Budapest University of Technology and Economics
Department of Networked Systems and Services

New Methods for Security and Privacy of CAN Bus Communication

Ph.D. Dissertation
of
András Gazdag

Supervisor: Levente Buttyán, PhD, DSc

www.crysys.hu

Budapest, Hungary
2023

Abstract

The last decades of the automotive industry have seen a significant change with the adoption of embedded controllers. Digital circuits and software components have taken control of processes previously controlled by analog methods. In addition to supporting more integrated functions and services, the primary motivation for this change was to reduce manufacturing costs. While delivering the expected results, this shift also created an undesirable problem: vehicles inherited the cybersecurity weaknesses of computers.

The emergence of cyber-physical systems (including vehicles) has brought new threats. If an attacker can take control of a computer-controlled physical process in an attack, it can cause physical damage by logical means. In the transportation industry, such an attack could endanger human lives or cause significant financial loss.

Fortunately, we are not yet aware any such attack happened in real life. However, the seriousness of the problem is illustrated by the millions of vehicles that manufacturers have recalled on several occasions to repair vulnerabilities found by security experts at considerable cost. The importance of the problem is further shown by the fact that several new regulations and standards, such as ISO/SAE 21434:2021 or UN Regulation No. 155, have been introduced in the European Union and elsewhere in the world to make cyber security a priority for new vehicles. Under UN Regulation No. 155, from 2024, only new vehicle types that meet cybersecurity requirements will be type-approved.

Researchers have also been working for years to address emerging cybersecurity issues. In this dissertation, we also address some of these challenges. We examine the entire cybersecurity problem set from several perspectives. We have conducted research to prevent and detect attacks and worked on testing new defensive solutions. In addition to security threats, we have also worked on the privacy issues of vehicular data release.

A specific feature of cyber attacks is that there can be a significant time lag between the execution of an attack and its discovery or the signs of the damage it causes. With this in mind, the first area where we have achieved new results relates to recording data on the internal network of vehicles. We propose a new compression algorithm to facilitate efficient data storage over a long period, thus allowing its analysis long after the attack. Our proposed method achieves significantly better results than widely used alternative compression methods.

We have also developed new attack detection methods. First, we show that the previously mentioned compression algorithm is suitable for detecting message injection attacks. This result further enhances the value of the compression method, as it allows us to identify a subset of attacks on compressed data, saving significant computational resources. Our following proposed detection method exploits correlations between transmitted signals and proves that attacks against significant signals can be efficiently detected using models built on correlations. Finally, we propose a third detection method that can be applied to signals individually. The latter method uses machine learning to predict for each signal an expected value. Measured values can be continuously compared to their prediction to identify any unexpected change caused by an attack.

The dissertation's final chapter focuses on the privacy issues of vehicular data release. We show that the data transmitted over the internal network can also be used to reconstruct the movement of vehicles for short and long distances. This result supports the assumption that the captured data can only be used with due care in order to avoid legal and ethical problems.

Kivonat

A járműipar utóbbi évtizedeiben jelentős változást hozott a beágyazott vezérlők elterjedése. A korábban analóg módszerekkel vezérelt folyamatok felett digitális logikák, és szoftver komponensek vették át az irányítást. A változás fő motívációja, az össztettebb funkciók és szolgáltatások támogatása mellett a gyártási költségek csökkentése volt. Ez az átállás bár meghozta a várt eredményeket, egy nem kívánatos probléma megjelenését is okozta: a járművek megörökölték a számítógépek gyengeségeit is.

A kiber-fizikai rendszerek (amelyek közé a járművek is sorolhatóak) új veszélyek megjelenésével jártak. Amennyiben egy támadás során egy számítógéppel vezérelt folyamat felett át tudja venni a támadó az irányítás, akkor azzal a valós életben okozhat már kárt. A közlekedés területén egy ilyen támadás könnyen emberéleteket veszélyeztethet, vagy jelentős anyagi kárt okozhat.

Szerencsére, egyenlőre nincs tudomásunk arról, hogy ilyen támadás történt volna, azonban a probléma súlyosságát jól mutatja, hogy a szakértők által talált sérülékenységek javítása miatt több alkalommal is járművek millióit kellett már gyártóknak visszahívniuk, ezzel jelentős költségeket vállalva. A problémát jól illusztrálja az is, hogy az Európai Unióban és világ egyéb területein is több új szabályozás jelent meg az elmúlt években, amelyek célja, hogy az újonnan tervezett gépjárművekben a kiberbiztonságra is kiemelt területként kelljen figyelni. Az ENSZ 155.-ös számú előírása alapján 2024-től csak azok az új járműtípusok kaphatnak típusengedélyt, amelyek a kiberbiztonsági előírásoknak is megfelelnek.

A felmerülő kiberbiztonsági problémák megoldásán a kutatók is évek óta dolgoznak. Ebben a disszertációban is a kihívások egy részére adunk megoldást. A kiberbiztonság javításáért a teljes problémakört több szempontból is vizsgáljuk. Végeztünk kutatást, amely célja a támadások megállítása, illetve felderítése, valamint dolgoztunk azon is, hogy a védelmi megoldásokat jobban lehessen tesztelni. A rendszerek támadásokkal kapcsolatos vizsgálatán túl a privátszféra védelmében felmerülő kérdéseket is vizsgáltuk.

A kibertámadások sajátossága, hogy egy támadás végrehajtása, és annak a felfedezése, vagy az az által okozott kár megjelenése között jelentős idő is eltelhet. Ezt a szempontot figyelembe véve, az első terület, ahol új eredményt értünk el, a járművek belső hálózatának adatrögzítéséhez kapcsolódik. Javasoltunk egy új tömörítési eljárást, amely elősegíti az adatok hosszútávú hatékony tárolását, ezzel lehetővé téve egy esetleges támadás után hosszú idő elteltével is az elemzését. Megmutattuk, hogy a javasolt eljárásunk jelentősen jobb eredményt ér el, mint a széles körben elterjedt alternatív tömörítési eljárások. A módszer alkalmazása számos előnnyel jár az adatok helyben tárolása, vagy távoli szerverre feltöltése szempontjából is.

A támadások felismerése területén is értünk el új eredményeket. Először a korábban ismertetett tömörítési eljárásról mutatjuk be, hogy üzenetbeszúrásos támadások detektálására alkalmas. Ez az eredmény tovább növeli a tömörítési eljárás értékét, mivel így a támadások egy részét már a tömörített adatokon is tudjuk vizsgálni, ezzel jelentős erőforrásokat spórolva. A következő javasolt detekciós módszerünk az átvitt jelek közötti korrelációt használja ki, és bebizonyítja, hogy a legfontosabb jelek elleni támadások a korrelációkra épített modellek segítségével hatékonyan detektálhatóak. Végezetül, egy harmadik detekciós eljárást is javaslunk, amely a jelekre egyesével alkalmazható. A módszer gépi tanulás segítségével minden jelre képes előrejelezni, hogy várhatóan milyen értéket kell érzékelnünk, így ha egy tamadás hatására egy jel nem várt módon változna, az ezzel a technikával detektálhatóvá válik.

A disszertációban vizsgált utolsó terület a jármű adatok érzékenységét vizsgálja a személyes adatok biztonsága szempontjából. Megmutatjuk, hogy a hálózaton átküldött adatok segítségével a járművek mozgása rövid, illetve hosszabb távon is rekonstruálható. Ez az eredmény alátámasztja azt a feltételezést, hogy a rögzített adatok csak megfelelő körültekintéssel használhatók fel jogi és etikai problémák nélkül.

Dedicated to my Grandfather.
Who showed me the beauty of engineering.

Acknowledgement

This thesis would not have seen daylight without the help and support of many people, for which I cannot be grateful enough.

First of all, I would like to express my deep and sincere gratitude to my supervisor, Professor Levente Buttyán, whose unquestionable dedication to quality has not only had an impact on me but on every member of our Lab.

I am thankful to Árpád Török and Zhendong Ma, the reviewers of this thesis, for the time and effort that they have invested in judging my work. Their questions and comments helped me a lot to improve this thesis.

I want to thank all my past and present colleagues in the CrySyS Lab not only for the pleasant atmosphere but also for that I could learn something from all of them. I am happy to have had the opportunity to attend Márk Félegyházi's class, which grabbed my attention from the very first moment and led to me getting to know the Lab. Many thanks also go to the infra team, which I'm happy to be a part of. During the countless hours of debugging, I have learned a lot.

I am grateful to all the colleagues with whom I could collaborate on papers or research projects, in particular, Tamás Holczer, Gergely Biczók, Gergely Ács, Szilvia Lestyán, Mina Remeli, Irina Chiscop, Joost Bosman, Rudolf Ferenc, and Zsolt Szalay.

I look back with pleasure on many of my former students. Many excellent quality theses and research papers were written with them during our years together, from which not only they but I, too, learned a lot. I am particularly grateful to those with whom I have co-authored scientific papers: Dóra Neubrandt, Csongor Ferenczi, György Lupták, and Ákos Boros.

I also owe a special thanks to Beatrix Koltai. The discussions with her have made me a better lecturer, supervisor, and researcher. I cannot wait to see her succeed in the future. I hope I will be able to give her the support I have received.

I am thankful to my colleagues at the Department of Automotive Technologies for allowing us to use their test vehicles and vehicle simulators for research purposes. I can finally say with relief that no cars were harmed during our research.

Last but definitely not least, I am grateful to my family. To my Mom, who has supported me with unwavering faith in achieving my goals since the day I was born. And to my Grandma, whose many hours spent with me on the tennis court were not in vain. Without the discipline and perseverance I learned there, I certainly would not have made it this far.

And...Anett, I am grateful for your love and for standing by me through whatever life brought. I hope you know that since splitting *that* KitKat, you have become an inseparable part of my future.

Finally, I would also like to acknowledge the financial support of the following research agencies and grants for their kind support that allowed me to focus on my work:

- National Research, Development and Innovation Office (NKFIH) of Hungary [1,2,3,4,5,6]
- ECSEL[7]
- Innovative Mobility Program of KTI[8]

[1]The research presented here have been supported by the NRDI Office, Ministry of Innovation and Technology, Hungary, within the framework of the Artificial Intelligence National Laboratory Program, and the NRDI Fund based on the charter of bolster issued by the NRDI Office.

[2]The research presented here have been supported by the NRDI Office, Ministry of Innovation and Technology, Hungary, within the framework of the Autonomous Systems National Laboratory Program, and the NRDI Fund based on the charter of bolster issued by the NRDI Office.

[3]The work presented here was partially supported from the grant GINOP-2.1.1-15. The project has been supported by the European Union, co-financed by the European Social Fund. EFOP-3.6.2-16-2017-00002.

[4]Project no. 138903 has been implemented with the support provided by the Ministry of Innovation and Technology from the NRDI Fund, financed under the FK_21 funding scheme.

[5]Project no. 2019-1.3.1-KK-2019-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme.

[6]Project no. 2018-1.2.1-NKP-2018-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

[7]This work has been funded by the European Commission via the H2020-ECSEL-2017 project SECREDAS (Grant Agreement no. 783119).

[8]The presented work was carried out within the MASPOV Project (KTI KVIG 4-1 2021), which has been implemented with support provided by the Government of Hungary in the context of the Innovative Mobility Program of KTI.

Contents

List of Figures

List of Tables

Introduction

The automotive industry has undergone several significant changes over the past 150 years. The first major innovation was the advent of the internal combustion engine. Nicolaus Otto's invention in 1876 fundamentally changed how cars were powered and thus their potential performance. From the introduction of the Ford Model T in 1908, there was also a steady increase in production, eventually leading to the vehicle industry becoming a driving force in modern economies. As the twentieth century progressed, cars became increasingly sophisticated, with the introduction of features such as automatic gearboxes (1939), air conditioning (1940), and electric fuel injection (1966). As the number of vehicles and the speed of transportation increased, so did the need for safety. The first seatbelt (1968) and the first airbag (1970) were signs of this.[9] These technological changes have always had a positive impact on society in the long term, but in the early stages, new technologies also brought new problems.[10]

Since the 1990s, Electronic Control Units (ECUs) connected together with a Controller Area Netword (CAN) have become common in cars. Embedded controllers and software running on them took control of processes previously controlled by analog control functions. This change was complemented in the 2000s by adding various smart functions to vehicles that required an internet connection. Overall, industry evolutions have led to a situation where modern cars are now better viewed as a network of computers on wheels. Indeed, a modern vehicle now has at least 50 but frequently far beyond 100 embedded controllers, and the amount of software running on these controllers is typically more than 100 million lines[11].

This latest change has also had negative side effects like the previous ones. The potential threats were first highlighted by Koscher et al. Their work [KCR+10] contributed to the emergence of a whole new field of research. They reviewed the communication interfaces and protocols used in vehicles and examined their level of exposure to external attacks. Their work was initially criticized on the grounds that the threats they raised were not a real problem, as they required physical access; thus, the physical protections built into cars would also prevent cyber attacks. In their subsequent paper, however, the authors showed that this no longer holds in practice. Remote access connections built into vehicles have rendered vulnerabilities remotely exploitable.

[9]https://www.britannica.com/technology/automotive-industry/Growth-in-Europe (Last accessed: Oct 1, 2023)

[10]https://www.qad.com/blog/2019/12/the-biggest-milestones-in-the-history-of-automotive-manufacturing (Last accessed: Oct 1, 2023)

[11]https://medium.com/next-level-german-engineering/porsche-future-of-code-526eb3de3bbe (Last accessed: Oct 1, 2023)

In addition to the researchers' attention, society became aware of the problem through the work of Miller and Valasek [MV15]. They have demonstrated that cars are at real risk by exploiting actual vulnerabilities. First, their research enabled them to take control of simple systems such as the multimedia system and air conditioning. Then, they demonstrated that controls critical to the operation of vehicles, such as the engine control unit, are also vulnerable to similar attacks. The research results were made available to the broader audience via an article in Wired magazine.[12]

When it comes to manufacturers, they put a great emphasis on the safety of the cars. There are ABS, ESP, crash avoidance systems, or even better and better crumple zones in every car. The inter-component communication channels are protected against errors caused by the high noise environment, but security seems to be lagging behind these other fields. Even today, the design process of internal components may still fail to apply basic cybersecurity principles to make them secure against a potential attacker, e.g., it is not unthinkable that a malicious device could be installed into a car and do some harm. It only takes one mechanic to plug such a device into an On-Board Diagnostics (OBD) port of a targeted car, to have access to the vehicle's CAN bus, should the network architecture allow such communication. Moreover, with the spread of Bluetooth OBD debugging probes, which can connect to the owner's smartphone, users themselves connect more and more, potentially compromised, devices onto the CAN bus. The statistics illustrate the situation: according to Upstream Security's 2023 report, 40% of cyber attacks against the automotive industry targeted vehicles' internal systems, and 60% targeted transport-related infrastructure. 20% of attacks on internal systems targeted infotainment systems, 35% ECUs, and 45% the remote keyless entry subsystems[13].

In this dissertation, we focus on the security issues of CAN networks. Since security was not considered during the initial design, the aim is to identify and at least partially solve the missing features. As a first step, we recognized that a retrospective communication data analysis is necessary for future cyber attack analysis, meaning the transmitted messages must be stored. The long-term storage of data can cause serious capacity problems. To remedy this situation, we propose a lossless compression technique that can store the information contained in the original data in a significantly smaller space.

We then show that the compression method is suitable for identifying certain attacks. The compression exploits the specific patterns of communication that can change during an attack, thus allowing the identification of attacks without the need to decompress the data first.

Beyond simple attacks, we also propose solutions for detecting more complex attacks. Two detection methods are proposed to model the characteristics of the internal vehicle processes. The first can identify anomalies based on the changes in the measured correlation between signals. The second models the characteristics of signals and predicts them, which then can compared to new values. Both methods allow the identification of deliberate signal modifications.

The last problem we have investigated is the privacy implications of CAN data release. New services might build on data gathered from vehicles. For example, by monitoring the driver's style, an insurance company can better predict the probability of an accident, thus providing a cheaper service for the more cautious drivers. However, this approach may also come with an

[12]https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway (Last accessed: Oct 1, 2023)

[13]https://upstream.auto/reports/global-automotive-cybersecurity-report (Last accessed: Oct 1, 2023)

additional privacy cost. Using extracted data from the communication, we show that it is possible to track the movement of vehicles in the short and long term. Short-term tracking helps restore the circumstances of an accident; e.g., this method allows one to show the car's movement in the last 100 meters. Building on short-term tracking results, our following algorithm implements long-distance tracking. By using map data, the inaccuracy of the method can be eliminated, enabling route reconstruction for over several kilometers, e.g., during a cross-town drive.

This dissertation is structured as follows. Chapter 1 discusses the CAN protocol. It describes the physical design as well as and the logical protocol. In Chapter 2, we introduce the attacker model. We give a description of the possible attack approaches and explain our large dataset created for the evaluation of defense mechanisms. Chapter 3 explains our proposed compression algorithm, to allow efficient long term storage of CAN logs for analysis. Chapter 4 contains our three newly proposed detection algorithms to identify different attack scenarios. In Chapter 5 we evaluate the privacy sensitivity of CAN data release. We show that it is possible to track a vehicle's movement in shorter and longer sections. Finally, in the last chapter, we summarize this dissertation and review the presented results.

Chapter 1

CAN bus

The CAN bus [ISO15, ISO16] is a broadcast, serial communication protocol widely used in vehicles. It was designed to be robust, withstand high external RF noise, while providing a high-speed communication link between the ECUs. The CAN bus is a cost-effective solution, because it enables the manufacturers to connect the ECUs to each other by connecting them to the central bus with only a twisted pair cable .

1.1 Physical properties

The CAN bus uses two wires called CAN high (CAN_H) and CAN low (CAN_L), in order to implement differential signalling. On each end of the cables, the wires are connected to each other using a terminating resistor in order to achieve a nominal 120 Ohm impedance. The CAN bus has two states, driven, and not driven. When it is not driven, the CAN_H and CAN_L wires get pulled to about the same 2.5V nominal voltage using the passive pull resistors placed in the CAN transceivers. This state is also called a "recessive" bit, which represents a logical binary 1. When the CAN bus is driven, at least one CAN node pulls the CAN_H wire to 3.5V and CAN_L wire to 1.5V nominal voltage. This state is also called a "dominant" bit, which represents a logical binary 0.

Bit Timing

Every CAN bus has a nominal bitrate, which gets preconfigured in the ECUs by the car manufacturer. The maximum bitrate specified by the standard is 1 Mbps, but 500 kbps and 250 kbps is also frequently used bitrates.

Every bit time can be divided into the following four segments (Figure 1.1):

- *synchronization segment (Sync_Seg):* This segment is used for synchronization. At the SOF (Start of Frame), every receiving ECU synchronizes itself to the edge of the first bit, which is called a "hard sync". There is also another kind of synchronization called bit resynchronization, which is performed at the synchronization segment of each bit, by the ECU fine-tuning its inner clock based on the deviance between the expected time of a potential edge, and the actual time of its detection.

- *propagation time segment (Prop_Seg):* This segment is used to compensate for physical delay times within the network (e.g., signal propagation time, internal delay of the ECUs, etc.)
- *phase buffer segment 1 (Phase_Seg1) and Phase buffer segment 2 (Phase_Seg2):* These segments are used for edge phase error compensation. The sampling of the bus occurs after the phase buffer segment 1. The length of these segments can be fine-tuned by resynchronization, and thus, the sampling point can be moved backward or forward.

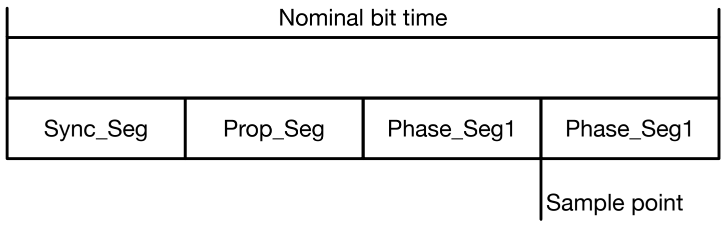


Figure 1.1: The four segments of the bit time.

Frame timing

Messages can be transmitted based on events (e.g., receiving a remote frame, a sensor value triggering a transmission) or by a trigger coming from an internal timer. The internal timer method is called Time-Triggered Communication (TTC). During communication, after every frame's EOF (End of Frame) segment, there is a 3-bit long intermission period. After these 3 recessive bits, the bus is considered idle, and any following dominant is considered as the SOF of the next frame. When the ECUs detects that the bus is idle, any of them may start to transmit. If two or more ECUs happen to transmit at nearly the same time, the conflict between them is resolved using contention-based arbitration.

ECUs communicate with each other mostly periodically using the TTC method. Regular repetition times enable a quite accurate prediction of the upcoming pattern of messages. As message transmission happen typically more often then changes in the vehicle state the content of messages is often repeating.

Contention-based arbitration

When a node starts to transmit a frame, it monitors the bus during the arbitration segment of the MAC frame in order to check whether the data on the bus is the same as it is transmitting. In case it detects that despite transmitting a recessive bit, the bus is still in dominant state, it knows that another ECU tries to transmit data, and it terminates the arbitration process and turns into a receiver. This allows the other ECU to transmit its message as nothing happened, and the other

ECU that lost the arbitration, can retry sending its message at a later time. Using this method requires the bus has to adhere to three key elements:

- The ID of the message types has to be unique.
- A data frame with a given ID and a non-zero Data Length Code (DLC) value may only be sent by one ECU.
- The remote frame's DLC value has to be the same as the data frame it requests.

Using this contention-based arbitration ensures that the ECU with the higher priority frame will always win the arbitration, because its ID contains more dominant bits, than the other potential transmitter ECU frame's ID.

1.2 Traffic capture

The broadcast nature of the CAN bus allows easy traffic captures. Connecting to the bus at any point gives access to all the messages transmitted in that segment. Modern vehicles typically have multiple busses with different speeds installed to provide communication for all ECUs.

In a vehicle with multiple CAN segments, traffic capture has to be performed separately in each segment, as the gateway connecting the different segments typically does not relay all packets between segments. A segmented CAN architecture can limit the effectiveness of CAN packet capture through the OBD port. In our test vehicles, we verified that only one CAN segment exists. Therefore, we were able to capture the entirety of the communication.

In our work, we captured and transmitted CAN packets with a Raspberry Pi based recorder. Using the PiCAN2[1] board, we were able to handle messages up to 1 Mbps speeds, the maximal transmission speed of the CAN bus. We verified with measurements, using commercial tools, that our device processes every CAN frame without packet loss.

We captured traffic in multiple formats supported by the Linux-CAN package[2]. Although some details differ in the text based formats, the contents are essentially the same. Every message has an ID which can be 11 or 29 bits long. The meaning and the range of the IDs are manufacturer specific. The lower the value of the identifier field the more prior is the message. After the ID comes the data length field then comes the data.

In Example 1.1 a CAN traffic log is shown. Each row corresponds to a message. The first column is the arrival time of the message in a Unix timestamp (not part of the CAN messages, only added during the capture process), the second column is the message ID, the third column shows the length of the data in the message, and the last column is the data.

[1] https://www.skpang.co.uk/collections/hats/products/pican2-can-bus-board-for-raspberry-pi-2-3 (Last accessed: Oct 1, 2023)

[2] https://github.com/linux-can/can-utils (Last accessed: Oct 1, 2023)

```
1481492674.734327  0x260  8  00 00 00 00 00 00 00 6a
1481492674.736055  0x2c4  8  05 c8 00 0f 00 00 92 3c
1481492674.738092  0x2c1  8  08 03 35 01 6a d9 00 4f
1481492674.754306  0x260  8  00 00 00 00 00 00 00 6a
1481492674.759605  0x2c4  8  05 c8 00 0f 00 00 92 3c
1481492674.769823  0x2c1  8  08 03 39 01 70 d9 00 59
1481492674.774302  0x260  8  00 00 00 00 00 00 00 6a
1481492674.783129  0x2c4  8  05 c2 00 0f 00 00 92 36
1481492674.794246  0x260  8  00 00 00 00 00 00 00 6a
1481492674.801541  0x2c1  8  08 03 3b 01 74 d9 00 5f
```

Example 1.1: Simplified CAN traffic log

1.3 Signals

ECUs continuously measure and transmit data of the vehicle's physical processes. The data field of CAN packets contain the signal values in an encoded format. However, the CAN matrix, which describes the correspondence between the data bits and the signals is not published by the manufacturers. Throughout this dissertation, we used a method proposed in [NGMK18] to identify signals and the boundaries. In our research, we handled these signals without knowing all of their exact meaning. An example plot of some decoded CAN signals can be seen in Figure 1.2.

In the industry, the CAN matrix is typically stored in a dedicated file format called CAN DBC files. These files facilitate the interpretation of data transmitted over a CAN bus. This standardized ASCII-based file format emerged in the 1990s and has since become ubiquitous across the automotive industry globally. These textual databases contain essential details for converting raw CAN bus data into tangible physical values, effectively serving as a repository of signals. However, as mentioned previously, the CAN matrix and the DBC files are vendor-protected secrets. Therefore, they are not publicly available.

1.4 Security shortcomings

Unfortunately, the basic CAN bus has no security, only safety measures [SDK19]. It was originally designed to be a robust communication bus that can withstand a high amount of noise while providing relatively high transfer speeds. At the end of every CAN message, a Cyclic Redundancy Check (CRC) code ensures that if the content of the message changes during transmission, the receiver will detect it. The CAN standard does not provide support for message authentication. Thus, just by receiving a message with a given ID does not guarantee that the source of the message was trustworthy. This leads to the issue that messages can be injected onto the bus without the communication partners ever noticing that a message was not sent by the correct device and could contain false data.

The CAN protocol was designed to be simple, causing only a small overhead in the communication. Therefore, the protocol generally lacks security properties rendering it defenseless against many attacks[MV13, MV14, MV15]. Some of the important properties and its consequences are the following:

- CAN frames do not contain sender or receiver fields and in general lack any authentication. This enables message spoofing.
- Access to the CAN bus is not restricted allowing eavesdropping or injection of messages.
- The arbitration process is not protected against misuse easily allowing the exploitation of the message priority for Denial of Service attacks.

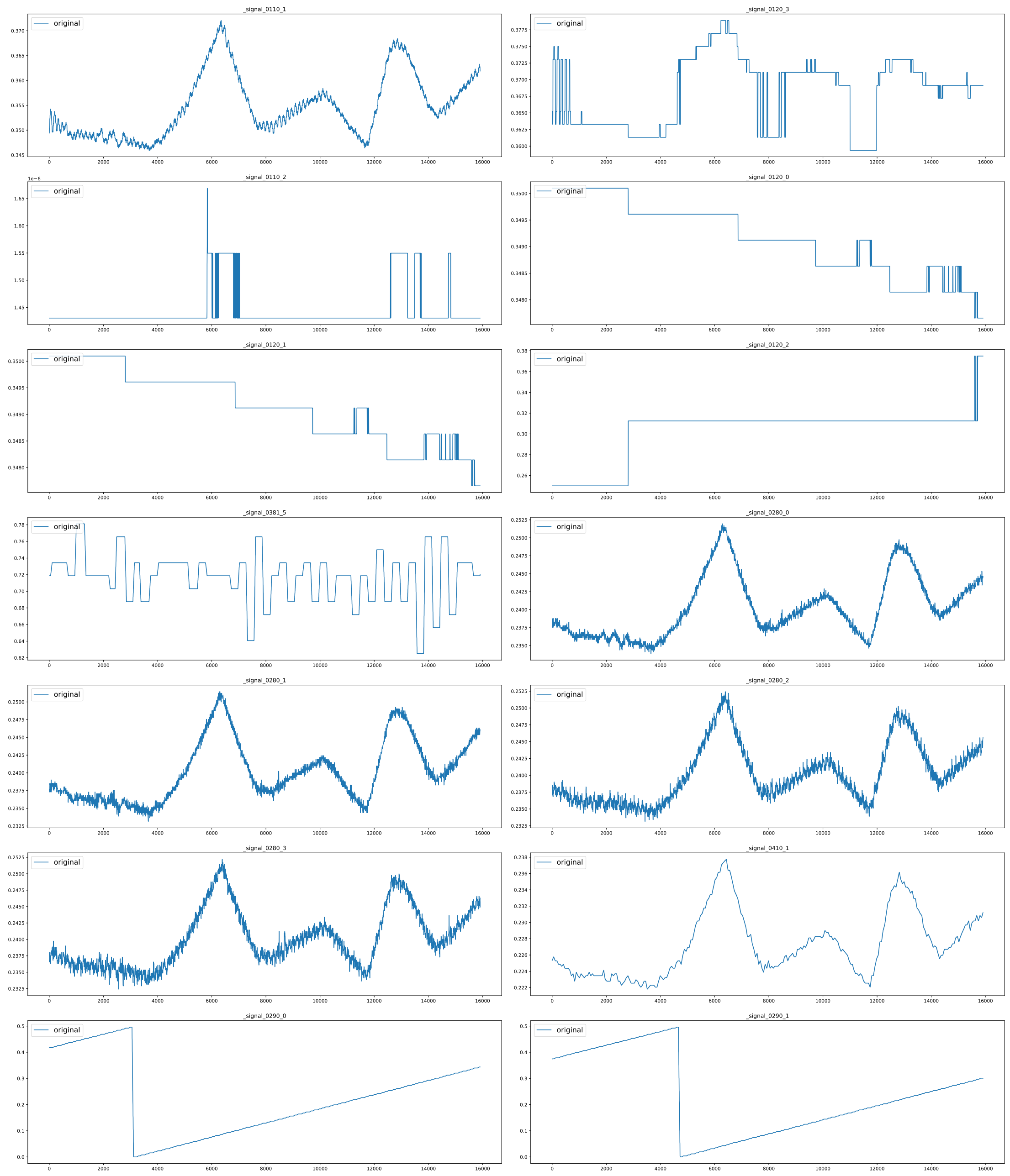


Figure 1.2: Signals encoded in a CAN communication.

Chapter 2

Attacks against the CAN bus

In this Chapter, we present the potential attacks against the CAN bus. We summarize previous results from the literature, and also demonstrate our capability to execute previously proposed and new attacks. This is an important prerequisite for our detection results in Chapter 4. By performing the attacks in practice, we demonstrate their executability and also capture a large amount of data.

During our research, we kept extending our datasets. This is the reason why our results are always tested on different data. Eventually, we have decided that our dataset can be considered sufficiently rich, as it contains enough benign and malicious traces to thoroughly test any anomaly detection algorithm (see Section 2.4).

2.1 Denial of Service attack

In a Denial of Service (DoS) attack, the attacker's goal is to render the CAN bus unusable, which can be achieved in two ways: by adhering to the CAN standard or by breaking it.

The first option is to send as many messages to the CAN bus with the lowest possible ID as physically possible. When the bus is idle, if two or more ECUs want to transmit at the same time, the one with the lowest ID will have priority. Thus, since the zero ID has priority over every other message ID, none of the regular messages will win the arbitration against the injected message, which will lead to the starvation of the regular ECUs.

The second method is to force the CAN_H and CAN_L wires into dominant state and hold it there as long as the attacker wants to. While the second method could be easier to implement, it triggers the error detection in the ECUs; thus, the connected subsystem will detect that there is an error with the CAN bus. On the other hand, the first solution does adhere to the rules of the CAN, and the ECUs might only think that everything is okay with the CAN bus, except it is busy at the moment. Nonetheless, using both solutions, an attacker can render a given CAN bus unusable.

2.2 Message injection attack

Message injection (or Fabrication) attacks [KCR+10] are possible due to various properties of the CAN bus. First of all, since the CAN bus is a broadcast channel, during operation, every ECU connected to the CAN bus receives all of the messages. This cost-saving measure saves a lot of money for the manufacturers, since they only have to wire a few twisted pair cables throughout the whole car in order to connect the ECUs to each other. When receiving a message, the ECUs decide whether they are interested in the given message based on its ID and process it or discard it. On the other hand, this principle also makes it easy for an attacker to eavesdrop messages on the bus, monitor the values of the sensors in real-time, or reverse engineer the messages and their purposes for a given vehicle type.

The second issue with the standard, also contributing to the possibility of message injection attacks, is that the basic CAN protocol does not include any kind of cryptographic message authentication measures by default, only optional extensions detailed in Section 4.1. Any ECU can create a message with an arbitrary ID and send it to the other ECUs via the CAN bus, and the ECUs will not be able to differentiate on the protocol level the messages coming from two different ECUs, if they have the same ID. This makes an attacker able to craft arbitrary messages and send them to the ECUs via the CAN bus. An attacker can have multiple goals to exploit these properties, for instance, overwriting values or forcing the network into a Denial of Service attack state.

Message injection attacks have several drawbacks. First of all, both the rapidly changing value behaviour and the at least doubled message periodicity is easily detectable, as it will be demonstrated in Section 4.2.3. Upon detection, the targeted ECU might switch into a fallback mode, where it ignores both the original and the injected values. Secondly, the ECU might have safety margins built-in. For instance, most, if not all of the lane assist systems have a maximum steering angle it is allowed to use in order to keep the car in the lane. If the attacker tries to induce a higher steering angle than this maximum value, the lane assist system might just deactivate.

The technical execution of a message injection attack is easy. Simply, by connecting a new device to the bus, the attacker can send messages. We performed a number of different message injection attacks during our dataset generation, which is described in Section 2.4.

2.3 Message modification attacks

Besides injecting messages, an attacker could also modify the content of the message, but achieving this is much harder. There are several safety measures in the CAN standard that makes it hard if not impossible to modify messages transmitted by another ECU on-the-fly (e.g., CRC, bit stuffing, sender ECU monitoring the bus during transmission, etc.). With the current techniques, there are two solutions to this problem. The first option is to compromise an ECU, and to send messages with modified values with malicious code. The second option is to first force an ECU into error mode, and then take its place in the communication[CS16]. Either way, this attack produces no extra messages, nor highly deviating values; thus, it is much harder to detect than simple message injection attacks. Until now, such a modification attack required a more in-depth knowledge[MV14] to carry out than just plugging in a device to the OBD port.

