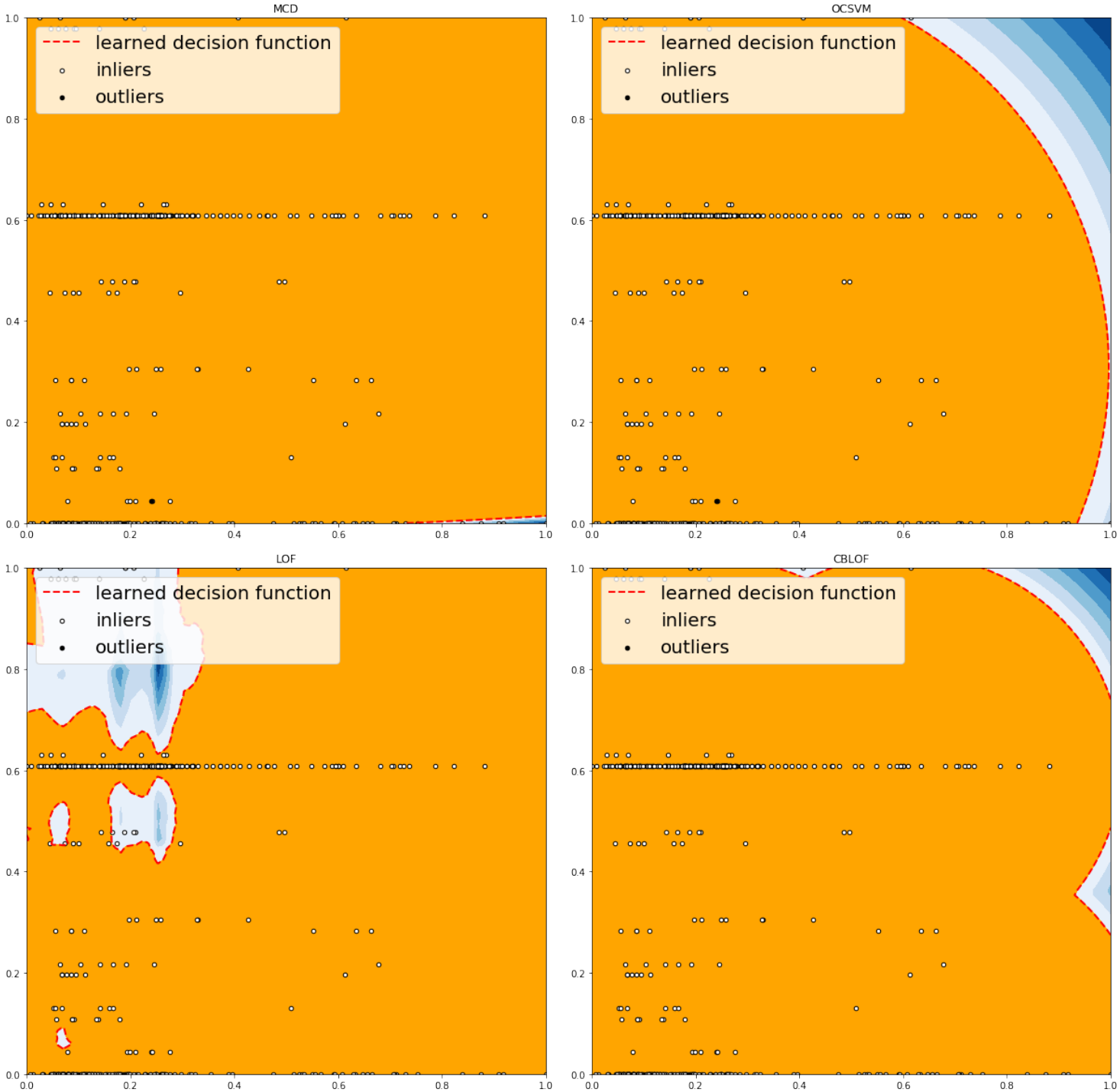
Figure 1: Distribution of 0x2C3 frame data over time.

