DOCTOR OF PHILOSOPHY

Detecting Cyber Attacks on the Automotive Controller Area Network

Tomlinson, Andrew

Award date:
2020

Awarding institution:
Coventry University

Detecting Cyber Attacks on the Automotive Controller Area Network

by

Andrew John Tomlinson

July 2019

Faculty of Engineering, Environment and Computing

Institute for Future Transport and Cities

A thesis submitted in partial fulfilment of the University's requirements for the Degree of Doctor of Philosophy

Certificate of Ethical Approval

Applicant:

Andrew Tomlinson

Project Title:

Development of big data analytics processes for continuous security monitoring in vehicle systems.

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

17 January 2019

Project Reference Number:

P85703

Acknowledgements

I would like to thank Dr Jeremy Bryans for his guidance, advice and kindness, and Professor Siraj Shaikh for wonderful support throughout. Together they made undertaking this PhD an interesting, rewarding and thoroughly enjoyable experience.

My thanks also go to Dr Harsha Kumara Kalutarage for collaborative work on the CAN timing anomaly detection; to Dr Daniel Fowler for assistance with CAN processes and CAN data capture, and to Jake Hayward for support on CAN simulator development. Thanks also to Dr Alice Lau for help with CAN data collection.

Finally I am deeply grateful to my wife, Hanida, and children, Carl and Edward, for their patience and support throughout.

Abstract

The rise in data usage and connectivity in cars has led to concerns about their risk to cyber-attack. The Controller Area Network (CAN), used by all cars for communication between safety-critical and performance-critical components, has been shown to be particularly vulnerable. Cyber-attacks have been demonstrated on the CAN that would compromise the safety of passengers, damage the car, or cause it to malfunction.

This thesis proposes and evaluates methods that might be used to detect the presence of an attack by monitoring the CAN traffic. The methods proposed detect attack-resultant anomalies in the CAN packet timings and packet data payloads. A one-class classification approach is adopted since the CAN attack detection solution would need to cope with constraints that make the gathering of sufficient labelled attack-data samples unlikely. These constraints are also discussed in the thesis. The test data is generated from models devised from studying the published attacks, which are reviewed. The attack detection is evaluated over a range of suitable machine learning algorithms and training options. Processes for capturing and parsing the CAN data for the detection are also proposed and tested.

The results show that some of the methods offer the potential for attack detection and deployment in an in-vehicle system. However, additional research would be required to reduce the number of false alarms they generate. Possible ways to achieve this are discussed.

The contributions of this thesis include the proposal of the detection methods suitable for the automotive CAN and their systematic evaluation, the creation and evaluation of algorithms for processing the CAN data into structures suitable for anomaly detection, and the synthesis of demonstrated attacks into representative models suitable for test data.

Publications

Presented here are publications where work that has been carried out has either informed or been included in the thesis.

Tomlinson, A., Bryans, J., Shaikh, S.A., Kalutarage, H.K., Detection of Automotive CAN Cyber-Attacks by Identifying Packet Timing Anomalies in Time Windows, *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 231-238, 2018, IEEE.
This paper discusses methods and results from the timing anomaly detection experiments described in Chapters 5 and 6.

Tomlinson, A., Bryans, J., Shaikh, S.A., Towards viable intrusion detection methods for the automotive controller area network, *2nd ACM Computer Science in Cars Symposium*, Munich, Germany, 2018.
This paper reviews methods that have been proposed for CAN intrusion detection and presents the case for one-class anomaly approaches, such as those evaluated in this thesis.

Tomlinson, A., Bryans, J., Shaikh, S.A., Using a one-class compound classifier to detect in-vehicle network attacks, *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1926-1929, 2018, ACM.
This paper describes the principles of the Compound Classifier, subsequently evaluated in this thesis for payload anomaly detection (Chapters 7 and 8), and presents the case for its suitability for CAN intrusion detection.

Hayward, J., **Tomlinson, A.**, Bryans, J., Adding Cyberattacks To An Industry-Leading CAN Simulator, *19th IEEE International Conference on Software Quality, Reliability and Security*, 2019, IEEE Computer Society.
The CAN attacks, test data generation options, and attack models discussed in Chapters 2 and 3 were incorporated into this paper.

Table of contents

List of figures

List of tables

Abbreviations

ACK	Acknowledgement
ARIMA	Auto-Regressive Integrated Moving Average
CAN	Controller Area Network
CAN ID	Controller Area Network Identifier
CAPL	CAN Access Programming Language
CC	Compound Classifier
CRC	Cyclic Redundancy Check
DoS	Denial-of-Service
ECU	Electronic Control Unit
FM RDS	Frequency-Modulation Radio Data System
FN	False Negative
FP	False Positive
FURIA	Fuzzy Unordered Rule Induction Algorithm
IDS	Intrusion Detection System
ISO	International Organization for Standardization
ITM	Interframe Space
KB	Kilobyte
KNN	K-Nearest Neighbours
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
Mbps	Megabits per second
ML	Machine Learning
MHz	Megahertz
OBD port	On-Board Diagnostics port
OCSVM	One-Class Support Vector Machine
OCTANE	Open Car Test-bed and Networks Experiments
REC	Receive Error Counter
ROC	Receiver Operating Characteristic

RTR	Remote Transmission Request
SOF	Start Of Frame
SVM	Support Vector Machine
TEC	Transmit Error Counter
TORCS	The Open Racing Car Simulator
TN	True Negative
TP	True Positive

Chapter 1

Introduction

Automotive cybersecurity is emerging as an important concern, especially with the move to autonomous and connected vehicles [65, 79, 84]. Modern cars contain interconnected wired and wireless networks, including the Controller Area Network (CAN) on which reside electronic control units (ECUs) that control safety-critical functions such as braking, steering and engine operation. CAN vulnerabilities have been demonstrated; for example: Koscher *et al.* [40] sniffed CAN packets, altered them, and reinjected them via the car's On-Board Diagnostics (OBD) port, affecting the braking and engine; Miller and Valasek [52] reinjected CAN packets to achieve predetermined actions; Lee *et al.* [43] reinjected CAN packets via an OBD port Bluetooth module, changing dashboard warning displays; Checkoway *et al.* [14] compromised the CAN through a variety of channels, including Bluetooth and cellular networks, CD-based firmware updates, and song files played via the CD unit. It is probable that the variety of possible attack modes will rise as vehicles become more connected [13, 79].

1.1 Problem Statement

The goal of this research is to determine and evaluate a method, or set of methods, that can be used in a CAN intrusion detection system to detect if cyber-attacks are taking place. It is envisaged that the attack would be detected from tell-tale characteristics in the CAN bus traffic. Thus comparing the current traffic pattern against those observed to be typical.

Such an intrusion detection system (IDS) would need to run in real time, detecting intrusions quickly, and rapidly eliciting an appropriate response without disrupting or delaying the broadcast of legitimate packets. It would need to cope with a high, continuous, volume of traffic: approximately 1000 packets per second, each packet carrying up to 8 bytes of payload data plus additional information, broadcast at up to 1Mbps. Ideally it would also allow easy

configuration, easy deployment, and adaption to new threats, facilitating compliance with the life-cycle usage pattern of cars.

Compared to PC systems, an IDS for the automotive CAN is likely to be constrained by limited processing power, limited multi-tasking capability and limited storage capacity [31]. The micro controller commonly used in car ECUs has a memory of 40-50 KB and the computing power is less than 10 MHz [66]. A retrospectively fitted, or more substantial, IDS would probably need creating as a dedicated unit, either attached to the CAN or attached externally, such as on the ODB port.

Of course, the IDS would also need to be reliable and accurate. Amongst the challenges considered in this thesis, is that of generating data and evaluation models that can be used to test and validate proposed detection methods. These challenges deepen if we expect any solution to cope with a long-term deployment, thus detecting attacks not currently envisaged. The variety of car makes and models, and their varied life-cycles and dispersed location, makes reliable and regular updating a challenge, and it is perhaps unwise to assume owners would be able, or motivated, to update their systems outside the annual service [31, 57]. An option might be over-the-air updating, though this adds a potential attack route and logistical challenges.

Research and development into CAN cyberattacks is hindered by two major challenges. First, although the CAN protocol is openly documented [36], the meaning of the data transmitted, and specifics pertaining to any car make or model, such as the expected frequency of broadcasts, or the matching of packet IDs to ECUs or functions, is not available to most researchers. Even when the specification is available, the behaviour of an ECU might not be fully known [76]. Such factors make the expected behaviour of the CAN traffic difficult to comprehensively map, and as a consequence, the boundary between a normal signature and an attack signature may be difficult to determine.

Second, obtaining labelled attack data is difficult. Again, this is not published by manufacturers, and generating it is difficult because: a) testing attack scenarios using cars is dangerous, costly, potentially damaging to the car, and may require ECU programming knowledge; and b) the nascent nature of CAN cybersecurity means the invented attack scenarios are speculative, incomplete and embryonic.

1.2 Motivation

The CAN bus was designed for speed and robustness, rather than security and authentication, and the security vulnerabilities inherent in its processes are frequently cited (e.g. [7, 11, 14, 40]). In spite of these vulnerabilities, the CAN bus remains the dominant network for critical

components and the most common system for accessing diagnostics [87]. Koscher *et al.* [40] point out that costs, production chains, component development speed and safety priorities, will form an obstacle to implementing security improvements that require modification to the CAN protocol. Therefore, the deployment of a sufficiently secured CAN replacement may be many years away.

The vast number of cars currently registered, and the continuing use of the CAN bus in new vehicles for the foreseeable future, means that the known cyber vulnerabilities will persist for many years. There are, for example, approximately 260 million passenger cars currently in the EU, and their average age is 11 years [21]. Solutions to secure those cars are unlikely to be comprehensive. Attack intrusion detection systems (IDSs) will therefore be necessary. Although such systems might be manufactured by the car manufacturer, and configured specifically to each make and model of car, a generic system that can be easily deployed across many car makes, models, and configurations, perhaps retrospectively, would offer commercial appeal and might be a more useful option for coping with the diverse range of cars. The maintenance and updating throughout the lifetime of the vehicle will also need considering.

1.3 Research Questions

The underlying hypothesis of the research is that, in spite of the operation of ECUs that broadcast the CAN packets being undisclosed by the car manufacturers, and also in spite of it being novel between car models, there exist characteristics of the packet timings and payload data that would be manipulated in an attack to produce detectable anomalies. This dissertation attempts to test this, both by the analysis of CAN properties and attack proposals, and through the testing of potential detection methods on data representative of manipulations that might result from attacks.

Determining and evaluating suitable CAN intrusion detection methods raises three core research questions:

1. Can we determine monitorable characteristics in CAN traffic that would provide an indication of an attack? To be useful, such characteristics would need to facilitate real-time monitoring. This raises a number of considerations that need resolving, including the extent to which we can comprehensively predict attack scenarios, and the likelihood to which identified characteristics are valid identifiers across makes and models of car.

2. What methods could be used to monitor the CAN traffic to detect the attack characteristics? The automotive use-case imposes a range of constraints, such as the variety of car makes and models, their life-cycle and restricted option for intrusion detection system updates, and the capacity to cope with unpredicted attacks.

3. How do we test and validate candidate detection methods? Staging actual attacks on cars is problematical for many reasons (discussed later in this thesis). Thus acquiring representative data, which covers comprehensive driving scenarios or attack scenarios, requires some ingenuity.

These questions are considered throughout this thesis and the attempts to answer them guide the research and discussion.

1.4 Contributions

The contributions of this PhD are:

1. Methods for detecting cyber-attacks to the CAN are proposed and evaluated. These methods are new to this context, and include machine learning approaches that consider changes to the timing of packet broadcasts, as well as changes to the data content of the packet. Primarily, these are considered as one-class problems, with proposed anomaly detection methods that do not assume pre-existing attack data.

2. Processes for transforming the CAN packet flow into a structure suitable for analysis and intrusion detection are proposed, constructed and tested. These include processes to automate the discovery of the data structuring used by individual manufacturers.

3. Attack models proposed across research studies are assimilated according to their CAN traffic implications and used to build a set of representative attack manifestations used for testing.

4. The empirical testing of the intrusion detection methods is considered as a stage that should be automated, facilitating comprehensive testing across a range of attack magnitudes and corruptions, as well as testing across makes and models of car.

1.5 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2 presents an overview of the CAN bus, discusses its cyber vulnerabilities and reviews the attacks that have been proposed or demonstrated. The description of the CAN focuses on the processes and packet structures pertinent to attack scenarios. The reviewed attacks are assessed for their CAN traffic implications, with commonalities identified which inform the attack detection options considered in this thesis. The motivation for attack detection, as well as the outcome choices, are also discussed.

Chapter 3 reviews existing studies into CAN intrusion detection, and considers the appropriateness of their proposed detection methods to the domain. These methods can be generally classified into signature based intrusion detection, and anomaly based intrusion detection; and the appropriateness of each approach is discussed, with suitable methods considered. Illuminating the discussion are relevant results from preliminary analysis of CAN traffic conducted in the early stages of this thesis. The chapter also discusses the options for generating experimental CAN data.

Chapter 4 describes the detection methods evaluated in this thesis. Those methods are: Auto-Regressive Integrated Moving Average (ARIMA), Z-score and mean-comparison - which were used to detect attacks producing packet timing anomalies; and One-Class Support Vector Machine (OCSVM), Compound Classifier (CC) and Local Outlier Factor (LOF) - which were used to detect attacks producing data payload anomalies.

Chapter 5 describes the design of the experiment to evaluate the packet timing anomaly detection methods (ARIMA, Z-score and mean comparison). The generation of data for the experiments is explained, as is the processing that would be conducted on the CAN traffic in an implemented system.

Chapter 6 presents the results from evaluating the packet timing anomaly detection methods. The impacts of the data characteristics on the result scores, are also considered.

Chapters 7 and 8 follow a similar structure to Chapters 5 and 6, but describe the design and results of the evaluation of the packet payload anomaly detection methods (CC, LOF, OCSVM).

Chapter 9 discusses the results from both experiments and considers the implications for a production implementation. Also discussed are potential adaptations and improvements that might be made to the methods, their implementation, and their testing.

Chapter 10 presents the conclusions and evaluates the extent to which the research aims have been achieved. It also considers future work that could be adopted to advance the work presented in this thesis.