In this section, we present a new Men-In-The-Middle (MitM) solution to modify CAN messages in real-time, without compromising any ECU. We have created a device, which is capable of modifying ISO 11898 high-speed CAN messages in real-time. It can handle up to 100% bus load with a bus speed of 500 kbps or less, and about 60-100% busload at 1 Mbps, which is the maximum speed specified in the standard. It introduces a delay of 260 μs, which is even at 1 Mbps is well within the delay of re-sending a message due to high traffic or a transient error on the bus. The device itself provides a wireless interface that can be used to remotely configure the attack parameters. All this, while consisting of cheap, commonly available parts.

2.3.1 Malicious CAN gateway

Since the CAN standard does not support message authentication, if we manage to insert a special device in the communication line between the two ECUs, then we can add, modify, or even delete messages with an arbitrary ID. The visualization of the attack can be seen in Figure 2.1.

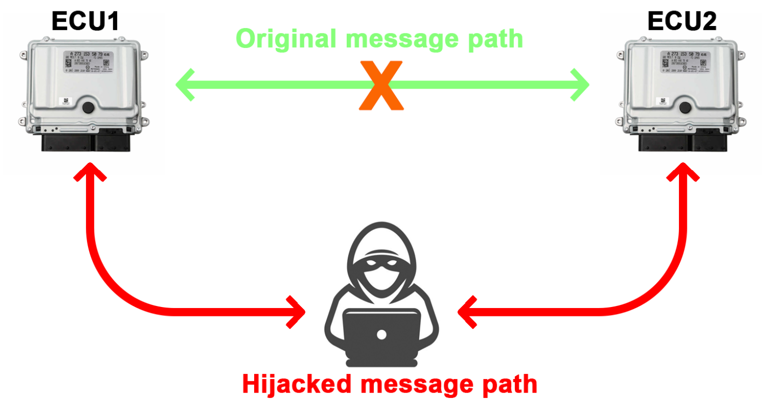


Figure 2.1: The concept of a MitM attack in the automotive industry.

Design

In order to create a CAN gateway, we used NodeMCU ESP32s microcontrollers. ESP32 is a two-core, 240MHz versatile microcontroller, with a built-in CAN controller and Wi-Fi module. It comes with the Espressif IoT Development Framework (ESP-IDF), which is based on the popular Free Real-Time OS (FreeRTOS) platform. Using the ESP-IDF allows us to create a hard-real-time device using its built-in scheduler, while providing convenient features, for instance, a built-in web server, which is useful for device configuration.

There were several requirements for the proof-of-concept malicious CAN gateway:

- It shall be able to handle ISO 11898 High-Speed CAN messages, with any bus speed.
- It shall have as low introduced delay as possible, preferably low enough that it is not significantly greater than a lost arbitration or a transient error.

- Despite being a proof-of-concept device:
  - It shall be robust enough to be used later at least for research.
  - It shall be easy to configure during testing, and configuration shall not require reprogramming of the device.
  - It shall be as serviceable as possible, without requiring specific hardware knowledge. The components shall be replaceable in case it is required.

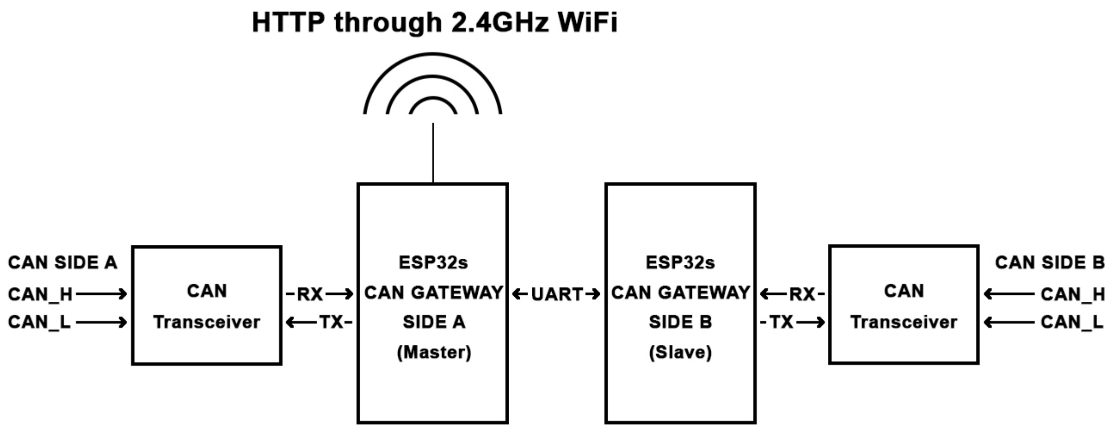


Figure 2.2: High-level architecture of the CAN gateway.

The first step of the design process was to create a high-level architecture of the CAN gateway, which can be seen in Figure 2.2. The CAN gateway consists of two CAN transceivers, and two ESP32s microcontrollers. One of the microcontrollers is the master, and the other one is the slave. Each microcontroller handles one side of the CAN bus via its built-in CAN controller, and the corresponding CAN transceiver. The communication between the microcontrollers is realized via a UART line. The master provides a web interface for configuration through its 2.4GHz Wi-Fi module, working as an access point.

Implementation

After designing the schematics and verifying it on a breadboard, we have built a soldered version of the circuit. We used a protoboard as the base of the device, which allowed us to apply minor changes to the hardware design without having to rebuild the complete circuit again. The components have been soldered onto the protoboard using sockets in order to make a potential component replacement easier to achieve. A picture of the finished proof-of-concept CAN gateway board can be seen on Figure 2.3.

Operation

After powering up the device, it provides a web interface for configuration through its Wi-Fi access point, as we mentioned before, which is a key element in making the device usable for

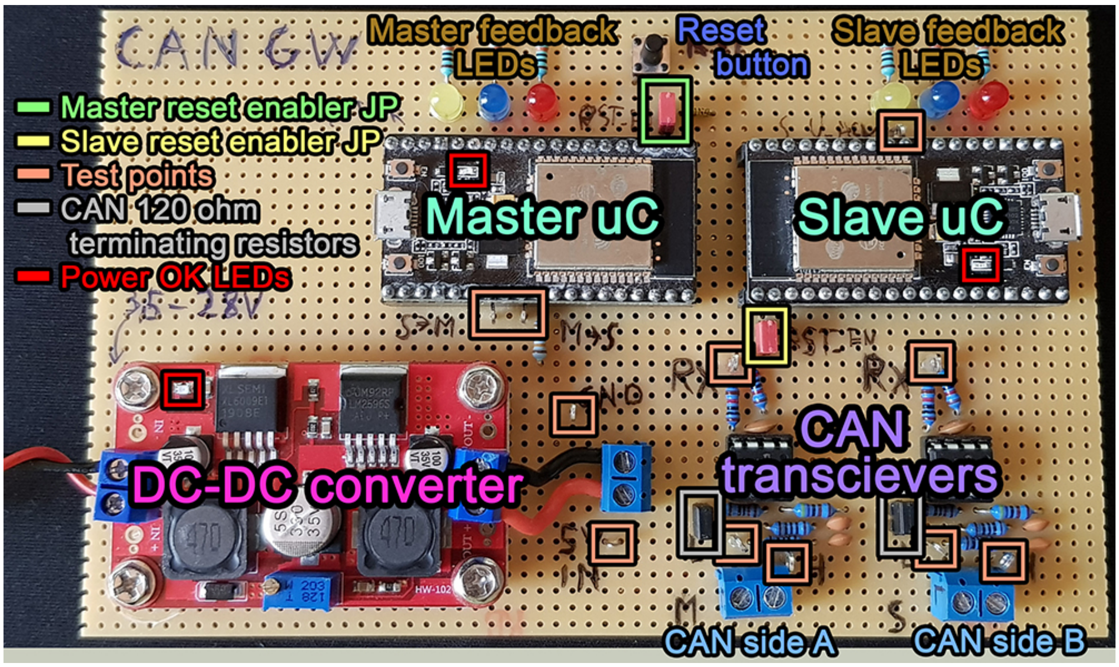


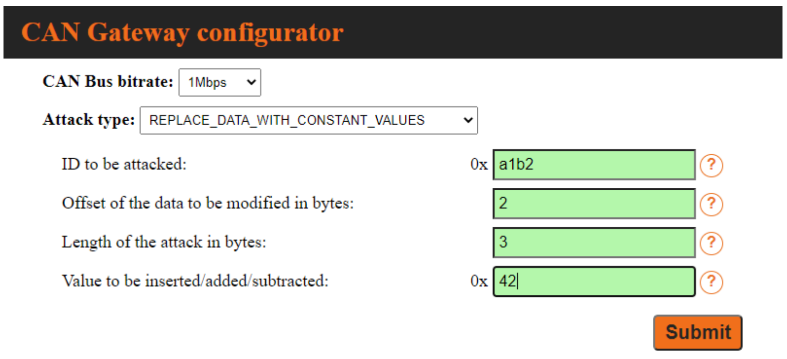
Figure 2.3: The top view of the proof-of-concept CAN gateway board.

research purposes. This web interface allows the user to configure it using a simple web browser, or via direct POST request (e.g., using curl). The graphical web configuration interface of the device can be seen on Figure 2.4. The configurable parameters are the following:

- *Bitrate:* This parameter can set the bitrate of the CAN bus.
- *Id:* The id of the CAN message to be attacked.
- *Offset and AttackLength:* The offset controls the position of the first byte to be attacked, and the attackLength determines how many bytes will be attacked.
- *ByteValue:* Some of the attack types require an additional parameter, which will replace the original or will be added or subtracted from the selected byte values.
- *AttackType:* There are several different attacks the device can perform:
  - `Passthrough`: In this mode, the device relays the traffic without modifying any of the messages.
  - `Replace-data-with-constant-values`: In this mode, the device replaces the selected bytes in the message with the given ByteValue parameter.
  - `Replace-data-with-random-values`: In this mode, the device generates random bytes for each of the selected bytes in the message, and replaces them.

- `Add-delta-value-to-data`: In this mode, we add the given ByteValue parameter to each of the selected bytes in the message. In case the resulting values would overflow, it gets capped at the maximum 255 value.
- `Subtract-delta-value-from-data`: Similar to the previous attack type, but the byte ByteValue is subtracted instead of added. In case the resulting value would underflow, it gets bounded at the minimum 0 value.
- `Increase-data-until-max-value`: In this mode, we take the lowest value from the selected bytes, increase it by one and replace all of the selected bytes if the increased byte is higher than the original. This is repeated until the max value (0xff) is reached.
- `Decrease-data-until-min-value`: This attack type is similar to the previous one, but at the start, we take the highest value from the selected bytes and decrease it every message, until we reach 0x00.
- `Replace-data-with-increasing-counter`: We start a counter from 0 and increase it by one at every occurrence of the message. The selected bytes get replaced with the counter. The counter can overflow.
- `Replace-data-with-decreasing-counter`: Similar to the previous attack type, but the selected bytes get replaced by a decreasing counter starting from 255, which can underflow.

After the user sends the configuration via the graphical web interface or via a direct POST request, the device validates the configuration on the server-side, and if it is correct, it starts the attack phase.



CAN Bus bitrate:	1Mbps
Attack type:	REPLACE_DATA_WITH_CONSTANT_VALUES
ID to be attacked:	0x a1b2
Offset of the data to be modified in bytes:	2
Length of the attack in bytes:	3
Value to be inserted/added/subtracted:	0x 42

Figure 2.4: The graphical web configuration interface of the CAN gateway.

2.3.2 Evaluation

During the evaluation, we performed two tests. First, we created a testbed using a Raspberry Pi with a PiCAN shield; and a NodeMCU ESP32s with an additional CAN Transceiver as our CAN nodes. Later, we used a vehicle testbed built from actual vehicle components to verify the functionalities of our device in a close-to-real-world setting.

Raspberry Pi testbed

The nodes were configured to send messages to each other via the CAN bus; however, they could only send these messages through the CAN gateway. The testbed can be seen in Figure 2.5.

During the measurements, we tested all attack types, with different IDs, offsets, and attack lengths, while logging both the UART lines, as well as the messages on both sides of the gateway. As we found, the CAN gateway managed to modify the messages with an introduced delay of 260us on the 1Mbps CAN bus. This delay is only approximately 2.3 times longer than a lost arbitration, which could be caused by a busy bus or a short transient fault.

One example of the tested attacks has the following parameters:

- *Bitrate:* 1 Mbps
- *Id:* 0x090
- *Offset:* 2; *AttackLength:* 3
- *ByteValue:* 0x08
- *AttackType:* `Replace-data-with-constant-values`

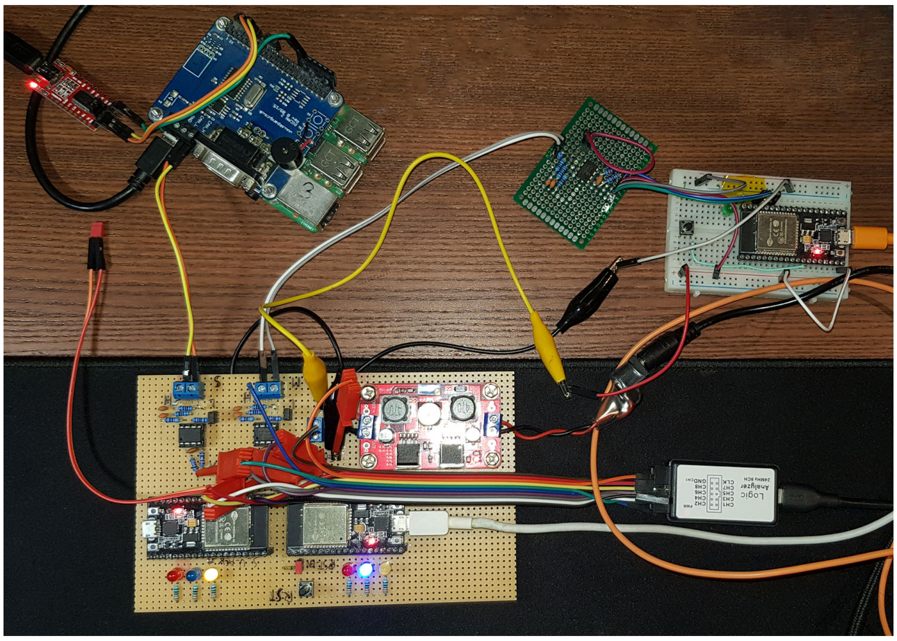


Figure 2.5: The CAN gateway testbed using a Raspberry Pi and a NodeMCU ESP32s.

A screenshot of a measurement can be seen in Figure 2.6. There are two messages traveling on the bus at the same time:

```
A→B: ID: 0x090, Data:  0x00 0x80 0x80 0x80 0x41 0x41 0x00
B→A: ID: 0x045, Data:  0x01 0xf2 0x03 0xf4 0x05 0xf6 0x07 0xf8
```

However, after both messages go through the CAN gateway, the targeted message with the 0x090 ID arrives with changed values. On the arriving side, the following messages are present:

```
A→B: ID: 0x090, Data:  0x00 0x80 0x08 0x08 0x08 0x41 0x00
B→A: ID: 0x045, Data:  0x01 0xf2 0x03 0xf4 0x05 0xf6 0x07 0xf8
```

Thus, we can say that the CAN gateway has successfully modified the preconfigured part of the message.

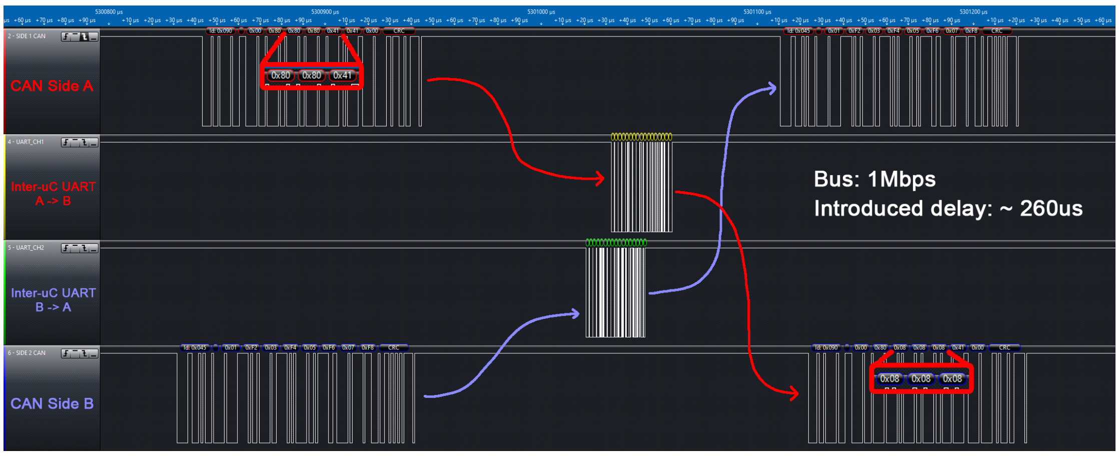


Figure 2.6: Simultaneous message transmission in both direction.

Vehicle testbed

After verifying that our attack worked in our local testbed, we tested the malicious CAN gateway on a Citroen C5 test bench, which contains the electronics of a real-life Citroen C5 (Figure 2.7).

We found a CAN bus connection point between the dashboard and the ECU, we connected the CAN gateway to this point. The attack we performed was to overwrite the tachometer value with a constant, and thus force the dash to show a modified engine rpm value instead of the real one. The attack parameters were the following:

- *Bitrate:* 250 kbps
- *Id:* 0x208
- *Offset:* 0; *AttackLength:* 1
- *ByteValue:* 0x30
- *AttackType:* `Replace-data-with-constant-values`

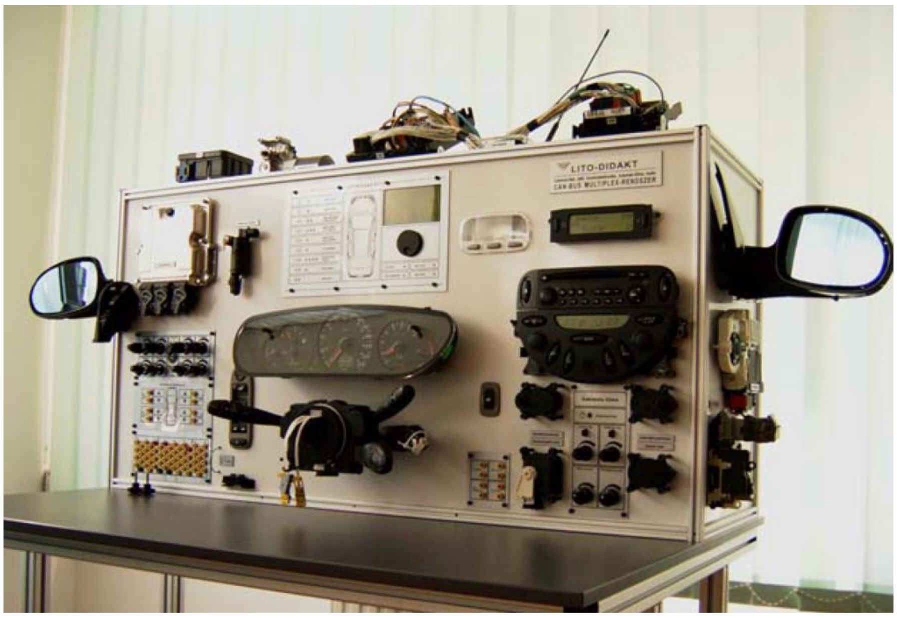


Figure 2.7: The Citroen C5 test bench.

As one can see in Figure 2.8, despite the engine idling at 810 rpm, the dashboard shows the modified values of around 1500 rpm.

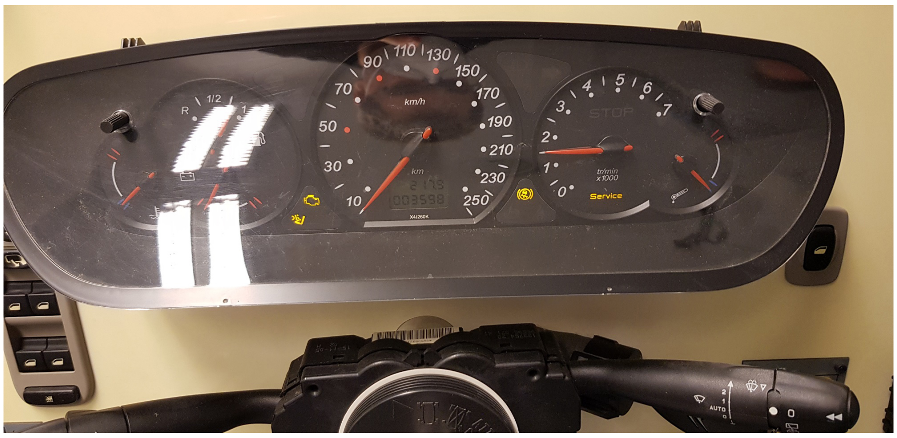


Figure 2.8: The attack of the tachometer displays different rpm than the real one.

2.3.3 Summary

Our CAN gateway device was designed to perform a Man-in-the-Middle attack by separating the targeted ECU and the rest of the CAN bus. Being in a man-in-the-middle position allows our device to modify the content of any message passing through the CAN gateway without any excess message or increase in the busload. It is capable of modifying CAN messages in real-time with a minuscule introduced delay of $260\,\mu$s, without being detectable by current measures. It can handle up to 100% bus load with a bus speed of 500 kbps or less, and about 60-100% busload at 1 Mbps. The device is built from low-cost, commonly available parts, and it provides a wireless interface that can be used to remotely configure the attack parameters.

After demonstrating our capability to execute message modification attacks, we developed an attack simulator[1]. This application can modify any benign CAN trace to contain an attack. With the added confidence that we validated this attack in real life, using the simulator is scalable solution for data generation.

2.4 CrySyS dataset of CAN traffic logs

In this Section, we describe our fullest dataset we generated during our research. It focuses on message injection and message modification attacks, as those two are challenging research problems from a detection point-of-view.

Similarly to our results presented in previous Sections, proof-of-concept demonstrations of attacks have shown the emerging threats against vehicles in recent years. As a response, the research community made several propositions to secure the protocol or introduce anomaly detection systems to stop the threats. Recent research has increasingly focused on using machine learning for anomaly detection. A typical property of these approaches is that they require a large dataset for proper model building and evaluation. However, there seems to be a shortage in appropriate datasets that contain a sufficient variety of attacks.

With our dataset, we would like to improve the situation by giving access to a large number of captured CAN logs in various traffic scenarios in both benign and attacked state. Our dataset not only addresses the data quantity requirements of machine learning-based anomaly detection approaches, but we also focus on the peculiarities of the field by capturing traces with different length. The dataset contains shorter traces (with IDs beginning with S-*), which are useful for rapid model development and idea-testing in addition to longer traces (with IDs beginning with T-*) captured in various traffic scenarios for robust real-life evaluation and results. In total, our dataset consists of 1274 CAN traces.

2.4.1 Methods

We captured multiple hours of traffic in different traffic scenarios to create a diverse benign dataset. In order to create realistic attacked traces, we chose two approaches to perform attacks. On the one hand, we built a testbed with a physical CAN network to execute attacks affecting the message repetition times. On the other hand, we developed an attack simulator to calculate the effect of timing in different attacks. This hybrid generation approach results in a scalable but still realistic solution.

Besides the previously shown anomaly patterns, where the attacker modifies a single signal, we introduce a new modification of the benign signals: double attacks, where the same (or different) attack takes place simultaneously against two CAN signals. Our goal with these

[1]https://github.com/CrySyS/can-log-infector (Last accessed: Oct 1, 2023)

anomalies is to test more thoroughly detection systems designed to exploit system-wide communication information, such as signal correlations. We performed all our attacks in single-signal (Figures A.11. and A.12.) and double-signal (Figures A.13. and A.14.) modes.

Benign CAN data captures

The CAN data was captured in our test vehicle through the OBD port. We built a device to record the raw messages as described in Section 1.2. The captures were performed in a variety of different driving scenarios. The dataset contains 26 recordings: 15 simple maneuver scenarios and 11 complex traffic scenarios, as shown in Table A.6. The complex traffic scenarios contain traces captured in an urban environment, on a country road, and during motorway drives.

The captured data was analyzed to determine the communication properties. The communication contains messages with 18 different CAN IDs. The data fields of the messages were processed with the method proposed by Brent et al.[NGMK18] to extract the vehicle signals. We managed to identify and extract 78 signals, which are shown in Table A.7.

Attacks

The inherent insecurity of the CAN bus allows for multiple attacks against vehicles. Taxonomies to categorize these attacks have been proposed in many papers [SDK19, CS16, TW17]. We describe our performed attacks following the widely used taxonomy of Cho et al[CS16].

According to this taxonomy, an attacker can achieve two types of compromise on ECUs: weak and full compromise. A weakly compromised ECU can be used to capture traffic and its normal message transmission can also be suspended (called a suspension attack). In addition to these misdeeds, a fully (or also called strongly) compromised ECU can also inject newly fabricated messages into the CAN bus (called a fabrication or injection attack). In the case of multiple compromised ECUs, if the attacker has weak control over one ECU and full control over another, a new type of attack also becomes possible: masquerade (or modification). In this scenario, the message transmission of the weakly compromised ECU is suspended, and at the same time, a synchronized fabrication attack is also performed using the fully compromised ECU. For the rest of the ECUs on the bus, this attack is transparent from the message repetition time of view: the inter-arrival times of the targeted frames on the bus remain unchanged. A masquerade attack can also be achieved by physically modifying the CAN bus to insert a malicious CAN gateway into the communication, as shown in Section 2.3

A suspension attack on a weakly compromised ECU has a similar effect on the CAN bus as a device malfunction or failure. As this can happen under benign circumstances as well, safety features are implemented in vehicles to handle such cases without severe consequences. Therefore, our work focused on attacks performed with a fully compromised ECU.

We performed 12 message fabrication and 12 masquerade attacks on our dataset of 26 traces. The attacks have been carried out in both single-signal and double-signal versions. All of the attacks have been performed for two different time durations. The resulting total number of traces in the dataset is 1274 (26 benign and 1248 attacked).

Fabrication attack During a fabrication attack, new messages are injected into the benign traffic. The attacker exploits the fact that ECUs may be implemented so that they accept data at any time. If this is the case, then sending modified CAN frames with a significantly higher frequency can reliably change the behavior of a receiving controller. The original and the injected messages appear on the CAN bus simultaneously.

We built a CAN testbed from three devices to safely reproduce such an attack in a laboratory environment. In order to remain as close to a real scenario as possible in our testbed, we replayed traffic captured from the test vehicle (with the simulator device) while executing the attacks (with the attacker device). We used a third device (the observer) to capture the effects of the attack on the replayed traffic. The schematic of the testbed is shown in Figure A.9.

Masquerade attack A masquerade attack is the most complicated to be performed on an actual vehicle because two ECUs have to be differently compromised in a coordinated way. This attack is also the most stealthy option for an attacker, as there are no additional messages on the CAN bus, and the timing of the normal packets remain unchanged. This property makes this attack easy to simulate: we modified the data contents of some messages of our benign capture logs, leaving all other aspects of the capture unchanged to achieve the effect of a masquerade attack. Overall, we performed the same number and type of attacks in the masquerade cases as during the fabrication attacks.

Signal modification strategies We chose two signals as the target of our tests: the vehicle speed and the engine revolution signals (Figure A.10). We found these signals in the CAN communication using manual reverse engineering steps and validated our finding with the method presented by Lestyán et al.[LÁBS19].

We defined six signal modification strategies that we performed during both the fabrication and the masquerade attacks. Furthermore, we executed the same attacks once only on one signal (Figures A.11 and A.12), then targeting two signals simultaneously (Figures A.13 and A.14). This wide range of attacks cover many strategies, allowing for a thorough evaluation of defense mechanisms. The chosen signal modification strategies are the following:

- CONST: The attacker replaces the CAN signal values with a constant in every message.
- REPLAY: The attacker replaces a CAN signal value with a previously captured value from the traffic. This attack takes twice as long compared to the others: first, the attacker records the signal values, then in the second half of the attack, it replays them.
- POS-OFFSET: The attacker adds a constant value to the CAN signal in each message.
- NEG-OFFSET: The attacker adds a constant value to the CAN signal in each message.
- ADD-INCR: The attacker adds a continuously incrementing value to the CAN signal in each message. This causes a slow but growing shift away from the original value.
- ADD-DECR: The attacker subtracts a continuously decrementing value in each message from the CAN signal. This causes a slow but growing shift away from the original value.

2.4.2 Data Records

Multiple files belong to each test case. The files containing the benign test cases are the following:

- *[TraceID]-benign.log*: CAN trace file with the raw messages.
- *[TraceID]-benign.json*: metadata about the trace (e.g. capture details).
- *[TraceID]-benign-speedAndRevolutionSignal.pdf*: plot of the speed and engine revolution signals.

The files containing the attacked cases are organized in the following way:

- *[TraceID]-malicious-[Attack-type].log*: CAN trace file with the raw messages.
- *[TraceID]-malicious-[Attack-type]-inj-messages.log*: in case of a message injection attack, the injected messages are stored separately as well.
- *[TraceID]-malicious-[Attack-type].json*: metadata about the trace (e.g. capture details and trace file information).
- *[TraceID]-malicious-[Attack-type]-speedAndRevolutionSignal.pdf*: plot of the speed and revolution signals in two format.

The TraceIDs for each of our scenarios can be found in Table A.6. The structure of a raw CAN trace follows the format used by the SocketCAN Linux package[2].

2.4.3 Technical Validation

CAN data recorder validation

We captured and transmitted CAN packets with a Raspberry Pi based recorder. Using the PiCAN2[3] board, we were able to handle messages up to 1 Mbps speeds, the maximal transmission speed of the CAN bus. We verified with measurements, using commercial tools, that our device processes every CAN frame without packet loss.

We executed "dry runs" of our testbed. During these executions, we replayed messages with the simulator device and recaptured them with the observer device without an attacker's intervention. These tests validated that all messages arrive in our testbed, and the inter-arrival times between messages remain unchanged.

Attack validations

Our attacks target two CAN signals: (i) the engine revolution signal and (ii) the vehicle speed signal. Both signals are displayed on the dashboard; thus, the effects of the attacks have been manually validated first to show that they have an actual impact on the vehicle.

[2]https://github.com/linux-can/can-utils (Last accessed: Oct 1, 2023)

[3]https://www.skpang.co.uk/collections/hats/products/pican2-can-bus-board-for-raspberry-pi-2-3 (Last accessed: Oct 1, 2023)

Fabrication attack validation We tested our testbed for the correctness of the fabrication attacks in two ways. First, we performed the testbed validation tests for every measurement to detect potential message loss. Second, we plotted the resulting signal after the attacks to verify the achieved effect visually. The result was rejected if any inconsistency was found, and the test re-executed. If the result passed all the checks, an automated visualization and documentation of the test case was executed.

Masquerade attack validation The execution of these attacks only modifies the data part of the messages. Therefore, we only had to check that the modifications were aligned with our signal modification goals. Similarly to fabrications attacks, we plotted the resulting signals and validated that the behavior of the modified signal matches the goal.

2.4.4 Usage Notes

Comparison to other datasets

The lack of available datasets has significantly hindered the research on CAN security [VIB+22]. Capturing real data and performing attacks require a significant effort and special expertise in the automotive field. Therefore, datasets with a wide range of attacks are required for advancements in the field.

Previous datasets primarily focus on fabrication attacks due to the relatively easy execution of these attacks. Although the significance of a fabrication attack has been shown in successful vehicle compromises, the drastic changes of these attacks in the frame repetition times allow the development of effective detection methods. Masquerade attacks are more powerful attack methods. Therefore, detection algorithms should also be tested against those. Currently available datasets either lack some of the desired features of the attacks or the attack circumstances are artificial.

The HCRL Lab released two CAN datasets with different attacks called "CAN Dataset for intrusion detection (OTIDS)"[LJK17] and "Car-Hacking Dataset"[SWK20]. Both datasets contain only fabrication attacks achieving different goals like, DoS, fuzzing, spoofing, or impersonation attacks.

The "Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2"[DLdHE19] dataset contains three different types of attacks: suspension, fabrication, and masquerade attacks. Their goal during the fabrication attacks is to perform a DoS, fuzzing, or replay attack. During a masquerade attack, they replace the frame data bytes with an FF value. Although this is a new type of attack, detecting this significant change is a manageable task.

The SynCAN (Synthetic CAN Bus Data) dataset[HSDU20] contains only extracted CAN signals instead of the original CAN frames. The attacks are synthetically generated and their impact is unknown. The attack generation tactics have a similar approach to that of ours (e.g. they also perform a CONST attack called Plateau, an ADD-INCR attack called Continous Change, and a REPLAY attack called Playback), but the dataset is significantly smaller compared to ours.

The ROAD dataset[VIB+22] can be considered the most complete dataset so far. It contains both fabrication and masquerade attacks that are physically verified to have an impact on the vehicle. Although their tests were performed on a real vehicle and not on a testbed, they executed their experiments on a dynamometer to remain safe during the test. This approach ensures that the attacks are executed on an existing CAN network; however, the vehicle is in a test environment during the execution. Therefore any external circumstance caused by a real environment (e.g. traffic scenarios) is missing from their data.

There are further CAN datasets available for purposes other than attack detection (see e.g., the "Automotive CAN bus data: An Example Dataset from the AEGIS Big Data Project"[4]). As their contents are unusable for our research goals, we excluded them from the comparison.