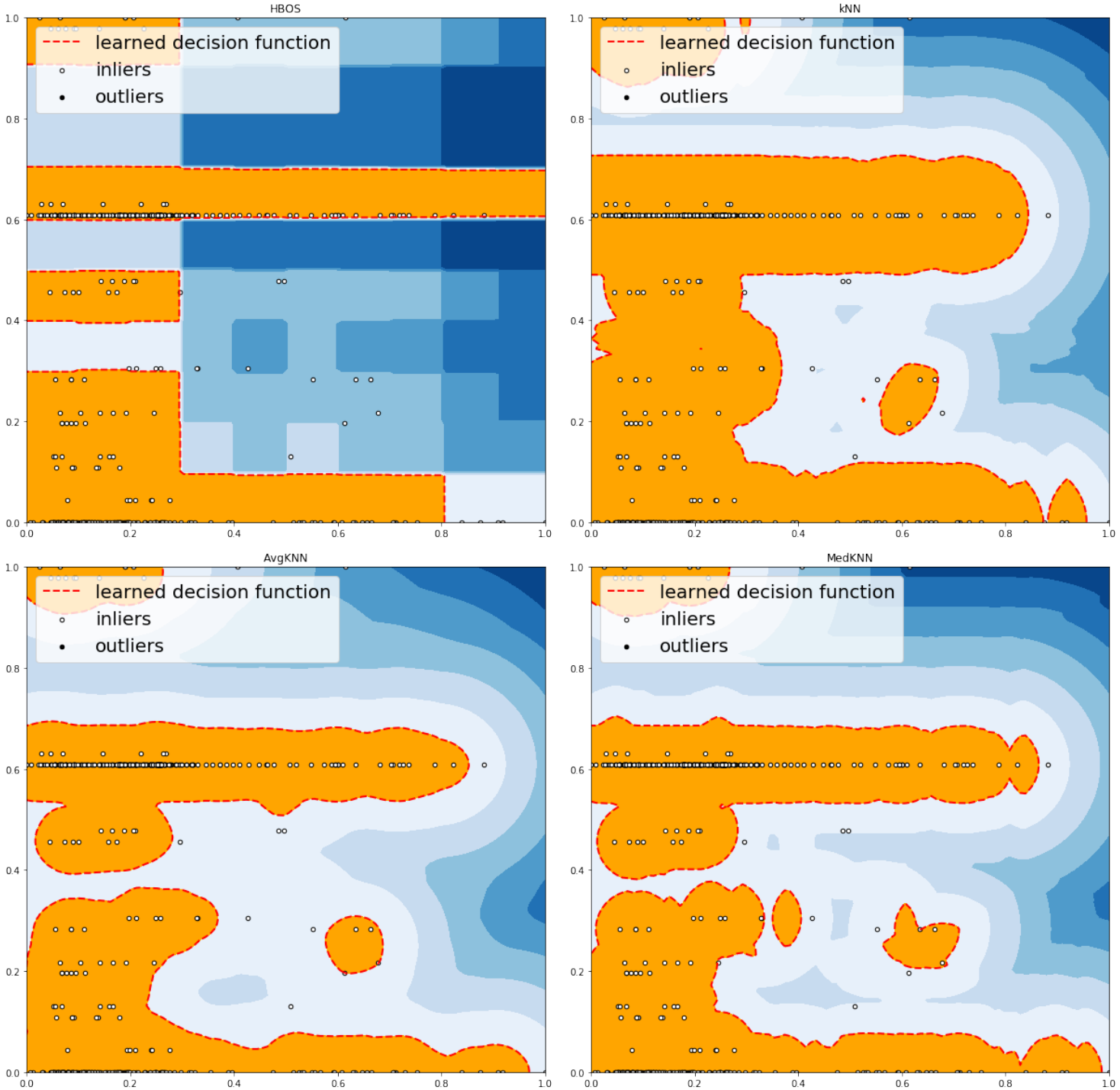
Let's start by showing the different detection algorithms we will use [4].

Type	Abbr	Algorithm
Linear Model	MCD	Minimum Covariance Determinant
Linear Model	OCSVM	One-Class Support Vector Machines
Proximity-Based	LOF	Local Outlier Factor
Proximity-Based	CBLOF	Clustering-Based Local Outlier Factor
Proximity-Based	HBOS	Histogram-based Outlier Score
Proximity-Based	kNN	k Nearest Neighbors
Proximity-Based	AvgKNN	Average kNN
Proximity-Based	MedKNN	Median kNN

As well as the training results of these algorithms on the driver frame positions, we can discuss these results at the end.

[3] see ref. 5 in references part
[4] see ref. 6 in references part for more informations about detection algorithms


MCD
learned decision function
inliers
outliers
OCSVM
learned decision function
inliers
outliers
LOF
learned decision function
inliers
outliers
CBLOF
learned decision function
inliers
outliers


HBOS
kNN
AvgKNN
MedKNN
learned decision function
inliers
outliers
0.0
0.2
0.4
0.6
0.8
1.0

8 Scenarios

8.1 Sniffing / Fuzzing

8.1.1 Identifier

Basic The first basic implementation tests each identifier incrementally with the same data at high speed. There was no prior sniffing to identify existing identifiers.

Advanced We start by sniffing the buses to recognize the identifiers already in use. Each sent frame contains a different data. For each tested identifier, a multitude of identical frames are sent. The frames are sent at a constant speed.