Chapter 2

CAN: Functionality, Vulnerability and Attacks

This chapter presents an overview of the automotive CAN network, focusing on the aspects relevant to cyber attack, and discusses the CAN vulnerabilities, and the attacks that have been demonstrated. The chapter commences with a description of the CAN protocol and a discussion of its cyber-vulnerabilities. The chapter concludes with a review of the attacks that have been demonstrated or proposed, and identifies the commonalities in these attacks with regard to the CAN traffic. Identifying such commonalities will inform the attack detection suggestions subsequently presented in this thesis.

2.1 CAN Description

Modern cars contain a few internal networks adopting different protocols tailored to the network's primary function. The networks are interconnected via gateways to facilitate secure communication between them. For example, the infotainment network might need to receive information from ECUs on a CAN so that visual or audible warnings can be displayed, volumes adjusted with speed, or voice commands processed. In fact, the expectation that ECUs throughout the car will communicate and share information is bolstered by the increasing emissions, fuel efficiency, safety and reliability standards that cars need to monitor [23]. Additionally, the sending and receiving of information from external systems is increasing as vehicles become more autonomous and connected [79]. The networks used to transfer information between components within the car tend to be wired networks. This is likely to be the case for the foreseeable future, since wired networks are better suited to the security and reliability needed by the critical car components, and in addition the requirement

to connect the components to the electric power source negates any advantage in wireless provision [83].

Car CANs follow the ISO 11898 standard [36]. Each car typically contains two or more CAN networks, which might operate at a high speed (500 Kbps or more) for real-time safety-critical functions, such as braking and engine management, or at a low speed (125 Kbps or less) for comfort functions, such as doors and windows. The CAN bits are encoded as 0 dominant, and 1 recessive. This is exploited by the CAN arbitration process, which decrees the order of precedence should multiple ECUs attempt to broadcast at the same time. Should this happen, the dominant 0 bit will overwrite the recessive 1 bit, and the node that is trying to send the recessive bit will detect the change and cease broadcasting. It will then wait and try to broadcast the frame later.

Although the CAN has been shown to be vulnerable to security violations, it has proven very reliable and robust. Protocol processes (discussed later) are included covering atomic broadcast, error detection, error signalling, error confinement, and recovery. Moreover, at the physical level, the CAN's twisted dual-line and differential-current signalling, cancel out the effects of electrical noise on the bus [23]. Di Natale *et al.* [19] report early CAN studies which concluded that it remained robust in the harsh vehicle environment, even with significant electro-magnetic interference. Ibrahim [35] cites that an undetected bit corruption within a CAN frame is unlikely within the lifetime of the car.

2.1.1 CAN Data Frames

There are four message frames broadcast on the automotive CAN:

- Data Frame: conveys data between nodes.
- Error Frame: sent by any node to indicate an error in the frame being transmitted.
- Overload Frame: set by a node to indicate that it is not yet ready to receive frames.
- Remote Frame: sent by a node to request data from another node.

Data Frame

Data frames convey data between nodes. A node will send a data frame either of its own accord, or in response to a request from another node. The structure of the Data Frame is given in Table 2.1.

Research into CAN cybersecurity often focuses on the data frame, since this structure is used to convey information between the ECUs and consequently might be altered to affect some component to the car's functionality.

The data component of the CAN data frame comprises 8 one-byte fields, though these might not all be used by the broadcasting ECU. Amongst the other fields in the packet is the identifier field, which contains an identifying ID that ECUs use to determine whether the packet is relevant to their needs. An ID should be unique to the broadcasting ECU, and is used by the other ECUs to decide whether the packet is relevant to them. However, the CAN protocol has no authentication, so any node could conceivably broadcast using an ID belonging to another ECU, or broadcast using a fully invented ID. Usually, broadcasting packets with invented IDs would not be useful to an attacker, since all other nodes would not recognise the ID and ignore the packet. However arbitration on the CAN dictates that when two ECUs attempt to broadcast packets at the same time, the packet with the lowest ID will have priority. Thus, as discussed later, packets with a very low fabricated ID value might be used to dominate the network in a Denial of Service (DoS) attack.

Table 2.1 CAN Data Frame Structure

Field	Description	Size	Purpose
SOF	Start Of Frame	1 bit	Indicates the beginning of the frame. Sent only when at least 11 successive recessive bits have been detected on the bus.
ID (Arbitration field)	Identifier	11 bits	Identifies the content of the messages on the bus. While two devices might typically broadcast using different IDs due to the content of the data, the ID does not identify any particular device. Also used in arbitration, with the most significant bit sent first.
RTR (Arbitration field)	Remote Transmission Request	1 bit	Indicates a request for data. 0 = transmission of a data frame; 1 = transmission of a data request. Also used in arbitration.
Control	Control	6 bit	Indicates the CAN format used and the size of the data field.

Data	Data	0 to 64 bits (0 to 8 bytes)	Contains the actual data, transmitted with the most significant bit first.
CRC	Cyclic Redundancy Check (Checksum)	16 bits	Calculated from the SOF, ID, RTR, Control and Data fields to detect a possible transmission error. Receiving nodes compare the transmitted checksum against their own calculation based on the received fields. The final bit in this field is always a recessive bit to allow time for the calculation.
ACK	Acknowledgement	2 bits	Always sent as recessive, which puts the transmitting node into receiving mode. The first bit in the field is updated to dominant by any node that successfully receives the frame based on the CRC calculation. The last bit in the field is always sent as recessive.
EOF	End Of Frame	7 bits	Recessive bits that always terminate the frame.
ITM	Interframe space	3 bits	Recessive state gap ensuring successive frames are separated.

Remote Frame

Remote frames are used by nodes to request data from other nodes. They are seldom used [35], and are not recommended by the international user's and manufacture's group Can In Automation (CiA) [10]. The remote frame structure is similar to the data frame except there is no data field and the Remote Transmission Request (RTR) bit is set to 1 (recessive). The remote frame's ID indicates the desired data, and both the remote frame and the data frame should have the same Control values. Because the remote frame's RTR is recessive, should

there be a clash with attempts to broadcast a remote frame and matching data frame at the same time, the data frame (with a dominant RTR) will win the arbitration.

Error Frame

A node might detect an error in a received frame, for example the Cyclic Redundancy Check (CRC) field might not tally with its own CRC calculation. As soon as it detects an error, a node generates a corresponding Error Frame. Error frames can be Active, which have an Error Flag comprising dominant bits; or Passive, which have an Error Flag comprising passive bits (this is discussed later).

Overload Frame

The overload frame allows a receiving node to force a wait period on the CAN bus until the node is ready to process incoming data. Overload frames have the same structure as the error frame (6-bit Overload Flag followed by 8-bit Overload Delimiter). However, unlike the error frame, the overload frame is not broadcast at a time that will curtail an existing frame broadcast. A mode might be temporarily unable to process incoming data or handle requests for data due to internal operations, for example. In this case, the node will transmit the overload frame starting at the first available interframe gap bit.

According to Di Natale *et al.* [19], the overload frame is unlikely to be needed by modern controllers, although they are still required to participate in the overload frame process.

Extended Frames

The 11-bit Identifier fields in the original CAN protocol (known as 2.0A) allow 2048 different values. To enable a greater range of Identifiers, CAN 2.0B, which was introduced in 1985, allows an extended Identifier. The extended Identifier incorporates an additional 18-bit Identifier element after the standard 11-bit Identifier. These two Identifier components are separated by two recessive bits, meaning that during arbitration, the extended frames carry a lower priority than standard frames.

2.1.2 CAN Bus Errors - Reduction, Detection and Confinement

The CAN protocol includes processes to detect and confine errors, as well as to reduce the likelihood of errors due to signal noise and timing misalignment. However, as we will see later in the chapter, some of these can be exploited during attacks. Of importance in error detection and confinement is the bit stuffing process used by the CAN bus. The CAN

protocol requires that for data frames and remote frames, the Start Of Frame (SOF), ID, RTR, Control and Data field portion cannot transmit more than 5 bits of the same polarity. Where the content of the frame would necessitate this, a stuff bit of the opposite polarity must be inserted after the 5^{th} bit. Receiving nodes ignore this stuff bit as they process the frame.

Bit Stuffing ensures the opportunities for timing-drift between nodes are reduced, thus preventing timing errors. Overriding it also provides a mechanism for signalling errors, as in the case of the 6-bit single polarity Error Flag of the error frame.

When any node detects an error in a frame, that node will globalise the error notification and cause the frame to be discarded by all nodes. The error detecting node immediately broadcasts an error frame comprising a 6-bit Error Flag of single polarity bits, followed by an 8-bit Error Delimiter (recessive).

The consecutive Error Flag bits of the error frame deliberately violate the bit stuffing policy. Other nodes will also broadcast an error frame, either because they also detect the error, or because they detect the 6-bit Error Flag being broadcast. In this latter case, the error frames of the responding nodes will consume the 8-bit Error Delimiter period that would have been broadcast by the original error detecting node. This results in effectively a 20-bit error frame (2x6-bit Error Flags + 8-bit Error Delimiter). After the 8-bit recessive Error Delimiter there is a 3-bit recessive interframe gap, after which the transmitting node will attempt to resend the same frame. Thus, an error not originally detected by all nodes could result in a 23 bit-time delay before the erroneous frames can be rebroadcast at the first opportunity [86].

If all nodes detect the error, hence send their error frames at the same time, the delay until the earliest opportunity to rebroadcast the frame is 17 bit-times (error frame + interframe gap).

Error Types

The CAN bus protocol identifies five types of error that might occur on the CAN bus [86]. Two of these (Bit Error and Bit Stuffing Error) operate at the bit level, while the remaining ones operate at the frame level. Any node detecting one of these errors will issue an error frame, causing the current frame to be flagged as faulty. The errors are presented in Table 2.2.

Table 2.2 CAN bus errors specified by the protocol.

Error	Description	Node detecting
Bit Error	A node will monitor the CAN bus during its own broadcasts. If the level on the bus is different to the level it has transmitted, it will immediately broadcast an error frame. Bit errors do not happen at the arbitration stage (since other nodes might legitimately be changing the CAN polarity) or at the ACK stage.	Transmitting Node
Bit Stuffing Error	If any node detects that the signal polarity is consistent for 6 bits, it will issue an error frame.	Receiving Node
CRC Error	The CRC is calculated based on the values in all fields of the frame up to and including the data frame. Each receiving node will compare the transmitted CRC against their own calculation. Any receiving node that observes a mismatch will broadcast an error frame commencing at the start of the incoming EOF.	Receiving Node
Frame Error	A missing required field or field element. Such errors might be detected with the CRC, ACK, EOF and ITM fields.	Receiving Node

ACK Error	The transmitting node will set the ACK bit to recessive, and expect it to be altered to dominant by any node that receives the frame and agrees the checksum. If the transmitting node does not detect a change in the ACK bit to dominant, it will assume no node received the frame and attempt to resend it.	Transmitting Node

Error Confinement

The CAN bus has two processes to try to confine errors. These processes also seek to prevent the bus from being jammed with frames from faulty nodes.

Firstly, erroneous frames get destroyed by the error frame sent by any node that detects an erroneous frame. The node that issues the error frame need not be a node that is interested in the erroneous frame, since frames are read by all nodes. In addition, a node might detect errors in its own frame, for example during broadcast of non-arbitration fields if the bit polarity it detects on the bus differs from the polarity it is broadcasting.

Secondly, the CAN protocol provides a mechanism to remove the transmit rights and receive rights of a node. Thus, a node that continuously broadcasts erroneous frames will be excluded from broadcasting onto the network. To achieve this, each node maintains the following counters:

- Transmit Error Counter (TEC): increments by 8 for each frame the node is unsuccessful in transmitting; and if > 0, decrements by 1 for each frame that the node successfully transmits.
- Receive Error Counter (REC): increments by 1 for every errored frame the node receives; and, if > 0, decrements for every frame the node successfully receives.

There are a few exceptions and additional rules governing the updating of counters, which are discussed in [19].

All nodes start out in Error Active mode (TEC < 128 and REC < 128), which means they can send and receive frames, including error frames with an Error Field comprising dominant bits.

A node will go into Error Passive mode whenever either of its counters reaches 128 or beyond, implying that the node is suspected of faulty behaviour. An Error Passive node cannot transmit error frames with dominant Error Fields, hence it cannot destroy frames, but it can still receive frames and send error frames with passive Error Fields. However, since the passive frames contain recessive Error Fields, they are likely to be overwritten on the network and so would not warn other nodes of the error. Should both of its counters drop to 127 or less, the node will once again be able to broadcast error frames (i.e. Error Active mode).

If a node's TEC counter continues to rise, reaching beyond 255, the implication is that the node is possibly corrupted. The node will switch to Bus Off mode, preventing itself from broadcasting any dominant bits until it is reset and 128 x 11 bit times has passed [10].

Ensuring CAN Bit-reading Timing Consistency

Frames need time to travel to the furthest nodes along the bus. In addition, the bus needs to ensure there is time for all nodes to check the frame and acknowledge an error-free receipt. This acknowledgement is done by the receiving node switching the single-bit ACK field from recessive to dominant. For the ACK, the transmitting node switches to receive mode. If it detects the change in polarity in its ACK during broadcast, it knows that at least one node has correctly received the frame. To ensure there is sufficient time on the bus to broadcast the bit to each node and register any discrepancies that might stem from another node broadcasting an opposing bit, as well as allow for changing the current between bits, the length of broadcast of each bit is important, and needs to be sufficiently long, consistent and recognised by all the nodes. Given that the network consists of individual nodes, each with its own clock, a drift in timing alignment between two nodes could easily result in an inconsistent parsing of the signal. The CAN protocol ensures bit rate alignment by bit stuffing, and by specifying the bit broadcast length on any particular network.

2.2 CAN Vulnerabilities

This section discusses the vulnerabilities of the CAN and summarises attacks that have been demonstrated which exploit those vulnerabilities. Somewhat ironically, the safety and efficiency features of the CAN already discussed are also an inherent source of security vulnerabilities. These are summarised in Table 2.3 and have been exploited in the attacks described later in this section.

Table 2.3 CAN features and associated vulnerabilities.