Code availability

The source code used for the dataset generation is open source[5], which allows others to extend or modify the dataset. Fabrication attack generation requires a few easily accessible hardware components, while the masquerade attacks can be generated on any general-purpose computer.

2.5 Summary

The CAN bus, which is one of the most common communication solutions for ECUs, has several security weaknesses, since security was not in focus during its development. The application of an authentication schema is not widespread; the source of an information can not be identified only inferred from the CAN ID field of the message. Hence, it is possible to inject fake messages, or modify existing messages on the CAN, and by doing that, to force some ECUs to act upon these fake messages, which may influence the vehicle's overall behavior.

While message injection attacks are easy to implement, they provide several side effects, making them almost trivial to detect. Message modification attacks are hard to realize and require a more profound knowledge of the field, but they are much less detectable.

We performed message injection and modification attacks as well, allowing us to obtain a large amount of data for later use. To aid the research community's effort in securing the CAN bus, we also released our dataset to the public.

[4]https://zenodo.org/record/3267184#.XpwVqS9h1hE (Last accessed: Oct 1, 2023)
[5]https://github.com/CrySyS/CAN-Dataset-Generator (Last accessed: Oct 1, 2023)

Chapter 3

Semantic compression of CAN traffic

This chapter focuses on the compression of CAN traffic. Even though the CAN protocol and the transmitted data are relatively simple compared to network traffic in other domains, the repetitive communication results in frequent transmissions, eventually generating a large amount of data. Currently, after a message is processed, the data is discarded. However, by increasing the exposure of automotive systems to cyber attacks to support forensics analysis, long-term storage of network traffic became desirable.The research question of this chapter is: *How can we best aid the data storing and offloading processes in a resource constrained environment to support forensics analysis?*

Historical analysis of CAN bus logs is only possible if the data storage issue is solved efficiently. There are two possible approaches: (1) storing traffic logs locally or (2) offloading captured traffic to a remote server. Whichever option the manufacturer chooses, traffic compression can significantly improve the efficiency of the process. In this Chapter, we propose a compression method that allows for the lossless, yet efficient storage of data. We also show in Section 3.4, that our compression method allows analysts to perform the log analysis on the compressed data, therefore, it contributes to reduced analysis time and effort. We achieve this by performing the semantic compression on the CAN traffic logs, rather than simple syntactic compression. Besides all these advantages, the compression ratio that we achieve is better than the compression ratio of the state-of-the-art syntactic compression methods, such as zip.

The syntactic compression methods operate on the low level byte stream representation of the data. In contrast to this, the semantic compression methods interpret the data being compressed and take advantage of its semantic understanding. The semantic compression has generated considerable interest in the recent years. It has been successfully applied in different fields such as general database compression [JNOT04b], video compression [MTTH13a] and virtual machine memory compression [RRA+13]. Here, we propose a new application area for it.

The rest of the chapter is organized as follows: In Section 3.1, we give an overview on crash data recorders and show that they are not appropriate for supporting forensic analysis of cyber-attacks. We describe our new semantic compression algorithm in Section 3.2, and we evaluate its performance in Section 3.3. In Section 3.4, we show that the proposed format is a useful approach to find traces of an injection attack. Finally, in Section 3.5, we summarize the results.

3.1 Related work

Data recording in road vehicles Data recording devices that can capture information continuously or triggered by an event have existed in the transportation industry for decades. The best known such devices are probably the "black boxes" used in aviation to record data that can be used by investigators to reconstruct some of the circumstances of an airplane crash. Such recording devices now also exist in road vehicles: since September 2014, a so called Event Data Recorder (EDR) is mandatory for every new passenger car and new light commercial vehicle (LCV) in the US.

The purpose of EDR devices is to collect data about the vehicle dynamics and the vehicle status that enable better accident reconstruction. It helps in validating insurance claims, encourages safer driving behavior, and extends the scientific knowledge about real accidents. The importance of an EDR-like "black box" increases with the deployment of highly automated functions in road vehicles, as there must be some objective evidence proving who was in charge of control in the vehicle in a critical situation. It is, however, not clear what would be the minimum set of data that needs to be collected in case of automated or highly automated vehicles; accident researchers and automated vehicle experts are currently working together on new regulations in this field.

While EDR devices collect data from the CAN bus, the recording of that data is not continuous in time, but triggered only by certain events that may indicate a forthcoming accident (e.g., events that trigger the airbag). In addition, the data recorded by EDR devices is limited to a short interval in time (typically a few seconds) surrounding the point in time of the accident. Unfortunately, a cyber attack that ultimately leads to an accident may happen long before the accident itself (at least, well beyond a few seconds interval around the time of the accident), and therefore, the data recorded by an EDR device will not contain satisfactory information about the cyber attack causing the accident. For detecting cyber attacks and for being able to analyze after an incident how the attack was executed, one needs to collect and record a continuous flow of CAN traffic for an extended period of time.

There exist data recording devices, such as tachographs, that perform continuous data collection in vehicles. Tachographs are mainly used on heavy trucks, buses, and emergency vehicles to continuously record certain parameters of the vehicle such as its speed, its engine RPM, and odometer values. Yet, the main purpose of tachographs is to monitor the duty status of the drivers of commercial vehicles, and they are not designed to record raw CAN traffic. They usually record only a few vehicle parameters with a certain recording frequency, and they are not available on all kinds of road vehicles. Hence, similar to EDRs, tachographs in their current form cannot really be used in investigations of cyber incidents affecting vehicles.

Hence, we can conclude that, although they have seemingly similar goals, existing data recording devices in road vehicles actually address a problem different from the one that we address in this Chapter, and they are not appropriate for cyber incident investigations.

Compression Semantic compression has generated considerable interest in the recent years. It has been successfully applied in different fields such as general database compression [BGR01, JNOT04a, GP16], video compression [MTTH13b], and network traffic compression [CRN08].

To the best of our knowledge, it has not been applied yet for forensic evidence handling in the automotive domain. Hence, what we propose in this Chapter is a new application area for it.

Many of the previously proposed semantic compression algorithms perform lossy compression. However, lossy compression is not appropriate for forensic purposes, as for forensic investigation, one needs to collect and retain accurate information that can potentially be used as evidence in front of court. Hence, in the sequel, we focus on semantic compression algorithms that provide a lossless service. In particular, we compare our work to [CRN08], which was proposed to compress IP traffic captures, and [GP16], which is a recent semantic compression algorithm for large data tables that outperforms some earlier proposals, such as [BGR01] and [JNOT04a].

IPzip [CRN08] is an algorithm for compressing IP network packet headers and payloads. One of the motivations for developing IPzip was to support forensic investigations and to help ISPs to comply with data retention laws. Hence, IPzip performs lossless compression on full network traffic captures. In addition, IPzip is a semantic compression algorithm that exploits the correlations exhibited by packets that belong to the same upper layer protocol session or have the same destination port (inter-packet correlation) and correlations of header fields within individual packets (intra-packet correlation). The basic idea of IPzip is to reorder the packets in the network log such that related packets are grouped together, and to separate the structured header part of packets from their unstructured payload. In this respect, our method is similar, as we also rearrange CAN packets based on their CAN IDs and separate the unstructured CAN payload from the CAN header and other meta-information, such as timestamps. We cannot exploit however correlations of header fields, because the CAN header contains a single ID field, and we cannot either exploit redundancy in upper layer flows, because we cannot interpret the proprietary, manufacturer specific upper layer protocols. Yet, we achieve a better compression ratio than IPzip: in [CRN08], the authors report that IPzip achieves a compression ratio between 30% and 40%, while our compression ratio is around 10-12% (in our binary format). The difference may stem from the highly periodic nature of CAN traffic, which is not a characteristic feature in IP traffic.

Squish [GP16] is a semantic compression algorithm that leverages the relational structure of large data tables in databases. It uses a combination of Bayesian Networks and Arithmetic Coding to capture multiple kinds of dependencies among attributes and to achieve near-entropy compression rate. More specifically, it learns a Bayesian network structure from the dataset, which captures the dependencies between attributes in the structure graph, and models the conditional probability distribution of each attribute conditioned on all the parent attributes. Then, it applies arithmetic coding to compress the dataset using the Bayesian network as the probabilistic model. Finally, it concatenates the model description file (describing the Bayesian network model) and compressed dataset file. Squish achieves a reduction in storage on real datasets of over 50% compared to its nearest competitors, including ItCompress [JNOT04a] and SPARTAN [BGR01]. It is difficult to compare it to our algorithm, because it was not used to compress network traffic logs. On tables containing discrete numbers, which are similar to a series of timestamps in our setting, it achieves a compression ratio of around 32%. Our 10-12% compression ratio is better, because we can exploit the periodic nature of the timestamps.

Finally, we must mention the compression algorithms BFC and SRA proposed in [WC15], which were specifically developed for lossless compression of CAN packets. However, their motivation is different: they address the problem of overload on the CAN bus. BFC and SRA can compress CAN packets in real-time, which results in reduction of the bus load. They achieve a compression ratio of around 80%, and they require special hardware or software in ECUs to perform compression and de-compression. Our algorithm is not intended to be used in real-time, it does not require any modification to existing ECUs in the vehicle, and it achieves a much better compression ratio of around 10-12%. However, this comparison is not entirely fair due to the different goals of BFC/SRA and our algorithm.

3.2 Traffic log compression algorithm

The usage of semantic compression and syntactic compression helps to achieve different goals. A clever combination of the two approaches could benefit from the advantages of both: semantic compression reduces the file size while maintaining accessibility to the data. whereas syntactic compression achieves the smallest possible file size.

To exploit the benefit of both approaches we propose to apply both methods at different operational phases. During data collection an on-the-fly semantic compression could reduce file sizes while keeping data available for immediate processing. The compressed files could be an input for IDS or other anomaly detection appliance. The compressed data format allows a fast analysis of data flows because they are stored in blocks after one another whereas investigation of causality relations is more computing-intensive.

An optional long term storage or cloud transfer of network logs requires a smallest minimum file sizes while the importance of immediate data accessibility is reduced. This implies the use of syntactic compression at this phase. It has been proven before that performing semantic compression before syntactic compression still pays off [JMN99].

Semantic compression

We propose a compression algorithm that takes advantage of the largely periodic nature of the CAN traffic. The high level approach of our algorithm is to separate the traffic into message flows, containing only messages that have the same ID, and then, compressing each message flow separately leveraging the previously identified properties. Algorithm 1 shows the pseudo code of the compression.

```
Algorithm 1: Semantic compression
  Input: raw CAN log
  Output: compressed CAN log
1 $messages \leftarrow read\ CAN\ traffic\ log$;
2 $flows \leftarrow separate\ Messages\ into\ message\ groups$;
3 for $flow\ in\ flows$ do
4     $calculate\_average\_inter\_arrival\_time(flow)$;
5     $group\_messages\_with\_identical\_data(flow)$;
6     for $message\ in\ flow.messages$ do
7         $compress\_timestamp(message)$;
8 for $flow\ in\ flows$ do
9     $write\_compressed\_flow\_to\_output(flow)$;
```

After reading the log file (where messages are stored similarly then showed in Example 3.1), the first step is to filter the messages based on the ID field of the protocol. This step generates separate lists of messages (a flow) where the only remaining information for each message to be stored is its timestamp in the log and the data content of the message. In many cases, the content shows only very low variations, allowing the compression to be even more efficient by grouping together identical messages.

```
1481492674.734327  0x260  8  000000000000006a
1481492674.736055  0x2c4  8  05c8000f0000923c
1481492674.738092  0x2c1  8  080335016ad9004f
1481492674.754306  0x260  8  000000000000006a
1481492674.759605  0x2c4  8  05c8000f0000923c
1481492674.769823  0x2c1  8  0803390170d90059
1481492674.774302  0x260  8  000000000000006a
1481492674.783129  0x2c4  8  05c2000f00009236
1481492674.794246  0x260  8  000000000000006a
1481492674.801541  0x2c1  8  08033b0174d9005f
1481492674.806689  0x2c4  8  05c2000f00009236
1481492674.814227  0x260  8  000000000000006a
1481492674.83034   0x2c4  8  05c5000f00009239
1481492674.833283  0x2c1  8  08033b0174d9005f
1481492674.834316  0x260  8  000000000000006a
1481492674.853767  0x2c4  8  05c8000f0000923c
1481492674.854213  0x260  8  000000000000006a
1481492674.865006  0x2c1  8  0803380172d9005a
1481492674.874181  0x260  8  000000000000006a
1481492674.877285  0x2c4  8  05c8000f0000923c
```

Example 3.1: Simplified CAN traffic log

Storing a complete and separate timestamp for each message in a flow would be a waste of storage. Our more efficient approach takes advantage of the periodicity of messages. Theoretically, a new message with the same ID should come at an exactly predictable time point based on the inter-arrival time of this message type. However, this behavior can be changed by a higher priority message on the CAN bus. If two messages are sent at the same time, then only the one with the higher priority will be sent, shifting the inter-arrival time of the messages with the lower priority. From this point on, the arrival times of this complete message flow will be shifted.

An efficient way to store the timestamp of a message is to store the number of periods (specific for that flow) passed since the last message of the same type and an additional offset value that is induced by either priority causes or measurement distortions. This approach also allows for an efficient description of message flows, where the same message data appears repeatedly from time to time.

It is important to note that the timestamp is not part of the original CAN message, but it is an essential part of the post-analysis to have this information; thus, we consider the timestamp a natural part of the CAN trace.

For each message flow, there are some additional metadata to be stored: the message ID, the first appearance of the flow in the log and the characteristic period length of the flow. These flow specific metadata should be followed by the message data and then the compressed timestamp for each message. An example of this compressed format can be seen in Example 3.2, where the # sign separates the period number and the offset value in each compressed timestamp.

```
0x260
start_time:1481492674.734327
period:19984
00 00 00 00 00 00 00 6a:  0#0,1#-5,1#12,1#-40,
                               1#-3,1#105,1#-87,
                               1#-16

0x2c4
start_time:1481492674736055
period:23540
05 c8 00 0f 00 00 92 3c:   0#0,1#10
05 c2 00 0f 00 00 92 36:   1#-16, 1#20
05 c5 00 0f 00 00 92 39:   1#111
05 c8 00 0f 00 00 92 3c:   1#-113, 1#-22

0x2c1
start_time:1481492674738092
period:31728
08 03 35 01 6a d9 00 4f:  0#0
08 03 39 01 70 d9 00 59:  1#3
08 03 3b 01 74 d9 00 5f:  1#440, 1#14
08 03 38 01 72 d9 00 5a:  1#-5
```

Example 3.2: Compressed CAN traffic log

A compression example

The operation of our semantic compression can be effectively demonstrated on Example 3.1 that shows a simplified CAN traffic log. It has been truncated and reduced to only contain messages from three different ID types. Other than that it is a real life traffic log.

In the first step the algorithm reads the messages separating them into groups with the same ID. In this case it would result in 3 groups: `0x260`, `0x2c4` and `0x2c1`. The following step is the same for each group, that is to compressing messages inside a group.

An efficient way to find repeated messages is to build a hash map of the messages using the message data as a key. At this level, the only remaining information to be stored is the arrival time of the message.

For a more efficient compression the timestamps are stored in a coded way taking advantage of the CAN traffic properties. The inter-arrival times of the messages can be calculated, based on the stamps, requiring only to store the small difference between the predicted and the actual arrival times.

It is possible, that a message data appears in the traffic from time to time. This also has an impact on the compression, i.e. we need to store the elapsed number of periods in each and every case too. This number usually has the value of 1 but for a recurring data this may vary.

The final result of the compression of this log can be seen in Example 3.2. For storing the compressed timestamp the number of cycles and the arrival shifts are separated with a # sign.

Output formats

We defined two output formats for our algorithm. One is a text base (ASCII) representation of the traffic log (like presented in Example 3.2), while the other is a binary format. Both formats contain the same lossless information.

It is worth having both of this options because various further usage may prefer one over the other. The binary format stores the compressed data in a more efficient way that can be seen in Table 3.1.

Table 3.1: Semantic compression ratio comparisons

	Original trace	Text format		Binary format	
Test case	file size (bytes)	file size (bytes)	file size percentage	file size (bytes)	file size percentage
1	10 095 971	1 710 920	16,94%	1 090 757	10,80%
2	7 040 165	1 334 902	18,96%	835 539	11,86%
3	19 143 383	3 747 229	19,57%	2 307 146	12,05%
4	21 936 245	4 233 994	19,30%	2 601 354	11,85%

Table 3.2: Semantic and Syntactic compression ratio comparisons

Test case	Original trace zip compressed file size (bytes)	Semantic and Syntactic compression combined			
		Text format		Binary format	
		file size (bytes)	file size percentage	file size (bytes)	file size percentage
1	1 291 315	546 725	5,41%	499 998	4,95%
2	937 319	429 234	6,09%	390 467	5,54%
3	2 569 118	1 194 758	6,24%	1 092 183	5,70%
4	2 895 039	1 332 585	6,07%	1 223 677	5,57%

3.3 Evaluation

We evaluated our algorithm in terms of run-time performance and efficiency. As the most important performance metric, we calculated the compression ratio and as for efficiency, we also measured the speed of our implementation. We performed our measurements multiple times with different datasets originating from different vehicles. We used vehicles of three different brands all belonging to the low mid-level category built between 2005 and 2010.

We captured traffic with a Raspberry Pi based CAN interpreter, similarly as introduced in Section 1.2. It allowed us to access the raw information on the CAN bus and we saved every CAN message with a timestamp. We performed traffic captures through the OBD interface where the design of the vehicle allowed for an uninterrupted access to the power-train CAN bus traffic through this connection. We were able to gather traffic logs of multiple hours in all three types of vehicles that we used in the evaluation.

3.3.1 Run-time complexity

The time complexity of our semantic compression algorithm is $O(n)$ where n is the number of messages in the traffic log. To compress a complete traffic log, the algorithm iterates over the messages 6 times. Every iteration is linear regardless the size of the input therefore its complexity is $O(n)$. For the python implementation of the compression we also only used data structures with either $O(n)$ or $O(1)$ data structure speed.

The algorithm was capable of efficiently compressing data gathered during the test scenarios in every case at least a magnitude faster than the incoming speed as shown in Table 3.1. This speed overall makes our algorithm a good candidate for on-board data compression for local usage of the information or as a preparation for a remote transmission.

3.3.2 Compression ratio

The measured compression ratios show significant progress in the data sizes (Table 3.1 and Table 3.2). We were able to achieve compression ratios of less than 20% using an ASCII representation of the output of our algorithm. The binary representation shows an even more efficient compression with the results being around 10% of the original file size.

If we applied the additional syntactic compression to our semantic compression it resulted in the smallest file sizes we were able to achieve. In the ASCII representation scenario the combined result shows an approximate 6% compression ratio while the binary case show an approximate 5% compression ratio.

This result can be considered as another proof that it is worth applying semantic compression before syntactic compression because with this combination additional efficiency can be gained.

3.3.3 Correctness

We also implemented a de-compression algorithm. It allowed us to restore the data in an identical form to the original files before the compression. We performed a bit-by-bit and a SHA-256 based comparison of the original and the de-compressed files to check the correctness of our algorithm. In every case, we could restore the original data without any loss.

3.4 The forensic use of the compressed format

In the recent years several articles have been published about the vehicle security. Perhaps the largest impact was achieved by the papers of Koscher at al. in [KCR+10] and Miller et al. in [MV13]. These papers described a series of attacks against the vehicle CAN busses using different attack approaches. Their attacks on a network level can be divided into two categories: sending already known messages with a frequency different from the usual frequency and inserting messages with a previously unused message ID. Based on our research, our proposed CAN compression format can also be helpful to find attacks described in these works.

The first type of the CAN attack inserts new messages with a known message ID. The purpose of this attack is to flood the CAN bus with a modified message data suppressing the information sent in the original messages. This approach was used, for example, to modify information displayed to the driver. The appearance of the original message on the CAN bus cannot be prevented leaving the only option of sending the crafted messages with a much higher frequency (up to 10-20 times the original value).

The second type of the CAN attack inserts completely new, previously unseen messages into the traffic. In the presented works, those new messages were diagnostic messages. The goal of those messages was to trigger functionalities of the car that would otherwise be turned off. As an example, it is possible to use the park assistant feature of certain cars during the normal driving circumstances to change position of the driving wheel.

Both of these attacks produce a very specific pattern in the CAN traffic that can be easily identified in the compressed format proposed in this work.

One of the properties of the CAN traffic, also utilized by our compression, is the highly regular arrival times. This property is harmed when a higher frequency flooding attack is inserted into the CAN communication. The proposed format represents arrival times in an "elapsed cycles # offset" format. During the normal operation, the value of elapsed cycles is the most probably 1 and the offset is a relatively small number. This behavior changes notably when a log file that includes a flooding attack is compressed. In this case, the value of the elapsed cycles is always 0 because the time between attacking messages is around 1/10 - 1/20 of the normal inter-message time. This also results in an offset field with a greater value than the offset values in the normal case. An example of a compressed attack traffic log can be seen in Example 3.3.

```
id:0110
start_time:1483093132166605
period:9994
0200000000270000:3#-392,1#426,
0200000000260000:1#-348,1#-47,1#-22,1#369,1#-37,1#-301,1#44,1#299,1#-209,1#-81
027d000000200000:2149#-3321,0#999,0#999,0#999,0#999,0#999,0#999,0#999,
                 0#999,0#999,0#999,0#999,0#999,0#999,0#999
```

Example 3.3: Compressed CAN traffic log

Using messages, as part of an attack, with completely new IDs generates an entirely new section in the compressed format. The first step of the compression is to separate messages into discrete groups of messages with the same ID. This step makes it very easy to find attacks using new message IDs. The easiest way to find this discrepancy is to compare multiple compressed traffic logs originating from the same vehicle. This allows the analyzer to significantly reduce analysis time.

Finding anomalies in the compressed format relies on the fact that the properties of the traffic are determined based on a benign traffic period. This can be rather easily achieved because only a very short time frame is required to calculate this information. This calculation can be repeated periodically and the result should be the same every time. If that was not the case, that could also be an indicator that probably an attack happened in that time frame.

3.5 Summary

In this chapter, we presented an efficient way to perform lossless compression of the CAN traffic logs. Based on our observations of the periodic properties of the CAN traffic, we designed a semantic compression algorithm for the CAN traffic. With the use of our algorithm, storage efficiency and communication costs can be significantly improved, while keeping the possibility to perform analysis on the compressed data.

Chapter 4

Anomaly detection

This chapter describes our efforts to improve the security of CAN networks with anomaly detection solutions. Our goal is to identify the previously introduced message injection and message modification attacks. We propose multiple algorithms, built on different prerequisites, that can effectively detect anomalies in the CAN communication. We formulated two research questions for this chapter:

- *Is it possible, and, if yes, how to detect anomalies on compressed logs?*
- *Is it possible, and, if yes, how to improve anomaly detection over the state of the art by applying the latest machine learning techniques?*

In Section 4.1, we give an overview of the potential approaches to improving the security of the CAN bus. Then, in Section 4.2, we show that our proposed CAN compression algorithm is suitable for message injection detection. In Section 4.3, we propose a new message modification detection algorithm against message modification attacks. Next, in Section 4.4, we explain a neural network based detection algorithm to identify further message modification attacks. Finally, in Section 4.5, we summarize our anomaly detection results.

4.1 Approaches to increasing the security of the CAN bus

Researchers have tried to tackle to problem of the insecurity of the CAN network (as described in Section 1.4) from many different perspectives. Some of the more notable options are (1) the improvement of the CAN specification with the introduction of security features, such as authentication, (2) the installation of a firewall between CAN network segments, (3) the redesign of the CAN transceiver hardware elements to enforce security controls, or (4) the introduction of various intrusion detection systems, such as a physical model-based intrusion detection or a network anomaly-based intrusion detection system to identify attacks. While all these measures could theoretically improve the security of the CAN bus, they are either not efficient enough or their usage is impractical from the manufacturer point of view. Next, we will detail these approaches and their shortcomings.

Authentication As mentioned before, one of the biggest deficiencies of the CAN bus is that it lacks security features, such as message authentication. CANAuth [VHSV11] solves this issue by using a symmetric key based HMAC. As described in Section 1.1, every bit transmitted on the CAN bus get sampled at the 75% percentile point of the bit time to ensure a reliable sampling. However, technology has developed a lot since the introduction of the CAN bus, and microelectronics are much faster nowadays. Thus we could use a higher sampling rate in order to hide authentication data in the propagation segment of every CAN bit.

CANAuth prevents message injection attacks introduced in Section 2.2 and also allows the detection of some message modification attacks, such as a MitM attack described in Section 2.3, it does not stop a modification attack occurring after a full ECU compromise. If the firmware of the ECU is modified, then the attacker is capable of transmitting modified data with correct message authentication codes.

Firewalls Another mitigation method is to use firewalls [Ari10]. By using a firewall, we can physically split a CAN bus into multiple separate segments and control the traffic going between them. For instance, we can apply allowlist or blocklist-based message filtering on each of the different segments, introduce rate limiting, etc. Thus, we can limit the possibility of message injection and DoS attacks. Adding a CAN firewall to an existing car should not require redesigning the car, since the firewall itself is just a simple device with two CAN interfaces, which can be inserted between the separated CAN segments.

While this solution could appear as the ultimate solution to the shortcomings of the CAN standard, it has several issues. For instance, we have to separate every important ECU or at least every important segment with a firewall, in order to be effective, which creates an excess cost for the manufacturer. An even bigger issue is the management of the firewall. Who gets to write the filtering rules? How are the rules updated, if an update is required? Who is responsible if an important packet gets dropped unintentionally? What if an airbag does not open during an accident due to a malformed firewall rule? While these edge cases could seem unimportant, not being able to address them satisfactorily may make manufacturers deciding not to use this technology. Adding a firewall in front of every important ECU naturally leads to the next approach, which is, improving every CAN transceiver with firewall-like features.

Secure CAN transceivers A third mitigation technique is the secure CAN transceivers proposed by NXP [EWO20]. Their idea is to introduce a new firewall-like security defense layer at the CAN transceiver level. Using this new layer, they can prevent message spoofing in both the transmitting and the receiving side; i.e., detect malicious ECUs and evade DoS attacks.

This approach is welcomed by manufactures as it does not result in a significant cost increase but from a rule-management point-of-view it suffers from the same problems as any other firewall solution. Furthermore, it can secure vehicles built in the future, it does not solve the security problems of the already existing billions of vehicles.

Modeling of the physical process As with any cyber-physical system, modeling the physical system parameters for intrusion detection can be a promising approach to identifying attacks.

This can be the foundation of an anomaly detection solution; however, compared to other traditional intrusion detection systems, it is worth discussing separately due to the significantly different challenges of physical system modeling and data capture. This approach has been shown to work effectively in general control systems, such as industrial control systems [GHM16] and in vehicular systems. In [WPW$^+$17] Wasicek et al. showed that by building a model of the engine, they can detect engine control modifications, such as a chip tuning. Although the experiment shows promising results, creating an accurate model and accessing all the necessary parameter readings remains challenging.

Network intrusion detection Identifying intrusions on the CAN bus can significantly improve vehicle security. In general, two intrusion detection approaches are available: signature-based or model-based anomaly detection [RKAK$^+$22]. Signature-based detection has a low false-positive rate as it can accurately identify previously known attacks. However, signature-based techniques fail to identify novel or previously unseen attacks. Model-based anomaly detection techniques can identify novel attacks; therefore, we only focus on these systems from hereon.

Anomaly detection solutions can recognize if the communication deviates from the normal state to inform the driver (and potentially the manufacturer) about a problem. Typically, an alert does not trigger any automatic countermeasure; therefore, the risk of a false intervention in the vehicle controls is low. Furthermore, an anomaly detector can simultaneously added to the CAN network to any other defense solution, as it operates only as an observer of the ongoing communication. Multiple potential detection mechanisms exist, e.g., in [MGF10], 8 categories have been defined as possible approaches, from simple formality and range checks to more complex plausibility and consistency checks.

Based on the information source, these are 4 different approaches to information gathering for network anomaly detection, each with an example:

- *CAN ID*: in [TJL15], Taylor et al. measure the frequency of the incoming packets with the same ID to identify any time-based deviation from the normal behavior caused by injected messages.
- *CAN Payload*: in [KTP20], Kukkala et al. propose a recurrent autoencoder network that can detect CAN messages in which signals have been tampered with. For each message ID, one such recurrent autoencoder is trained to reconstruct the signals within that particular message ID.
- *CAN frame*: in [ABM$^+$21], Ashraf et al. proposed an LSTM autoencoder-based model, that uses the packet count and bandwidth of the traffic in a fixed window as features.
- *Physical characteristic*: in [CS16] Cho et al. introduce a Clock-based IDS (CIDS) solution to identify any change between the transmission times of CAN frames originating from the different ECUs due to various manufacturers. Their approach shows very promising accuracy. However, it can only be applied by physically modifying the network.

Previously demonstrated results all have benefits, but their detection accuracy does not reach the expected level, or their resource needs could be more suitable for embedded systems. In the following Sections, we propose new anomaly detection algorithms to address these problems.

4.2 Attack detection in compressed CAN traffic

Cyber attacks on vehicles can cause physical accidents. This means that when an accident happens, forensic analysis must be extended into the cyber domain, and investigators must analyze whether the accident was caused or made possible by a cyber attack. Imagine, for example, that a compromised ECU provides false data and as a consequence, misleading information is displayed to the driver on the dashboard, or the airbag is disabled silently before a crash, or some autonomous driving function is enabled and the driver loses control over the vehicle. All these can either lead to an accident or increase accident severity. As the cyber attack on the vehicle may occur well before the accident that it causes, forensic analysis can be successful only if detailed logs are recorded for an extended period of time, not just for a few seconds before the accident[1].

In our view, in the future, especially with the increased penetration of autonomous vehicles, it will be indispensable to continuously record CAN traffic in vehicles and efficiently store these logs for later forensic analysis. Efficient storage of CAN logs requires compressing them. Compression not only saves storage space, but it also makes it easier to off-load logs from the vehicle. Usually, the compressed log must be decompressed for analysis purposes, and the analysis is carried out on large amount of decompressed data. This increases the inefficiency of the analysis itself. In this Section, we study the problem of detecting anomalies that may indicate cyber attacks on the compressed CAN traffic log, generated after applying our previously introduced compression algorithm from Chapter 3, hence making analysis faster by not requiring a decompression first.

Anomaly detection cannot be performed on any kind of compressed CAN log, but the compression method must support the analysis of the compressed data. Our compression algorithm achieves a higher compression ratio than traditional syntactic compression methods such as gzip. Besides this advantage, in this Section, we show that it also supports the detection of certain types of attacks in the CAN log without decompression. More specifically, we can easily detect injection attacks, where the attacker (e.g., a compromised ECU) injects a given type of periodic CAN message with a smaller repetition time (higher frequency) than its normal repetition time. Most of the attacks demonstrated in prior work were of this kind[MV15][CMK+11]. The increased frequency of injected false messages usually results in "overriding" the information carried in the legitimate messages. We show that such an attack causes a well-identifiable anomaly pattern in the compressed log even when the frequency of the fake messages is just slightly larger than the normal frequency.

The remainder of this section is organized as follows: In Section 4.2.1, we give an overview on the existing anomaly detection works on CAN traffic focusing mostly on injection attacks detection, aiming for the same goal as our solution. In Section 4.2.2, we discuss the attack scenario and the possible attacks against the CAN protocol that we and recent works take into consideration. We describe the dataset, we used during this research, in Section 4.2.2. We present our anomaly detection approach and its evaluation in Section 4.2.3. Finally, we conclude in Section 4.2.5.

[1]https://www.nhtsa.gov/research-data/event-data-recorder (Last accessed: Oct 1, 2023)

4.2.1 Related Work

Anomaly detection on the CAN bus has been an actively researched field recently. Multiple approaches have been proposed varying in the interpretation of the CAN traffic. If the interpretation of the CAN messages are accessible (e.g. because the CAN matrix is available), it is possible to collect and analyze the actual vehicle parameters. Approaches using this knowledge usually perform anomaly detection on this high level data. The researches not using a CAN matrix are mainly focused on the communications properties such as repetition times of the messages.

A. Taylor et al. proposed a method from the first approach in [TLJ16]. They interpreted the CAN massages to build a current state of the vehicle. Then with a Long Short-Term Memory Network (LSTM) predicted the next state of the car. If the actual state, based on the following messages, is diverging from the predicted state, due to injected messages with false data, they detect it as an anomaly.

S. N. Narayanan et al. proposed a hidden Markov models based approach to anomaly detection[NMJ16]. They used the OBD port available in every modern car to access the CAN bus. Packets captured through this interface are interpreted then and used to build the Markov model. They also understand states of the vehicle and define the possible state transitions. If an unexpected state transition is detected that means an anomaly in their model.

In [MS17] M. Marchetti et al. showed that anomaly detection can be efficiently performed based on CAN ID sequences. From the CAN traffic they only use the ID field of the messages. They build a transition matrix to understand the connection between messages. If during normal traffic an ID follows another then this transition is marked as normal in the matrix. Their anomaly detection method analyzes whether a not allowed transition appears in the traffic.

In another paper Taylor et al. [TJL15] presented an anomaly detection approach that is based on repetition times of the messages on the CAN bus. They first splitted the traffic into flows. For every flow various measures are calculated such as the number of packets in the flow, the average Hamming distance between successive packet data fields and the average time difference between successive packets. During their analysis they show that the only reliable parameter for anomaly detection is the average time difference between successive packets. They use a one-class support vector machine (OCSVM) to classify the benign traffic and to detect anomalies. They measure the efficiency of their work only on syntactically generated traffic.

Although anomaly detection on compressed traffic has several advantages, this idea was not researched so far. We aim to close this gap by analyzing normal and attacked compressed CAN traffic to determine what kind of anomalies could be detected with this approach.