8.1.2 Data

On an unknown identifier The data is sent incrementally, there is no need to make the data sending random since this could be a normal behavior. The frames are sent at a constant speed.

On an existing identifier In the same way as before, the data is sent incrementally. But now we are sending our frames as fast and constant as possible.

9 Results

9.1 Sniffing / Fuzzing

9.1.1 Identifier

Basic No real solutions for bypassing the filters have been implemented. The detection filter looking for identical data with different identifiers can work perfectly here.

Detection	True positive	False positive	Attack frame
Specification signatures	6 141	5	6 141
Attack signatures	6 138	77	6 141

There are 5 false positives for the detection of manufacturer specifications because 5 malicious frames have been sent through the 5 legitimate traffic identifiers. This breaks the sending regularity of these identifiers. There are 2 detections for each irregularity: the distance between the current (malicious) frame and the previous (legitimate) frame, then at the next turn, the distance between the current (legitimate) frame and the previous (malicious) frame. 5 legitimate frames are therefore counted as malicious before the sending regularity takes place again.

Advanced Several solutions to bypass the filters have been implemented. By not fuzzing already used identifiers, the regularity of the cyclic sending of these frames is not broken. Constant speed sending aims to lower the unique identifier ratio of the last frames sent. Sending random data on different identifiers is also less suspicious than with the same data.

Detection	True positive	False positive	Attack frame
Specification signatures	306 037	16	306 300
Attack signatures	0	0	306 300

By not discarding already used identifiers:

Detection	True positive	False positive	Attack frame
Specification signatures	306 831	20	307 050
Attack signatures	250	117	307 050

9.1.2 Data

Since the data area can contain anything, it is difficult to judge that a frame is malicious without knowing the manufacturer's specifications. However, it is still possible to identify a fuzzing behavior that would look like a DOS attack.

On an unknown identifier No real attack detection solution has been implemented. None of the filters presented earlier can be effective on data fuzzing.

Detection	True positive	False positive	Attack frame
Specification signatures	65 025	0	65 025
Attack signatures	0	0	65 025

On an existing identifier The filter on the frame sending frequencies for the same identifier can be useful here.

Detection	True positive	False positive	Attack frame
Specification signatures	65 024	10	65 025
Attack signatures	432	7	65 025

Due to a very frequent and constant sending, we can only detect the beginning and the end of the fuzzing. Between the two, the speed is so fast that the legitimate frames disappear in the malicious traffic.

10 Conclusion

Without knowledge of the manufacturer's specifications, it is possible to detect anomalies related to specific attacks. We rely on generalities, such as the number of CAN identifiers used on a bus, which could appear to be spurious in some legitimate cases.

In our very restricted test environment, no false negatives were found during normal bus operation. All attacks that occur at least once on a legitimately used CAN identifier are reported as alerts without the need to use manufacturer's specifications.

Outlier detection algorithm training shows positive results. Some algorithms come as close as possible to legitimate data. Aside the first linear models, there are a significant number of false negatives as opposed to signature-based detection. The cost of computer security remaining low in a car, the computing power needed to train a model represents a big disadvantage compared to signature-based detections.

Whether by anomaly or by signature, a balanced detection rate can be found. However, the limit between legitimate and malicious frames will hardly be perfect.

Therefore, as we said before, implementing a system that blocks frames flagged as malicious is not really possible for user security. Unless this prevention system is applied on non-critical CAN buses. For the critical frames we are interested in, the implementation of an intrusion detection system will be useful as an informative tool for the different car actors.

Comparisons could be made with other automotive data buses such as Media Oriented Systems Transport (MOST), Local Interconnect Network (LIN) or FlexRay. But also on buses that look like or rely on the CAN bus such as the SAE J1939 or the Vehicle Area Network (VAN). An approach in other domains such as aviation or naval with NMEA 2000 can be considered.

11 References

1. N. Salman, M. Bresch, “Design and implementation of an intrusion detection system (IDS) for in-vehicle networks”, 2017
2. D. Paret, “Multiplexed Networks for Embedded Systems”, 2007
3. G. Dupont, J. Den Hartog, S. Etalle, A. Lekedis, “Network intrusion detection systems for in-vehicle network - Technical report”, 2019
4. Zhao, Y., Nasrullah, Z. and Li, Z., “PyOD: A Python Toolbox for Scalable Outlier Detection”, 2019
5. L. Arora, “An Awesome Tutorial to Learn Outlier Detection in Python using PyOD Library”, analyticsvidhya.com, 2019
6. Zhao, Y., Nasrullah, Z. and Li, Z., “Implemented Algorithms”, pyod.readthedocs.io, 2019
7. Garcia, F. D., Oswald, D., Kasper, T., and Pavlidès,P. “Lock it and still lose it - on the (in)security of automotive remote keyless entry systems”, 2016