Characteristic	Description	Advantages	Vulnerability
Unencrypted, untargetted broadcast.	Any device on the bus can send any frame and receive all frames.	Speed and efficiency of broadcast and processing.	Fuzz testing and network traffic reading for reverse engineering.
Weak authentication	Sending devices are not identified, and receiving devices cannot be individually targeted. Instead, messages are broadcast to all nodes, and it is up to individual nodes to decide whether to process the received message.	Bus is "hot-pluggable". Flexibility for devices to determine their own data needs.	Spoofing and unauthorised ECU reflash.
Arbitration by message priority	Each message has a priority (the lower the ID, the higher the priority).	Important messages take precedence.	DoS by broadcasting high priority messages.
Error detection response and self-removal.	All devices on the bus can detect errors and notify all other devices. Devices can remove themselves from the network if they frequently receive erroneous data, or frequently transmit it.	Lessened disruption of normal operation.	ECUs can be forced off-bus.

Although malicious access to sub-networks in a vehicle should be protected by sufficiently secured gateways, this has been shown not to be the case. Moreover, the requirement for components across the car to communicate and share information, often as legal requirement such as monitoring emissions, introduces problems and complexities to ensuring a robust gateway [7]. Checkoway *et al.* [14] and Tencent's Keen Security Lab [78] were both able to bypass CAN gateways in production cars, as well as reprogram the gateway functionality.

Hoppe *et al.* [32] also managed to compromise a gateway ECU connecting the internal subnetworks, as well as connecting the external network which is accessible from the OBD port during normal operation. They achieved this by identifying, through running and analysing a diagnostic session, a flaw in the gateway that allowed some unauthorised messages to pass through the gateway when requested via forged diagnostic requests.

Some of the attacks discussed below used the ODB port to access the CAN (e.g. [24, 32, 43, 85]), including remotely via a phone app [91]. However other access points have been demonstrated. A detailed analysis of potential attack surfaces was carried out by Checkoway *et al.* [14]. As well as direct access via the OBD port, they were able to inject packets onto the CAN bus via the car's media player. They were also able to gain access to systems in the car via the car's Bluetooth, FM RDS and cellular systems, which they could then exploit to compromise the CAN ECUs. For example, they reflashed the media ECU to trigger the broadcast of pre-determined CAN packets when a particular Program Service Name message was received over the FM RDS channel. Koscher *et al.* [40] demonstrated situations in which CAN ECUs were illicitly reflashed via the OBD port and used to stage subsequent attacks. They were able to reprogram ECUs that should have prevented this from happening, including reflashing whilst the car was being driven. They concluded that "many ECUs in our car deviate from their own protocol standards, making it even easier for an attacker to initiate firmware updates or DeviceControl sequences" (p7). In a well publicised attack, Miller and Valasek [51] remotely hacked a Jeep Cherokee via a cellular connection to its infotainment system, enabling them to inject CAN messages which controlled the steering, brakes, and acceleration. Although, prior to the attack, the authors needed to perform a preliminary analysis of the car requiring physical access, no modifications or additional devices needed to be added to the car during the attack.

What is clear from these studies is that the impenetrability of the automotive CAN cannot be assumed. Access might be gained through inadequacies or imperfections in gateways; devices might be connected directly to the CAN; or internal ECUs might be illicitly reflashed to act a internal rogue units. Increases in connectivity, and the uptake of over-the-air updating, are likely to present more opportunities for breaching.

2.3 CAN Attack Models and Effects

Having discussed the vulnerabilities of the CAN and the possibilities for malicious access to the CAN bus, this section summarises the effects of attacks that have been demonstrated. The goal of this section is to understand the potential intentions of an attack, as well as the

likely effects on the bus. It should be appreciated that the attacks described here present only a flavour of what might be possible, depending on the ingenuity of the attacker.

Checkoway *et al.* [14] reflashed media and telematics ECUs so that they broadcast predetermined packets onto the CAN bus disrupting operations such as tire pressure monitoring. In their detailed exploration of attacks possible via the OBD port, Koscher *et al.* [40] were able to manipulate many functions by injecting one or more packets. In a car being driven on the road, they were able control braking, engine-control and engine component configuration, locks, windscreen wipers and dashboard readings. They were able to control additional functions when the cars were stationary. Their attacks comprised three approaches:

- Packet Sniffing and Target Probing - observing the packet traffic as various car functions are tested, to try to determine what packets pertain to what functions.
- Fuzzing - iterative testing using random, or partially random packets. The authors also discovered this to be a simple way to stage disruptive attacks.
- Reverse Engineering - using a memory reader and debugger to understand ECU code.

As already discussed, a series attacks have been demonstrated and publicised by Valasek and Miller [51–53, 85], including attacks via direct access through the OBD port, and remote, indirect access via cellular networks connected to the internal infotainment systems. They replayed packets onto the CAN to achieve a wide range of adverse effects, such as: causing the display of erroneous information, such as speedometer or odometer readings; manipulating functions, such as steering the car during auto-parking or braking the car via the pre-collision system; or failings caused by overpowering the CAN network, such incapacitating the power assisted steering, preventing the driver from making hard turns.

The ability to manipulate data packets on the CAN does not automatically mean that we can infer the ability to control some functionality of the car. In their study of the CAN traffic of two cars, Valasek and Miller [85] reported observing many instances where spoofing a packet and its data values did not trigger any action. For example, a packet that indicated how much the brake was pressed did not cause the brake to engage when replayed. Reasons for this, the authors cite, could be that i) the original ECU might still be broadcasting the correct data, confusing the recipient ECU with conflicting data; ii) the ECUs have many safety features built in (such as only acting on a Lane Keep Assist steering wheel turn that is less than 5%). The authors point out that circumventing the first challenge requires broadcasting spoofed data at a higher rate than the legitimate data. The second challenge might be circumvented by tricking the ECU in some way – indeed Valasek and Miller were able to determine the car operations and ECU interactions that rendered a variety of

attacks achievable on the two cars they studied. Clearly, overcoming both challenges implies dedication and depth of the knowledge from the attacker. Not only would the attacker need to understand the packet data, but would need to ensure the normal CAN processes are not breached. For example, legitimate packet broadcasts need to adhere to the Cyclic Redundancy Check (the CRC field should have a checksum value that considers the data field amongst others) and bit stuffing (which requires a change in bit value after five consecutive matching values). In normal running, these safety features might reduce the likelihood of erroneous data being accepted.

Hoppe *et al.* [32] used a simulated automotive CAN network built using a Vector CANoe package and attached production ECUs to test attack scenarios. In the first scenario an ECU was recoded to disrupt the window functioning by issuing an open window request when the car's speed exceeded 200 km/h. For their second attack the ECU was recoded to disrupted functioning by injecting counteracting messages, e.g. a close window request each time an open window request was observed on the CAN. Similar "Read and Spoof" actions were also tested that overrode legitimate indicator signal light requests. Their third attack emulated the diagnostic session response of the airbag control unit, masking that it had been removed. The authors also concluded that it would be possible to spoof the regular messages broadcast for the ECU gateway, which would control any driver-side airbag warning lights. This masquerade attack would not necessarily alter the frequency of messages, since the legitimate node is assumed to be removed or incapacitated.

Lee *et al.* [43] conducted a systematic fuzzing attack on three mid-range automobiles, by sampling their CAN traffic via the OBD port and then creating fuzzed packets comprising the CAN IDs and random data in one of the data fields, with zero values in the remainder. These were reinjected via the OBD port every 10 ms. Acknowledging the tightness with which the manufacturers guard their CAN databases, and thus the challenges facing any hacker, the authors deliberately avoided making any analysis of the packet contents or attempts to reverse engineer, or interpret the data, or to identify any recognizable ECU activity. They identified 26 CAN IDs, and in 14 of these they observed changes in the activity on the instrument panel displays, which the authors assumed would map to corresponding ECUs. In addition, the authors believed that three CAN IDs produced a change in the actual behaviour of the car, though they were unable to clarify the pattern.

Cho and Shin [17] demonstrated bus-off attacks, where attack collisions generated by injecting spoof packets, forced CAN ECUs to self-deactivate in accordance with the CAN protocol specification for error containment. Cho *et al.* [16] extended these into attacks that might take place whilst the vehicle is parked and switched off. Even in this state, many ECUs are monitoring the bus using reserve power supplies. The authors state that the bus

activity needed to wake up the ECUs is simple, and standard to the CAN specification; and would be met by pretty much any legitimate structured message pertaining to the CAN ID. Battery drain could thus be achieved by continuously broadcasting the message to keep the ECU awake, or by waking, and then controlling, the ECUs that govern processes that are high battery drainers. Significantly, the attacked ECUs needn't be those directly causing the battery drain. For example, the authors found the light controls difficult to reverse engineer, but were able to invoke the lights in spite of this, by controlling the doors and boot, thus activating the interior lights. In another ignition-off attack (Denial of Body-control), the ECUs were woken by injecting packets onto the CAN and then the CAN bit rate was temporarily switched from 500 kBits/s to 250 kBits/s and packets broadcast, resulting in the ECUs registering errors that sent them into a bus-off state. Since many ECUs are not programmed to automatically recover from being bus-off, many functions were incapacitated, including key-less entry. Functionally was only returned to the car after the battery had been disconnected for a few minutes and reconnected. Instead of changing the bit rate, the authors state that the bus-off attack can also be induced by changing the internal/net resistances or capacitance. The authors tested their attacks on 11 cars of differing makes or models. In post 2015 cars they found that typically over 50% of CAN IDs sent when the ignition was on, would still be transmitted if the ECU was awakened whilst the ignition was off; which they cite as an indication of the wide use of ECU wake-up functionality that might be exploited by an adversary.

Froschle and Stuhring [25] considered a range of attacks that might be mounted once the attacker has access to the CAN, and tested these on a CAN simulation rig comprising the SILAB driving simulator and connected ECUs. They assumed the attacker had been able to compromise one node on the CAN, such as the gateway ECU, so was able to run her own code on it. The attacker would not need further access to the vehicle during the attack, launching the attack via the malware on the compromised node, and would not need to compromise any other node during the attack. Adopting this assumption, they demonstrated fuzzing attacks, impersonating attacks and injection attacks. For simple injection attacks they injected at high frequency, packets with very high priorities (e.g. 0x0) to block lower ranking IDs. They also flooded the network with dominant bits to fully incapacitate it. This typically sent at least one node into a bus-off state within 1ms (or 260 bit-time). Another of their attacks targeted specific IDs by mimicking and synchronising their broadcast, hence exploiting the arbitration and error containment mechanisms to suppress the legitimate broadcast. These attacks resulted in the mimicked ECU eventually being forced off-bus. For example, in the case of an ID with a repeat broadcast period of 20ms, the ECU was forced off-bus typically

within 8.8ms to 13.9ms when its original broadcast and immediate repeat attempts were suppressed by spoofing.

2.3.1 Attack Characteristics

The attacks described above have a variety of approaches and effects, such as exploration or probing, denial of service or disruption, control or manipulation. They have been instigated from gateway breaches, devices attached to the CAN, and from compromised nodes acting as rogues. In spite of their variety, they are likely to have two basic effects on CAN traffic: changes to packet timings; and, changes to packet data payload.

Many of the attacks will alter the frequency of CAN packet broadcasts. These might be delayed or missed, such as in the disruptive attacks by Froschle and Stuhring [25] that flood the bus with dominant bits to disrupt broadcasts, or that targeted individual IDs by synchronously broadcasting spoofed packets, or prevented ECUs from broadcasting by forcing them into a bus-off state. The packet broadcast can also be expected to be missed if the ECU is temporarily deactivated during a reflash, as proposed by Koscher *et al.* [40]. For many attacks, though, the broadcast of a particular packet will increase. Attacks, such as those by Hoppe *et al.* [32] result in an increase in the number of messages for a CAN ID – the legitimate messages still being broadcast, plus the injected, counteracting, spoofed messages. The injection of spoofed packets is likely to need to be done at rates comparable to, or above, the broadcast of the legitimate packet in order to override the legitimate values [76]. Similarly, fuzzing attacks are likely to results in increased appearance on the CAN of the target CAN ID.

However, not all attacks can be expected to affect timings and broadcast rates [30]. The masquerade attacks proposed by Miller and Valasek [52, 85] and Froschle and Stuhring [25], for example, have a legitimate ECU that has been incapacitated, with a masquerading ECU broadcasting fake packets in its place at the same frequency. It is also possible that other action might be taken to cover any broadcast time-shifts. Sagong *et al.* [64], for example, have demonstrated a *cloaking attack*; an intelligent masquerade attack, in which the adversary adjusts the time of spoofed messages to correlate with the mimicked ECU.

The second aspect that the attacks are likely to alter is the payload data values. In fuzzing attacks these may be randomly generated, while predetermined values might be used in attacks that seek to purposefully manipulate the vehicle. Of particular interest here are fabrications that include faked, but plausible, values. These values are likely to registered as legitimate by the receiving ECU.

The attacks might entail subtle and convoluted data fabrications. For example, Miller and Valasek [52] describe an attack that invoked the Park Assist System to manipulate the

steering on a Toyota Prius. Their attack had to first inject messages to fool the ECUs into believing the car was in reverse and travelling at less than 4 mph. Although the sensor value messages the authors injected did not alter the car's speed, they were read by some ECUs as signifying the actual speed, thus enabling activation of the speed-restricted Park Assist Mode, which in turn responded to fake steering command messages injected onto the CAN. The forged messages injected during this attack were legitimate and plausible, containing values that would be expected in another situation.

2.4 Conclusion

Although the automotive CAN is robust and suited to the in-vehicle environment, it lacks the security features expected in a connected network, and has subsequently been shown to be vulnerable to cyber attack via a range of channels. Attacks have been demonstrated that affect safety and performance functions of the car ([52, 14, 32, 51, 53, 85]). However, these have been specific to each car, requiring either existing knowledge of the CAN dictionary for that car model, reverse engineering, or systematic fuzz-testing to derive CAN dictionary knowledge. Such attacks have focused on manipulating specific CAN packet IDs pertinent to a given car model. Whilst testing the detection of such actual attacks would provide useful proof of an attack detection system's potential for identifying a predefined attack, we also need to consider how the detection systems could work across a broader range of potential attacks, including those not yet envisaged. Moreover, the testing of such systems needs to be demonstrated across car makes and models. The testing in this thesis therefore attempts to meet this need.

Whilst attacks might affect both packet timing and payload, attacks that would be manifest only in changes in payload have been proposed (e.g. [85, 25, 30, 64]). The focus of this thesis is therefore to determine methods to independently detect anomalies in both CAN packet timings, and in the CAN packet payload. Where possible, the detection should be applicable across all CAN packets, and not tailored to one attack model or car, allowing a system that might be deployed broadly and not constrained to detecting only currently specified attacks.