4.2.2 Realized CAN injection attacks

It is possible to achieve an anomaly with just a few messages but in most cases for an attack to be successful, a large number of messages are necessary. In Chapter 2, we introduced the attack organizing them by the technical complexity of the execution. In the following paragraphs, we only focus on message injection attacks and we describe the various possibilities of an attacker, organized by the number of messages required. The amount of extra messages greatly influences the attack detectability.

DOS against the CAN bus In this scenario the goal of the attacker is to completely disable the communication on the CAN bus. This can be achieved at least with two extreme approaches.

An attacker could disturb the transmission of every CAN packet by starting its own dummy transmission in the middle of every other packet. This way an error will occur during the reception of every packet. This attack does not need a full packet to be sent by the attacker just a few bits with the correct timing.

Similar effect can be achieved with the transmission of packets with the ID 0. The ID field of the CAN packet is also determines the priority of the message. The value of the ID decides which packet can be transmitted in case of multiple colliding packets. The smaller the ID of a packet is the higher its priority is. If an attacker sends continuously packets with the ID 0 then there will not be any resource left for the normal traffic.

Both of these attacks are operation critical for a vehicle. A complete DOS against the CAN bus isolates the ECUs from each other disabling most of their operations. These scenarios are trivial to detect, as the number of extra messages or interventions are extremely large, but very difficult to handle.

Messages with new IDs It is common in car manufacturing that the same hardware parts are used in various car models. This practice makes it possible for an attacker to try to trigger functionality in a car that would not be used otherwise. On the CAN level this means that messages could appear with previously unseen IDs.

Some attacks are realized with the usage of debug packets[MV13]. These scenarios also introduce packets with new IDs on the bus.

Although the number of messages is rather low in this scenario, if all benign IDs are known in advance, identifying these attacks is simple. Messages with IDs not seen before can effectively be found with basic allow-listing or simple anomaly detection.

Irregular messages with known IDs Some CAN messages are only transmitted as a response to certain events. These messages are encountered rarely because they are responses to environmental changes and are not part of the regular operation of a vehicle. An attacker could inject any of these messages at random times to force an inconsistency in the operation.

Without an external source of information the only way to detect these messages is to correlate information from other packets. This is a challenging task in most cases if even possible.

Messages with regular repetition times To interfere with the normal operation of a vehicle the regular communication of the ECUs should be altered. It is hard to remove messages from the CAN bus (if en error occurs during transmission usually a re-transmission logic is triggered at the sender) thus the best possible option for an attacker is to send malicious packets additionally. A packet with fake content could force the vehicle into a compromised state until the next packet with correct content arrives. Most attacks aim to keep the vehicle in a compromised state for the majority of the time. This means that the attacker is required to send a lot of malicious packets to minimize the effect of the original benign traffic.

Based on the goal of the attacker the frequency of the malicious packets could be anything between 1x and 10x of the original traffic. Our measurements and previous research[MV13] results also showed that a malicious traffic with ∼10x the frequency of the original traffic forces the vehicle to stay in a compromised state almost constantly.

Datasets

During the research we created two datasets. First, we created a synthetic dataset where the attacks were manually injected into a clean CAN traffic log. Then we also performed some attacks against a real vehicle that gave us real life infected traffic logs.

Synthetic data set

We have captured a few hours of benign traffic from a mid class vehicle. With reverse engineering we found the signal used to display the RPM of the engine on the dashboard. We used this signal during our attacks to simulate an attack where false information is displayed to the driver. The RPM value is sent by an ECU in a message with the ID 110. Normally this message is send in every 10 millisecond. This attack belongs to the "Messages with regular repetition times" category described in Section 4.2.2.

We created a packet with a malicious content to insert into the traffic. The packet contained a higher RPM value than found in normal traffic.

We generated the malicious traffic with multiple steps. First we splitted the normal traffic into smaller chunks. Each chunk contained approximately 1 minute of traffic. As a base rule we decided that every attack should be at least 5 seconds long because a shorter attack on the dashboard would probably not disturb the driver thus it would not achieve any goal. We also generated longer attacks. For each attack scenario we increased the attack length with 5 seconds. This resulted in attacks with random length in these intervals: 5-10; 10-15; 15-20; 20-25 and 25-30 seconds.

For every attack scenario we generated 100 malicious samples. They were each tested in our algorithm together with 100 benign samples.

We generated the malicious traffic simulating the normal operation of the CAN bus (including the bus arbitration). First, we generated 10000 malicious packets. The time stamp of the first packet was randomly chosen from the first half of the benign sample, the rest of the time stamps were calculated based on the chosen attack frequency. Then, we merged the benign and the malicious packets according to the time stamps. If two messages overlapped than the one with the higher time stamp was shifted after the other, simulating the CAN arbitration process. If there was enough time until the next message then the malicious message was simply inserted. Otherwise the same logic was repeated again to resolve further conflicts in the time stamps. Generally, the bus load was relatively low in our test vehicle resulting a low number of those conflicts.

Once we had the 100 malicious and 100 benign samples for every scenario we compressed all of the logs with the chosen compression algorithm. Furthermore we examined how the detectability of such an attack changes with the modification of the message injection frequency. We generated attacks where the injection frequency was 10 times, 5 times and 2 times higher than the original frequency of the given ID. We considered the 10 times higher frequency the default frequency for an injection attack as our real life tests and other researcher results also demonstrated it is an adequate frequency for an attack to have a stable effect.

In our captured traffic there are 18 different IDs. There are IDs with regular (14) and irregular (4) repetition times. We only focused on the regular IDs.

Real-life attacks

We used a test vehicle to demonstrate some of the attacks described previously. It allowed us to test our anomaly detection approach in a real life scenario as well. During the attacks, we targeted both the speed and the transmission indicator of the vehicle. For the speed indicator, we used 3 different attack frequencies: ∼10 times higher, 2 times higher and the exact same frequency as the original messages have. For the transmission indicator we also used a frequency 10 times of the original. We also collected benign traffic from the vehicle to compare it to the malicious logs.

Speed indicator modification In this attack we were able to change the displayed speed of the vehicle. We achieved that even when the car was standing still without the engine running.

We performed this test with different attack frequencies. In the first attempt, shown in Figure 4.1, the frequency of the forged packets was the same as the original one effectively doubling the number of packets with the given ID. This caused the speed indicator to oscillate between the real speed (0 km/h) and the forged speed (30 km/h).

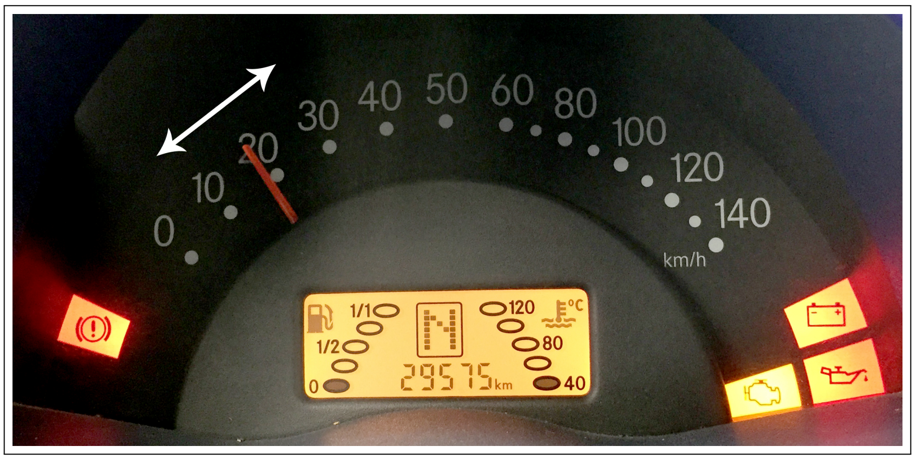


Figure 4.1: Speed indicator attack with 1x frequency caused oscillation of the indicator needle.

In our second attempt, shown in Figure 4.2, we increased the frequency of the malicious packets to 10x the frequency of the normal traffic. This created a stable attack where the indicator showed continuously the speed defined by our attack.

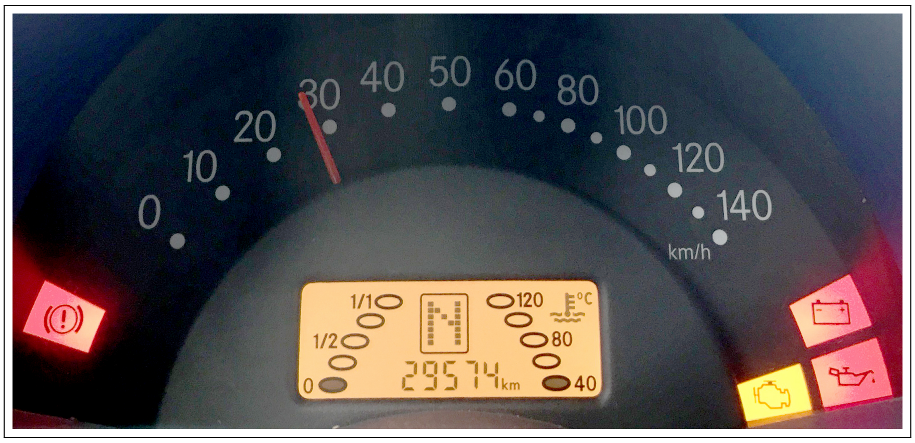


Figure 4.2: Speed indicator attack with 10x frequency. The indicator shows 28 km/h while the real speed was 0 km/h.

Transmission dashboard modification We also attacked the transmission signal for the dashboard, of which the normal state is shown in Figure 4.3. The engine was still not running but we were able to force the display to show that the vehicle was in gear 1.

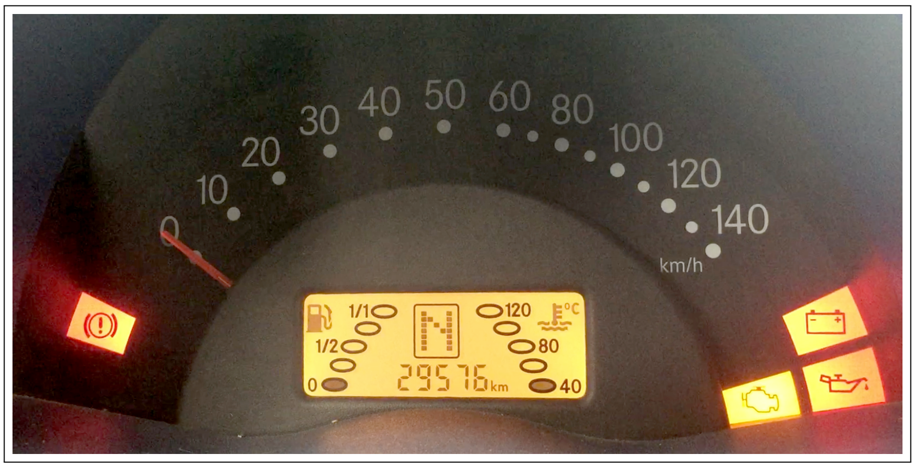


Figure 4.3: Original state of the transmission display.

To achieve the attack goal, we used a packet observed during previous test drives. The malicious packet injection frequency was also 10x of the original rate. As an unintended side-effect we also modified the fuel level indicator and some control lights from the engine. In the original state, the fuel level was low whereas during the attack it showed that the tank is half full (Figure 4.4). This indicates that the fuel level and some of the control signals are transmitted in the same packet as the current gear.

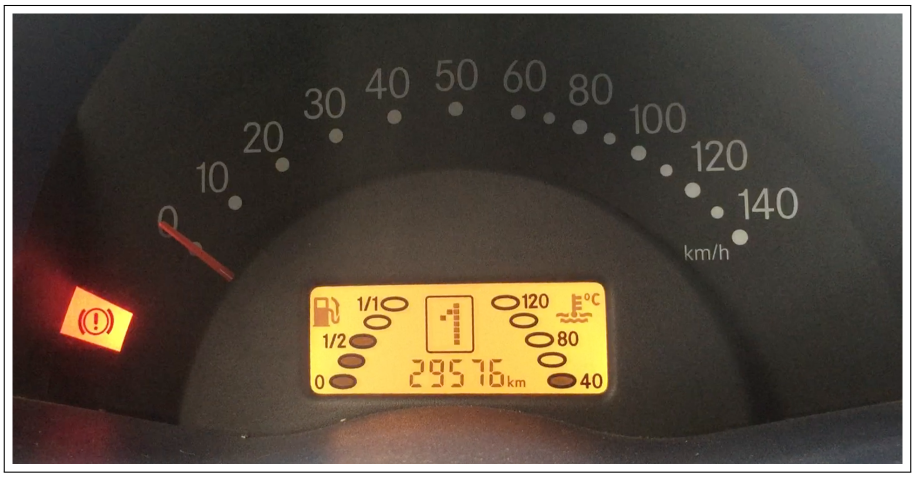


Figure 4.4: Attack on the transmission display. The engine was not running but the indicator showed gear 1. The control lights were switched off and the fuel level was increased.

4.2.3 Anomaly detection algorithm

Our detection algorithm has the goal to decide whether a given message in the compressed CAN traffic log belongs to an attack or not. To address this, first we split the compressed log into separate ID files, where each file contains messages of a given ID. These files are analyzed separately.

We calculated different features of the malicious and benign logs to find the ones that distinguish them the most efficiently. Although, the changes of the repetition times had a significant impact on the structure of the compressed traffic log, the simplest and most powerful feature turned out to be the number of messages during a constant time window.

In a time window of 1 minute we count the number of messages for each ID. Thus we get a feature for each ID: the number of messages in a minute. This will be different in a normal and an attacked traffic log. This approach is also intuitive. If we inject additional messages of an ID that has an approximately constant message rate per minute, the increase in the message rate per minute will indicate an attack.

This feature proved to be reliable for attacks both with higher and lower frequencies.

Based on the previously suggested feature, attacks can be detected efficiently. As can be seen in subsection 4.2.4, this approach separates malicious traffic logs from benign logs even visually making the decision easy.

4.2.4 Results

We evaluated our method on both synthetic and real life data with different attack frequencies.

On synthetic data we used the above mentioned 100-100 normal and attacked samples for attacks with different frequency. The histogram of the distribution of the attacks can be seen in Figure 4.5. They demonstrate that the attacked traffic is efficiently distinguishable from the normal traffic even when the attack frequency is as low as 2 times of the original.

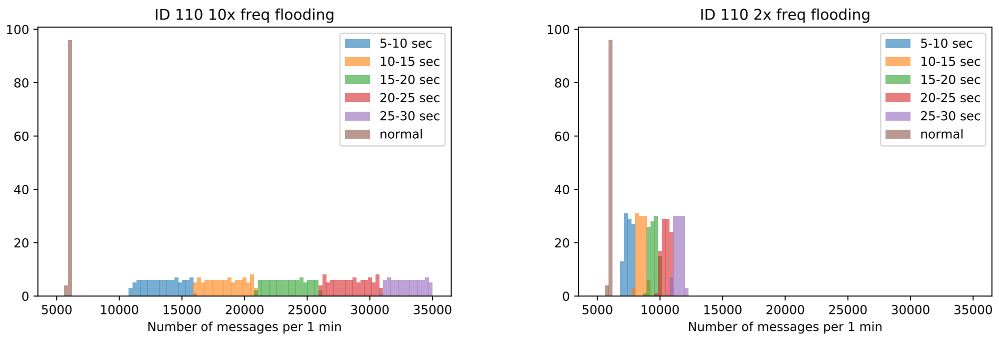


Figure 4.5: Comparison of the number of messages feature for 100 - 100 benign and synthetically attacked samples.

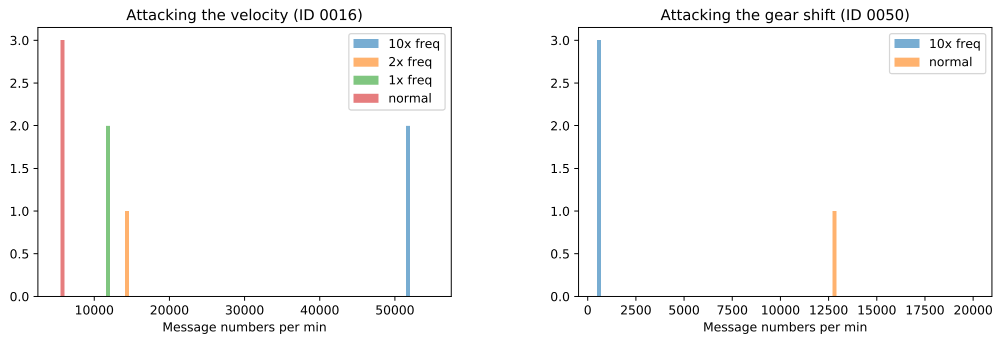


Figure 4.6: Comparison of the number of messages in normal and attacked scenarios during real attacks.

On the data from the real world attacks we performed the same calculations. Figure 4.6 shows that our algorithm achieves the same reliable results in the real life scenarios as well.

These results show that with this approach it is possible to achieve correct classification in every case. For the stable attacks, where a high message frequency is used, the proposed method produces a reliable result with 0 false positive and false negative rates. As the message injection rate decreases the confidentiality is also reduced but even for attacks with 1x injection frequency it remains high enough for a correct decision.

4.2.5 Summary

In this section, we argued that cyber attacks on vehicles may cause physical accidents, therefore, forensic investigations must be extended into the cyber domain. In order to support this, CAN traffic in vehicles must be logged continuously and stored efficiently for later analysis. Our main contribution was a novel anomaly detection method that works on compressed CAN traffic logs. The advantage of running anomaly detection on the compressed logs is that less amount of data needs to be analyzed, hence, the efficiency of forensic investigations can be increased.

Our anomaly detection algorithm is based on analyzing the average frequencies of messages with given CAN IDs. The compression algorithm that we use preserves the number of messages per unit time in an easily extractable form in the compressed CAN log, which makes it possible to use our anomaly detection algorithm on the compressed logs. We demonstrated that this approach works reliably in a range of scenarios, including using data sets captured in real vehicles and modified with synthetically generated attacks as well as data sets captured in real vehicles under real attacks. Our algorithm was capable to identify attacks in both cases.

Observing the average frequencies of messages with given CAN IDs may appear to be a simplistic approach for anomaly detection; nevertheless, it works reliably for detecting injection attacks. In addition, many prior works suggested that injection attacks are easy to carry out and they have noticeable effects, hence, this type of attack is one of the most important attacks to consider.

4.3 Correlation-based anomaly detection

An overwhelming majority of the demonstrated attacks rely on injecting fake messages into the CAN bus. Some ECUs can be easily misled by the fake information in those injected illegitimate messages if they overweight the legitimate ones carrying the correct information. These injection attacks, however, are not so difficult to detect: one can observe the timing statistics of different types of messages and detect deviations from expected values by simple heuristic rules. Similarly to our proposed method for forensics analysis, [MV14] introduced the idea of analyzing the rate of messages for in-vehicle attack detection. In the normal state of vehicle operation, most message IDs appearing on the CAN bus have a regular frequency. When an attacker injects messages to achieve some malicious goal, the frequency of some message IDs will unexpectedly increase, as the legitimate ECUs will still send messages periodically with those message IDs besides the attacker's injected messages. Moreover, the frequency may be increased by a factor ranging from 2 to 100, as pointed out in [MV14]. Hence changes in frequencies can be detected quite easily by simple comparison to some fixed thresholds within a certain size of observation window. Equivalently, increased frequency of a message ID can be translated to decreased inter-arrival times for that message ID, and hence, changes in the statistics of inter-arrival times of certain message types can also serve as the basis for attack detection.

While the majority of attacks on the CAN bus indeed relies on message injection, this is not the only technique to achieve malicious goals. The predictability of message ID frequencies alone is not sufficient for detecting attacks in case of irregular or unpredictable CAN messages and in case of attacks that do not inject new messages on the CAN bus. In this Section, we address the latter problem: we propose a method to detect message modification attacks. Message modification attacks are more difficult to carry out, but they are also much more difficult to detect, therefore, attackers may consider them in the future, especially, given that message injection attacks will likely be detected by future vehicles. Achieving a message modification attack is difficult because the built in safety features of the CAN bus prevent a deliberate modification of a message on the fly. Hence, as described in Chapter 2, three options remain: (1) either the attacker compromises the relevant ECU to modify the message before it is even transmitted; (2) or a CAN gateway between two segments is compromised to modify a message during the hand-off between segments; (3) or the CAN bus is divided into two segments with a new malicious gateway inserted, allowing for message modifications on the gateways between the segments. The first two can be performed purely with software exploits, while the last requires physical modification of the network. Despite the increased complexity, the first and the last approaches to message modification attacks have already been demonstrated in [MV13]. We also demonstrated the executability of the third option in Section 2.3.

In this Section, we introduce a new anomaly detection solution that utilizes the fact that vehicle signals are correlated. The speed, the engine revolution, the current fuel consumption and many other values change together, representing the physical changes of the vehicle state. The solution proposed here can understand these high level relationships between the vehicle signals with the correlation values that correspond to the normal state of the vehicle. During an attack, if a vehicle signal is overwritten by the attacker in a CAN message, some of the measured correlation values may deviate from those of the normal state, and this can be detected as an anomaly. An advantage of this approach, compared to previously proposed algorithms where only a single signal value is used in the anomaly detection, is that if the attacker does not modify all the correlating signals precisely at the same time, the attack can likely be detected.

The rest of this Section is structured as follows. Section 4.3.1 presents the related work. Section 4.3.2 introduces our proposed anomaly detection method and Section 4.3.3 summarizes its testing and validation. Finally, Section 4.3.4 concludes this work.

4.3.1 Related work

Academic papers proposing solutions for securing in-vehicle networks can be divided into three groups: (1) a relatively large body of work (e.g., [VHSV11, GMvHV12, NR16]) is concerned with adding extensions to the CAN protocol, and by doing so, fixing its inherent security weaknesses; (2) a few papers (e.g., [MHT+12]) discuss intrusion prevention either by introducing firewalls or other techniques; and (3) another set of papers (which we discuss in more details below) deal with intrusion detection on the CAN bus using various approaches. Given the considerable amount of work done in the field, a few surveys have also been published: [SMAV19, KKJ+21] have the broad scope of in-vehicle security as a whole, and [LOAB19, YZOB19, WLX+20] are more focused on in-vehicle intrusion detection. As our work falls in the domain of anomaly-based intrusion detection, we focus on that domain in the rest of this section.

Anomaly-based intrusion detection can take two approaches: *specification-based* and *model-based* anomaly detection. In case of the former, the normal behavior of the monitored system is explicitly specified. For instance, in the automotive case, the car manufacturer can have specifications for the normal frequency of different periodic messages at which they appear on the CAN bus. Deviations from the specification can easily be identified as signs of attack. In the case of model-based anomaly detection, on the other hand, no explicit specification of normal behavior is given, but instead, a model of the normal behavior is somehow obtained (e.g., learned by observing the system in the non-attacked state), and deviations from this model are detected as attacks. Different academic proposals differ in what modelling technique they use and what features of the system are modelled.

As for the modelling technique, many papers propose to use some statistical model (e.g., mean, variance, or entropy) of some parameter, with simple heuristic rules (e.g., comparison to a threshold) [MV14, TJL15, SKK16, GMT16, MBC+17] or more sophisticated statistical hypothesis testing methods [MS17, TBSK18] to detect deviations from the model. Other papers (e.g., [The14, TLJ16, MW17, KK16a]) use some machine learning model (e.g., classifier or neural network), with the corresponding model specific method to decide if some input deviates from the learned model. Regarding the features that are modelled, many papers consider properties of the network traffic itself, such as packet timing features (e.g., frequency of certain types of packets) [TJL15, SKK16, MBC+17, TBSK18] and features related to the content of the packets (e.g., the time series of packet IDs or certain signal values) [MW17, KK16b, MS17], whereas some papers use physical characteristics as features, such as voltage level and clock drift of physical signals on the CAN bus [CJJ+18, JWQ+18].

In this Section, we propose a model-based anomaly detection method for the CAN bus that uses correlations across different types of messages as features. To the best of our knowledge the only other paper using correlation-based anomaly detection is [BODA+20] therefore, we make a more in-depth comparison here. The method proposed in [BODA+20] computes the correlation between two specific signals, velocity and RPM, and detects attacks where extra messages containing incorrect values of these signals are injected into the bus. In contrast to this, our method computes the correlation between all pairs of signals, and identifies those pairs that have high correlation without identifying the actual signals. In addition, we detect message modification attacks, which are more difficult to detect than injection attacks. We simulate seven different types of modification attacks and evaluate the performance of our method for each of them. Otherwise, both [BODA+20] and our work use the Pearson correlation function, while in [BODA+20], the authors applied other analysis techniques (such as K-Means and Hidden Markov Models) as well.

4.3.2 Anomaly detection algorithm

Our anomaly detection algorithm focuses only on the detection of message modification attacks, where the original repetition times of the messages are unchanged. As a result only the content of the messages can be used for anomaly detection.

Overview

We propose an anomaly detection algorithm that uses the correlation between signals encoded in CAN messages. Under normal conditions, the correlation between different signal pairs stays within a (signal pair specific) interval. In case of an attack where the attacker modifies only one member of a correlating signal pair, the resulting correlation may no longer stay within the interval, and this can be detected as an anomaly.

In the training phase, the correlation values between signals has to be determined. We measured multiple times the pairwise Pearson correlation between every signal pair in a one minute long time window and in a three minutes long time window. Next, based on these measurements, we decided whether the measured values represent an actual correlation. We achieved this by fitting different continuous probability distribution functions onto the measured correlation values. When we found a proper fit, we added the signal pair to our model. For every signal pair, we also calculated four thresholds to identify the boundaries of normal behaviour: (1) two thresholds define a narrow normal interval, such that measurement outside of this interval are considered potential anomalies; (2) and another two thresholds define a wider interval, such that measurements outside of this interval are considered anomalies immediately.

In the detection phase, correlation values are determined in both a one minute long and a three minutes long window. Then the measured values are compared to the previously defined threshold for anomaly detection.

Data preprocessing

In the training and testing phases we used a 31 minutes long CAN traffic log captured from a middle class vehicle. The traffic contains both periodic and non-periodic messages. This means that some messages arrive regularly with a fix repetition times, while others are only transmitted upon specific events. Before the training, we filtered out the non-periodic messages and those periodic messages that appear less than once per minute. After the filtering step, 92% of the original data remained.

The next step was the signal extraction from the traffic log. For this we used an algorithm from the Automated CAN Payload Reverse Engineering[2] project introduced in [NGMK18]. This algorithm separates the bits of the CAN data field into signals based on bit flip frequencies, called Transition Aggregation N-Grams. The method builds on the property that the MSB bits of a signal change less frequently than the LSB bits. If there is a significant change in the bit flip frequencies of two neighboring bits that shows the boundary between two signals. The same statistical information can also be used to determine the signal encoding.

We calculated correlation values pairwise for the identified signals in different time windows. We tested window sizes from 1 to 8 minutes. For every interval, we re-sampled the signals to have two signal values per second within the chosen window. This rarefying speeds up the correlation calculation. Our measurements showed that the best results can be achieved by choosing 1 minute and 3 minutes as final time windows. This allows us to detect significant anomalies fast and smaller anomalies in a reasonable time.

Model training

Five matrices are calculated for both time windows for the purpose of anomaly detection. A matrix C contains the correlation values and there are four additional threshold matrices that store two upper ($C_{th+,1}$ $C_{th+,2}$) and two lower thresholds ($C_{th-,1}$ $C_{th-,2}$) for the detection.

Each cell $c_{i,j}$ in matrix C contains the Pearson correlation value calculated between signals i and j in the given time window. The correlation value is stored the following way:

- if the calculated coefficient indicates constant values then a *NaN* value is stored.
- otherwise if the two-tailed p-value of the Pearson correlation coefficient is less than or equal to 0.05, then the calculated value is stored.
- otherwise a 0 is stored.

[2]https://github.com/brent-stone/CAN_Reverse_Engineering (Last accessed: Oct 1, 2023)

During the training phase, we randomly select a starting point in the CAN log and calculate the correlation values for all signal pairs for both time intervals. Selecting the starting point randomly allows us to use the original trace multiple times generating a correlation matrix with small differences for every starting point. With this method, we created 300 training matrices for the threshold calculations.

These training matrices gave us a good representation of the typical correlation value for all pairs of signals. In order to find thresholds characterizing the normal behaviour, we fitted different continuous probability distribution functions onto the $c_{i,j}$ values of every training matrix. For every distribution, we performed a Kolmogorov-Smirnov (K-S) test to find the distribution that fits best the correlation values. The K-S test gave us two results: a D statistics and a p-value. For the former, we calculated[3] a critical value at significance level $\alpha = 1\%$ using (4.1) (where n is the number of samples in the dataset):

$$d_{1\%} = \frac{1.6276}{\sqrt{n}} \tag{4.1}$$

Those distributions, where the resulting D statistic was less than or equal to the critical value and the resulting fitted probability distribution's standard deviation was greater than 0 and less than 0.2, we accepted the distribution as a potential candidate. Then, for all these candidates, we calculated the probability distribution's percent-point (or quantile) function value for the $10^{-3}, 1-10^{-3}, 10^{-6}, 1-10^{-6}$ probabilities. These gave us candidates for the minimum ($min_{i,j}$), maximum ($max_{i,j}$), significant min. ($sigmin_{i,j}$) and significant max. ($sigmax_{i,j}$) thresholds.

To choose the best option from the candidates, a scoring technique was used, which is based on the length of the normalized significant min-max interval ($sigmax_{i,j} - sigmin_{i,j}$) and the normalized min-max intervals ($max_{i,j} - min_{i,j}$). The candidate with the minimum final score was selected as the final probability distribution. We used the following function (Figure 4.7 and Equation 4.2) with a minimum value of 0.6 for the significant min-max, and a minimum value of 0.5 for the min-max intervals:

$$score(x) = \begin{cases} (x-min)^2, \text{if } x \leq min \\ \frac{x-min}{min*x}, \text{if } x > min \end{cases} \tag{4.2}$$

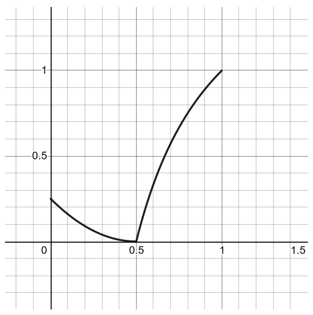


Figure 4.7: Visualization of the function used to score the normal min-max threshold intervals.

[3]https://www.real-statistics.com/statistics-tables/kolmogorov-smirnov-table (Last accessed: Oct 1, 2023)

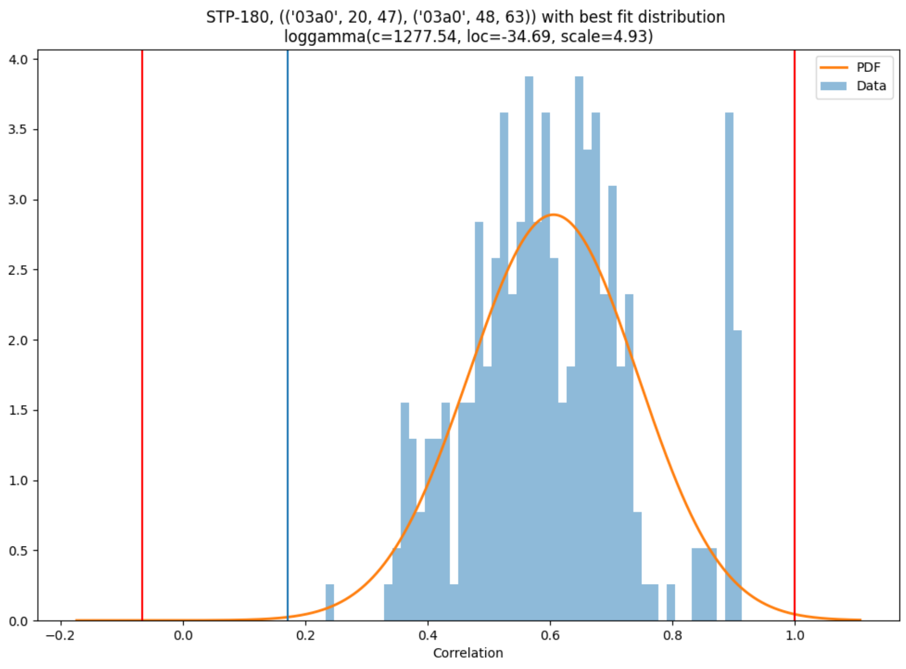


Figure 4.8: One example of a signal pair with a fitted 'loggamma' probability distribution

Then, for the final candidate score we used the formula (4.3):

$$final\_score = 0.65 * minmax\_score + 0.35 * significant\_minmax\_score \tag{4.3}$$

The used scoring system assigns the smallest score to the candidates with a min-max interval length closest to 0.5 and a significant min-max interval length closest to 0.6. This approach prefers the candidates where the intervals are relatively small but not too tight. Our measurements showed that these typically used statistical constants in the calculations with this weighting gives the best trade-off between false positive and false negative results. We used a larger weight for the min-max score to reflect that it is more important to detect an attack than the speed of the detection.

These final chosen threshold values are stored in the threshold matrices, given that there was at least one candidate distribution, in the following way: the minimum values are stored in matrix $C_{th-,1}$; the maximum values are stored in matrix $C_{th+,1}$; the significant minimum values are stored in matrix $C_{th-,2}$, and finally, the significant maximum values are stored in matrix $C_{th+,2}$. If there was no candidate probability distribution found, the signal pair was excluded from the study.

Some correlations between signals are not linear, but as we focus on detection accuracy and not correlation measurement accuracy, we used the Pearson correlation in all our measurements, resulting in the most precise model.

In Figure 4.8, we present an example of the measured correlation values of a signal pair and the fitted probability distribution function ('loggamma'). The vertical blue and red lines show the determined minimum and maximum, and significant minimum and maximum values, respectively.

Detection

In the detection phase, the current correlation values are calculated for the last 1 and 3 minutes of the traffic, and then, the results are compared to the threshold matrices in the following way:

- if a correlation value is outside the significant min-max interval, it is marked as an attack;
- if the value is only outside the normal min-max interval, but not outside the significant one, we consider it a potential attack only, which will be a real attack if our model with the other time interval also gets a similar result.

4.3.3 Evaluation of the algorithm

Benign dataset In the testing phase, we took 270 1-minute long samples of the original trace, which we also used for training. As the starting points of the samples are chosen randomly, this can be considered a separate set of samples from the training dataset. We performed the detection step of the algorithm on these new samples with a previously trained model and found the following result: the model incorrectly signaled an attack in 14 out of 270 cases, resulting in a false positive rate of 5.2%.

Infected dataset The validation of the proposed algorithm was performed on an infected dataset. We simulated different message modification attacks (no new message is added to the log) with a previously developed attack simulator[4] that can take a clean CAN log and modify a selected subset (specified by ID and time interval) of its messages according to 7 different attack scenarios:

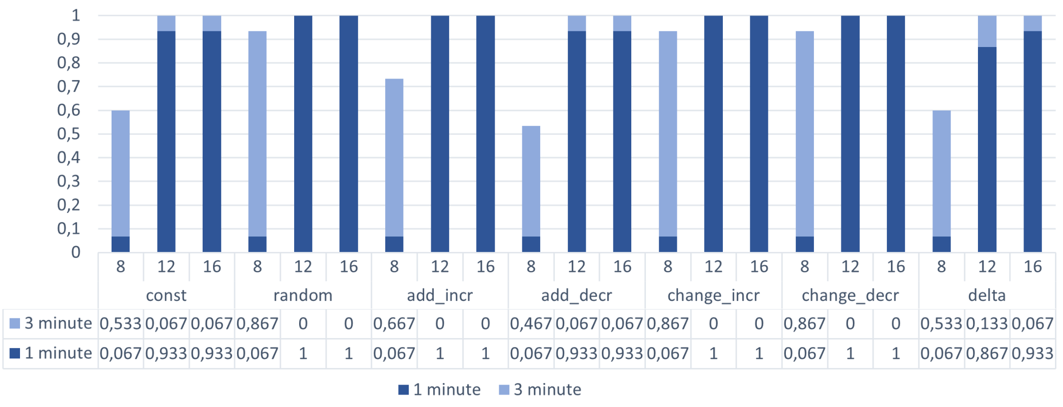
1. **const:** the original data value is replaced by a given attack data.
2. **random:** the original data value is replaced by a new random value.
3. **delta:** a given attack data is added to the original data value.
4. **add_incr:** an increasing value is added to the original data value.
5. **add_decr:** an increasing value is subtracted from the original value.
6. **change_incr:** the original data value is replaced by an increasing value.
7. **change_decr:** the original data value is replaced by a decreasing value.

[4]https://github.com/CrySyS/can-log-infector (Last accessed: Oct 1, 2023)

As it can be seen from this list, this simulator can synthetically generate a large number of infected traces following the same attack strategies we used in the in real-life demonstrated version of a message modification attack, described is Section 2.3.

Measurements In order to evaluate the performance of the algorithm in more details, we divided the signals into three different groups and validated the algorithm in each group separately. The first group contain signals that strongly correlate with multiple other signals. Typically, the most important signals of a vehicle belong to this group. The second group contains signals that have a strong correlation with one other signal, and the third group contains signals with only weak correlation values. We considered a correlation strong between two signals if the mean of the absolute correlation value was above or equal to 0.9 for all the 300 training data samples.

We chose from each group 4 or 5 signal pairs for the validation. For these signals, we simulated all previously mentioned 7 attacks on 15 randomly chosen segments of the original trace. Each attack was performed multiple times. First, only 8 bits was modified according to the attack description in one of the target signals, than the number of affected bits in the upcoming test was increased by 4 until the signal length was reached. All of the attacks were theoretical, but based on previous real life attack descriptions. For this simulation, we did not check whether a specific attack would actually have any impact in real life.

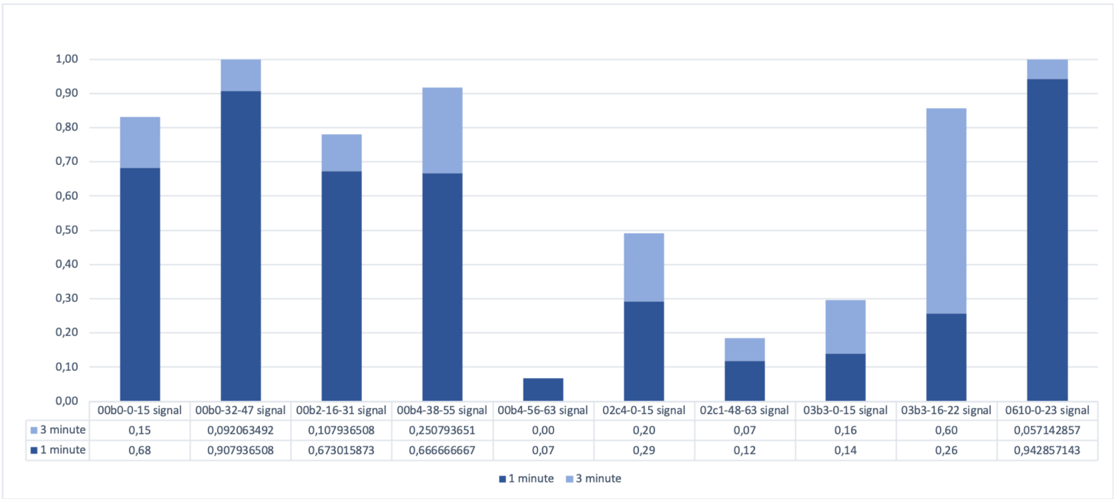


	8	12	16	8	12	16	8	12	16	8	12	16	8	12	16	8	12	16	8	12	16
	const			random			add_incr			add_decr			change_incr			change_decr			delta		
3 minute	0,533	0,067	0,067	0,867	0	0	0,667	0	0	0,467	0,067	0,067	0,867	0	0	0,867	0	0	0,533	0,133	0,067
1 minute	0,067	0,933	0,933	0,067	1	1	0,067	1	1	0,067	0,933	0,933	0,067	1	1	0,067	1	1	0,067	0,867	0,933

Figure 4.9: Testing results for 16 bit long signal with strong correlations.

Figure 4.9 shows detailed results for a signal with strong correlations. The 16 bit long signal was attacked with all attack types. For each type, 3 attacks were performed where the affected number of bits increase from 8 to 16. The two colors of the columns indicate which time window was successful for the attack detection. The detection rate varies between 55% and 100% with an above 90% result for attacks modifying more than 12 bits.

The results found in the others groups, as expected, are less accurate. The average detection accuracy of attacks of signals with one strong correlation is 58% while this falls to ∼20% for the third group where the signals only have weak correlations.



	00b0-0-15 signal	00b0-32-47 signal	00b2-16-31 signal	00b4-38-55 signal	00b4-56-63 signal	02c4-0-15 signal	02c1-48-63 signal	03b3-0-15 signal	03b3-16-22 signal	0610-0-23 signal
■ 3 minute	0,15	0,092063492	0,107936508	0,250793651	0,00	0,20	0,07	0,16	0,60	0,057142857
■ 1 minute	0,68	0,907936508	0,673015873	0,666666667	0,07	0,29	0,12	0,14	0,26	0,942857143

Figure 4.10: Detection accuracy of high priority signals.

Figure 4.10 shows a summary of the results for messages with the highest priority (based on the CAN ID field). It can be seen, that most messages with the highest priority contain signals with high correlation, making them ideal candidates for a correlation based anomaly detection.

4.3.4 Summary

In this section, we proposed a novel correlation based anomaly detection method for the CAN bus with a focus on message modification attacks. We showed, that our solution efficiently detects most of the attacks, making it a promising candidate for real life anomaly detection. Furthermore, a significant advantage of this correlation based detection is that it can detect even the most sophisticated attacks, assuming that the attacker does not modify every related signals consistently.

In our future work, we plan to investigate if the proposed threshold based detection mechanism can be replaced with other potential solutions that increase accuracy. A potential option for this is a machine learning based classification. Moreover, the efficiency of the correlation calculation could also be increased with better data preprocessing, in order to further improve the applicability of our solution in real life scenarios.

4.4 Signal anomaly detection with TCN

Detecting message modification attacks is a difficult task. Building any model of the CAN traffic based only on the message data is particularly challenging, as we do not know how to interpret the data; therefore, we cannot exploit any semantic information. As we showed in Section 4.3, exploiting data correlations between messages can be a powerful detection mechanism. However, not every signal correlates strongly with others, so that approach is limited. In this Section, we propose a new detection method that works on a signal-by-signal base to supplement our previous solutions. We propose a TCN-based approach for detecting modified CAN bus messages. We construct and train the TCN in an unsupervised fashion, since, in practice, labelling CAN bus messages is a very difficult task. In the training process, the TCN will learn to accurately reconstruct the individual signals of CAN bus messages through its causal convolution layers, which allows for information retention from past data samples. Finally, the classification of new data samples will resume to setting an appropriate threshold on their reconstruction loss value. The core idea here is that signals whose data have been altered will be poorly reconstructed by the model, and thus be easy to recognize. Note, that it is not a prerequisite for us to know CAN bus signal semantics which is usually kept confidential [LÁBS19, RLÁB19]. The contribution of this Section is two-fold:

1. We then propose a TCN architecture to learn and reconstruct the normal behaviour of CAN bus signals, and use this information to pinpoint anomalies that do not conform to the reconstruction given by model.

2. We compare the detection performance of our approach to a state-of-the-art GRU-auto-encoder [KTP20] (shown to outperform other existing solutions) through numerical experiments on both our own dataset and the *de facto* standard SynCAN dataset [HSDU20]. Results show that our simple TCN-based approach compares favorably to the state-of-the-art, i.e., it achieves similar or better accuracy with a significantly lower false positive rate.

The rest of this Section is structured as follows. Section 4.4.2 presents our proposed TCN architecture in detail. Section 4.4.3 describes the design of our experiments including choosing the baseline, introducing our two datasets and the training process, and defining evaluation metrics. Section 4.4.4 presents the results of the comparative performance evaluation. Finally, Section 4.4.5 concludes this research.

4.4.1 Related work

In recent years, a considerable amount of literature has been published on CAN bus intrusion detection. These works can be split into three categories: frequency-, statistics-, or machine learning based methods. Most of these approaches are particularly useful for detecting cyber-attacks in which additional messages are being injected into the CAN bus. The simplest of the three, frequency-based models focus on testing inter-arrival times of CAN messages against a predefined normal baseline [TJL15, SKK16, MBC$^+$17]. As the name suggests, statistics-based detection approaches exploit the statistical properties of CAN bus traffic such as entropy [MA11], Z-score [TBSK18] or Mahalanobis distance [MCWW19]. Machine learning based methods imply the usage of artificial neural networks, clustering and supervised models for classification and regression. In the specific field of CAN bus intrusion detection, popular machine learning approaches include autoencoders [LOMH19, LCX$^+$21, NLY$^+$20], recurrent neural networks (RNN) such as Long Short-Term Memory (LSTM) networks [TLJ16, NJCF19, KCI$^+$20, HSDU20, HIO$^+$20], Gated Recurrent Unit (GRU)-based networks [KTP20], replicator neural networks [WWSZ18], and deep convolutional networks [SWK20]. The scrutinized literature shows that recurrent architectures are often the preferred choice for modeling the time series of CAN bus signals, whilst convolutional networks are used when data is transformed to a two-dimensional grid dataframe to resemble an image format [SWK20]. In particular, only one approach was found to combine these two techniques in the form of a convolutional LSTM [TLW20] which is trained on labeled data in a supervised fashion.

Temporal Convolutional Networks To the best of our knowledge, no existing solution employs (causal) convolutions to model the time series representation of CAN signals; we argue that such an approach makes perfect sense given the successful application of convolutional networks to sequence modeling tasks. Specifically, a Temporal Convolutional Network (TCN) is a type of convolutional network whose architecture consists of causal (and dilated) convolutions [BKK18]. It has been shown that this new type of network outperforms recurrent architectures, such as LSTM and GRU, on a multitude of sequence modeling tasks including the adding problem and image classification on sequential MNIST and P-MNIST [BKK18]. In fact, TCNs have also been successfully applied to anomaly detection in general time series data [HZ19].

4.4.2 Anomaly detection algorithm

In this section, we present the motivation behind choosing temporal convolutional networks as an anomaly detection mechanism for the CAN bus. We first provide some background on convolutional networks and then describe our proposed TCN architecture in detail.

Convolutional networks Convolutional neural networks are a particular kind of deep neural networks that enables the extraction of relevant spatial and temporal features from the input (e.g., an image) by learning a set of filters. These filters represent multi-dimensional arrays sliding over the input image, and are initialized randomly. During the forward pass, the dot product between the entries of each filter and the image sub-block is computed, resulting in a feature map. When another convolutional layer is added, the features learned in the first layer are combined to create new ones. To account for as many (non-linear) combinations of features as possible, it is customary to increase the filter size in the subsequent layers. The deeper the network becomes, the better it gets at extracting refined patters from the data. A more detailed description of different convolutional architectures can be found in [AG17].

Temporal convolutional networks (TCN) are a category of convolutional networks particularly suitable for modeling long-term dependencies in sequential data [vdODZ+16, BKK18]. Consider for instance the following task: based on input sequence $x_0, x_1, \dots, x_T$, predict corresponding output $y_0, y_1, \dots, y_T$ at each time step. There are two constraints associated with this task. First, the predicted output y_t should only be influenced by previously observed inputs $x_0, x_1, \dots, x_t$, and, second, the size of the network output must be identical to that of the input sequence. TCNs tackle the first constraint by sliding a filter only over the past input values. In other words, the convolution filter has positive weights only for past inputs. TCNs also employ dilated causal convolutions which, unlike regular causal convolutions, enable an exponential growth of the receptive field by skipping over the inputs while convolving. Moreover, a larger receptive field allows the neural network to infer the relationships between different observations in the input data. The second constraint is addressed by padding the input data with zeros at the borders, to control the dimension of the output. These two architectural elements can be observed in Figure 4.11, depicting a dilated causal convolutional network with two hidden layers. Here, the zero-padding is represented by the white squares on the left side. The filter size of $k = 3$ is indicated by the blue lines. The dilation factor d, applied at each layer, indicates how many input values are being skipped by the filter. Increasing the dilation factor by 2 at each subsequent layer results in a receptive field of size 15: the value of a neuron in the output layer is influenced by fifteen neurons from the input layer.

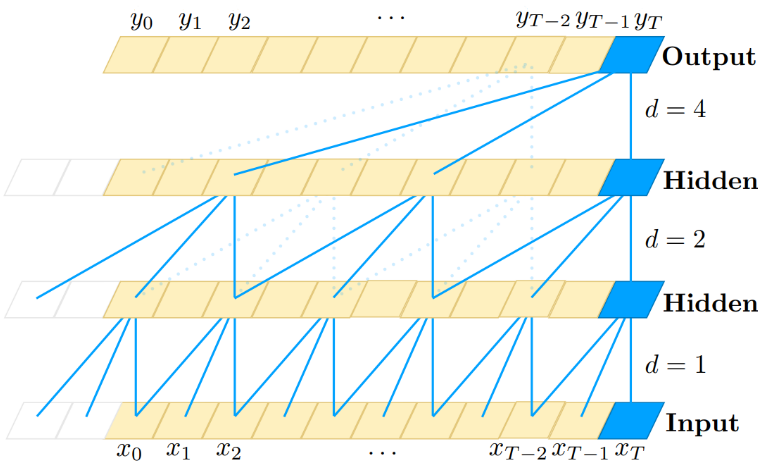

y_0 y_1 y_2 $\cdots$ y_{T-2} y_{T-1} y_T
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 $\cdots$ x_{T-2} x_{T-1} x_T

Figure 4.11: A dilated causal convolutional neural network with two hidden layers, dilation factors $d = 1, 2, 4$ and filter size $k = 3$ [BKK18].

TCNs possess numerous advantages when compared to recurrent architectures [BKK18]. Convolutions within TCNs can be computed in parallel, thus allowing the entire data sequence to be processed. That is not possible with RNNs, where the computation of the output at a specific timestep requires the complete computation of all its predecessors. Moreover, TCNs require less memory during training than RNNs, where partial values of cell-gates need to be stored, and exhibit stable gradients, as backpropagation does not happen through multiple different time samples. In theory the receptive field of RNNs is infinite; in TCNs the field is finite, and its size depends on the number of layers (dilations) and filters used. Apparently, there exists a trade-off between how lightweight the network is, and its ability to capture long-term dependencies in the data. Both aspects are equally important to obtain a scalable and reliable CAN bus intrusion detector. In the remainder of this Section we show that a TCN model is a suitable candidate for this purpose. Furthermore, we argue that an architecture with scalable size is advantageous in a resource limited embedded environment, such as a vehicle.

TCN architecture The proposed TCN to be used for CAN bus intrusion detection follows the general framework from [BKK18] and is shown in Figure 4.12. The network consists of an input layer, three residual blocks, and an output layer. As shown in the figure, the input for the TCN

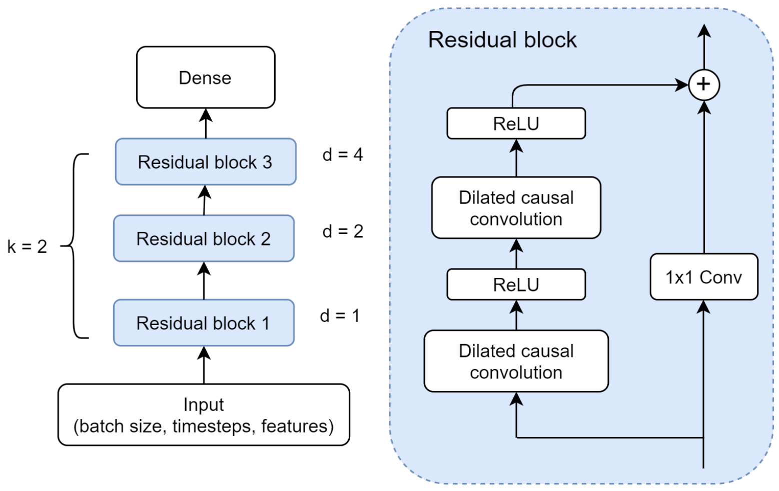


Figure 4.12: Our TCN architecture with three residual blocks with convolutional dilations and filter size of $k = 2$.

must be three-dimensional. Each residual block contains two dilated causal convolution layers each having 64 filters and the same dilation factor d. The Rectified Linear Unit (ReLU) is used as an activation function on these layers. The filter size is kept at the same value of $k = 2$ across all residual blocks. A skip connection is also enabled, which adds the output from the previous layer to the next layer. This is marked by the element-wise addition $\oplus$. Due to zero-padding, this operation may receive inputs that differ in shape. To circumvent this, a 1x1 convolution is added.

The network is kept simple deliberately: no weight normalization or dropout layers have been used. Our main objective here is to investigate whether this lightweight TCN can successfully learn to reconstruct CAN bus signals, and achieve results comparable to or better than other, more complex state-of-the-art classifiers.

Intrusion score and output We distinguish between benign and malicious messages by applying a threshold to the reconstruction loss. We therefore monitor the squared error between the signal value at a given time and its latest reconstructed value. This defines an intrusion score for each signal in a message. To compute an intrusion score per message, we calculate a set of thresholds given by the 99.9th percentile of the validation loss for each signal in the data. A message is then labeled as malicious if one of the signal's intrusion scores exceeds the threshold set for that signal. We opted for this approach to label messages based on individual signal thresholds: in practice, depending on the complexity and correlation of the signals, some may be better reconstructed during training than others.

4.4.3 Experiment design

In this subsection we describe the design of our numerical experiments, including the baseline model, datasets, training process and our choice of evaluation metrics.

Selecting the most suitable baseline For evaluation purposes, we identified the best-performing CAN bus anomaly detection algorithms by scrutinizing recent literature. We used the following selection criteria:

- Unsupervised learning: the algorithm requires no labeled data for training.
- Generalization: the algorithm is easy to generalize, and thus does not depend on data pre-processing such as identifying and pre-selecting specific CAN signals.
- Fully-reproducible: the algorithm needs to be accompanied by sufficient information in order to have a fully reproducible implementation.

To the best of our knowledge, the most recent and suitable candidate is the INDRA framework [KTP20]. It proposes a recurrent autoencoder network that is able to detect CAN messages in which signals have been tampered with. For each message ID one such recurrent autoencoder is trained such that it learns to reconstruct the signals within that particular message ID. This approach is shown to outperform other recent unsupervised methods such as Predictor LSTM [TJL15], Replicator Neural Network [WWSZ18], and CANet [HSDU20], on most attack classes of the SynCAN dataset, in terms of accuracy and false positive rate. Moreover, Predictor LSTM is designed to predict the raw message data in string form, and thus does not directly fall within the scope of time-series-based intrusion detection. Note that the CANet model is also more complex since its architecture combines the LSTM models of individual messages to account for capturing the correlations between different IDs. Finally, the convolutional LSTM proposed in [TLW20] is a promising method for predicting multi variate time series data. However, it was designed for supervised learning which requires labeled data for training and for this reason, it falls outside the scope of this paper.
In view of these arguments, INDRA is the most sensible baseline for comparative performance evaluation.

Datasets

When evaluating machine learning classifiers, it is considered best practice to employ multiple datasets in order to assess the impact of the number of data samples and different features on the model's performance. Moreover, publicly available CAN bus datasets for intrusion detection are labelled differently, either per message ID or per signal. To account for both, we consider two datasets: the SynCAN dataset with message labels and the CrySyS dataset with individual signal labels.

SynCAN dataset The SynCAN (Synthetic CAN Bus Data) dataset was introduced in [HSDU20], and is publicly available[5]. The dataset contains 10 different CAN message IDs, whilst the number of signals in each ID varies between 1 and 4. Overall, the dataset covers 20 signals. The training data spans approximately 16.5 hours of traffic, while the testing data about 7.5. Moreover, testing data includes a 0/1 label per individual message, to indicate whether it is malicious

[5]www.github.com/etas/SynCAN (Last accessed: Oct 1, 2023)

or not. However, there is no indication as to which signal has been attacked within a malicious message. Since this dataset is only meant for unsupervised learning purposes, the training data does not include explicit labels. Finally, the test data is split across six different files, each corresponding to a different simulated attack:

- *Plateau attack:* the value of a single signal is overwritten by a constant value over a certain period of time.
- *Continuous change attack:* the value of a signal is overwritten at a slow pace, such that it increasingly deviates from its true value.
- *Playback attack:* the values of a signal within a time interval is overwritten with the values of the same signal from a randomly selected past interval.
- *Suppression attack:* signal values contained in a certain message ID simply do not appear in the CAN traffic for a period of time.
- *Flooding attack:* messages with a certain ID are sent with a higher frequency to the CAN bus.

Detection of message injection attacks (suppression attack and flooding attack) is not a goal of this Section. Nonetheless, in Section 4.4.4, we evaluated our TCN architectures performance on those as well for a better comparison with the INDRA model.

CrySyS dataset The CrySyS dataset was created by the CrySyS Lab in the context of the SECREDAS project [6], and it is also publicly available[7]. It is significantly smaller compared to the SynCAN dataset, however, the driving environment and the behavior of the vehicle are better known. It contains 7 smaller ($<$1 minute) captures of specific driving and traffic scenarios, and a longer trace ($\sim$ 25 minutes). There are 20 different message IDs in the traces, and the number of signals varies between 1 and 6.

We modified the original CrySyS traces with the attack generator script, also used in Section 4.3.3 to simulate attacks. After we identified the different signals in the traces, using the method presented in [NGMK18], we replaced a chosen signal with a modified value for the second half of the trace. Note that the simple change-to-constant/plateau attack was enough to demonstrate the capabilities of our approach over INDRA (see Section 4.4.4). Also note that we focused on IDs with 1 to 4 signals per message, similar to SynCAN, to be able to compare the results across the two datasets.

Training the models

Training both the TCN and INDRA models required the normalization of signal data (values between 0 and 1), and then re-shaping the input data to three-dimensional. This was done by sliding a fixed-size window over the time series, one timestamp at a time. As in [KTP20], we

[6]www.secredas-project.eu (Last accessed: Oct 1, 2023)

[7]www.crysys.hu/research/vehicle-security (Last accessed: Oct 1, 2023)

Table 4.1: Overview of datasets used in the numerical experiments.

Dataset	Message ID	No. of signals	Train samples	Test samples
SynCAN	2	3	4139826	909869
	3	2	2070144	1884235
	10	4	1380087	610294
CrySyS	280	4	157472	3895
	290	5	15748	389

applied a rolling window of 20 timestamps or, equivalently, of 20 messages, to the training datasets shown in Table 4.1.

The rest of the training parameters were set to the same values as in [KTP20] to ensure an accurate reproduction of the INDRA model. Concerning the optimizer and loss function, both models used the *Adam* optimizer with learning rate 0.0001 and mean square error. The models were trained for 100 epochs with a batch size of 128 on 85% of the training data, whilst the other 15% was kept for validation. An early-stop mechanism terminated the training if the validation loss did not improve in the last 10 epochs. Note that during initial experiments, a higher number of epochs was considered, but the training stopped before the 100th epoch in all cases. All models were implemented using the *keras* and *keras-tcn*[8] libraries in Python 3.7, and trained on a GeForce GTX 960 GPU. The two models have only been trained offline, not on live CAN bus data.

Evaluation metrics

To evaluate the performance of the TCN model, we use the intrusion score defined in Section 4.4.2. The INDRA model uses the same squared error as a signal intrusion score, but applies a generic threshold set to the 99.9th percentile of the validation loss (computed across all signals). The message intrusion score is then given by the maximum signal intrusion score contained in that message, and is then compared to the threshold. We use three standard performance metrics for the evaluation of the models: accuracy, false positive rate and precision. Accuracy measures the ratio of the predicted labels exactly matching the ground truth, and is defined as follows:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}, \tag{4.4}$$

where TN, TP, FN, FP denote the number of true, and false, positives and negatives, respectively. Accuracy gives an indication of the general classification capabilities of a certain model.

The false positive rate (FPR) measures the amount of samples wrongly classified as malicious, whilst in fact being benign. The false positive rate is extremely relevant from the practical

[8]www.github.com/philipperemy/keras-tcn (Last accessed: Oct 1, 2023)

point of view: in the CAN bus context, the messages marked as malicious may need to be further analyzed before deciding on mitigation actions. To keep operation efficient, the false positive rate needs to be minimized as much as possible. Precision, on the other hand, measures the capabilities of the model to actually detect the relevant attacks (positive samples). This is another important quantity to monitor since imbalanced datasets, with far more negatives than positives, may render accuracy a deceiving metric. In fact, CAN bus datasets are usually imbalanced, since most (simulated) attacks have a very short duration. The FPR and precision are defined as follows:

$$FPR = \frac{FP}{TN+FP}, \tag{4.5}$$

$$Precision = \frac{TP}{TP+FP}. \tag{4.6}$$

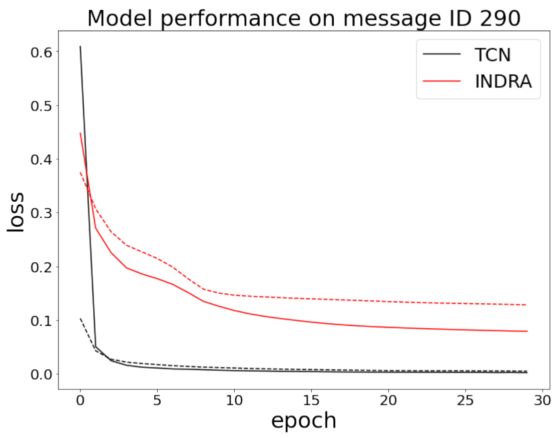


Figure 4.13: Training loss (continuous lines) and validation loss (dashed lines) of the two models on message ID 290 of the CrySyS dataset.

Table 4.2: Accuracy of the models on SynCAN dataset.

Model	Data	Normal	Cont.	Playb.	Flood.	Suppress	Plateau
TCN	ID 2	**0.9977**	**0.8660**	**0.8674**	**0.7678**	**0.8402**	**0.8336**
INDRA		0.9811	0.8584	0.8660	0.7600	0.8347	0.8133
TCN	ID 3	**0.9992**	**0.8664**	**0.8680**	**0.6422**	**0.8390**	**0.8394**
INDRA		0.9965	0.8653	0.8672	0.6420	0.8377	0.8386
TCN	ID 10	**0.9977**	**0.8637**	0.8577	0.7399	**0.8446**	**0.8282**
INDRA		0.9858	0.8546	**0.8638**	**0.7923**	0.8370	0.8100

4.4.4 Results

SynCAN We first assessed the performance of the two models on the SynCAN dataset. The accuracy and false positive rate, calculated for the normal test set and for each attack class, are shown in Table 4.2 and 4.3. A first observation is that TCN achieves a higher accuracy than INDRA in most cases, with the exception of playback and flooding attacks on ID 10. Moreover, the false positive rates are quite low for both models, which can be explained by looking at the precision values in Table 4.4. Overall, there are large variations in the precision values across different message IDs which may be related to how the attacks were performed (target signals chosen, attack duration, etc.) and the different signal correlations. Also, the relatively low precision values in Table 4.4 show that the models manage to capture only a limited set of temporal characteristics of the SynCAN data. This is a direct consequence of the stopping mechanism implemented during training, and in the case of TCN, of the choices made to keep a lightweight architecture. For playback attacks, precision is very low for both models, which leads to the similarly low false positive rates achieved in this class. This is not surprising since during a playback attack, a portion of past data is written over its current values, making the signal look normal, and thus the attack difficult to detect. TCN clearly achieves a better performance than INDRA in detecting continuous attacks. Moreover, for message IDs 2 and 3, TCN detects suppression attacks with a much larger precision compared to INDRA. This result appears to be influenced by the number of signals in the message, since precision significantly decreases as the number of signals increases. As for plateau attacks, the two methods achieve similar results. INDRA is more precise than the TCN model is on detecting flooding attacks. This is an expected result, mainly due to the TCN accurately reconstructing data from a flooding attack since the data values are not altered during such an attack. To sum up, the TCN model is capable of detecting all message modification attacks (continuous change, playback and plateau) effectively. Although detecting attacks which modify the arrival rates of CAN bus messages was not part of the original goal, TCN also proved successful at detecting suppression attacks.

CrySyS The message IDs in the SynCAN dataset contains signals that are physically interdependent, but are very weakly correlated; this also increases the difficulty of the detection task. In order to assess how the two models perform in a different setting, we consider two message IDs of the CrySyS dataset which contains more signals with a strong correlation. Here, similarly to SynCAN, only one signal was attacked. The results are shown in Table 4.5. We notice that

Table 4.3: False positive rate of the models on SynCAN dataset.

Model	Data	Normal	Cont.	Playb.	Flood.	Suppress	Plateau
TCN	ID 2	**0.0022**	**0.0018**	**0.0013**	**0.0026**	**0.0001**	**0.0066**
INDRA		0.0188	0.0121	0.0046	0.0157	0.0101	0.0495
TCN	ID 3	**0.0007**	**0.0009**	**0.0002**	**0.0011**	**0.0004**	**0.0012**
INDRA		0.0034	0.0033	0.0012	0.0033	0.0025	0.0036
TCN	ID 10	**0.0022**	**0.0072**	0.0160	**0.0001**	**0.0011**	**0.0136**
INDRA		0.0141	0.0176	**0.0070**	0.0047	0.0105	0.0447

Table 4.4: Precision of the models on SynCAN dataset.

Model	Data	Cont.	Playb.	Flood.	Suppress	Plateau
TCN	ID 2	**0.4457**	0.2458	0.0205	**0.9027**	0.3022
INDRA		0.1992	**0.3696**	**0.1577**	0.2812	**0.3033**
TCN	ID 3	**0.5231**	0.0000	0.1028	**0.5854**	**0.7809**
INDRA		0.0143	0.0000	**0.3766**	0.3261	0.6192
TCN	ID 10	**0.3706**	**0.1949**	0.000	0.0212	0.2036
INDRA		0.1779	0.1668	**0.9413**	**0.0386**	**0.2224**

Table 4.5: Results for the CrySyS dataset.

Model	Data	Acc.	FPR	Precision
TCN	ID 280	**0.8833**	0.0426	**0.7766**
INDRA		0.7989	**0.0000**	0.0000
TCN	ID 290	**0.9159**	0.0687	0.7701
INDRA		0.8617	**0.0378**	**0.7755**

both models still achieve high accuracy and a low false positive rate, with TCN showing a high precision for both attacks, as opposed to INDRA, failing to detect the attack in message 280.

Results combined The simple TCN architecture achieves a slightly better accuracy compared to the INDRA model on both datasets. A remarkable achievement of TCN is the significant reduction of false positives (by a factor of 10) in nearly all cases: this translates to a more reliable detector in practice. Further advantages of the TCN are that it is quick to train, has a much smaller resource need, and achieves in general lower training and validation loss (see Figure 4.13 for an example).

4.4.5 Summary

In this section, we examined the applicability of temporal convolutional networks to CAN bus anomaly detection, with a focus on message modification attacks. To this end, we proposed a lightweight TCN, and showed that its classification performance compared favorably to the state-of-the-art baseline INDRA across different datasets and attack classes. Specifically, we demonstrated that our computation-efficient and compact TCN model achieves similar or better accuracy, while reducing false positives with an order of magnitude. This shows that TCNs have a great potential both in modeling CAN bus signal and being deployed in practical settings.