Chapter 3

Related Work - CAN Intrusion Detection

This chapter discusses the options that have been proposed for CAN attack detection, as well as considering the options for generating the data to test those detection proposals. Whilst inspiration and ideas can be drawn from reviews of computer-network intrusion detection (e.g. [9, 26, 71]), the nature of the automobile CAN bus warrants some considerations that differ from an intrusion detection method that might be implemented on a server or computer. As discussed in Chapter 1, differentiating factors include the low processing power available to the ECUs on the CAN; the lifecycle, maintenance and usage patterns of the car; the rapid generation of CAN data records, and the limited CAN dictionary sharing by automotive component manufacturers. Also, the safety critical nature of the CAN bus and the nature of driving make it important to achieve high accuracy, especially reducing false positives, which would cause unwarranted intervention during driving or distractions for the driver.

3.1 CAN Intrusion Detection

Intrusion detection is based on the assumption that the behaviour of intrusions will differ from the behaviour of legal activity, and a few analytical methods have been proposed as potential candidates for a CAN intrusion detection system (IDS). Such methods can be categorised broadly as *signature based* and *anomaly based*; a distinction made by other researchers across the computing field [55, 60, 61, 71]. The anomaly based approaches are further categorised into their methods employed, such as statistical, knowledge based, clustering/density based, support vector machines, neural networks, and Markov models. The latter four categories are examples of Machine Learning (ML).

Table 3.1 presents a general summary of computing IDS methods in terms of overheads, advantages and disadvantages. The Set-up overhead column lists the relative machine effort to establish or up-date the decision algorithm, based on opinions expressed in published

sources [9, 20, 29]. The training for clustering algorithms can be particularly high since it may involve nested iterations through the training data-set to identify the cluster neighbours. That said, the training overhead across all the methods is high, especially considering the low processing available on ECUs on the CAN bus. Training therefore might not be viable in-line. Training of signature, statistical and knowledge-based methods relies mainly on human and disparate processes that are unlikely to be amenable to automated training, so the overheads are left blank in the table.

Of course, general estimates of the overheads, such as shown in Table 3.1, are not definitive since implementation variations will affect processing and workload, so would require a thorough analysis of the training algorithms. For example, grouping the data points as dense and connected regions can reduce the overhead in clustering methods [4, 9]; likewise, the adjustment of parameters for the SVM [29]. Nevertheless, the overheads would need to be considered and minimised if in-line CAN training is considered.

Table 3.2 summarises the CAN studies that have tested the IDS methods, and which are discussed later in this Chapter. It also shows the attack manifestation that the studies sought to identify: packet flow (changes in the timing or number of packets), and payload manipulation (changes to the values in the packet data fields).

Table 3.1 Intrusion detection method overheads, advantages and disadvantages.

Method Category	Overhead			Advantage	Disadvantage
	Set-up	Processing	Memory		
Signature Based		Low	Medium	Low false positive rate for known attacks.	Assumes the attack signature is already known. Easily bypassed by modified attacks [71]

Continued on next page

Table 3.1 – continued from previous page

Method Category	Overhead			Advantage	Disadvantage
	Set-up	Processing	Memory		
Statistical		Medium	Low	May require no prior knowledge about normal activity, and can identify attacks evolving over a long period [26, 61].	Can be retrained by attacker; and not all behaviours can be stochastically modelled [26, 61]. Thresholds may be difficult to determine [61]. Unlikely to capture the interactions between attributes [12].
Knowledge Based		Medium	Medium	Low false positives.	Knowledge may be difficult to develop. May be difficult to maintain [61].
Clustering or Density Based	$O(n^3)$	High	High - may require matching against full training data set.	Simplicity and understandability [9, 60].	Needs feature reduction to reduce high dimensionality [9]. Classification time can become large as the number of neighbour comparisons increases.
Support Vector Machines	$O(n^2)$	Medium	Low	Can deal with high-dimensional data and small samples [9, 71].	Resource-hungry training [60]. Prone to high false positive rate [61]

Continued on next page

Table 3.1 – continued from previous page

Method Category	Overhead			Advantage	Disadvantage
	Set-up	Processing	Memory		
Neural Networks	$O(emnk)$ where: e epochs, m features, n observations, k hidden neurons.	High	Low	Adaptability to environmental changes [26]. Ability to generalise and cope with noise [41].	Lack of descriptive model explaining decision [26]. Prone to high false positive rate [61].
Hidden Markov Models	$O(nc^2)$ where: c number of states	Medium	Low	Used extensively in IDS [26]. Suitable for assessing transition sequences [55].	Results are dependent on assumptions about the system behaviour[26].

Table 3.2 Intrusion detection methods used in automotive CAN research.

Category	Study	Intrusion Detection	
		Packet Flow	Payload Manipulation
Signature Based	Studnia *et al.* [75]: attack signatures modelled using language theory.	X	X
Statistical	Ling and Feng [45]: consecutively broadcast ID threshold count.	X	
	Cho and Shin [17]: model of clock behaviours using Recursive Least Squares, with Cumulative Sum analysis to detect anomalies.	X	X
	Song *et al.* [73]: attack score based on message frequency.	X	
	Gmiden *et al.* [28]: time intervals between matching CAN IDs.	X	

Continued on next page

Table 3.2 – continued from previous page

Category	Study	Intrusion Detection	
		Packet Flow	Payload Manipulation
	Taylor *et al.* [76]: Message time intervals combined with payload statistics, assessed using T-Tests	X	
	Lee *et al.* [44]: Time interval of a remote request broadcast and the received response compared against mean.	X	
Knowledge Based	, , Marchetti and Stabili [47]. Transition matrix.	X	
	Studnia *et al.* [75] Legitimate state transitions.	X	X
Clustering or Density Based	Martinelli *et al.* [49]: Four nearest neighbour classifiers.	X	X
Support Vector Machines	Taylor *et al.* [76]: Message time intervals combined with payload statistics, assessed using One Class Support Vector Machine.	X	
Neural Networks	Kang and Kang [38]: Supervised Deep NN trained using normal and attack CAN packets.	X	
	Taylor *et al.* [77]: Anomaly detection in data bit words using Long Short-Term Memory recurrent neural network.		X
	, Wasicek and Weimerskirch [89]: Root-Mean-Square function to decide the degree of anomaly from trained Bottleneck NN output.		X

Continued on next page

Table 3.2 – continued from previous page

Category	Study	Intrusion Detection	
		Packet Flow	Payload Manipulation
Hidden Markov Models	Narayanan *et al.* [59]: Hidden Markov Model detecting anomalous changes to speed and RPM.		X

3.2 Signature Based Intrusion Detection

Signature based intrusion detection assumes that the signature patterns of the attack are known. The discriminatory features from the signatures can thus be stored in a database, against which the features of subsequent packets are monitored. Packets with features that match the signatures on the database are flagged as violations.

Since they do not need to adapt to the deployed environment, signature based methods can be easy to deploy initially [71]. They can detect reliably and generate a very low false positive rate when the attack signature is known, and they can determine which type of attack the system is experiencing, which is useful for assessing the epidemiology of attacks [61].

However, signature based intrusion detection techniques have sometimes been rejected for CAN anomaly detection because the development, maintenance and deployment of signature databases is seen as impractical due to the nature of the industry sector, the life-cycle of the vehicle, and the still emerging patterns of attack [31, 57]. The non-disclosure of CAN databases makes the expected behaviour of the CAN traffic difficult for independent researchers to comprehensively map [52], and as a consequence, the boundary between a normal signature and an attack signature may be difficult to determine. Also, the CAN databases are proprietary and vary by manufacturer, further hindering the creation of useful signatures. In addition, the behaviour of an ECU might not be fully known [76]. As a consequence, the meaning of data, or indeed an understanding of what packets pertain to what functions, is unlikely to be readily available.

With regard to attack signatures, the newness of the automotive cybersecurity field, and the diversity of potential attack modes being uncovered (e.g. Checkoway *et al.* [14]), together with the ingenuity of hackers, suggests that it is probably unwise to assume all attack scenarios will be predictable. A further hindrance is that the efficiency of signature based detection may be degraded if the attack activity spans multiple packets [61].

Additional complications stem from the frequent updates required to keep the signature databases up to date. The long lifespan of vehicles and their dispersed locations makes updating the signature database difficult, and it is perhaps unwise to assume owners would be able, or motivated, to update their systems outside of the annual service [57].

In spite of the drawbacks, Studnia *et al.* [75] proposed a formal language-based approach to model both normal ECU behaviour and behaviour to known attack signatures. They designed models to detect four categories of attack: modified packets that do not conform to the communication protocol; periodic forged packets broadcast while the legitimate versions are still being broadcast; normal packets replaced with malicious forged ones; and, forged event-triggered packets. Their proposed system would first check whether the packet was compliant with the protocol, then perform a signature-based consistency check against the current system state and the ECU's model.

When devising the models, Studnia *et al.* had access to the specifications of the CAN network and the ECUs of the sampled car. Though they acknowledge that parts of the actual system might not adhere to those specifications. They present their system as a early proposal, present limited details of their attack simulations, but report that their system could detect intrusions in real-time, but it sometimes failed to detect an attack if it missed the first packets of the attack broadcast. Also, they recognise that their models were based on simple representations and attacks, and that the modelling of more complex situations (such as taking into account the timings between matching ID packets), might require a method that offers more expressive representations. When considering similar approaches, Taylor *et al.* [77] state that assuming a finite symbol dictionary might be unrealistic for CAN traffic, and the number of words for a particular ID might be infinite over the lifetime of the car.

3.3 Anomaly Based Intrusion Detection

Anomaly based IDS seek to detect behaviour that deviates from the normal. The assumption is that the normal behaviour can be sufficiently profiled and thresholds established to enable the meaningful identification of deviations.

In anomaly based intrusion detection, the detection is not based on set signatures, giving these methods the potential to generalise, and making it possible to detect previously unknown attacks [60, 61]. Further, the profiles used by the system are customised by the learning algorithm, which makes it difficult for the attacker to know what activity might trigger an alarm [61]. However, such systems require training and the collection of data that is sufficiently representative of normal traffic to enable accurate, reliable anomaly boundaries to be determined. In the case of CAN data this can be problematical both in terms of accessing

data from the CAN bus, and in terms of gathering data covering a representative range to driving situations.

Anomaly detection is prone to higher rates of false positives (wrongly classifying a legitimate event as an attack) and false negatives (failure to detect an attack) [61, 71]. Either of these outcomes is especially detrimental in the automobile CAN network. False negatives obviously so, while false positives would be a distraction to the driver, and might lead to ignoring future warnings.

Even so, the unpredictability of attack scenarios, together with the difficulties in determining comprehensive robust signatures, has led to many studies adopting the anomaly detection approach. Broadly, these approaches can be categorised into statistical based approaches; knowledge based approaches; and machine learning, which can be further categorised according to the machine learning model applied.

3.3.1 Statistical Based

Statistical methods compare the currently observed statistical profile of the system against a previously determined statistics profile, such as mean, median and mode. In the case of time series data, such as the CAN bus traffic, statistical methods might employ a rolling window. Statistics based techniques can by univariate or multivariate. Univariate techniques model each variable independently. Some factors regarding the manner of broadcast, such as timing intervals between CAN ID broadcasts, might be amenable to univariate analyses, but they are likely to be less useful for applying to the data contents of the CAN messages. Here for example, what appears to be a normal profile (for example, for the steering position) might actually be malicious in the given context. Therefore, multivariate methods might be better suited for detecting intrusions affecting the CAN ECU sensor data.

The complexity of the automobile network is cited as making it difficult to make a precise statistical model that would allow rare but legitimate patterns [75]. Even so statistical models have been considered for CAN intrusion detection, and some researchers proposing other methods have suggested statistical models as a supplemental check [77].

Ling and Feng [45] proposed using resettable counters and thresholds to detect any ID that is broadcast consecutively beyond a threshold-defined number of times. Such broadcasts might be indicative of a DoS attack. However, their reliance on known thresholds has been challenged [11].

Gmiden *et al.* [28] selected IDs broadcast at fixed time intervals, and used the ID's own regular frequency to monitor for increased broadcast frequencies indicative of injection attacks. The authors present their system as an early proposal, so do not present evidence

for the validity or scope of the application, likewise the range of timing variance is not empirically tested.

Song *et al.* [73] also considered the time intervals, and observed in their study car that the average time interval for IDs was reasonably regular when driven at normal speeds. Consequently, intrusion attacks were detected where an ID's packets were injected at half the normal time interval. It is, though, unclear from their paper how many of the CAN ID set the testing was conducted on.

Cho and Shin [17] proposed a clock-based IDS to detect fabrication, suspension and masquerade attacks. In fabrication attacks, a compromised ECU injects forged packets with a view to distracting receiver ECUs. Suspension attacks entail an ECU being compromised so as to suspend its transmissions. The masquerade attack is potentially harder to detect since it does not necessarily alter the frequency of any ID broadcast, and involves an ECU learning the broadcast patterns of another ECU, which it then forces to suspend broadcasting and itself takes over the broadcasting. Cho and Shin proposed that their method could detect three attacks, and could pinpoint the attacking ECU. Their system uses message periodicity to estimate the transmitter's clock skew (the difference in its frequency compared to clocks in other ECUs). A norm model is then constructed using Recursive Least Squares, and intrusions detected using a Cumulative Sum analysis to detect patterns of change which are used to determine attacks.

Hoppe *et al.* [31] devised an attack scenario in which hazard warning lights are rendered inactive by a malicious component that broadcasts a mimic packet to turn off the lights whenever it detects a legitimate packet calling for them to be activated. Modelling the flow of packets using an attack simulation, they observed that the frequency of broadcasts for the packet rose beyond the maximum $22(\pm 1)$ per second seen in normal operation (when only the legitimate version of the packet was broadcast). In addition, the semantic meaning of the packet's data ("on" or "off") in the previous 8 packets switched more frequently than the 0 or 1 changes observed during normal operation. Thus, they treated more than 28 packet broadcasts per second, combined with more than 4 inversions in the previous 8 packets, as signifying an anomaly.