Future work First of all, the elimination of the early termination mechanism would potentially yield better performance; early termination was necessary in our experiments due to hardware-related constraints. Second, the TCN architecture was kept very simple on purpose to ensure a computationally lightweight model. However, the learning abilities of the network could be improved by increasing the filter size and the dilation factor between causal convolutions, and by stacking additional residual blocks together. Third, it is worthwhile to investigate how message-based and signal-based intrusion thresholds, and the underlying intra-message signal correlation influence the performance of both models for different attack classes. Finally, correlations between signals across different message IDs could be considered leading to a more accurate representation of normal CAN bus behaviour. To this end, an architecture combining multiple TCN blocks (modeling individual message IDs a la CANet [HSDU20]) could be used.

4.5 Summary

In this chapter, we focused on detecting anomalies in CAN networks. We investigated new methods to detect the attacks presented in Chapter 2.

Our anomaly detection on compressed traffic, presented in the first section, further extends the defense against injection attacks. In the following two sections, we proposed two new methods for defending against message modification attacks. The first one can efficiently detect anomalies by evaluating the signal correlations. In contrast, the second one processes the signals individually with a machine learning method to predict the future signal values and then detect anomalies by comparing the prediction with the measured values.

In conclusion, the presented methods address different attacks against CAN networks, and their combined or separate application can significantly improve vehicle security.

Chapter 5

Privacy problems

Data is constantly being generated within the systems of vehicles, particularly in contemporary cars equipped with an array of controllers and sensors. These vehicles capture an extensive range of signal values during operation, offering a comprehensive insight into the vehicle's performance and its driver's behavior. As discussed in preceding sections, storing and managing this influx of data is becoming increasingly crucial, particularly in identifying and mitigating potential cyber-attacks targeting automotive systems.

However, beyond its significance in fortifying cybersecurity measures, the data harvested from vehicles is versatile. Rather than solely serving defensive purposes, it possesses the potential to fuel a new wave of data-driven services tailored to enhance user experience and convenience. This paradigm shift opens new ways for innovative applications within the automotive industry, paving the way for personalized services and intelligent recommendations based on intricate data analytics.

Yet, a pressing concern emerges amid the promising prospects of data-driven services - protecting personal data. With access to an abundance of sensitive information, including driving patterns, vehicle usage habits, and even geolocation data, a paramount obligation arises to safeguard individual privacy rights. In their pursuit of offering tailored services, manufacturers must navigate a delicate balance between utility and privacy, ensuring that data utilization remains within the bounds of legal frameworks and ethical standards.

The potential applications of vehicle-generated data are manifold. Manufacturers can leverage this information to deliver proactive maintenance recommendations, optimize performance, and enhance the overall driving experience for vehicle owners. Moreover, third-party entities may capitalize on behavioral insights from driver data to offer supplementary services, ranging from personalized insurance premiums to location-based promotions.

However, the exploitation of such data must be underpinned by a steadfast commitment to compliance with pertinent regulations and adherence to ethical principles. Striking a harmonious balance between innovation and data protection is imperative to foster trust and sustain the integrity of data-driven services in the automotive domain. Thus, while the potential benefits of leveraging vehicle data are undeniable, they must be pursued with a commitment to ethical conduct and regulatory compliance, ensuring that the privacy and autonomy of individuals remain sacrosanct amidst the ever-evolving landscape of automotive technology.

The volume of data generated and collected by networked information systems is already staggering and ever-increasing. With the advent of data analytics and machine learning, this data inflow enables thousands of data-driven services which make our lives easier everyday. On one hand, the modern vehicle is such a networked information system, generating abundant data both on its in-vehicle network and, when V2X-enabled, towards other vehicles and the smart infrastructure. On the other hand, a prominent service category is, e.g., location-based services which help us navigate efficiently, hail a ride swiftly and cheaply, and get informed on potentially relevant businesses nearby. However, such services do not come for free: more often than not we pay for them with our personal data.

When service-enabling data is personal, the data controller (and data processors involved) must observe regional and national data protection regulations; in Europe this points to the European General Data Protection Regulation or the GDPR [PtC16]. While we do not intend to introduce GDPR fully in this chapter, it is essential to outline three of its concepts to facilitate comprehension. First, GDPR demands legal basis for data controllers to use personal data: the primary way to achieve this is getting the *informed consent* of the data subject. Second, the term "personal data" is defined as "any information relating to an identified or identifiable natural person; an identifiable natural person is one who can be identified, directly or indirectly." *Singling out* is a fundamental method explicitly mentioned to identify a person in data; only data that do not allow singling out the record of *any* individual may be exempt from the GDPR [CN20]. Third, certain personal data is deemed *sensitive*, and thus enjoy even stronger protection: personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs; trade-union membership; genetic data, biometric data processed solely to identify a human being; health-related data; and data concerning a person's sex life or sexual orientation.

The research question for chapter is: *Should CAN logs be treated as personal data, as by offloading data processing to third-party providers, there might be strict legal requirements for data handling?*

Vehicular data often *directly* carries personal or even potentially sensitive information such as location data, heart rate, or driver fatigue among others. Indeed, four spatio-temporal data points of an individual's daily trajectory identify 95% of all individuals uniquely [DMHVB13]; this facilitates singling out. Also, as evidenced by the publicly released NYC taxi dataset, individual trips might allow the inference of the driver's (or the passenger's) religious beliefs or health status[1]; these are sensitive personal attributes. Moreover, the Strava incident[2] showed us that location data create patterns on the aggregate that can still give away an individual's personal information.

Meanwhile, data collection through the in-vehicle network, notably the CAN bus, is ongoing by OEMs (Original Equipment Manufacturers) for maintenance, recall, and, increasingly, monetization purposes. Therefore, even third-party service providers can get access to that data; vehicle data hubs collect and standardize such data and sell it for applications including insur-

[1]fpf.org/blog/implications-of-broad-data-collection-by-the-nyc-taxi-limousine-commission (Last accessed: Oct 1, 2023)

[2]www.theguardian.com/world/2018/jan/28/fitness-tracking-app-gives-away-location-of-secret-us-army-bases (Last accessed: Oct 1, 2023)

ance[3], traffic management, electric vehicle infrastructure planning, fleet management, advertising, mapping, city planning, and location intelligence[4]. The vehicle data market is predicted to be worth between 300 billion and 800 billion USD by 2030[5]. However, if CAN data i) reveals the trajectory and location of the vehicle and its passengers and/or ii) enables singling out a specific driver, it should be treated as personal data; in this case the GDPR should be in full effect. Adding fuel to the fire, CAN data is often collected without the informed consent of the owner/-driver and without even the alternative of opting out. Such a hypothesis carries great weight: for handling private data, GDPR defines strict requirements for data controllers and processors (reasonable technical and organizational security safeguards, Art. 24 and 32 GDPR, in some cases data protection impact assessment, Art. 35 GDPR) and rights for data subjects (right to access, right to erasure, right to object, as per Art. 12-23 GDPR). Satisfying these requirements carry a substantial cost and might even perturb the century-old process of how cars are sold. Moreover, not meeting these requirements can result in hefty fines, evidenced by much publicized cases in other domains[6].

To avoid potential repercussions of data breaches and legal non-compliance, data controllers and processors either apply anonymization, or ask drivers' consent in order to process their data. However, since vehicle data are composed of the fine-grained measurements of a multitude of vehicular sensors, they are inherently high-dimensional, potentially unique to a driver, and therefore challenging to anonymize without significant utility loss. Standard pseudonymization techniques that remove only direct identifiers but keep the sensor measurements intact generally do not work, and vehicular data are no exception[7]. Moreover, the "ad-hoc" application of different standard anonymization techniques, such as the aggregation or removal of apparently sensitive measurements, often results in volatile, empirical privacy guarantees to the population as a whole but fails to provide a strong, worst-case privacy guarantee to every single individual; a specific requirement of privacy regulations. Indeed, as the measurements of different sensors are strongly correlated, we demonstrate later that removing some apparently sensitive measurements (e.g., GPS coordinates) may still allow their inference from the data of other "proxy" sensors (e.g., steering wheel angle and speed). This not only makes anonymization very difficult, but it also stifles fine-grained consent control which provides a common legal basis to process personal data if anonymization is not viable. However, data controllers (can) hardly explain the potential privacy implications of sharing such interdependent attributes which means that a driver's consent will arguably not be *informed* and hence becomes invalid.

In this chapter, we show exactly that indeed, analyzing CAN logs and using the combination of different sensor measurements make it possible infer the trajectory of the vehicle even if precise GPS locations cannot be accessed. Specifically, our contribution is twofold. First, we reconstruct both short (microtracking) and long (macrotracking) driving routes (including destination) only from the speed, steering wheel position, and the starting location of the vehicle with high accuracy. Second, we demonstrate that intuitive but ad-hoc anonymization methods

[3]www.nationwide.com/personal/insurance/auto/discounts/smartride (Last accessed: Oct 1, 2023)
[4]themarkup.org/the-breakdown/2022/07/27/who-is-collecting-data-from-your-car (Last accessed: Oct 1, 2023)
[5]www.capgemini.com/insights/research-library/monetizing-vehicle-data (Last accessed: Oct 1, 2023)
[6]www.cnet.com/tech/gdpr-fines-the-biggest-privacy-sanctions-handed-out-so-far (Last accessed: Oct 1, 2023)
[7]www.vice.com/en/article/4avagd/car-location-data-not-anonymous-otonomo (Last accessed: Oct 1, 2023)

providing empirical average-case privacy guarantees cannot be relied on to transform CAN logs into anonymous data exempt from the GDPR [PtC16], while still preserving meaningful utility.

The rest of this chapter is organized as follows. Section 5.1 provides an overview on related research works. Section 5.2 describes the attacker model we consider in this chapter. Section 5.3 shows how we can reconstruct both the short- and long-term trajectory of a vehicle based on its CAN log. Section 5.4 investigates multiple naive signal processing and statistical methods meant to provide defense against tracking efforts, shows that these are ineffective for CAN data. At last, Section 5.5 concludes this chapter.

5.1 Related Work

The automotive location privacy problem has been researched for well over a decade. A significant boost to this field was the emergence of vehicle usage based services such as new insurance constructions. These services offer a lower cost for users who shared their vehicular data, while promising user privacy via only collecting driving behaviour information and not location data. Unfortunately, Dewri et al. [DAET13] already showed that the collection of vehicle parameters can reveal the driving destination without a GPS signal. In the following years multiple papers showed that various types of vehicular data can be used to reconstruct driving traces partially, or completely. Gao et al. [GFS$^+$] showed that based on a starting position and the speed signal, their elastic pathing algorithm could predict destinations with an error smaller than 500m for 26% of the cases. This is significantly less accurate than our approach which has an error of 5-100 meters for endpoint reconstruction (see Table 5.1). Zhou et al. [ZCL$^+$17, ZDZ$^+$19] used also only the starting position and the speed signal to reconstruct driving traces. Their approach combines these with additional knowledge about the environment, such as road condition, real-time traffic, and even traffic regulations, to filter out the potential trace candidates. Results showed that the real route was in the top 10 candidate routes with 60% probability. Kaplun et al. [KS19] showed that without the detailed speed signal, it was also possible to reconstruct the trajectory with a high probability. They used cornering events, average speed and total driving time for reconstructing trajectories. Waltereit et al. [WUW19] showed that based on the distance of each driving section and turning directions, it was possible to find the correct trajectory in a large area of a map without knowledge of starting point or destination. Our approach works with similar type of data, however, both the objective and the internal operation of our algorithm differ significantly. The authors in [WUW19] assumed that the signals extracted from CAN messages are accurate and reconstructed route segments separately. However, as we show, signals are inherently erroneous which results in inaccurate reconstruction for longer trajectories. Pesé et al. proposed the RoCuMa algorithm for route identification based on only the steering wheel position in [PPS20]. The algorithm first creates a road curvature database of the analysed city, and then tries to identify the curves during the trips. RoCuMa achieved over 70% accuracy in the most suitable regions, but was completely inefficient in others (e.g., Manhattan like grids). The problem RoCuMA tried to solve is similar to ours discussed in Section 5.3.2, as the steering wheel position can be extracted from the CAN traffic in most cases. Even though the objective is similar to ours, our approach is more robust, i.e., its correctness does not significantly depend on the selected region, and also more accurate. Sarker et al. [SQS$^+$20] proposed Brake-Based

Location Tracking (BBLT). Their approach extracts the break signal values from the CAN bus, which are then used to reconstruct the movement of the vehicle. The extracted signal values are processed in three steps. First, they categorize the signal sub-sequences into different driving maneuvers, then based on the maneuvers they estimate the parameters of the movement, such as the number of intersections or the speed profile. Finally, they search for trajectories on the regional map, that are the closest to the determined parameters. The goodness of a candidate edge is determined by a custom score function. Their evaluation showed that in 89% of the cases the reconstruction was successful.

Finally, the location privacy of a driver can also be violated via smart devices carried on board while driving. E.g., Han et al. [HON+12] described a location inference technique using smartphone accelerometers to successfully locate drivers within a small radius of the true location.

In summary, prior works use different signals (only speed [DAET13, KS19, GFS+], speed and traffic information [ZCL+17, ZDZ+19], only steering wheel position [PPS20], braking [SQS+20], acceleration [HON+12]) for reconstruction, while we rely on speed and steering wheel position extracted from CAN messages exclusively. Earlier approaches first partially, or completely reconstruct the trajectory from the signals, then fit the result to the map for correction. By contrast, we always predict the next location from only the previous one using speed and steering wheel position, and immediately correct the prediction with map data, that is, we do not let the error accumulate over successive predictions. As we show later, this approach provides larger reconstruction accuracy even for longer trajectories than what has been reported before. Our technique is simple and efficient, and works even on resource-constrained devices as long as map data can be stored on the device.

5.2 Adversary model

The adversary aims to reconstruct the trajectory of a specific individual's vehicle from its CAN data as accurately as possible. The adversary may know the identity of this driver and can only access the vehicle's CAN data, except its GPS trajectory, as well as the geographical map of the region R (e.g., a bounding box of the entire trajectory). This is the only background information the adversary has about the driver's whereabouts. In particular, the GPS location is not present in the CAN data explicitly but can only be inferred indirectly from other signals extracted from the CAN log, such as speed, steering wheel position, as well as the start of the trajectory which is presumably observable by the adversary. These are not far-fetched assumptions since the majority of available cars nowadays still do not have built-in GPS receiver, or if they do, it is unlikely that potentially sensitive location information is shared with third parties. The adversary aims to localize the trajectory within R with as little error as possible, which is measured as an average distance between the reconstructed and the original trajectory. Although larger error implies stronger privacy preservation in general, more precise assessment of the potential privacy risks is difficult due to the varying contextual factors and therefore is beyond the scope of this work. For example, an error within a few hundred meters may not be enough to decide if the driver visited a hospital or a church in a city, but is sufficient to infer which village it traversed in the countryside.

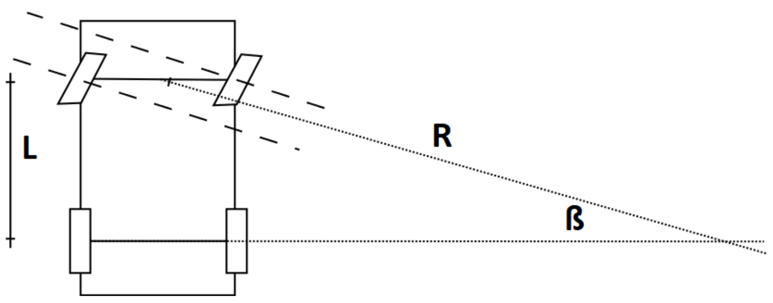


Figure 5.1: Computation of the radius as $R = L/\sin(\beta)$, where L and β denote the vehicle length and wheel position, respectively.

5.3 Trajectory reconstruction

In this section, we show that by releasing CAN data, vehicles and thus drivers become traceable. The tracing of the vehicle is achieved in two steps. First, we show how a vehicle can be traced accurately over short distances based exclusively on CAN messages. We refer to this concept as microtracking. Second, we show how to extend tracing for longer trips using additional, publicly available information. We refer to this second problem as macrotracking. Recall that positioning data (GPS) is not included in the CAN traffic.

5.3.1 Microtracking

The objective of microtracking is to reconstruct the trajectory of a given vehicle over a short distance (10-100 meters). If the initial position is of the vehicle is known, the next position at any time can always be predicted from the previous one using the speed and the direction of the movement. Fortunately, these values can be computed solely from the signals of speed and steering wheel position which are directly accessible in the vehicle's CAN messages transmitted and recorded during the trip.

First, the speed and the steering wheel position signals are extracted from the CAN message stream. These can be relatively easily found even for an unknown vehicle based on the method described in [LÁBS19]. Next, the extracted signals together with the timestamps of the recorded CAN messages are used to calculate small delta movements of the vehicle. If the vehicle is going along a straight line, then the calculation is straightfoward (distance = velocity × elapsed time). If the steering wheel is not in the central position, then the radius of the path segment can be calculated based on the geometry of the vehicle as illustrated in Figure 5.1. In particular, the radius R can be calculated from the sine of the angle β of the wheels as $R = L/\sin(\beta)$, where L denotes the distance between the two axles. The wheel position is inferred from the steering wheel position, which is extracted from the CAN messages, through measurements. Finally, the small delta movements are aggregated sequentially to uncover the trajectory of the vehicle. We validated our method successfully in different short but characteristic test scenarios.

In general, this type of microtracking is reasonably accurate for short trajectories, but it can be quite inaccurate for longer trajectories as the small approximation errors stemming from the noisy and often biased sensor measurements can add up over a longer trip.

5.3.2 Macrotracking

Next, we describe the reconstruction of the movement of a vehicle from CAN traffic captures over longer trips (>100 meters). To achieve this goal, we use some auxiliary information in order to mitigate the problem of error accumulation. As for microtracking, the speed and steering wheel angle values extracted from the CAN messages are required for the reconstruction. Additionally, the starting position and the initial heading are also a prerequisite for our algorithm. Provided with these input data, we show that the trajectory of a vehicle can be effectively reconstructed revealing the destination of the drive, which constitutes a privacy breach with respect to the driver. This implies that CAN logs have to be handled or processed carefully to avoid this privacy issue and comply with data protection regulations.

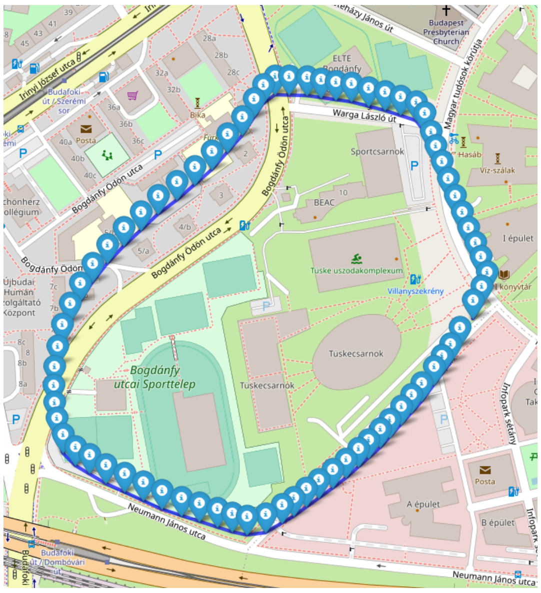
Cumulative errors in microtracking

Our microtracking can be considered as a dead reckoning process which estimates the vehicle position from the previous one using inherently erroneous sensor measurements. Indeed, these measurements are (1) noisy (e.g., steering wheel position, wheel speed measurements are not always accurate due to random environmental factors such as wheel slippage or surface irregularities), (2) potentially biased (e.g., deliberately higher velocity is reported for safety reasons), or (3) some parameters of the vehicle may not be known exactly (e.g., axle distance). Even if these errors result in a small deviation relatively to the previous position, they accumulate over successive predictions making the recovery of longer trajectories very inaccurate. To mitigate this error accumulation, we first performed a thorough calibration by multiplying the extracted values of speed and steering wheel position by a correction factor so that they match the manually measured ground truth values. Although this improves the results, the problem of error accumulation still persists, which is also illustrated by Figure 5.2a; some parts of the trajectory are off the road and still far from the original path.

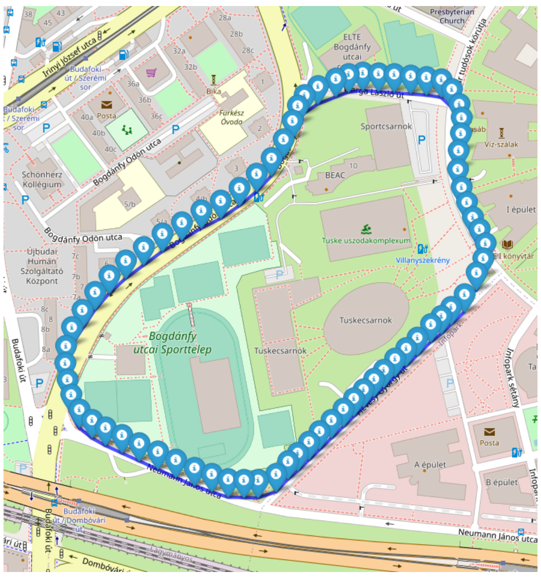
In fact, as long as location is predicted only from inaccurate internal parameters and measurements of the vehicle, it remains subject to cumulative errors. In engineering practice, this problem is generally solved by periodically performing external measurements to correct the prediction and thus reduce the accumulated errors. Similar solutions are used, e.g., for the guidance of spacecraft systems [LMS82]. Next we show how map data can be used to perform periodical measurements to improve the prediction of our model.

Measurements based on map data

The state of a vehicle at any time is characterized by its heading and position. Following our prediction model in microtracking, the state is predicted only from potentially inaccurate internal parameters (axle distance) and sensor readings (speed and steering wheel position), referred to as *model-based prediction* in the sequel. However, using a map allows us to regularly correct the predicted state of the vehicle, i.e., reducing the accumulated errors. In particular, if the vehicle should always be on a road, then a predicted position which is off the road can be corrected. Similarly, the direction of movement should also be aligned with the road, allowing corrections for the heading as well.



(a) C1 reconstruction without map



(b) C1 reconstruction with map

Figure 5.2: C1 test trajectories. Map-corrected trajectories (right) follow the roads more faithfully than only model-based trajectories produced by microtracking (left).

Starting from the position predicted by our microtracking model only from internal measurements as described in Section 5.3.1, we calculate a new position on the map by projecting this model-predicted position on the nearest road segment. This gives us a new, corrected position and heading value aligned with the road that we use in our improved algorithm as a precise external measurement. Along roads without intersections, this approach works smoothly and prevents error accumulation. However, near intersections, it might be unclear which the correct road segment is for the projection. To address this duality in our algorithm, we use a simple linear estimation of the next state as properly weighing both the model-predicted and map-corrected states based on the nearby conditions: the model-predicted states have larger weight near intersections, while map-corrected states have larger weight away from the intersections.

More specifically, given the model-predicted and map-corrected position and heading values, denoted by $state_{model} = [\mathsf{pos}_{model}, \mathsf{head}_{model}]$ and $state_{map} = [\mathsf{pos}_{map}, \mathsf{head}_{map}]$, respectively, the next state of the vehicle is computed as their linear combination: $w \cdot state_{map} + (1-w) \cdot state_{model}$, where $w = \min(d_{model}, \gamma)/\gamma$ is the map weight, and d_{model} denotes the distance from pos_{model} to the nearest intersection. In other words, if the model-predicted position is within a distance of γ to the nearest intersection, then its weight is inversely proportional to the distance, otherwise mainly map-corrected position and heading are used for prediction.

Our linear estimation is a simple instantiation of the Kalman filter [WB95], which is a powerful tool to estimate past, present or future states of a system with uncertainties. It needs to have a dynamic model of the target system and multiple sequential measurements to form its estimations. A weighted average is calculated over the output of the dynamic model and measurement results, taking into account the uncertainty of said measurement. In our algorithm, the dynamic model is the movement reconstruction of the vehicle in microtracking when only internal parameters and sensor readings are utilized. The measurements are the projected positions on the map, and the uncertainty of the measurement can be approximated with the distance to the closest intersection.

The pseudo code to reconstruct the movement of a vehicle is presented in Algorithm 2. First, the next state is always predicted from the previous state with model-based prediction as in microtracking (Line 5-8), and then map-based correction (Line 9-14) is only performed if the distance from the last correction is sufficiently large (Line 10 in Alg. 2). Executing map-based correction after capturing every single CAN message with a speed or steering wheel position value is unnecessary and would prolong reconstruction significantly.

The result of a successful reconstruction of our C1 test is shown in Figure 5.2b. Although a slight deviation from the road can be observed near intersections due to model-based prediction, these errors are corrected and hence do not accumulate as moving away from the intersection due to the increasing weight of map-based correction (γ is set to 150 meters).

Algorithm 2: Macrotracking for CAN logs

```
   Input: starting position and heading value, CAN log
   Output: Reconstructed trajectory $\mathbb{T}$
1  initialize current state to starting position and heading;
2  load data from CAN log;
3  filter relevant messages;
4  while there is message to process do
5      Model-based prediction:
6          extract speed and steering wheel position from messages;
7          compute heading from axle distance and steering wheel position;
8          calculate next state from current state using heading and speed;
9      Map-based correction:
10         if distance from last correction > minimum required then
11             find nearest road segment on map;
12             project current position and heading to selected road segment;
13             update map weight $w$ based on distance from closest intersection;
14             update next state using the projected state with map weight $w$;
15     append next state to reconstructed trajectory $\mathbb{T}$;
16     update current state to next state;
```

We built our algorithm on OpenStreetMap[8], which we accessed using the OSMnx library [Boe17]. This service allowed us to get access to precise position and shape information of road segments. We projected the map and all coordinate data in OSMnx from the World Geodetic System 1984 (WGS84)[9] (used in maps and GPS) to a Spherical Mercator projection coordinate system (EPSG:3857)[10] to be able to perform more precise distance calculations in 2D.

[8]https://www.openstreetmap.org (Last accessed: Oct 1, 2023)
[9]https://epsg.io/4326 (Last accessed: Oct 1, 2023)
[10]https://epsg.io/3857 (Last accessed: Oct 1, 2023)

Validation

The accuracy of our algorithm depends on the correctness of model-based prediction and the density of the road network. On one hand, areas with many intersections does not allow the map based corrections to improve the model prediction as much, therefore the reconstruction error will dominate over longer distances. On the other hand, if the trajectory of the drive follows long sections without intersections, our algorithm will hardly suffer from any errors.

We defined two metrics to evaluate the accuracy of our algorithm as follows:

1. *Endpoint reconstruction error:* we measure the distance (in meters) between the actual endpoint of the movement and the destination predicted by the algorithm.

2. *Average trajectory reconstruction error:* along the actual movement of the vehicle (the ground truth trajectory) we calculate test points every 15 meters. At each test point we find the closest point in the reconstructed trajectory and measure the distance to the test point. Finally, we calculate an average of the measured distances.

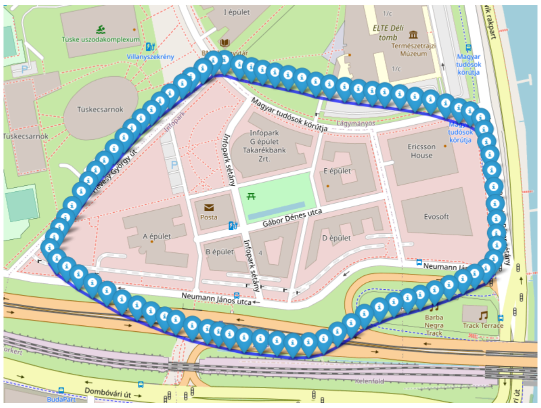
Table 5.1 shows data about our test cases. In these tests, we drove along different circular trajectories to be able to visually show the result of our first metric as well. We also counted the number of decision points (intersections), where the algorithm had to make a correct decision. In the C3 test case, we drove with frequent movements of the steering wheel even on straight road segments to make the reconstruction harder. The table shows that our algorithm was able to reconstruct the trajectories with only small errors. Furthermore, the corresponding Figures (5.2-5.4) show that the reconstructed trajectories follow along the roads actually used, with small deviations only around intersections.

Privacy implications for CAN logs

In this chapter, we showed that it is possible to reconstruct the movement of a vehicle for both short and long distances from CAN messages. This can be a valuable forensics tool, e.g., in case of accident reconstruction; but it can also cause a privacy breach if CAN logs are not handled with proper care and/or they fall into the wrong hands. If the starting position and heading is known, then our algorithm can effectively reconstruct a vehicle' trajectory, revealing private information about the driver. Given that personal location information is categorized as sensitive data in the GDPR, data controllers, whether OEMs or third-party companies, managing such data have to comply with strict requirements. This comes with added cost and operation complexity; yet, non-compliance could result in prohibitively large fines.

Table 5.1: Summary of macrotracking test cases

Test case		Average trajectory reconstruction error (meter)	Std. deviation of error (meter)	Endpoint reconstruction error (meter)	Total distance travelled (meter)	Number of decision points on map
C1 (Figure 5.2)	without map	30.2	25.13	9.3	2025.17	20
	with map	**9.37**	**8.99**	**4.2**	2039.11	20
C2 (Figure 5.3)	without map	39.37	34.02	**35.58**	2139.03	18
	with map	**9.13**	**8.74**	41.12	2158.48	18
C3 (Figure 5.4)	without map	55.04	36.07	82.17	1751.07	19
	with map	**7.45**	**6.05**	**6.05**	1817.81	19


(a) C2 reconstruction without map

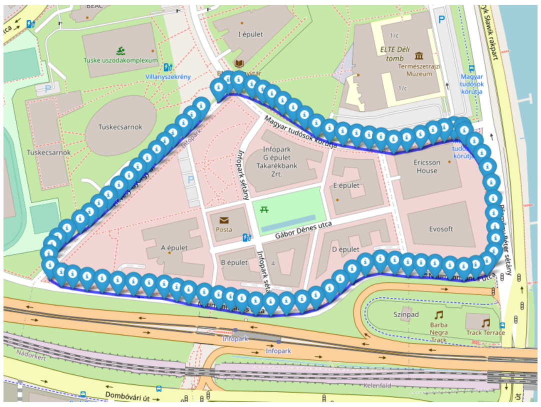
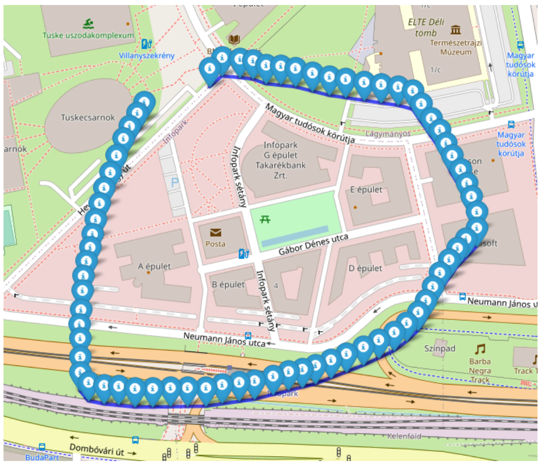

(b) C2 reconstruction with map

Figure 5.3: C2 test trajectories


(a) C3 reconstruction without map

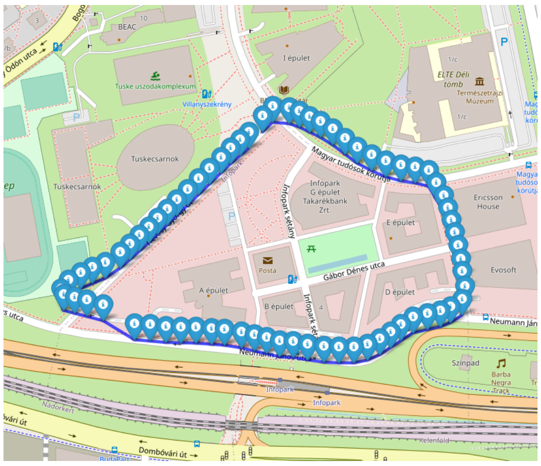

(b) C3 reconstruction with map

Figure 5.4: C3 test trajectories

5.4 Potential defenses

In this section, we describe and evaluate different defense techniques against location reconstruction (macrotracking) described in Section 5.3.2. We consider here some well established signal processing techniques including low pass filtering and smoothing which might be tempting to employ by a data controller or processor in order to distort CAN data so that unique features of traces are no longer recognizable. However, as we show, such techniques fail to provide strong privacy guarantees, or introduce so much distortion to counter our attacks that renders the anonymized data practically useless.

5.4.1 Smoothing

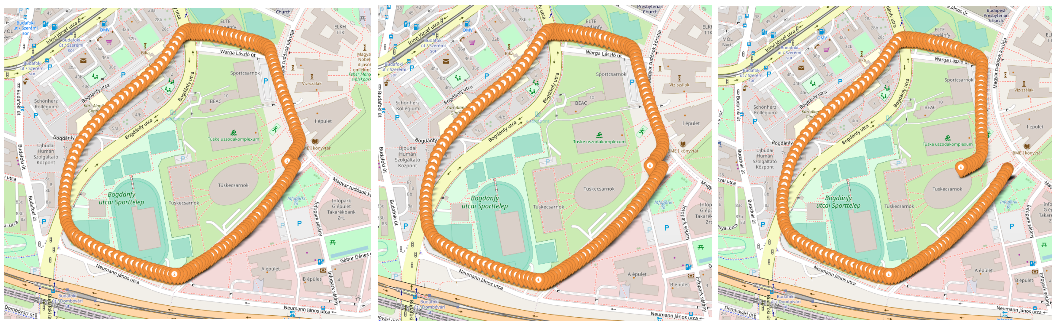
Smoothing is a type of downsampling technique. It is used to remove short-term fluctuations and highlight longer-term trends in the signal (or time-series). It has many variations, the main idea is to shift a fixed-size moving window through the signal and apply a transformation on each window, then publish the transformed signal. We apply a moving window by passed time and not by data points, i.e. the average of the data points that is in a fixed time frame (called *smoothing window*), whose size in time is given as a parameter w, is calculated and used as a single replacement of all points within the given time frame. As the mean is reported per window, consecutive windows do not overlap. This technique is expected to smooth out local variations of every signal within w seconds. Therefore, as long as such local variations correspond to unique features of a trace, the transformed signal should mitigate tracking.