Lee *et al.* [44] considered the time intervals between the broadcast of a remote request and response packets as a means of detecting three types of attack: a) DoS by packet injection; b) fuzzing attack by packet injection; c) impersonation attack in which an ECU impersonates another ECU that it has incapacitated. The authors reasoned that these attacks might incur changes that would affect the interval between a node making a remote request and receiving the next message bearing the requested ID. Such timing changes might be brought about by the changed traffic affecting packet arbitration sequences, or, in the case of impersonation

attack, changes in the distances between nodes, the clock cycles, the processing mechanisms or the physical ECU qualities. They measured variations in the request/response timings in four ways: 1) offset (number of messages on the CAN between the remote request and message with the matching ID); 2) the proportion of requests with an offset of one (i.e. the next message after the remote request was one with the requested ID); 3) the proportion of requests not responded to in the next six messages (an ECU was deemed to be out of action in this case), and 4) time interval between the request and the matching ID'd message. Conducting DoS and fuzzing attacks that injecting records via the ODB port, for each of several IDs tested they observed a decrease in offsets, and an increase in none-responses.

3.3.2 Knowledge Based

Knowledge based techniques deduce a set of rules from the training data. Subsequent events are then classified against these rules. These techniques can produce a high capture rate if the knowledge base is comprehensive enough; and a low false positive rate since such techniques are less likely to automatically classify previously unseen evens as attacks [26]. However, their drawback is that knowledge might be difficult to acquire or encode [26]. This is potentially the case in CAN traffic, and the issues discussed above for the Signature based techniques are also applicable to the Knowledge based ones.

Marchetti and Stabili [47] proposed building a transition matrix, which contained the legitimate paired sequences of CAN IDs. Consecutive messages broadcast onto the CAN would then be validated against the matrix; and messages broadcast out of sequence, flagged as anomalies. Such an approach would require very little storage and have low computational costs. Based on tests conducted from 10 hours of driving data in different traffic conditions, they concluded that this approach gave no false positives, and it reliably detected simulated attack data where random sequences of IDs were inserted into the readings. However, the method was less successful at detecting replay attacks wherein known sequences are rebroadcast. For these attacks, the sequence is previously seen, so treated as legitimate, consequently, any anomaly can only be observed at the beginning or end of the injected sequence. The authors propose the addition of statistical detection methods to cope with such possibilities.

Whilst such a matrix approach based on the next legitimate ID might usefully detect mismatches where an ID is known to be followed by an ID from a very small subset, it would be less successful where an ID could be followed by an ID from a large subset. My analyses of CAN data from a popular family car shows this to be the case for many of the IDs (Fig. 3.1).

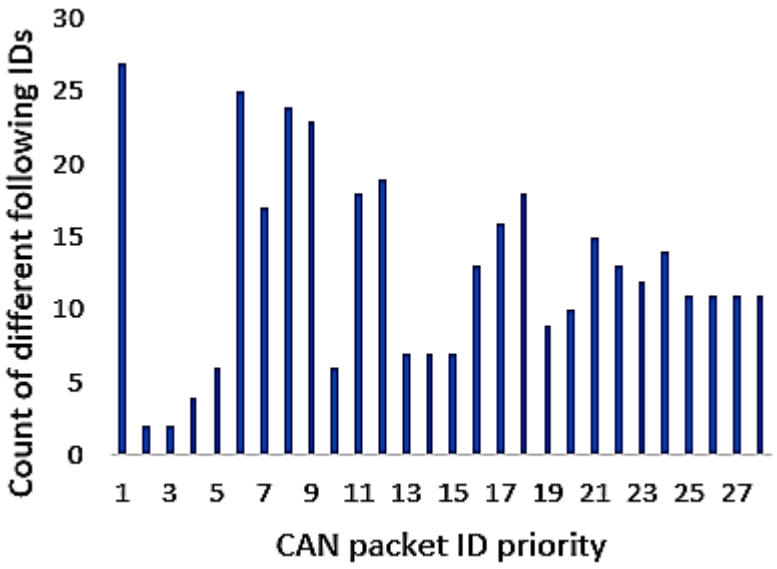


Fig. 3.1 Count of different packet IDs observed to follow each CAN ID in a popular UK family car. The CAN ID rank, rather than ID value, is represented on the x axis. The data was obtained from logs spanning over 10 hours driving over a range of roads types and conditions.

3.3.3 Open Source IDS

Within the field of intrusion detection for general computer networks, a range of rule-based open source packet sniffers with included IDS suites have been developed, such as Snort [18] and Suricata [80]. Although usually run on a general purpose computer, some, including Snort, have been shown to be usable on a Raspberry Pi [42]. The reliance on predefined rules and known signatures, which must be stored and regularly updated, as well as interrogated for each packet during detection, has been seen as barrier to their use in front end detection systems (e.g. [9, 1, 5, 56]). Such systems have been devised for detection on common computer networks and have not been considered in any of the CAN papers reviewed for this thesis, moreover no mention of any compatibility with the CAN protocol could be found on their web sites or forums, although there is some discussion suggesting current incompatibility [69]. However, even if compatibility could be gained, the determination of suitable signatures and rules for detection would still need resolving, as would the mechanisms for deployment and regular rule-list updating.

3.4 Machine Learning

Machine Learning (ML) methods typically attempt to learn patterns in the data by processing samples on which the status of the packet (e.g. "normal", or "intrusion") is known. Learning can either be supervised or unsupervised. Supervised learning uses a training data set containing labelled normal and anomalous data, whereas unsupervised learning uses a training data set in which the labelling of the data is unknown; the machine learning algorithm will then attempt to derive the classes based on some similarity. Some unsupervised machine learning methods offer variations that seek to determine outliers after being trained just using data from known normal instances.

During one-class classification training, data taken from normal operation is analysed to determine a threshold boundary that encompasses the normal instances. Subsequent instances are then classified as normal if they reside within the boundary, or anomaly if they lie beyond.

One-class classification fits situations that are relevant to automotive network traffic analysis. It is, for example, suitable where training data instances of attack behaviour are difficult to generate or predict, or where the simulated attack class risks being too narrowly defined to enable generalisation [90].

One-class ML classifiers have been successfully used across a number of domains [39] and are clearly relevant for the analysis of automotive network traffic where the variety of the attack modes, and the complex lifecycle of the vehicle, suggest it would be unwise to assume knowledge of the entire range of attack profiles. The one-class classification approach has therefore been proposed as a likely CAN intrusion mechanism [76, 81, 46].

3.4.1 Clustering and Outlier Detection

Clustering ML techniques adopt the assumption that instances of a particular class have data profiles that group them into clusters around an archetype. Each new instance can then be classified according to its distance from that archetype. Approaches to outlier detection have employed multidimensional distance calculations, enabling multivariate analysis, often combined with density calculations [61]. There are a number of distance calculations, but often the Euclidean distance is used, usually with the data normalised to ensure the dimensions carry equal influence.

Clustering approaches can be applied to unsupervised learning [61]. For unsupervised training on data that includes multiple classes, the assumption is that the anomalies will form the smaller of the clusters. For unsupervised training that contains only normal data, a density measure and distance calculation could determine the edge of the normal class boundary.

Martinelli *et al.* [49] tested nearest neighbour classifiers in an attempt to distinguish between normal CAN packets and simulated attack CAN packets solely from the data fields, irrespective of IDs. They staged four types of attack that injected packets, and initially compared sample metrics for each data field to establish that it showed a statistical significance between the normal data and the fields generated in the attacks. Having established this, they then constructed classifiers that incorporated all eight data fields, which they trained and evaluated using labelled normal and attack generated packets. The classifiers were: Fuzzy-rough K-nearest neighbour; Discernibility kNN Classifier, and Fuzzy Unordered Rule Induction Algorithm (FURIA). The four types of injection attack tested were: DoS (a dominant 0000 CAN ID, with 0 value data fields, injected every 0.3ms); fuzzing (random ID and data values every 0.5ms); and, gear and RPM fabrication (CAN IDs related to gear or RPM information, injected every 1ms). Other than describing that the normal data and attack data resulted in statistically different data sets, the authors discuss little of the gear and RPM attacks, such as the extent of the data manipulation, though they have published the raw data. All the classifiers were able to classify inserted packets containing the gear and RPM data with full accuracy, though they were not as accurate on the DoS and fuzzing attacks. The authors concluded that the best results across all attack scenarios were achieved with the Fuzzy-rough K-nearest neighbour, which showed false positives rates ranging from 0 to 0.038 across the data sets. It achieved a precision of 0.963 and recall of 1 in the DoS attack; a precision of 0.85 and recall of 1 in the fuzzing attack, and precision and recall of 1 in the gear and RPM attacks.

3.4.2 Hidden Markov Models

Hidden Markov Models attempt to predict the subsequent state of a system from the current state. Although they cannot be applied to some processes; for example, where the future state is independent of the current state (such as predicting the results in a series of coin tosses), they have been suggested as having potential in CAN anomaly detection [77].

Narayanan *et al.* [59] used gradient data derived from CAN packets, rather than absolute values, to implement a Markov-based anomaly detector. From the CAN readings, they built a time-series vector containing speed and RPM data, which they were able to identify in the CAN logs they obtained from three different model cars. They used a Hidden Markov Model, generating transition probabilities (i.e. probability of changing to the next state) and emission probabilities (i.e. the probability of the observed output for the given state). During anomaly detection, the posterior probability of the next read observation was determined using a sliding window of immediately prior observations. Their test data comprised CAN logs in which they changed the data to show anomalies representative of specific behaviours,

e.g. a sudden increase in speed. The authors report that the system successfully detected the anomalies they generated covering speed and RPM data, both individually, and in speed and RPM data combined, though they acknowledged that they need to test with more anomalous states of varying degrees.

3.4.3 Support Vector Machines

Support Vector Machines (SVMs) seek to separate classes using a hyperplane that follows the centre of the largest margin between the classes. The instances on the edge of each class that determine the margin become the support vectors.

SVMs can learn from small samples; can cope with high-dimensional data, and have the ability to self-learn [71]. A variant of the SVM particularly relevant to anomaly detection is the One-Class SVM (OCSVM), which attempts to create a suitable boundary when trained using only normal instances.

Observing that many CAN IDs seem to be broadcast at fixed frequencies, Taylor *et al.* [76] restricted their study to examples of these, ignoring IDs with less regular periodicity. They measured broadcast frequencies and average data-content changes, which they compared with historical values to determine anomalies. Their study concentrated on attacks that might alter the ID frequency, either by injecting extra packets or erasing packets, and compared a statistical approach against an OCSVM.

For their statistical approach, against each ID, the authors measured:

- The ID.
- Number of packets in the flow.
- Average Hamming distance between successive packet data fields.
- The variance of the Hamming distance between successive packet data fields.
- The average time between successive packets.
- The variance of the time difference between successive packets.
- T-test values comparing the time difference mean to the historic values.
- T-test values comparing the Hamming difference mean to the historic values.
- Combined time difference T-test value and Hamming distance T-test values.

They chose a window length of one second, and eliminated IDs with a period of less than 50ms, since these would not occur frequently enough in each flow to calculate meaningful statistics. Windows were advanced in half-second increments, with sequences of window scores combined to produce a single anomaly score. For the OCSVM, the same measurements were included, with the exception of the T-tests and the CAN IDs.

Data was captured from five-minute drives of an SUV, and attack traffic was simulated by deleting packets, or inserting packets gathered from other captures. During insertion, a set of rules for modifying time-stamps was followed to conform to the CAN arbitration rules. Simulated insertion attacks lasted 100ms to 1s, with rates 1x, 5x and 10x the average rate.

The authors found that the Hamming distance added no discriminatory power, and dropped it from their subsequent comparisons. The statistical method was able to usefully detect the longer of the attacks (1s), though not the shorter ones. In comparison, the OCSVM was able to perfectly detect attacks lasting 0.5s or higher, and showed superior performance on the shorter attacks. The authors conclude that in spite of the promising results, their systems needs testing with more data. They also acknowledge that their approach ignores IDs with non-periodic broadcasts, and would not detect attacks purely resulting in changes in the data fields.

3.4.4 Neural Networks

Inspired by biological neural functioning, a neural network comprises many simple processing nodes working in parallel to produce a decision based on their accumulated outputs. The functioning of the neural network is determined by the network structure, together with the processing carried out in each node and the adjustment of connection strengths (or weights) applied to the connections between the nodes.

Kang and Kang [38] tested a neural network on data generated using the Open Car Test-bed and Networks Experiments (OCTANE) simulator, with three CAN IDs used in the simulation. Attack data was generated by manipulating the data in the packets to deceive the system (though a detailed description of this is not provided by the authors) and, in addition, Gaussian noise was added to the value information to add randomness.

They used an unsupervised deep belief network structure for pre-training to derive the initialisation parameters, followed by a supervised deep neural network to tune the parameters to determine the classifications. Thus, their system required known, labelled, attack packets during training. Taking only packets with a data field of 64 bits, for the normal and attack classes they used the neural network to determine the probability of bit values in any position of the data field. Two training methods were compared: one in which all the bits in the data field were assessed, and another in which training was done using only the bits corresponding

to value data (as opposed to mode or status data, e.g. “on” or “off”). The vectors fed into the neural network comprised the probability that the bit-symbol at a given position was 1, and the “attack” or “normal” label assigned to the training sample. Weights in the neural network were tuned using back-propagation to minimise the mean squared error between the predicted value and the output.

Comparing their system against a conventional feed forward neural network, they concluded that theirs had a more accurate and consistent detection performance, and that the speed of testing suggested real-time decision making was viable. The training involving just the value data (as opposed to value and mode data) gave the better performance.

Taylor *et al.* [77] considered the example of an attack that transmits legitimate packets that are anomalous only in terms of the context of recent packets on the bus. Such an attack might occur, for example, where malicious Keep Lane Assist messages falsify the steering requirement. For each ID, they trained a Long Short-Term Memory (LSTM) recurrent neural network (RNN) to predict the next packet data bit values, the output being predicted values of between 0 and 1 for each of the 64 bits in the word. They found that the LSTM worked well for detecting unusual bit patterns (such as when bit values are randomly flipped) for some IDs, though there were others for which the anomaly detection was poor, though the reasons were not readily apparent to the authors. Another weakness the authors point out is that the IDs were assessed independently, so any anomalies in the interactions between them would not necessarily be picked up, although this is common across many methods tested.