Evaluation

We evaluate our algorithm on the smoothed data produced with different values of smoothing window size w. The goal is to find a window size where the accuracy of macrotracking is sufficiently small but the accuracy of the transformed data is still meaningful.

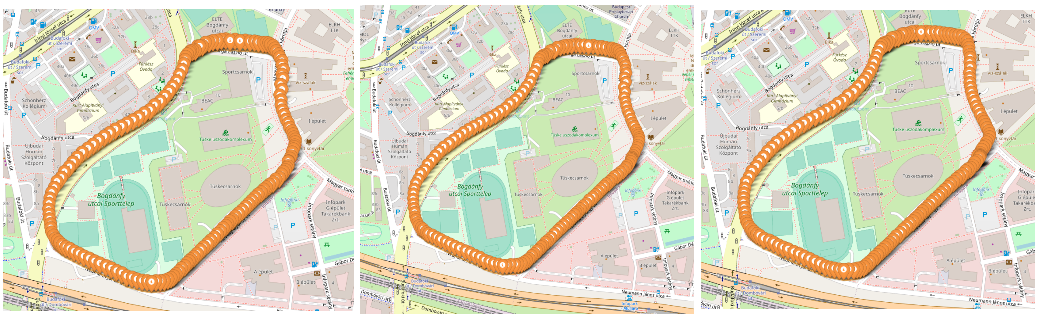
Figure 5.5, Figure A.15, and Figure A.17 show the effect of smoothing on the model-based trajectory reconstruction without map correction (i.e., microtracking) for the C1, C2, and C3 cases respectively. Although smoothing negatively impacts reconstruction, applying smoothing even with the largest window size still allows a relatively accurate reconstruction of many parts of the trajectory. Interestingly, due to an encoding of the steering wheel value in the messages, applying smoothing on this signal has an inverse effect: it results in sharper turns than in the original trace.

Our macrotracking algorithm with map correction is significantly more accurate than only model-based reconstruction and can successfully reconstruct the original traces in all cases. In the C1 (Figure 5.6) and C2 (Figure A.16) cases, increasing the window size also increases the reconstruction error, but all results can still be considered as successful reconstructions. Although the C3 test case is driven with frequent steering wheel changes (as described in Section 5.3.2), the effect of these changes are reduced by smoothing, and therefore the reconstruction results are actually improved with a larger window size (Figure A.18). Table 5.2 contains the trajectory reconstruction errors with their standard deviation and the endpoint reconstruction error for all test cases with three different window sizes.



(a) Smoothing window: 0.201s (b) Smoothing window: 1.608s (c) Smoothing window: 6.4s

Figure 5.5: C1 test case macrotracking result on smoothed data without map



(a) Smoothing window: 0.201s (b) Smoothing window: 1.608s (c) Smoothing window: 6.4s

Figure 5.6: C1 test case macrotracking result on smoothed data with map

Table 5.2: Effects of smoothing on the location tracking algorithm.

Test case	Smoothing windows size (second)	Average trajectory reconstruction error (meter)	Std. deviation of error (meter)	Endpoint reconstruction error (meter)
C1 (Figure 5.6)	0.201	32.2	26.4	22.76
	1.608	33.51	26.97	33.02
	6.4	38.58	27.97	132.94
C2 (Figure A.16)	0.201	37.75	28.68	75.74
	1.608	38.92	32.83	76.72
	6.4	42.22	32.47	126.84
C3 (Figure A.18)	0.201	50.78	32.74	64.71
	1.608	47.53	29.6	66.52
	6.4	20.47	18.15	70.09

5.4.2 Low pass filtering

Low pass filtering is a common technique not only to compress signals, but to reduce noise, eliminate aliasing, or attenuate resonances [Ell12] without heavily degrading utility. Moreover, with the growing need for data privacy, low pass filtering has been used for signal anonymization as well [CHCMB19]. Low-pass filters attenuate or eliminate all signal components above a specified frequency. By deleting these high frequency components, one can get rid of the idiosyncrasies of the signal and end up with the more general parts. Unlike smoothing, low pass filtering is expected to provide a finer-grained control over utility loss, and the mean squared error is precisely quantifiable since the transformation is orthonormal and therefore preserves the L_2-norm of the signal.

In this section, we show that low pass filtering still implies mostly unsatisfactory privacy guarantee in practice. Below we explain how we applied a low-pass filter on CAN bus signals with different degrees of utility, then we evaluate the resulting filtered signals with our macrotracking algorithm.

We apply low-pass filtering as follows. First, the signal is transformed to its frequency domain using orthonormal Discrete Cosine Transform (DCT) which has better energy compaction property than other Fourier-related transforms. After DCT transformation, the number of removed high frequency components is determined. In general, the more components is dropped from the signal the lower the utility becomes. The resulting utility is measured by calculating the normalized euclidean distance between the original and the low passed signal, i.e. we delete as many of the highest frequency components as many needed to reach a predefined *error distance* (aka., reconstruction error) from the original signal. As orthonormal DCT preserves the L_2-norm of the original signal, the transformed signal has the same L_2-norm as the original one. For example, in order to have a reconstruction error of 10% at most, the maximum number of the highest frequencies of the transformed signal are removed such that the L_2-norm of the removed components is not greater than the 10% of the total L_2-norm of the whole signal. A naive algorithm would start deleting components one-by-one starting from higher to lower frequencies, instead we apply logarithmic search and calculate the resulting error after each iteration. Once the desired error rate is reached, the filtered signal is transformed back to the time domain and published.

Evaluation

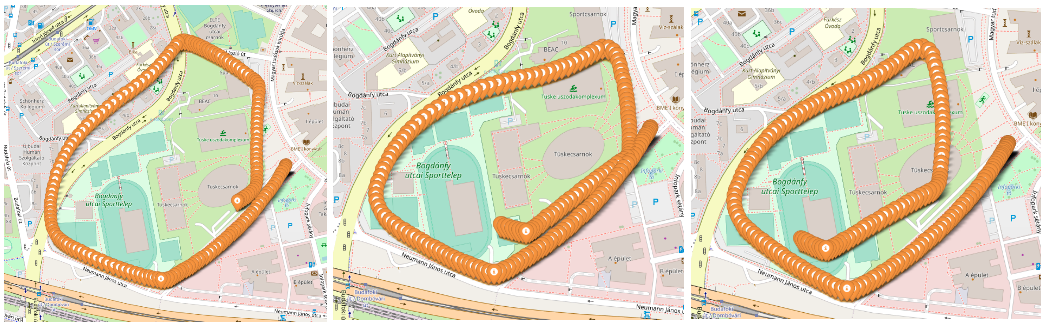
Figure 5.7 shows the result of trajectory reconstruction without map (i.e., only model-based prediction) after low-pass filtering the C1 test case. In comparison with smoothing, low-pass filtering with the chosen parameters distort the original traces more significantly. The counter-intuitive changes of the turn angles can also be observed here. Similar results can be observed in Figure A.19 about the C2 case, and in Figure A.21 about the C3 case.

Figure 5.8 shows that our macrotracking algorithm (i.e., reconstruction with map correction) is capable of reconstructing the original C1 trajectory if the prescribed error rate of low pass filtering is below 40%. Above this value, the algorithm cannot reconstruct the vehicle movement in one of the intersections at least. We also observed similar behaviors in the C2 (Figure A.20) and C3 (Figure A.22) test cases.

The accuracy of the reconstruction for all cases with different amount of low-pass filtering are depicted in Table 5.3. In summary, the reconstruction is successfully prevented at a low pass filtering error of 40%.

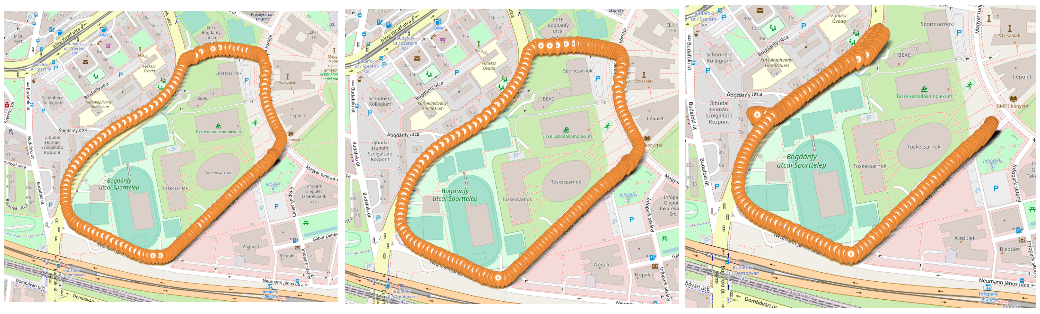
Table 5.3: Effects of low pass filtering on the location tracking algorithm.

Test case	Allowed reconstruction error	Average trajectory reconstruction error (meter)	Std. deviation of error (meter)	Endpoint reconstruction error (meter)
C1 (Figure 5.8)	10%	8.63	9.07	8.45
	20%	8.25	7.33	12.9
	40%	47.71	74.23	602.64
C2 (Figure A.20)	10%	8.71	8.94	11.43
	20%	10.98	11.9	13.37
	40%	175.08	159.73	589.17
C3 (Figure A.22)	10%	7.01	5.92	9.36
	20%	7.58	5.93	8.16
	40%	207.08	151.12	124.05



(a) Filtering: 10% (b) Filtering: 20% (c) Filtering: 40%

Figure 5.7: C1 test case macrotracking results on low pass filtered data without map



(a) Allowed reconstruction error: 10% (b) Allowed reconstruction error: 20% (c) Allowed reconstruction error: 40%

Figure 5.8: C1 test case macrotracking results on low pass filtered data with map

5.5 Summary

We showed that CAN logs carry personal and sensitive information and therefore are subject to privacy regulations. We reconstructed the potentially sensitive trajectory of the vehicle from different sensor measurements other than the exact GPS coordinates. Our attack does not rely on any special background knowledge for trajectory reconstruction besides the speed, steering wheel position, and the starting location of the vehicle. In addition, our reconstruction technique is simple and efficient, and can also serve as a useful forensics tool, e.g., in case of accident reconstruction. As this attack is feasible under mild assumptions, even unprocessed raw CAN logs are undoubtedly regarded as personal data according to most privacy regulations. Our attack exploits the subtle dependencies among different sensor measurements which makes their anonymization, as well as consent control, challenging. Indeed, we also showed that naive anonymization approaches, such as smoothing or low-pass filtering, either do not provide a strong worst-case privacy guarantee to every single individual or they yield very inaccurate data. Therefore, we advocate the application of more principled approaches, such as Differential Privacy, to anonymize aggregated CAN logs, and provide identical privacy guarantees to every single driver irrespective of any background knowledge of the adversary. On the other hand, Differential Privacy generally provides meaningful utility only with a sufficiently large number of drivers, and can be used for synthetic data generation indirectly through the post-processing of already noisy aggregates. Therefore, further research is needed to improve the accuracy of the differentially private release of CAN data with sufficiently strong privacy guarantees.

Finally, instead of data anonymization, data controllers may rather ask for the consent of drivers in order to process or share their vehicular data, and still remain compliant with privacy regulations. Even if fine-grained controls to review and grant permissions to certain signals are provided to drivers with the description of how data will be shared and used, there is usually less explanation on how much data they can share without revealing sensitive information due to the interdependency of different signals or attributes. However, in the spirit of the GDPR, providing *informed* consent should also entail the comprehension of such potential privacy implications.

Conclusion

CAN networks have been the foundation for in-vehicle communication since their inception. They have been used for decades without problems, but by connecting vehicles to the internet, the isolation of the CAN bus communication has fundamentally changed. Suddenly, the internal components of vehicles became potential attack targets. These new threats inspired our research to efficiently store internal communications for later analysis, detect emerging attacks, and shed light on potential personal data privacy threats.

In Chapter 2, we introduced our attacker device, which has since generated interest among our industrial partners and can perform the latest attacks from the state-of-the-art. In addition, we presented a large dataset of benign and attacked traffic logs, which is a valuable asset for the research community as machine learning methods get more widespread.

The CAN compression algorithm, presented in Chapter 3, is a purpose-designed algorithm for a specific problem. It exploits the traffic patterns of typical ECU communication and the physical operation of a vehicle to efficiently solve the long-term data storage problem. Due to the total number of cars driven nowadays, the scale of this problem globally is enormous, hence the need to research domain-specific solutions.

Anomaly detection on the CAN bus, similar to our results presented in Chapter 4, has generated significant interest in the research community in the last ten years. New results are presented almost every other week. We consider our results to be valuable contributions to the research area. However, as the field of machine learning is growing enormously, we expect new anomaly detection methods will appear. Our CAN-specific ideas will likely remain usable, but applying better-performing base models may improve the results presented here. Regardless, as future work, we plan to combine the two anomaly detection methods presented in Section 4.3 and 4.4 to test if the simultaneous application of these two approaches leads to better results. Furthermore, we will investigate whether a traffic situation-specific detection model can achieve better results than a general approach.

In Chapter 5, we show that privacy problems can arise from CAN data release. The proposed two algorithms can successfully reconstruct the movement of a vehicle for short and long-distance drives. These threats to the driver's privacy should be appropriately mitigated. We also show that naive anonymization techniques are not sufficient. A potential future work of this research topic is the application of differential privacy to CAN data release.

All the results presented in this thesis are the authors' own contributions to the field of vehicle security research. Among those, the most significant ones are the following.

THESIS 1.1: I proposed a semantic compression method for compressing CAN traffic and measured that this alone compresses data to 10% of the original size in [C2]. I showed that combining semantic and syntactic compression can reduce the required storage space to 5% of the original size in [J1]. This approach thus provides a significantly more efficient result than syntactic compression alone, which only reduces the size to 30% of the original.

THESIS 1.2: To support forensics analysis, I showed through measurements on two datasets that the compressed format is suitable for high-confidence identification of message injection attacks in [C4][11]. The previous results show that to implement a successful message injection attack, at least five times the normal message frequency is required during the attack. The proposed detection solution, on the other hand, can detect the anomaly from as low as twice the normal frequency.

THESIS 2.1: I showed in [C6][12] that our model, built on the correlation measurements between CAN signals, can successfully detect message modification attacks. I tested the accuracy of the proposed method against seven different attack strategies. The results show that for attacks targetting signals strongly correlating with other signals, the accuracy of our detection is $\sim$90% with a 0% false positive rate due to applying a double threshold system. It is worth highlighting that for the RANDOM, ADD-INCR, and ADD-DECR attacks modifying at least 8 bits of a signal, we were able to achieve 95% accuracy. Similarly, for all attacks modifying at least 12 bits of a signal, the detection accuracy is 95%.

[11] Dóra Neubrandt implemented the measurement algorithm.

[12] György Lupták implemented the correlation calculation and statistical testing.

THESIS 2.2: I proposed a TCN-based detection model that can detect CAN message modification attacks by predicting future values to CAN signals and then comparing the prediction with the actual values in [C7][13]. Based on measurements from two datasets, I demonstrated that my TCN-based detection method detects attacks with an accuracy between 83% and 99% while keeping a false positive rate below 0.2%. I compared the proposed method to the previously best-performing solution and showed that my detection algorithm performs better in 27 out of 30 cases.

THESIS 3.1: I proposed a (macrotracking) algorithm that can reliably reconstruct the trajectory of a vehicle over longer trips only from raw CAN data and publicly available map information in [J2]. I have verified the method's accuracy with measurements: the algorithm was able to reconstruct all several kilometers-long test cases, consisting of at least 20 intersections, with just a few meters of inaccuracy.

THESIS 3.2: I have showed that the proposed macrotracking algorithm is robust to typical signal distortion techniques for protecting privacy in [J2]. The method's robustness has been verified by several measurements: even after applying a low-pass filter to achieve a 20% distortion, the inaccuracy remained below 8 meters. In the case of smoothing, the algorithm is even more robust: it accurately restored the original trajectory even after applying a 6.4s long smoothing window.

[13]Irina Chiscop implemented the TCN network architecture.

List of own publications

Conference and Workshop Papers

[C1] András Gazdag, Levente Buttyán, and Zsolt Szalay
Towards Efficient Compression of CAN Traffic Logs
34th International Colloquium on Advanced Manufacturing and Repairing Technologies in Vehicle Industry, 2017.

[C2] András Gazdag, Levente Buttyán, and Zsolt Szalay
Efficient lossless compression of CAN traffic logs
25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2017.

[C3] András Gazdag, Tamás Holczer, Levente Buttyán, and Zsolt Szalay
Vehicular can traffic based microtracking for accident reconstruction
Vehicle and Automotive Engineering, Springer, 2018.

[C4] András Gazdag, Dóra Neubrandt, Levente Buttyán, and Zsolt Szalay
Detection of Injection Attacks in Compressed CAN Traffic Logs
Security and Safety Interplay of Intelligent Software Systems, Springer, 2019.

[C5] András Gazdag, Csongor Ferenczi, and Levente Buttyán
Development of a Man-in-the-Middle Attack Device for the CAN Bus
1st Conference on Information Technology and Data Science, 2020.

[C6] András Gazdag, György Lupták, and Levente Buttyán
Correlation-based Anomaly Detection for the CAN Bus
Euro-CYBERSEC, 2021.

[C7] Irina Chiscop, András Gazdag, Joos Bosman, and Gergely Biczók
Detecting Message Modification Attacks on the CAN Bus with Temporal Convolutional Networks
Proceedings of the 7th International Conference on Vehicle Technology and Intelligent Transport Systems, 2021.

Journal Papers

[J1] András Gazdag, Levente Buttyán, and Zsolt Szalay
Forensics aware lossless compression of CAN traffic logs
Scientific Letters of the University of Zilina, 2017

[J2] András Gazdag, Szilvia Lestyán, Mina Remeli, Gergely Ács, Tamás Holczer, and Gergely Biczók
Privacy pitfalls of releasing in-vehicle network data
Vehicular Communications, 2023.

[J3] András Gazdag, Rudolf Ferenc, Levente Buttyán
CrySyS dataset of CAN traffic logs containing fabrication and masquerade attacks
Nature: Scientific Data, 2023.

Bibliography

[ABM$^+$21] Javed Ashraf, Asim D. Bakhshi, Nour Moustafa, Hasnat Khurshid, Abdullah Javed, and Amin Beheshti. Novel deep learning-enabled lstm autoencoder architecture for discovering anomalous events from intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4507–4518, 2021.

[AG17] N. Aloysius and M. Geetha. A review on deep convolutional neural networks. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 0588–0592, 2017.

[Ari10] Arilou. Feasible car cyber defense. In *Proceedings of the 21st escar Europe*. ESCAR, 2010.

[BGR01] Shivnath Babu, Minos Garofalakis, and Rajeev Rastogi. SPARTAN: a model-based semantic compression system for massive data tables. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 283–294, New York, NY, USA, 2001. ACM.

[BKK18] Shaojie Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, abs/1803.01271, 2018.

[BODA$^+$20] Lotfi Ben Othmane, Lalitha Dhulipala, Moataz Abdelkhalek, Nicholas Multari, and Manimaran Govindarasu. On the performance of detecting injection of fabricated messages into the can bus. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2020.

[Boe17] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[CHCMB19] Alice Cohen-Hadria, Mark Cartwright, Brian McFee, and Juan Pablo Bello. Voice anonymization in urban sound recordings. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2019.

[CJJ$^+$18] Wonsuk Choi, Kyungho Joo, Hyo Jin Jo, Moon Chan Park, and Dong Hoon Lee. VoltageIDS: Low-level communication characteristics for automotive intrusion detection system. *IEEE Transactions on Information Forensics and Security*, 13(8):2114–2129, 2018.

[CMK$^+$11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX Security Symposium (USENIX Security 11)*, San Francisco, CA, aug 2011. USENIX Association.

[CN20] Aloni Cohen and Kobbi Nissim. Towards formalizing the gdpr's notion of singling out. *Proceedings of the National Academy of Sciences*, 117(15):8344–8352, 2020.

[CRN08] S. Chen, S. Ranjan, and A. Nucci. IPzip: a stream-aware ip compression algorithm. In *Proceedings of the 2008 Data Compression Conference*, pages 182–191, March 2008.

[CS16] Kyong-Tak Cho and Kang G. Shin. *Fingerprinting Electronic Control Units for Vehicle Intrusion Detection*. USENIX Association, 2016.

[DAET13] Rinku Dewri, Prasad Annadata, Wisam Eltarjaman, and Ramakrishna Thurimella. Inferring trip destinations from driving habits data. In *Proceedings of the 12th ACM Workshop on privacy in the electronic society*, pages 267–272, 11 2013.

[DLdHE19] Guillaume Dupont, Alexios Lekidis, J. (Jerry) den Hartog, and S. (Sandro) Etalle. Automotive controller area network (CAN) bus intrusion dataset v2, 2019.

[DMHVB13] Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3(1):1–5, 2013.

[Ell12] George Ellis. Filters in control systems. *Control System Design Guide*, 9:165, 2012.

[EWO20] Bernd Elend, Thierry Walrant, and Georg Olma. Securing can communication efficiently with minimal system impact. Technical report, NXP, 2020.

[GFS$^+$] Xianyi Gao, Bernhard Firner, Shridatt Sugrim, Victor Kaiser-pendergrast, Yulong Yang, and Janne Lindqvist. Elastic pathing: Your speed is enough to track you.

[GHM16] András Gazdag, Tamas Holczer, and Gyorgy Miru. Intrusion detection in cyber physical systems based on process modelling. In *Proceedings of 16th European Conference on Cyber Warfare & Security*. Academic conferences, 2016.

[GMT16] M. Gmiden, H. Mohamed, and H. Trabelsi. An intrusion detection method for securing in-vehicle CAN bus. In *Proceedings of the 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2016.

[GMvHV12] Bogdan Groza, Stefan Murvay, Anthony van Herrewege, and Ingrid Verbauwhede. LiBrA-CAN: a lightweight broadcast authentication protocol for Controller Area Networks. In *Proceedings of the International Conference on Cryptology and Network Security (CANS)*, pages 185 – 200, 2012.

[GP16] Yihan Gao and Aditya Parameswaran. Squish: near-optimal compression for archival of relational datasets. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1575–1584, New York, NY, USA, 2016. ACM.

[HIO$^+$20] M. A. Hossain, Hiroyuki Inoue, H. Ochiai, D. Fall, and Youki Kadobayashi. LSTM-based intrusion detection system for in-vehicle can bus communications. *IEEE Access*, 8:185489–185502, 2020.

[HON$^+$12] Jun Han, Emmanuel Owusu, Le T Nguyen, Adrian Perrig, and Joy Zhang. Accomplice: Location inference using accelerometers on smartphones. In *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*, pages 1–9. IEEE, 2012.

[HSDU20] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer. CANet: An unsupervised intrusion detection system for high dimensional can bus data. *IEEE Access*, 8:58194–58205, 2020.

[HZ19] Yangdong He and Jiabao Zhao. Temporal convolutional networks for anomaly detection in time series. *Journal of Physics: Conference Series*, 1213:042050, jun 2019.

[ISO15] ISO. 11898-1:2015 - Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling, International Organization for Standardization, 2015.

[ISO16] ISO. 11898-2:2016 - Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit, International Organization for Standardization, 2016.

[JMN99] H. V. Jagadish, J. Madar, and Raymond T. Ng. Semantic compression and pattern extraction with fascicles. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 186–198, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[JNOT04a] H. V. Jagadish, R. T. Ng, Beng Chin Ooi, and A. K. H. Tung. ItCompress: an iterative semantic compression algorithm. In *Proceedings. 20th International Conference on Data Engineering*, pages 646–657, March 2004.

[JNOT04b] H.V. Jagadish, R.T. Ng, Beng Chin Ooi, and A.K.H. Tung. Itcompress: an iterative semantic compression algorithm. In *Proceedings. 20th International Conference on Data Engineering*, pages 646–657, 2004.

[JWQ+18] Haojie Ji, Yunpeng Wang, Hongmao Qin, Xinkai Wu, and Guizhen Yu. Investigating the effects of attack detection for in-vehicle networks based on clock drift of ECUs. *IEEE Access*, 6:49375–49384, 2018.

[KCI+20] Z. Khan, M. Chowdhury, Mhafuzul Islam, Chin-Ya Huang, and M. Rahman. Long short-term memory neural network-based attack detection model for in-vehicle network security. *IEEE Sensors Letters*, 4:1–4, 2020.

[KCR+10] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, 2010.

[KK16a] Min-Joo Kang and Jewon Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PLOS ONE*, 11:e0155781, 06 2016.

[KK16b] Min-Ju Kang and Je-Won Kang. A novel intrusion detection method using deep neural network for in-vehicle network security. In *Proceedings of the 83rd IEEE Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2016.

[KKJ+21] Kyounggon Kim, Jun Seok Kim, Seonghoon Jeong, Jo-Hee Park, and Huy Kang Kim. Cybersecurity for autonomous vehicles: Review of attacks and defense. *Computers & Security*, 103, 2021.

[KS19] Vladimir Kaplun and Michael Segal. Breaching the privacy of connected vehicles network. *Telecommunication Systems*, 70, 04 2019.

[KTP20] Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and S. Pasricha. INDRA: Intrusion detection using recurrent autoencoders in automotive embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39:3698–3710, 2020.

[LÁBS19] Szilvia Lestyan, Gergely Ács, Gergely Biczók, and Zsolt Szalay. Extracting vehicle sensor signals from CAN logs for driver re-identification. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Proceedings of the 5th International Conference on Information Systems Security and Privacy, ICISSP 2019, Prague, Czech Republic, February 23-25, 2019*, pages 136–145. SciTePress, 2019.

[LCX+21] Yubin Lin, Chengbin Chen, Fen Xiao, Omid Avatefipour, Khalid Alsubhi, and Arda Yunianta. An evolutionary deep learning anomaly detection framework for in-vehicle networks – CAN bus. *IEEE Transactions on Industry Applications,* to appear, 2021.

[LJK17] H. Lee, S. H. Jeong, and H. K. Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, volume 00, pages 57–5709, Aug 2017.

[LMS82] E.J. Lefferts, F.L. Markley, and M.D. Shuster. Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*, 5(5):417–429, 1982.

[LOAB19] Siti-Farhana Lokman, Abu Talib Othman, and Muhammad-Husaini Abu-Bakar. Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review. *EURASIP Journal on Wireless Communications and Networking*, 184, 2019.

[LOMH19] Siti Farhana Lokman, Abu Talib Othman, Shahrulniza Musa, and Abu Bakar Muhamad Husaini. Deep contractive autoencoder-based anomaly detection for in-vehicle controller area network (CAN). In Abu Bakar Muhamad Husaini, Mohamad Sidik, and Andreas Ochsner, editors, *Progress in Engineering Technology: Automotive, Energy Generation, Quality Control and Efficiency*, pages 195–205. Springer International Publishing, Cham, 2019.

[MA11] Michael Muter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 1110–1115, 06 2011.

[MBC$^+$17] Michael R. Moore, Robert A. Bridges, Frank L. Combs, Michael S. Starr, and Stacy J. Prowell. Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: A data-driven approach to in-vehicle intrusion detection. In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, CISRC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[MCWW19] Xiuliang Mo, Pengyuan Chen, Jianing Wang, and Chundong Wang. Anomaly detection of vehicle can network based on message content. In Jin Li, Zheli Liu, and Hao Peng, editors, *Security and Privacy in New Computing Environments*, pages 96–104, Cham, 2019. Springer International Publishing.

[MGF10] Michael Müter, André Groll, and Felix C. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *2010 Sixth International Conference on Information Assurance and Security*, pages 92–98, 2010.

[MHT$^+$12] Tsutomu Matsumoto, Masato Hata, Masato Tanabe, Katsunari Yoshioka, and Kazuomi Oishi. A method of preventing unauthorized data transmission in Controller Area Network. In *Proceedings of the 75th IEEE Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2012.

[MS17] Mirco Marchetti and Dario Stabili. Anomaly detection of can bus messages through analysis of id sequences. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, pages 1577–1583, 2017.

[MTTH13a] Tao Mei, Lin-Xie Tang, Jinhui Tang, and Xian-Sheng Hua. Near-lossless semantic video summarization and its applications to video analysis. *ACM Transactions* *on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 9, 06 2013.

[MTTH13b] Tao Mei, Lin-Xie Tang, Jinhui Tang, and Xian-Sheng Hua. Near-lossless semantic video summarization and its applications to video analysis. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(3):16:1–16:23, July 2013.

[MV13] C. Miller and C. Valasek. Adventures in automotive networks and control units. Technical report, IOActive Labs Research, 2013.

[MV14] C. Miller and C. Valasek. A survey of remote automotive attack surfaces. Technical report, IOActive Labs Research, 2014.

[MV15] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle. Technical report, IOActive Labs Research, 2015.

[MW17] Moti Markovitz and Avishai Wool. Field classification, modeling and anomaly detection in unknown can bus networks. *Vehicular Communications*, 9:43–52, 2017.

[NGMK18] Brent C. Nolan, Scott Graham, Barry Mullins, and Christine Schubert Kabban. Unsupervised time series extraction from controller area network payloads. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–5, 2018.

[NJCF19] Naman Singh Negi, Ons Jelassi, Stephan Clemencon, and Sebastian Fischmeister. A LSTM approach to detection of autonomous vehicle hijacking. In *Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems - Volume 1: VEHITS*, pages 475–482. INSTICC, SciTePress, 2019.

[NLY$^+$20] E. Novikova, V. Le, Matvey Yutin, M. Weber, and C. Anderson. Autoencoder anomaly detection on large CAN bus data. In *Proceedings of DLP-KDD 2020.*, New York, NY, USA., 2020. ACM.

[NMJ16] Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. OBD-SecureAlert: An anomaly detection system for vehicles. In *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, 2016.

[NR16] Stefan Nürnberger and Christian Rossow. vatiCAN – vetted, authenticated CAN bus. In *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pages 106 – 124, 2016.

[PPS20] Mert D Pesé, Xiaoying Pu, and Kang G Shin. Spy: Car steering reveals your trip route! *Proceedings on Privacy Enhancing Technologies*, 2020(2):155–174, 2020.

[PtC16] European Parliament and the Council. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

[RKAK$^+$22] Sampath Rajapaksha, Harsha Kalutarage, M.Omar Al-Kadri, Andrei Petrovski, Garikayi Madzudzo, and Madeline Cheah. Ai-based intrusion detection systems for in-vehicle networks: A survey. *ACM Computing Surveys*, 11 2022.

[RLÁB19] Mina Remeli, Szilvia Lestyán, Gergely Ács, and Gergely Biczók. Automatic driver identification from in-vehicle network logs. In *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*, pages 1150–1157. IEEE, 2019.

[RRA$^+$13] Anshul Rai, Ramachandran Ramjee, Ashok Anand, Venkata N. Padmanabhan, and George Varghese. MiG: Efficient migration of desktop VMs using semantic compression. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 25–36, San Jose, CA, June 2013. USENIX Association.

[SDK19] Florian Sommer, Jürgen Dürrwang, and Reiner Kriesten. Survey and classification of automotive security attacks. *Information (Switzerland)*, 10, 2019.

[SKK16] H. M. Song, H. R. Kim, and H. K. Kim. Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. In *Proceedings of the International Conference on Information Networking (ICOIN)*, 2016.

[SMAV19] Chandra Sharma, Samuel Moylan, George T. Amariucai, and Eugene Y. Vasserman. An extended survey on vehicle security. *Computing Research Repository (CoRR)*, abs/1910.04150, 2019.

[SQS$^+$20] Ankur Sarker, Chenxi Qiu, Haiying Shen, Hua Uehara, and Kevin Zheng. Brake data-based location tracking in usage-based automotive insurance programs. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 229–240, 2020.

[SWK20] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[TBSK18] Andrew Tomlinson, Jeremy Bryans, Siraj Ahmed Shaikh, and Harsha Kumara Kalutarage. Detection of automotive CAN cyber-attacks by identifying packet timing anomalies in time windows. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 231–238, 2018.

[The14] Andreas Theissler. Anomaly detection in recordings from in-vehicle networks. In *Proceedings of the International Workshop on Big Data Applications and Principles*, 2014.

[TJL15] A. Taylor, N. Japkowicz, and S. Leblanc. Frequency-based anomaly detection for the automotive can bus. In *2015 World Congress on Industrial Control Systems Security (WCICSS)*, pages 45–49, 2015.

[TLJ16] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 130–139, 2016.

[TLW20] Shahroz Tariq, Sangyup Lee, and Simon S. Woo. CANTransfer: Transfer learning based intrusion detection on a controller area network using convolutional lstm network. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, pages 1048–1055, New York, NY, USA, 2020. Association for Computing Machinery.

[TW17] Vrizlynn L.L. Thing and Jiaxi Wu. Autonomous vehicle security: A taxonomy of attacks and defences. *Proceedings - 2016 IEEE International Conference on Internet of Things; IEEE Green Computing and Communications; IEEE Cyber, Physical, and Social Computing; IEEE Smart Data, iThings-GreenCom-CPSCom-Smart Data 2016*, pages 164–170, 2017.

[vdODZ$^+$16] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, Oriol Vinyals, A. Graves, Nal Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *ArXiv*, abs/1609.03499, 2016.

[VHSV11] Anthony Van Herrewege, Dave Singelée, and Ingrid Verbauwhede. CANAuth - a simple, backward compatible broadcast authentication protocol for CAN bus. In *Proceedings of the ESCAR Conference*, 2011.

[VIB$^+$22] Miki E. Verma, Michael D. Iannacone, Robert A. Bridges, Samuel C. Hollifield, Pablo Moriano, Bill Kay, and Frank L. Combs. Addressing the lack of comparability & testing in CAN intrusion detection research: A comprehensive guide to CAN IDS data & introduction of the ROAD Dataset, 2022.

[WB95] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical Report TR95-041, University of North Carolina at Chapel Hill, Department of Computer Science, 1995. The article has also been translated into Chinese by Xuchen Yao, a student at The Institute of Acoustics of The Chinese Academy of Sciences. See also our Kalman filter web site at https://www.cs.unc.edu/ welch/kalman/index.html.

[WC15] Yu-jing Wu and Jin-Gyun Chung. Efficient controller area network data compression for automobile applications. *Frontiers of Information Technology & Electronic Engineering*, 16(1):70–78, Jan 2015.

[WLX$^+$20] Wufei Wu, Renfa Li, Guoqi Xie, Jiyao An, Yang Bai, Jia Zhou, and Keqin Li. A survey of intrusion detection for in-vehicle networks. *IEEE Transactions on Intelligent Transport Systems*, 21(3), March 2020.

[WPW$^+$17] Armin Wasicek, Mert D Pesé, André Weimerskirch, Yelizaveta Burakova, and Karan Singh. Context-aware intrusion detection in automotive control systems. In *Proc. 5th ESCAR USA Conf*, pages 21–22, 2017.

[WUW19] Marian Waltereit, Maximilian Uphoff, and Torben Weis. Route derivation using distances and turn directions. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, pages 35–40, 03 2019.

[WWSZ18] M. Weber, G. Wolf, E. Sax, and B. Zimmer. Online detection of anomalies in vehicle signals using replicator neural networks. In *ESCAR 2018*, 2018.

[YZOB19] Clinton Young, Joseph Zambreno, Habeeb Olufowobi, and Gedare Bloom. Survey of automotive controller area network intrusion-detection systems. *IEEE Design & Test*, October 2019.

[ZCL$^+$17] Lu Zhou, Qingrong Chen, Zutian Luo, Haojin Zhu, and Cailian Chen. Speed-based location tracking in usage-based automotive insurance. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2252–2257, 2017.

[ZDZ$^+$19] Lu Zhou, Suguo Du, Haojin Zhu, Cailian Chen, Kaoru Ota, and Mianxiong Dong. Location privacy in usage-based automotive insurance: Attacks and countermeasures. *IEEE Transactions on Information Forensics and Security*, 14(1):196–211, 2019.

Appendix

Dataset Description Figures & Tables

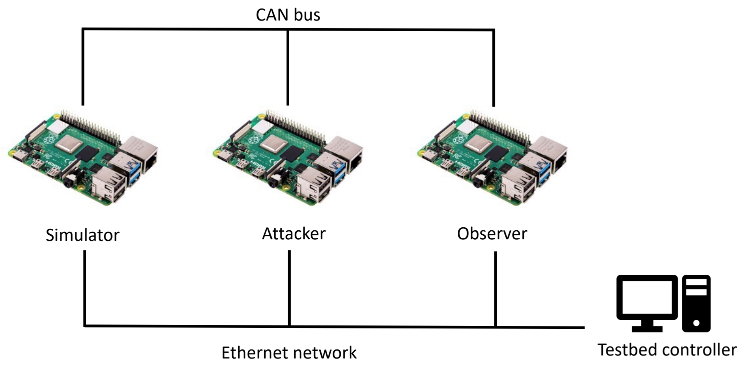


Figure A.9: CAN testbed schematics

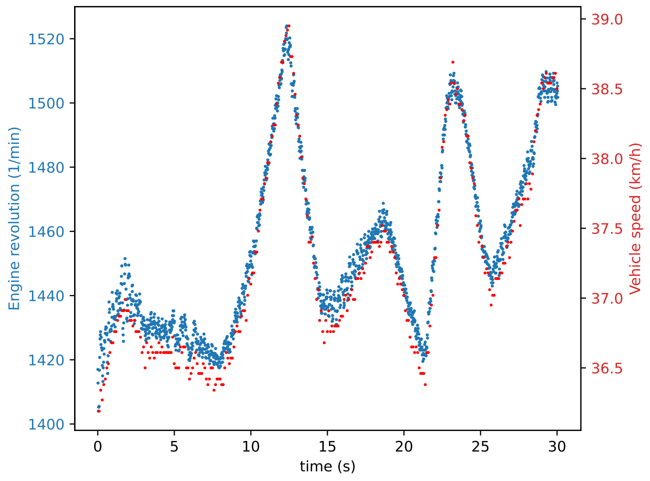


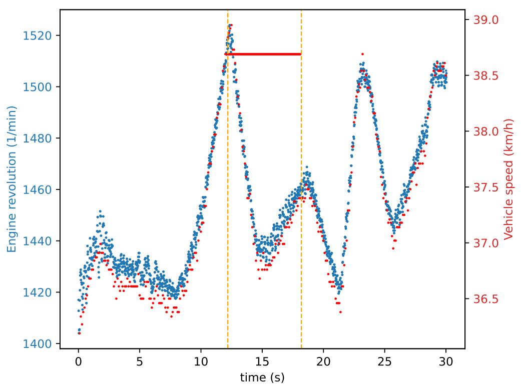
Figure A.10: Example benign CAN signal (S-1-4)

Table A.6: CAN trace capture scenarios.

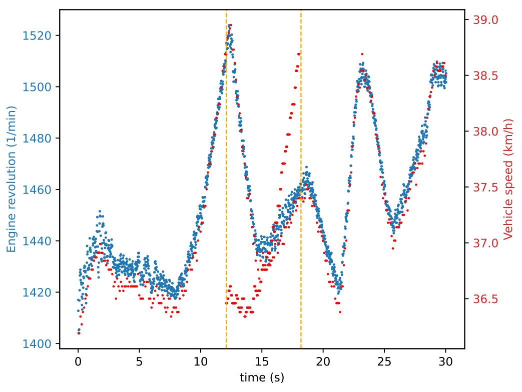
Trace ID	**Scenario description**	**Trace length**	**Trace size**	**Number of messages**
S-1-1	Driving with $\sim$ 36 km/h.	30.08 s	693 KB	17,935
S-1-2	Driving with $\sim$ 36-37 km/h.	30.16 s	691 KB	17,888
S-1-3	Driving with $\sim$ 36-37 km/h.	30.16 s	695 KB	17,982
S-1-4	Driving with $\sim$ 37 km/h.	30.06 s	692 KB	17,911
S-1-5	Driving with $\sim$ 35 km/h.	32.72 s	752 KB	19,443
S-1-6	Driving with $\sim$ 37 km/h.	29.99 s	672 KB	17,404
S-2-1	Driving with $\sim$ 60 km/h.	30.06 s	692 KB	17,909
S-2-2	Driving with $\sim$ 60 km/h.	30.01 s	692 KB	17,897
S-2-3	Driving with $\sim$ 59 km/h.	30.93 s	713 KB	18,445
S-2-4	Driving with $\sim$ 60 km/h.	30.19 s	696 KB	18,000
S-2-5	Driving with $\sim$ 61-62 km/h.	31.98 s	738 KB	19,077
S-2-6	Driving with $\sim$ 62 km/h.	31.11 s	717 KB	18,553
S-3-1	Acceleration then deceleration: 0 km/h to 50 km/h to 0 km/h.	29.82 s	683 KB	17,669
S-3-2	Acceleration then deceleration: 0 km/h to 40 km/h to 0 km/h.	32.36 s	747 KB	19,327
S-3-3	Acceleration then deceleration: 0 km/h to 40 km/h to 0 km/h.	30.72 s	709 KB	18,335
T-1-1	Driving in urban environment.	430.17 s	10,211 KB	256,921
T-1-2	Driving in urban environment.	1,253.81 s	30,015 KB	748,241
T-1-3	Driving in urban environment.	1,106.71 s	26,433 KB	660,880
T-1-4	Driving in urban environment.	1,576.21 s	37,884 KB	940,154
T-1-5	Driving in urban environment.	1,055.67 s	25,158 KB	629,786
T-1-6	Driving in urban environment.	1,232.86 s	29,431 KB	733,933
T-1-7	Driving in urban environment.	261.73 s	6,189 KB	156,371
T-2-1	Driving on country road.	359.32 s	8,519 KB	214,625
T-2-2	Driving on country road.	371.97 s	8,810 KB	221,907
T-3-1	Driving on motorway.	552.92 s	13,090 KB	328,901
T-3-2	Driving on motorway.	562.09 s	13,333 KB	334,980
	Total:	2h 33m 43s	219,655 KB	5,500,474

Table A.7: Identified CAN signals.

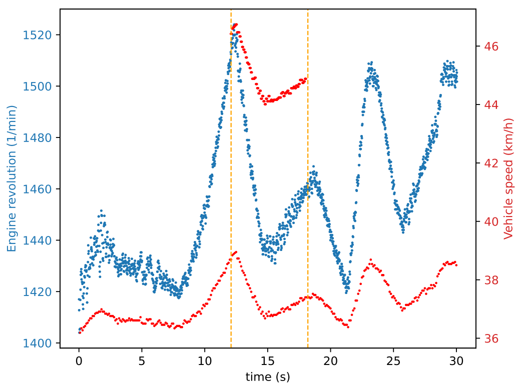
Message ID	**Signal Index**	**Start bit offset**	**End bit offset**
0x110	0	6	23
	1	24	39
	2	40	47
	3	48	55
	4	56	63
0x120	0	9	19
	1	21	31
	2	34	39
	3	41	51
	4	52	63
0x140	0	1	7
	1	14	39
	2	40	63
0x180	0	1	12
	1	13	14
	2	15	20
	3	21	28
	4	32	36
	5	37	38
	6	39	47
0x1a0	0	12	20
	1	25	31
	2	32	63
0x280	0	3	15
	1	19	31
	2	35	47
	3	51	63
0x290	0	2	8
	1	18	24
	2	34	40
	3	50	56
	4	57	63
0x295	0	6	18
	1	23	31
0x300	0	2	3
	1	4	7
	2	8	10
	3	14	25
	4	26	27
0x301	0	19	47
	1	54	55
0x380	0	0	1
	1	2	3
	2	8	11
	3	13	23
	4	32	33
	5	34	35
	6	36	39
	7	45	48
	8	55	56
	9	57	63
0x381	0	0	2
	1	3	4
	2	7	15
	3	24	30
	4	31	38
	5	40	47
0x383	0	0	4
	1	6	7
	2	10	39
0x410	0	9	23
	1	24	32
	2	33	38
	3	39	40
	4	41	48
	5	49	54
0x440	0	3	4
	1	5	8
	2	13	22
0x4a0	0	16	33
	1	34	47
0x510	0	5	15
	1	17	23
	2	25	31
	3	32	63
0x531	0	6	39



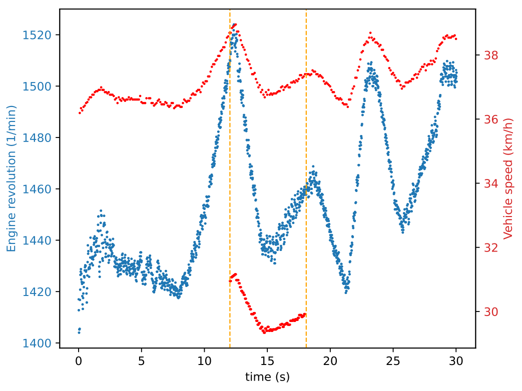
(a) CONST signal injection attack



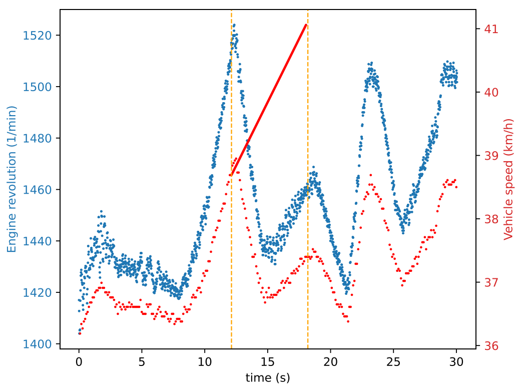
(b) REPLAY signal injection attack



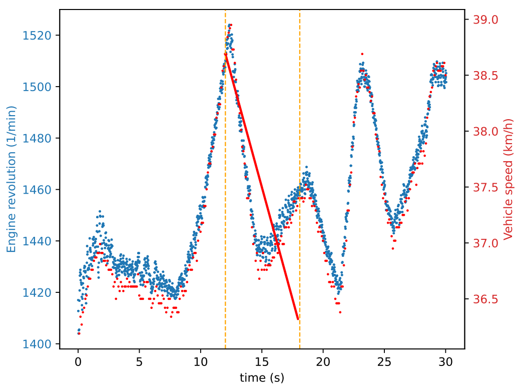
(c) POS-OFFSET signal injection attack



(d) NEG-OFFSET signal injection attack

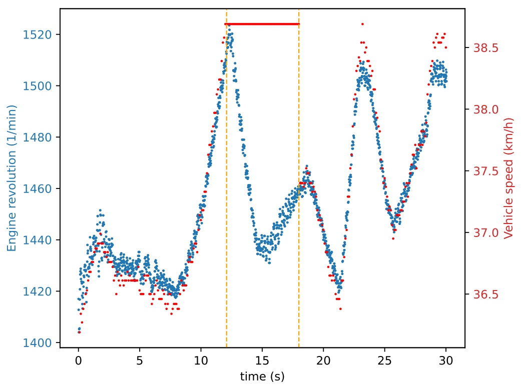


(e) ADD-INCR signal injection attack

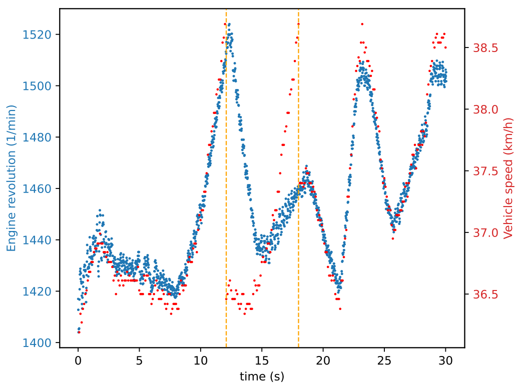


(f) ADD-DECR signal injection attack

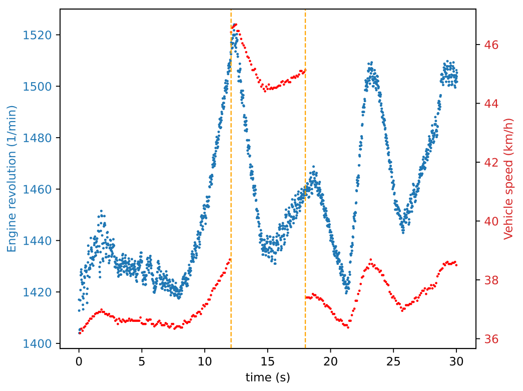
Figure A.11: Single signal injection attacks (S-1-4)



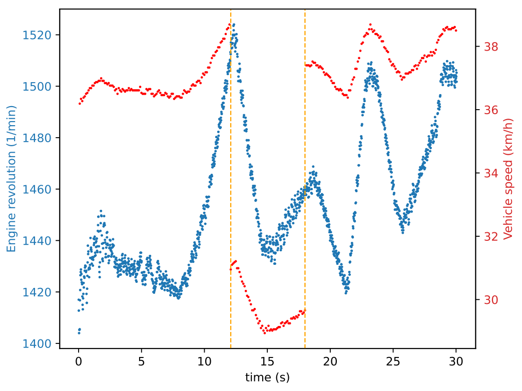
(a) CONST signal modification attack



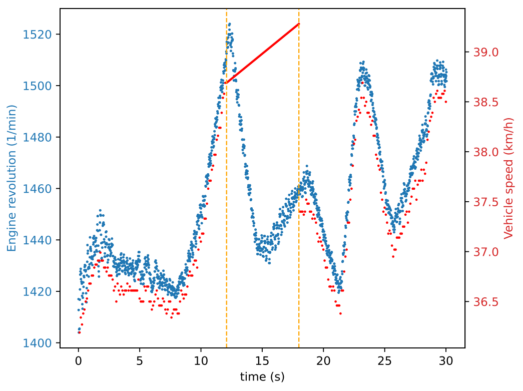
(b) REPLAY signal modification attack



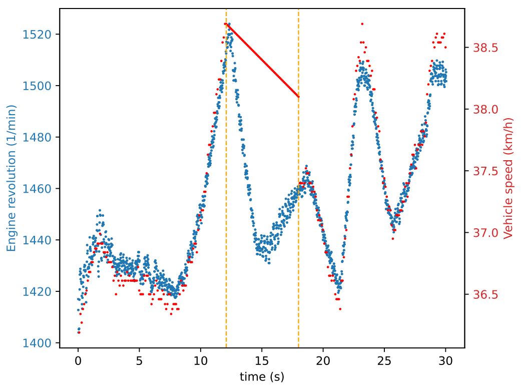
(c) POS-OFFSET signal modification attack



(d) NEG-OFFSET signal modification attack

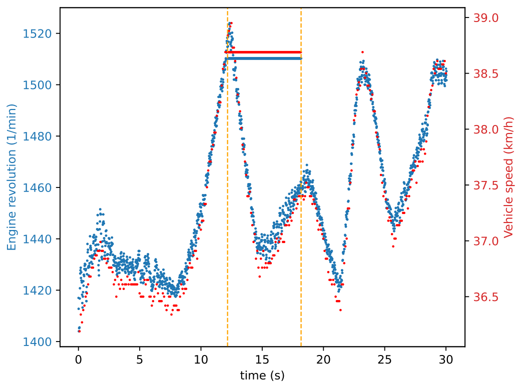


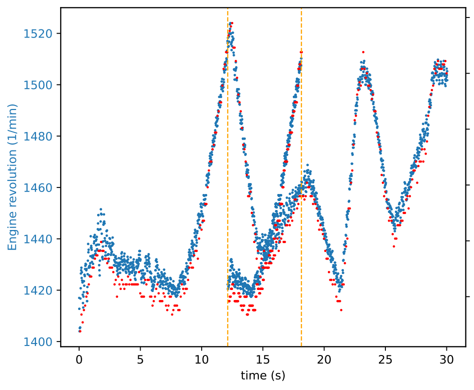
(e) ADD-INCR signal modification attack

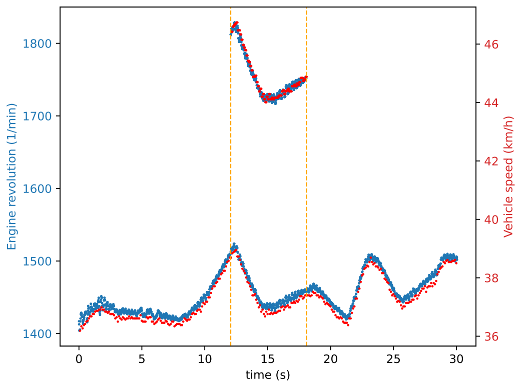


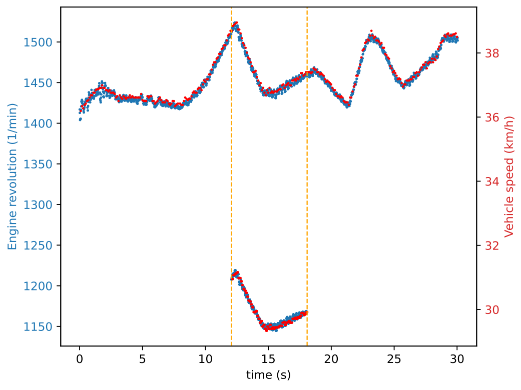
(f) ADD-DECR signal modification attack

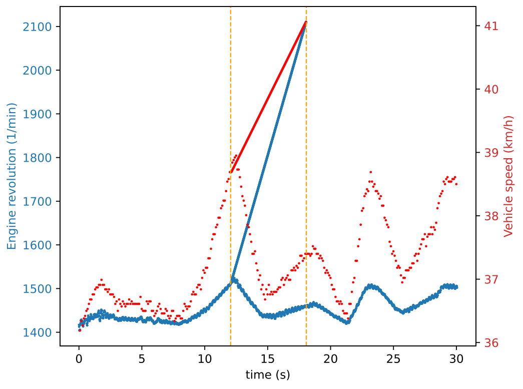
Figure A.12: Single signal modification attacks (S-1-4)


(a) Double CONST signal injection attack


(b) Double REPLAY signal injection attack


(c) Double POS-OFFSET signal injection attack


(d) Double NEG-OFFSET signal injection attack


(e) Double ADD-INCR signal injection attack

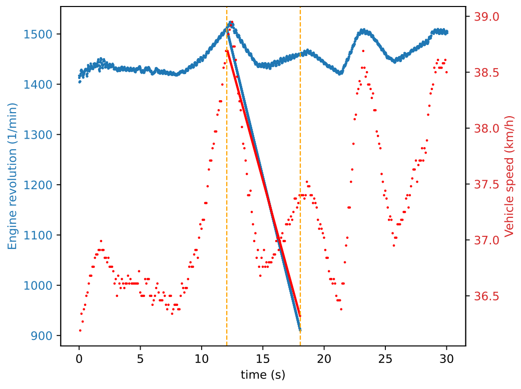
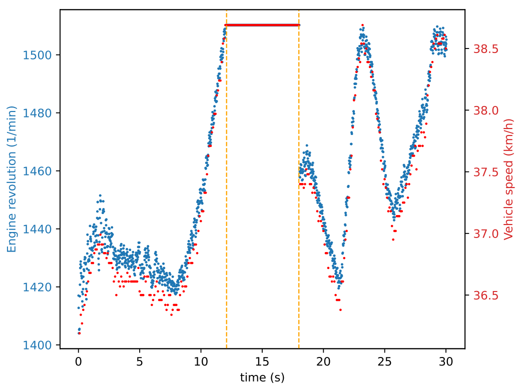
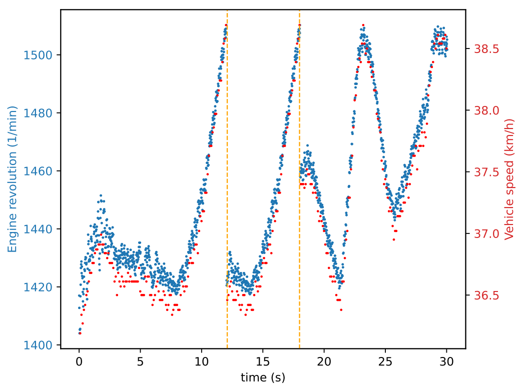

(f) Double ADD-DECR signal injection attack

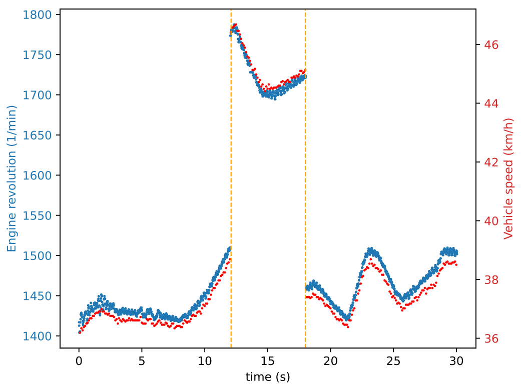
Figure A.13: Double signal injection attacks (S-1-4)



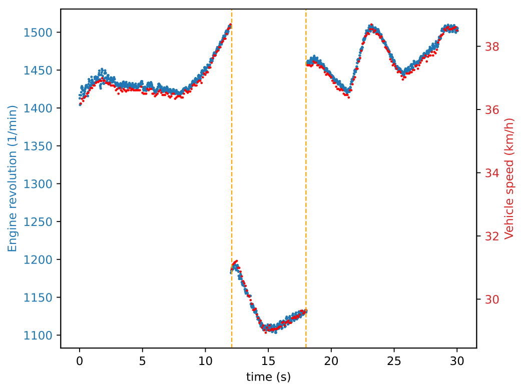
(a) Double CONST signal injection attack



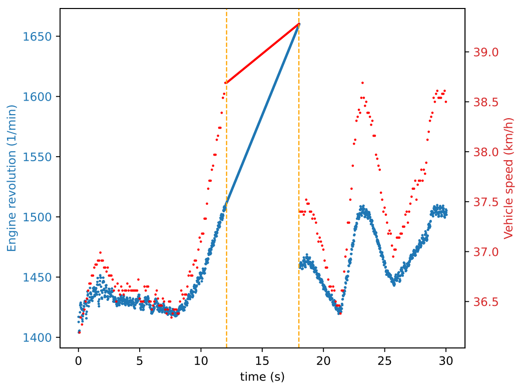
(b) Double REPLAY signal injection attack



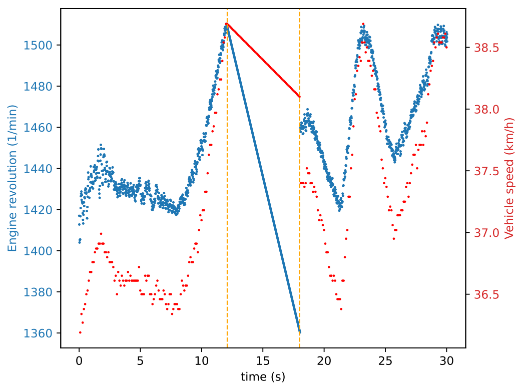
(c) Double POS-OFFSET signal injection attack



(d) Double NEG-OFFSET signal injection attack



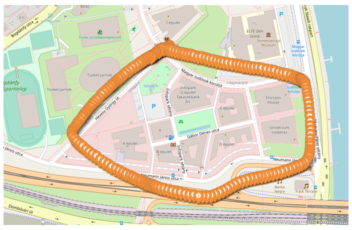
(e) Double ADD-INCR signal injection attack

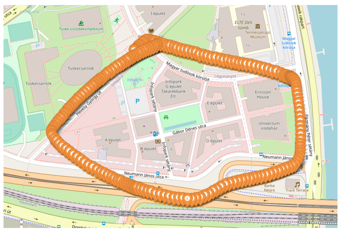


(f) Double ADD-DECR signal injection attack

Figure A.14: Double signal modification attacks (S-1-4)

Effect of smoothing and low pass filtering on macrotracking


(a) Smoothing window: 0.201s


(b) Smoothing window: 1.608s

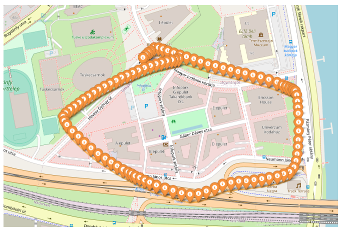
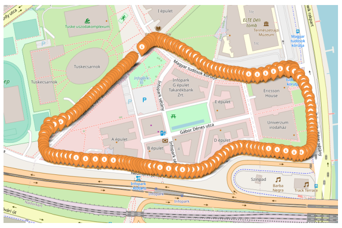
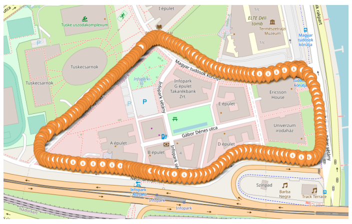

(c) Smoothing window: 6.4s

Figure A.15: C2 test case macrotracking results on smoothed data without map


(a) Smoothing window: 0.201s


(b) Smoothing window: 1.608s

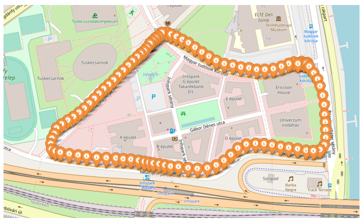
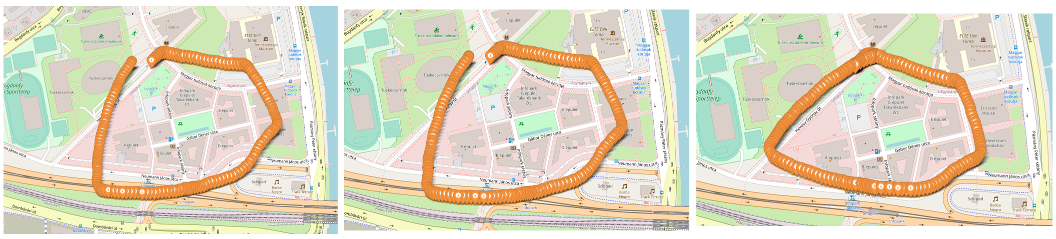
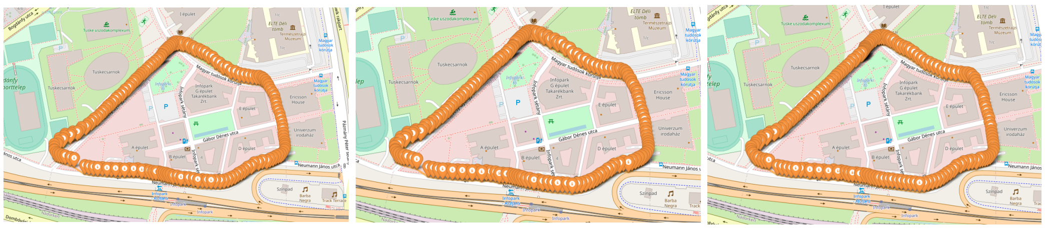

(c) Smoothing window: 6.4s

Figure A.16: C2 test case macrotracking results on smoothed data with map



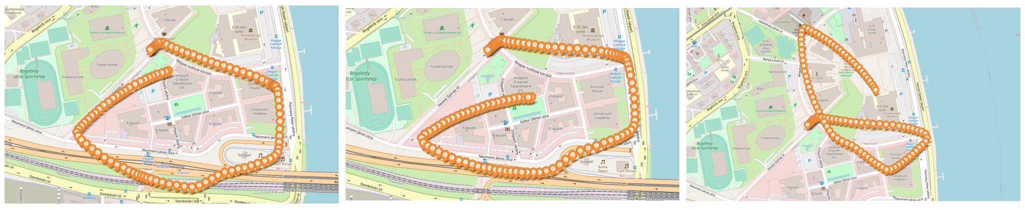
(a) Smoothing window: 0.201s (b) Smoothing window: 1.608s (c) Smoothing window: 6.4s

Figure A.17: C3 test case macrotracking results on smoothed data without map



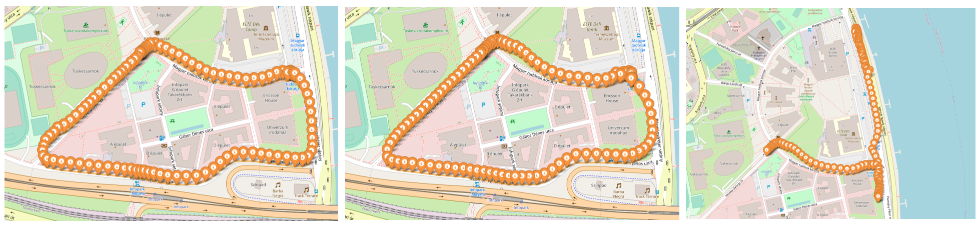
(a) Smoothing window: 0.201s (b) Smoothing window: 1.608s (c) Smoothing window: 6.4s

Figure A.18: C3 test case macrotracking results on smoothed data with map



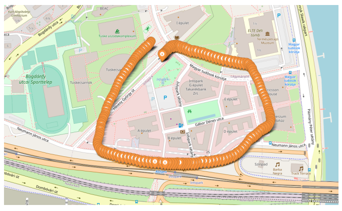
(a) Allowed reconstruction error: 10% (b) Allowed reconstruction error: 20% (c) Allowed reconstruction error: 40%

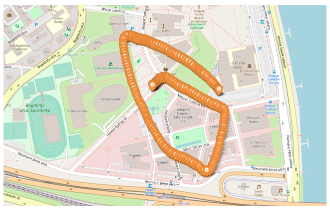
Figure A.19: C2 test case macrotracking results on low pass filtered data without map



(a) Allowed reconstruction error: 10% (b) Allowed reconstruction error: 20% (c) Allowed reconstruction error: 40%

Figure A.20: C2 test case macrotracking results on low pass filtered data with map


(a) Allowed reconstruction error: 10%


(b) Allowed reconstruction error: 20%

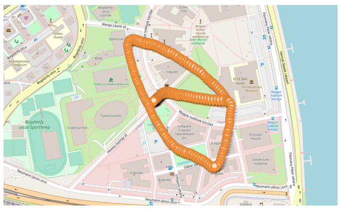
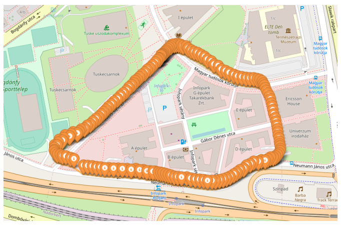
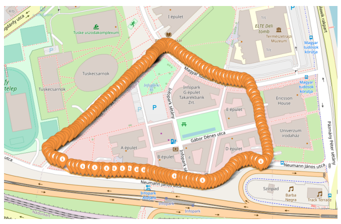

(c) Allowed reconstruction error: 40%

Figure A.21: C3 test case macrotracking results on low pass filtered data without map


(a) Allowed reconstruction error: 10%


(b) Allowed reconstruction error: 20%

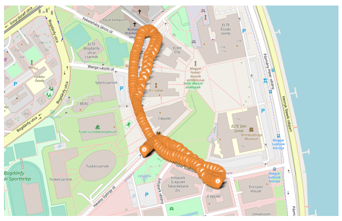

(c) Allowed reconstruction error: 40%

Figure A.22: C3 test case macrotracking results on low pass filtered data with map