Wasicek and Weimerskirck [89] considered options for detecting chip tuning. This involves an attacker deliberately changing the software or parameters of an ECU, or adding new hardware that acts in an abnormal manner. The authors selected three features, speed, RPM and torque, which they felt would characterise the vehicle engine’s power behaviour in different traffic situations. Data was collected from simulated CAN readings from the TORCS racing simulator. Vectors comprised mean value, standard deviation, variance, skewness and kurtosis for the features for a time period and for a shifted time period. These were recorded for each of the three features, giving 30 input values. Including a time-shift period, they propose, would reduce the impact of any noise in the data. The training vectors were fed into a bottleneck artificial neural network, trained using Levenberg-Marquart back-propagation. For the test data, the outputs from the neural network were combined using a Root-Mean-Square function to generate an anomaly score. Although they found some success in that the true positive rate was higher than the false positive rate, the authors concluded that there was room for improvement, a conclusion borne out in the close proximity of their presented Receiver Operating Characteristic curves to the random guess curve.

3.5 Limitations and Assumptions

The methods described above have reported some successes, but hold assumptions and limitations which could reduce their applicability.

The system investigated by Lee *et al.* [44] required transactions involving remote requests, which would, of itself, add to CAN traffic unless implemented only where such transactions are already used. Moreover, it is restricted to detecting only masquerading ECUs. Conversely, Marchetti and Stabili [47] acknowledge that their CAN ID sequence matrix would not detect replay or masquerade attacks. The models used by Narayanan *et al.* [59] pertained solely to speed and RPM information, and assumed the known derivation of the related CAN data.

For many of the methods, the need for the prior determination of thresholds or signatures, and the assumption that these are constant, could be problematical. For example, the methods proposed by Hoppe *et al.* [31], Marchetti and Stabili [47], Ling and Feng [45], Studnia *et al.* [75] and Gmiden *et al.* [28] required prior determination of specific packet frequencies or sequences, and assume these are consistent and not prone to seasonal cycles or variations. Although the constancy of CAN ID broadcasts has been observed (e.g. [54, 85]), such studies have relied on empirical testing to justify the consistency, rather than offering any protocol justification. It is difficult to conclude therefore that broadcast rates will always be immutable and not vary irrespective of changes due to conditions, ECU configuration, or updating of ECU software. Procedures for determining the normal broadcast rates and sequences in a production deployment would need addressing, and the many permutations and configurations in vehicles would also need accommodating. The patterns determined for one vehicle might not transfer to another vehicle, even of the same model. Foster and Koscher [23] warn that: "sometimes even different generations of the same vehicle will use different frames. For example, the CAN frame to unlock the trunk on one vehicle may activate the windshield wipers of another vehicle." Whilst the car manufacturers might be able to tailor an IDS to the CAN dictionary of a specific car make, model and version, the option becomes more difficult for the independent developer, or when seeking a more generic solution that might be deployed across many car models.

3.6 Data Generation Options

The experiments described later in this thesis, as with the experiments described above, require representative CAN data, including data that is representative of attacks. This section considers the options for generating the attack data. The capture of the data from the test

cars used in this thesis, and the precise details of generating the attack representations, are subsequently presented in chapters 5 and 7.

3.6.1 Simulator vs. Data Manipulation

Whilst some researchers have been able to stage attacks on real cars (e.g. [40, 52]), for others this has not been an option due to the cost of acquiring a vehicle which might be broken in an attack, and the lack of an expanse of unused road on which an an attack can be staged. Ethical approval and insurance implications also need resolving. Researchers have therefore adopted one of two approaches: either use a simulator (e.g. [32, 89]), or manipulate captured data to mimic an attack (e.g. [38, 76, 77]).

Some commercial vendors offer potential candidates for building an automotive CAN simulator, for example, the prebuilt CAN simulator in Vector's CANoe [88]. Such simulators potentially offer replication of CAN processes that are hard to mimic without attacking a real car. Such an example being the replication of latent delays and rescheduling in packet broadcasting that might result from the CAN arbitration process (as discussed in Section 2.1) when an attack inserts packets. In addition, the derivation of the data payload, as well as the triggering of packet broadcasts, might be documented in the simulator specification, and might be programmable (such as in CANoe using the Communication Access Programming Language [CAPL] language), giving scope for the control of testing and comprehensive coverage. The setting up and programming of such simulators, though, can be complex and challenging; and licensing would need to be resolved. Also, there will always be doubts about validity since the simulator can only represent a simplifications of the car's CAN network and the driving situation. The CANoe simulator [88], for example, represents far fewer ECUs than would be found in a production car (though it can be adapted and expanded by savvy CAN programmers). Additionally, simulators are unlikely to incorporate attack options. Kang and Kang [38] used the Open Car Test-bed And Network Experiments (OCTANE) simulator to simulate a vehicle CAN bus. But they had to generate attack samples from these artificially by manually adding frames to the generated logs.

The second approach, adopted for this thesis, is to simulate the effects of the attack by manipulating CAN data recorded under normal operation. This approach might miss the subtle packet broadcast changes that an actual vehicle, or even a simulator, might automatically capture. Also, detailed changes in car behaviour that would result from the attack, cannot be known. Even so, this approach ensures the underlying training and test data represents actual driving data, captured from the full range of ECUs, driving situations, and interactions on the automotive CAN. It is an approach that has been adopted by other studies. As mentioned above, Kang and Kang [38] simulated attacks by manipulating the logs

generated from a CAN simulator, though they give few details of the precise manipulations in their paper. Studnia *et al.* [75] added new frames or modified existing ones to simulate their attacks, but also only provide limited detail. Taylor *et al.* [76] modified captured CAN logs to mimic attacks that injected packets (such as data spoofing), and attacks that caused packets to miss broadcast. For the latter, erased packets were removed from the capture with no other modifications. For the former, they inserted packets at 1x, 5x or 10x their average rate for periods ranging from 100ms to 1s. Data fields were copied from an earlier instance of that packet ID. A limitation of the their data is that it was generated from a car driven only at a steady speed for five minutes, with no user controls (such as lights or windows) being operated. In addition, for detecting payload anomalies using context provided across ECU data fields (as is proposed in this thesis), reusing previously captured data fields leaves uncertainty in determining whether the injected data values are different to those expected for the current system state.

Unlabelled CAN attack data has been published by Lee *et al.* [44], and data generated using CAN simulation software has been published by Kang and Kang [38] though the precise details of the attacks simulated using this software are not stated. The decision was therefore taken to capture real CAN data. Capturing dedicated data for this thesis ensured the following:

1. Data capture could be combined with a functional analysis of the study cars, for example, by systematically manipulating the controls of the car during data capture, as well as providing large data sets across journeys and driving situations to enable profiling of the CAN packet streams.

2. Sufficient data sets could be captured across journeys, enabling between journey comparisons during testing.

3. Attack records generated using the captured data could be accurately labelled.

4. Whilst the data used for the testing was captured using a Kvaser CAN reader (discussed in Chapters 5 and 7), this was compared with data captured from other readers (such as Vector's CAN loggers [88]) providing data for triangulation to validate that the capture process was reliable.

3.7 Conclusion

A range of methods have been proposed for CAN packet anomaly detection. Requiring the capture and dissemination of known attack signatures for profile matching (e.g. [75])

presents a problem given the lifecycle of cars, their variety, and the unpredictability of future attack scenarios. Similar problems are presented by machine learning approaches that require existing examples of attack data for the classifier training (e.g. [49], [38]). Statistical approaches have been proposed which require the profiling of individual CAN IDs (e.g. [45], [31]), which again presents challenges for a system that might be deployed across car makes and models.

Approaches that might not need existing attack-data for training have been proposed by Narayanan *et al.* [59] and by Taylor *et al.* [76] [77]. Narayanan *et al.* used a Hidden Markov Model to detect anomalous changes in speed and RPM, but training their system required knowledge of how these were manifest in the CAN packets. Taylor *et al.*'s One Class Support Vector Machine approach used a combination of packet timing and payload measurements to try to detect attacks in which both the timing and the payload had been altered, although the method did not extend to detection where only one of these had been manipulated. Their proposal for payload anomaly detection operated at the bit level and showed success in detecting attacks in which the packet payload bits were randomly changed. But it is unclear how their method would perform with attacks that produced a more systematic and strategic update of the packet payload. Moreover, since their method considered each packet ID in isolation (basing normality on observing the previous broadcasts for that packet), their system ignores contextual information that might detect stealthy anomalies, such as a fabricated sensor reading that drifts steadily further from the true value over a sustained attack.

As discussed in Chapter 2, the anomaly detection in this thesis will be considered as independent for timing anomalies and payload anomalies. This chapter has discussed the benefits of the proposed methods being ones that enable a one-class detection, thus they do not assume the availability of attack signatures or attack data examples. The chapter has also discussed the few one-class approaches that have been proposed, and highlighted their limitations.

Chapter 4

Anomaly Detection Methods Used In This Thesis

This chapter presents the anomaly detection methods that will be evaluated in this thesis. Using those methods requires suitable processing and parsing of the CAN broadcast flow, which is also discussed in this chapter. The configuration of the methods and the optimisation of their parameters will be discussed in Chapter 5, for the timing anomaly detection, and Chapter 7, for the payload anomaly detection.

As already discussed, the nature of CAN attacks reveals the follow traits:

- The time and frequency of packets may be affected, though not always (e.g. masquerade attack). Many attacks involve the injection of packets, increasing the ID's frequency, although broadcasts could be delayed or disrupted through arbitration and error processes triggered during the attack, or by malicious activity such as illicit ECU reflashing.

- The data payload might be fabricated; randomly in a fuzzing attack, or with specific, plausible values in spoofing attacks.

- The challenges in maintaining updated attack dictionaries in the car, together with the diversity of car models and CAN dictionary configurations, reduces the viability of signature approaches and suggests unsupervised anomaly detection.

- The unpredictability of future attacks, and the difficulties in gathering actual attack data, suggests that the anomaly detection should employ one-class approaches.

The IDS methods considered so far suffer limitations such as requiring prior analysis of the CAN traffic to decide the values for the discrimination factors which are then fixed in the

classifier. One objective of this thesis was to find methods that could be used unsupervised, so that they might be easily adapted for cars without detailed knowledge of the CAN dictionary. Therefore, considered here are approaches that might be used for the one-class, unsupervised detection of anomalies in packet broadcast rates and in the data values transmitted in the payload. These will be considered separately, in the expectation that a CAN IDS would employ both.

4.1 CAN Packet Timing Anomaly Detection

This section discusses the methods used to detect anomalies in packet timings. The time-defined window approach used to parse the CAN bus flow for the detection methods is discussed in Section 4.3.

Two unsupervised approaches were considered for the packet timing anomaly detection: Z-score and Autoregressive Integrated Moving Average (ARIMA). These two methods were used to predict the time intervals between the consecutive broadcast of matching packets, which were then compared against the actual observed time gaps. For comparison, a supervised method of comparing each broadcast interval against the mean for that packet ID, was also tested.

4.1.1 ARIMA

ARIMA time-series forecasting models seek to make the data stationary (i.e. the mean and variance do not vary through time) by removing trends and seasonality from the data. An example of a data set showing both seasonality and trend is shown in Figure 4.1. ARIMA attempts to make the data stationary through the use of differencing (subtracting the observation in the current period from the previous one), which requires the determination of parameters covering the number of past forecast errors used for prediction (p), the number of times the differencing should be carried out (d), the number of moving average time periods to be used to predict the current values of the series (q). Tuning the ARIMA algorithm and determining the optimal p,d,q settings entails a detailed analysis of the data set. However, Auto ARIMA determines the best combination of parameters [34]. It is implemented in the Forecast package in R [33], which was used in this project, although it has recently been implemented in Python [72].

For the detection, the CAN packet flow was parsed into discrete time windows (discussed later), and within each window, auto ARIMA was used to determine the time gap residuals (i.e. the actual time gap value minus the predicted time gap value). These were then standardised

using R scale function, to produce a Mean Squared Error for the window, which could be compared against an error deviation threshold.

Fig. 4.1 Example data series showing diurnal seasonality and gentle upward trend. Source: Chio and Freeman [15].

4.1.2 Z-Score

Similar to the ARIMA-derived error scores, the Z-score (time gap minus the window gap mean, divided by the standard deviation) was also used to produce a Mean Squared Error for the window, which could be compared against an error-deviation threshold.

4.1.3 Mean Comparison

The mean comparison merely compared the absolute difference between the mean time gap and the observed time gap, against a predetermined fixed value. The mean time gap was calculated within the window for the monitored packet.

4.1.4 Timing Method Supervision

Z-score and ARIMA have been used by Kalutarage *et al.* [37] and Shaikh and Kalutarage [70] with time-based data to assess the likelihood of attack on complex computer networks. Whist these methods require an error score to be optimised for detection, they otherwise require little supervision in terms of being able to adapt to variance and difference between data sets. The Z-score and ARIMA methods could be regarded as unsupervised when used in time-based windows (such as those used in the timing anomaly detection evaluated this thesis, which are described later), in that any variance is scaled to be of a similar magnitude across the windows and CAN IDs. This could facilitate specifying a common threshold, without needing to be tailored according to CAN behaviour differences across cars or situations.

The third method used for the timing anomaly detection (comparison of the broadcast interval against the window mean), was included as a baseline for comparison with the other two methods. This method should perform best since it uses a threshold tailored according to the observed variance between the broadcasts for the tested set of journeys. A preliminary analyses of CAN data from the cars used in this experiment suggested testing thresholds of around 0.003 (this is discussed in Chapter 6).

4.2 CAN Data Payload Anomaly Detection

Whilst packet timings are fairly easy to determine and monitor, detecting attacks from anomalies in the data payload presents a more challenging prospect. Values that are extreme or random are unlikely to be of value to the attacker, other than for fuzz testing and probing, since they would be meaningless to the ECU. Of more value to the attacker are values that are plausible. Although the CAN dictionary is unpublished, fabrication attacks (e.g. [14, 32, 40, 52]) have demonstrated that reverse engineering to determine the data meaning and derivation is possible, and that disruptive, but plausible, values can be broadcast to tamper with the car. Moreover, the attacks might entail fabricating data for seemingly unrelated functions. Consider, for example, the steering attack by Miller and Valasek [52], which required the broadcasting of fabricated data for speed in order to tamper with the steering. Thus an attack might span packets pertaining to a variety of functional areas.

The anomaly detection methods considered in this project provide the potential to determine an anomaly in a multivariate array based on values across all the elements in the array. Thus, fabrication of one variable might be detected if the rest remain unfabricated. For such methods to work, the variables in the vector should be connected and correlated in some way. Of course, such methods would be useless if the attacks fabricated the values for all of the variables in the array; though this would require the attacker to have full knowledge of the variables that are being monitored, and the ability to fabricate them together in a coordinated and correlated fashion. The identification of the sensor data vectors, and the algorithms that were developed to automate the process, are discussed in Chapter 7.

Three methods that enable unsupervised novelty detection were evaluated. These are: One-Class Support Vector Machine (OCSVM), Local Outlier Factor (LOF), and One Class Compound Classifier (CC). The testing was conducted using Python 3.6, with the CC built from scratch employing scipy.spatial packaged Euclidean distance calculations [68], and the LOF and OCSVM constructed using the Scikit-learn [62, 67] v0.20.0 package.

4.2.1 One Class SVM

The OCSVM is deemed an especially suitable choice where data has a multimodal distribution and is high-dimensional [71, 60], or where training data sets might be small [15]. Multi-class support vector machines construct a hyperplane that separates the classes of data by the largest margin. In contrast, the OCSVM attempts to find the largest margin between the projected training data and its origin, thus demarking data-dense regions in input space. Target instances are thus classified according to their position relative to the margin.

4.2.2 Local Outlier Factor

The LOF relies on distance measurements and density calculations. The LOF has been shown to be resilient to data sets with varying cluster densities [15]. Its calculation entails determining the local density of data points as defined by the distance to the other data points in the immediate surrounding area [8]. The LOF score is derived by comparing the density score of a point with the density scores of its K nearest neighbours (KNN). Each instance acquires a score which is higher in instances that lie away from their own K nearest neighbours. Thus, the score can be used as a threshold to classify outliers.

4.2.3 Compound Classifier

The Compound Classifier (CC) was devised for rapid classification using a reduced instruction set processor [3]. Primarily used in rapid image processing applications [4, 50], it has also been used in medical diagnosis [2]. Its authors proposed that the CC: is usefully able to generalise, learn and make rapid decisions; lends itself to visualisation which is important for understanding and checking; and, can be easily adapted for multi-class and one-class problems [3, 4]. It uses nearest neighbour calculations and probability density functions to replace the training data points with a smaller number of hyperspheres, thus potentially leading to faster decision making compared with many nearest neighbour classifiers [3]. Hyperspheres are constructed as the training data is observed, and are changed in size and location according to whether each training instance is inside them or not. During training iterations, the hyperspheres thus evolve to encompass the shape of the training data set. The CC training and decision processes are presented in Appendix A.

For this research, a CC was built using Python following the specifications and parameters outlined in [3, 4].

4.2.4 Payload Detection System Model

The payload anomaly detection proposal is that a suitable classifier method would be integrated into a CAN intrusion detection system that could be attached to the CAN network, either directly or via the OBD port in the driver's compartment. The CAN packets would be grouped into functional clusters, and the trained classifier would assess the CAN packet fields in real-time by comparing the packet's values against those most recently broadcast by other fields in the same cluster. An anomaly would be flagged when a broadcast is observed that contains sensor data values that are inconsistent with the most recent values from other data fields in the cluster. Thus, the system would process snapshots of the CAN traffic, with each snapshot comprising data payload values for the currently broadcast packet, together with the most recently broadcast payload values for the other packets assigned to the same cluster (Algorithm 1). This method could be easily implemented on a live system since it requires just the trained classifier, plus a one dimensional array holding the most recent values from the desired CAN data fields, with only the changed data fields being updated as each new record is detected.

Algorithm 1 Classification Process

```
ℂ ← cluster comprising associated sensor data fields
for each broadcast of packet ℙ do
  if ℙ fields correspond to ℂ fields then
    ℂ ← updated data for fields corresponding to ℙ
    Apply anomaly test to ℂ
  end if
end for
```

Developing the classifier can be summarised into the following broad stages, which are considered later in this thesis:

1. Determine which CAN data fields represent sensor data (including fields that are composite).
2. Assign data fields into functional clusters (i.e. each $\mathbb{C}$ in Algorithm 1).
3. Generate a classifier for each cluster.
4. Use the classifier to continuously monitor CAN traffic.

4.2.5 Normalisation

All the methods used for payload anomaly detection require the data to be first normalised before it is processed by the classifier. This ensures all variables are on the same scale and have equity in their influence.

Normalisation was done by standardisation using the Scikit-learn unit variance StandardScaler function [67]. The scalers were first fitted to the training data, with the fit then used to transform the training and test data sets. The scaling is described in Section 7.2.1.

4.3 Parsing the CAN Packet Flow

For both detection systems (timing and data payload) the mode of parsing the CAN flow needs to be considered. The broadcast of CAN packets onto the bus is continuous whilst the car is in operation. The detection needs to be instant, and cannot rely on the storage and processing of large quantities of data. This section discusses the parsing of the CAN flow that was adopted for the two detection strands.

4.3.1 Timing Anomaly Detection - Time Defined Windows

The timing anomaly detection adopted a time-defined window approach. Such approaches process the data as discrete windows. Keeping the calculated metrics and consequent decisions local to the window reduces the impact of any longer-term system changes reflected in the data. There are a few window approaches that could be adopted for packet-timing anomaly detection. One option is to process the window to derive metrics and then apply these to classify the next broadcast. The window slides through the data one broadcast at a time, recalculating the metrics at each broadcast. This approach requires the storage of each time gap for each monitored CAN ID, though these only need storing whilst that window is being monitored.

In this study the timing anomaly data was processed in discrete, non-overlapping, contiguous windows, which has the advantage of reducing the recalculation of metrics – beneficial in the limited CPU capacity in vehicles. The metrics were calculated for a window and then applied to all the broadcasts within that window for classification. After this, the assessment moves on to the next non-overlapping window. This approach reduces the frequency with which the window metrics need recalculating compared to a sliding window. Assessing the metrics against broadcasts within the window ensures the assessed broadcasts are within the scope of any naturally occurring adjustments captured by the detection methods. It also ensures that records at the immediate start of the journey are evaluated for anomalies.

4.3.2 Data Payload Anomaly Detection - Rolling Updates

As highlighted in Algorithm 1, for the payload classifier training, and for the subsequent payload anomaly detection, we need to amalgamate data from the most recently broadcast packets in the cluster. This refines the data to just the elements required for the detection, which can be processed in a compact structure. In a production version, the amalgamated data would not need to be retained after each classification has taken place, so a single, continuously updated array could be used. However, to facilitate test replication, and allow attack data to be simulated, each change to the array was saved as a separate snapshot record. Thus each snapshot record combined the most recent field values across all the chosen fields with the values in the newly broadcast packet.

TIME	CAN_ID	D8	D5D6
27.08628	324	165	0
27.08653	383	34	561
27.18604	324	180	0
27.18628	383	53	903

(a) Sample from the CAN packet flow. For illustration only two IDs and and two data fields are shown.

TIME	CAN_ID	17C_D3D4	158_D1D2	191_D5	18E_D1D2	1DC_D2D3	324_D8	383_D5D6
27.18481	17C	32	0	55	0	32	165	561
27.18496	18E	32	0	55	0	32	165	561
27.18520	158	32	0	55	0	32	165	561
27.18544	191	32	0	57	0	32	165	561
27.18561	1DC	32	0	57	0	32	165	561
27.18604	324	32	0	57	0	32	180	561
27.18628	383	32	0	57	0	32	180	903

(b) Section of the array of amalgamated CAN data. Each row shows the change in the data elements over time as each CAN ID is broadcast. The Time and CAN_ID columns are not included in the actual array.

Fig. 4.2 The rolling data payload array used for payload anomaly detection.

Figure 4.2 illustrates the updating of the snapshot array. 4.2a shows CAN broadcasts between 27.08628 and 27.18628 seconds for two of the CAN IDs. 4.2b shows the array state after the broadcast of the CAN packets. The array is updated as each new packet in its CAN ID scope (indicated by the prefix in each column heading) is broadcast. For example, the broadcast of CAN ID 324 at 27.18604 seconds, results in an array update at that time for element 324_D8. Likewise, the broadcast of CAN ID 383 at 27.18628 seconds, results in an update of array element 383_D5D6. Also observable in the Figure is that the first few rows

of 4.2b show that the array values did not change when packets were broadcast on the CAN. This would have resulted from packet broadcasts that contained the same field value as the previous broadcast for that CAN ID, thus causing no observable update to the array.

4.4 Conclusion

The methods proposed in this thesis adopt a one-class approach; offering the potential to classify anomalies without requiring an anomaly class to be manifest in the training set data, which was necessary for many of the methods proposed by other researchers discussed in Chapter 3.

For the timing anomaly detection, a time-boxed window provides the mechanism for determining normal operation in terms of temporally localised events. This reduces reliance on assumptions that packet broadcast patterns will be immutable across journeys, car configurations or driving situations; which some proposed detection methods are predicated on (e.g. [45, 73, 28]). A slightly different, sliding, window approach was adopted by Taylor *et al.* [76], who used a half-second window, although some of the CAN IDs were eliminated from their detection for being broadcast too infrequently to generate analytical statistics within each window. As well as using different analytical techniques and data measurements to Taylor's, this thesis aims to test a range of window sizes. The proposed use here of ARIMA and Z-Score for the timing anomaly detection, provide potential mechanisms for defining normal flow without pre-training.

For the payload anomaly detection, the three selected one-class methods were chosen for their unsupervised potential to determine outliers in any one of the assessed dimensions, thus presenting the prospect of detecting outliers according to the context provided by a consensus of ECUs. This might facilitate the detection of attacks, such as those demonstrated by Miller and Valasek [52], that have involved the fabrication of values that are plausible even though they are of malicious intent.

To be able to instantaneously compare payload data across CAN packets, a rolling-update snapshot record is proposed, wherein the most recent data values from the analysed data fields are inserted into an updating array. Proposed mechanisms for updating the array, and automatically determining the data to be used in it, are considered in Chapter 7.

Chapter 5

Detecting Timing Anomalies - Experiment Design

This chapter describes the set-up of the experiment to compare ARIMA, Z-score and supervised mean-comparison for detecting anomalies in packet timing. Results are presented and discussed in Chapter 6.

Data for the experiment was captured from two popular, unmodified UK family cars from different manufacturers, thus providing an indication of the validity of the methods across car models. The two cars are henceforth referred to as Car A and Car B. Prior to running the experiments and deciding the optimal values for their parameters, the profile of the CAN bus packet-flow rates for both cars was analysed. Those results are also presented in this chapter.

5.1 Data Capture and Profile Analysis

For each car (Car A and Car B), four journeys, totalling over two hours of driving per car, were logged over a two-week period, and covered a range of typical driving conditions to capture a representative spectrum of situations and CAN behaviours. CAN messages were logged from the OBD port using a Kvaser Leaf Light[1] reader, designed for loss-free CAN message capture with high time-accuracy, attached to a laptop computer. Figures 5.1, 5.2, 5.3 and 5.4 show the mean broadcasts per second, and the times between broadcasts, of matching CAN IDs for each journey.

[1] www.kvaser.com/product/kvaser-leaf-light-hs-v2-obdii

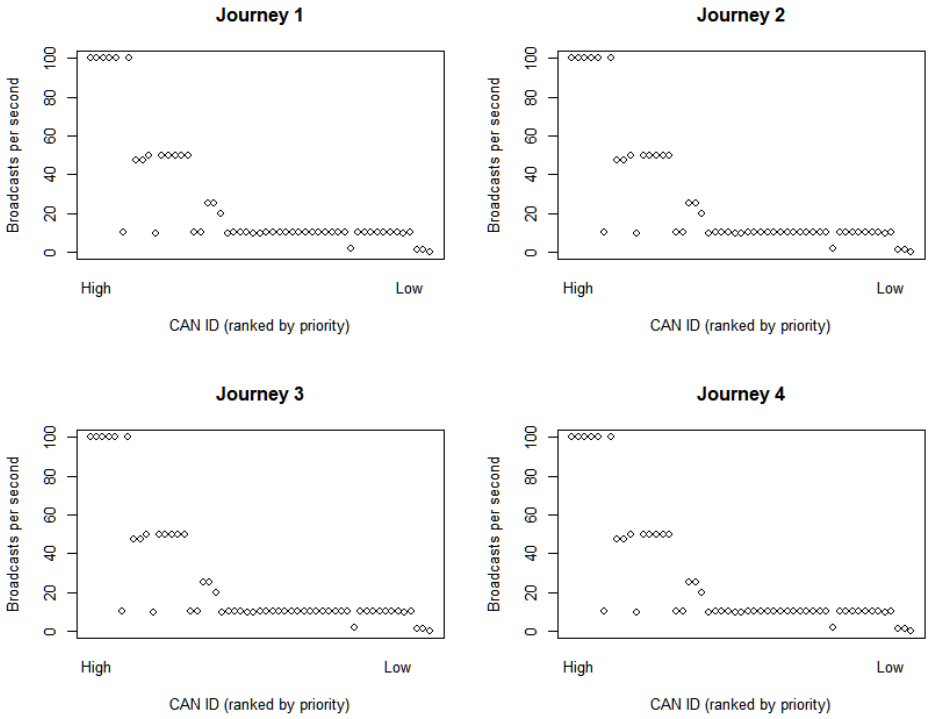


Fig. 5.1 Car A: Average broadcasts per second of CAN IDs ranked by priority.

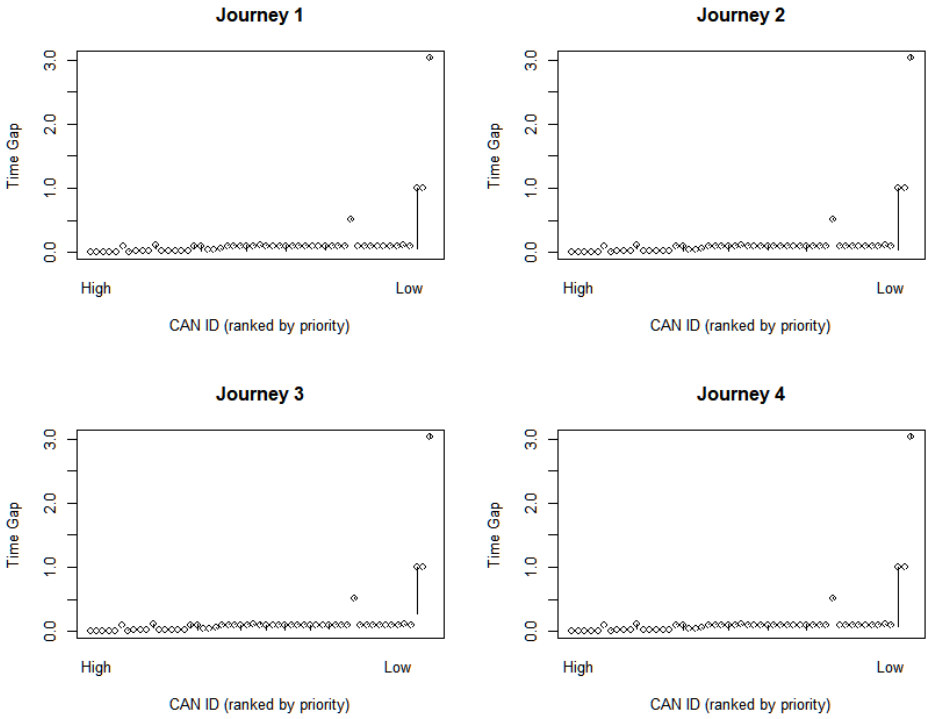


Fig. 5.2 Car A: Average broadcast gaps and range (shown by vertical lines) of CAN IDs ranked by priority.

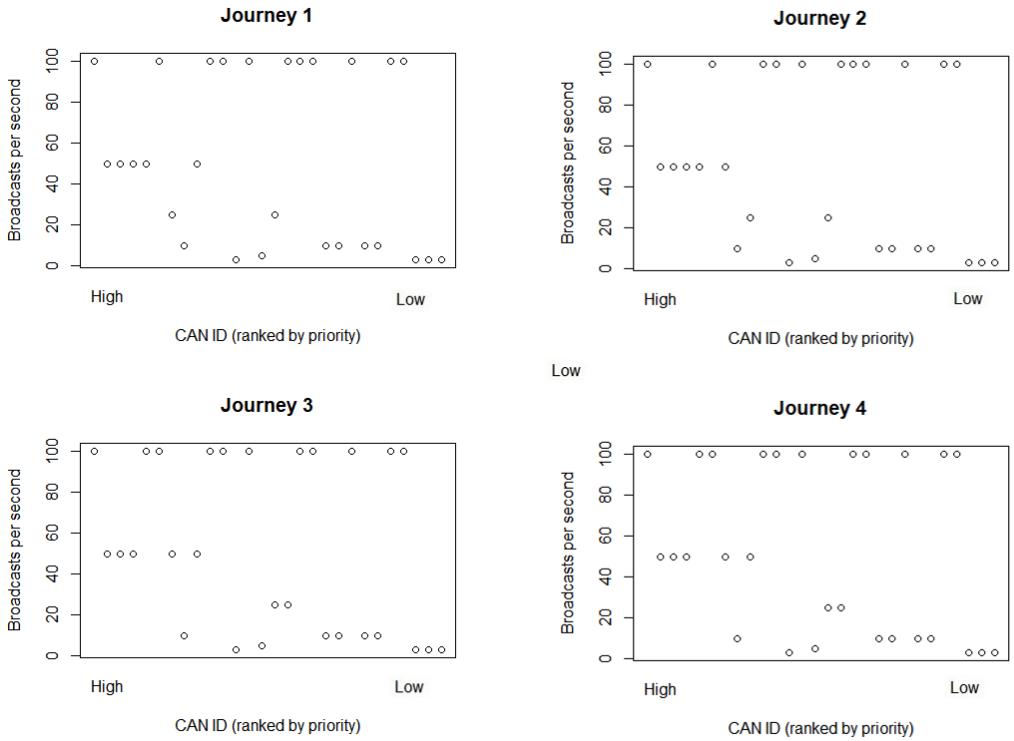


Fig. 5.3 Car B: Average broadcasts per second of CAN IDs ranked by priority.

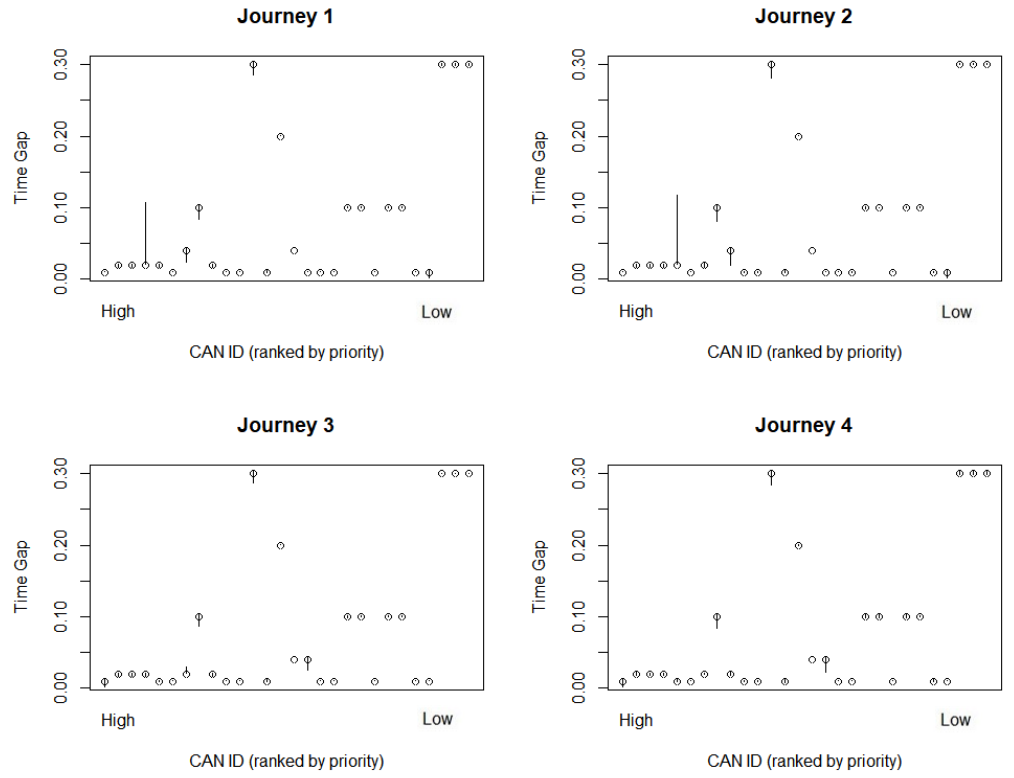


Fig. 5.4 Car B: Average broadcast gaps and range (shown by vertical lines) of CAN IDs ranked by priority.

5.1.1 Broadcast rates

For Car A, fifty-four CAN IDs were detected, of these all but two were broadcast throughout the journeys. Two CAN IDs occurred briefly at the start of one journey and were not observed again. These were excluded from the subsequent testing, leaving fifty-two in the data set. In Car B, twenty-eight CAN packets were identified. All were broadcast across each journey and included in the testing.

The CAN IDs were broadcast at reasonably regular intervals throughout the journey. With a couple of exceptions, especially in Car B, the pattern of broadcast frequencies and timings was similar across journeys.

Car A Broadcast Rates

Most CAN IDs in Car A were consistent in their broadcast intervals. The higher priority CAN IDs tended to be broadcast more frequently, many being broadcast at about 100 per second, while only 4 CAN IDs showed a broadcast rate of less than 10 per second in (Figure 5.1).

Any variation in the broadcast interval was small for the majority of CAN IDs (Figures 5.2). All but one CAN ID showed a maximum repeat-broadcast time-difference range of 0.1 second or less, with 39 (71%) showing a range of less than 0.01 second. Forty CAN IDs had a repeat-broadcast standard deviation of less than 0.0007 seconds. Whist in the remaining CAN IDs, all but four had a standard deviation of less than 0.01 seconds.

Car B Broadcast Rates

A similar pattern was observed in Car B, wherein ten packets (33%) were broadcast with an average frequency of 100 per second across all journeys. The remainder typically had an average broadcast rate of around 50, 25 or 10 per second, with five (18%) having a broadcast rate of less than 10 per second across any journey. As with Car A, all but one CAN ID showed a maximum repeat-broadcast time-difference range of 0.1 second or less, with 22 (78%) showing a range of less than 0.01 second. Twenty four CAN IDs had a repeat-broadcast standard deviation of less than 0.0007 seconds. In spite of the general homogeneity, there was some variation between journeys for a couple of the CAN IDs, as shown in Figures 5.3 and 5.4. The CAN ID with priority ranked 4 showed a relatively large range in repeat broadcast intervals in two of the journeys shown in Figure 5.4, and there was also a shift in broadcast rates between journeys for a couple of other packets. Without knowledge of the CAN dictionary, determining the cause of these shifts would require a detailed analysis involving additional data capture. However, the cause might be car specific, and so might

not of itself offer more insight into the implications for a car-agnostic anomaly detection proposal. However, the fact that such phenomenon are observed is important, and shows that not all CAN IDs might be consistent enough for timing anomaly detection without additional interpretation and situational analysis.

Implications

Overall, the analysis suggests ECUs are broadcasting their information at a fairly consistent, predictable, rate throughout the journey, and for each journey. Such consistency in CAN message broadcasting has been noted by other researchers [52, 54]. If we assume that individual CAN IDs are specific to individual ECUs, then almost all the broadcasting ECUs would be broadcasting at 10 times or more per second, with the majority broadcasting at a consistent rate, more than 100 times per second.

In general, across both cars the CAN IDs with the higher priority (i.e. lowest ID value) were broadcast most frequently and with least timing variation. Whilst the planning of ID broadcast schedules by manufactures might account for this, it also might be influenced by the lower priority IDs having their broadcasts rescheduled by broadcasts of higher priority IDs as part of the CAN arbitration process.

This observed regularity of CAN ID broadcasts suggests the possibility that they might be regular enough to enable the easy detection of broadcast variations that might signify an attack. As discussed in Chapter 2, an attack that seeks to take control of an ECU to force it to broadcast erroneous data (for example, a re-flash attack that overwrites ECU code) is likely to render the ECU inoperable whilst the attack takes place, causing a gap in the broadcasts. Similarly, an attack from an added malicious source that injects false data by broadcasting forged packets under a legitimate CAN ID, is likely to lead to an increase in broadcasts of those packets, changing the time-gap between broadcasts, especially if the legitimate ECU is still broadcasting.

5.2 Timing Anomalies: Attack Simulation

For the anomaly testing, the log from a journey lasting 30 minutes and covering a range of driving situations, including town traffic and faster roads, was selected for Car A, and a similar journey was selected for Car B. Fresh copies of these log files were manipulated so that each copy include fabricated attack representations based on one of the following:

- **Dropped packets** of a specific CAN ID could result from either a reflash attack, an attack in which an ECU is disabled and replaced by a masquerading ECU, or an

injection attack. In a reflash attack the firmware of an attacked ECU is overwritten by an attacking vector, thus being replaced by adapted firmware [40]. With regard to changes in broadcast timings for both the reflash attack and the masquerade attack, the ECU would miss broadcasts that would otherwise have occurred during the reflash or change between ECUs respectively. Taylor *et al.* [76] tested record outages spanning between 0.3 and 1 second for the latter attack. An increase in dropped broadcasts could also be triggered by an injection attack, when a higher priority ID carrying malicious data is broadcast at a high rate so as to ensure the malicious data out-broadcasts the legitimate data [52, 77]. The purpose of the dropped broadcast detection was thus to see whether the methods could detect when packet broadcasts had been skipped.

- **Injected packets** might occur when a compromised or illicitly added ECU broadcasts packets that mimic those from a legitimate ECU, thus broadcasting false data or false control signals. A variation on this is the denial of service attack, where packets are submitted with rapid frequency in order to overwhelm the bus. The purpose of the injection attack detection was thus to see whether the methods could detect when additional packet broadcasts had been inserted.

```
0,344,8,38 235 0 0 38 252 69 43,533.96919,R
0,344,8,38 229 0 0 38 239 69 62,533.97918,R
0,344,8,38 225 0 0 38 250 69 9,533.98918,R
0,344,8,38 227 0 0 39 2 69 28,533.99918,R
0,344,8,38 221 0 0 38 244 72 47,535.00904,R
0,344,8,38 215 0 0 38 223 72 59,535.01904,R
0,344,8,38 211 0 0 38 241 72 14,535.02903,R
0,344,8,38 215 0 0 38 244 72 22,535.03903,R
0,344,8,38 211 0 0 38 217 72 38,535.04903,R
```

Fig. 5.5 Example of the dropped packet attack data. Records for CAN ID 344 have been dropped for a one second slot after the record broadcast at 533.99918. Note that only records for CAN ID 344 are shown in this illustration.

Attacks were simulated for every CAN ID identified in each selected journey's log. For each attack, 20 attack points were created in the captured log file at time-points chosen randomly throughout the duration of the log. For the dropped broadcast simulation, packets pertaining to the ID being tested were removed for a one second slot at the chosen attack point (Figure 5.5). For the injection detection, a broadcast of the relevant CAN ID was inserted into the log file at the next available gap after the randomly selected time (Figure 5.6). It was observed that across all car journeys, the shortest gap between any two packet broadcasts was 0.00008 seconds (shown in 1.3% of broadcasts). Therefore, so as not to

```
0,356,8,2 0 0 54 124 0 0 60,253.89509,R
0,884,7,0 0 0 0 0 0 10 ,253.89539,R
0,398,3,0 1 0     ,253.89598,R
0,344,8,0 0 0 0 0 0 24 62,253.89952,R
0,401,8,8 0 128 154 154 0 127 47,253.89976,R
0,344,8,47 40 0 6 156 8 25 0,253.90019,R
0,310,8,0 0 0 14 0 0 0 31,253.90027,R
0,314,8,0 0 0 0 0 0 0 25,253.90052,R
0,319,8,0 139 0 192 0 0 0 21,253.90076,R
0,380,8,0 0 3 218 1 0 0 24,253.90101,R
```

Fig. 5.6 Example of the inserted packet attack data. A fabricated record for CAN ID 344 has been inserted at time 253.90019.

violate any physical limit that this sized gap might indicate, the randomly chosen attack points for the injected records were selected to ensure at least a 0.00008 second gap before and after the inserted record. The times for the inserted attack packets were fabricated in the logs accordingly.

5.3 Conclusion

Data captured from multiple journeys from the two study cars showed that the CAN ID broadcasts for each car were regular and consistent in their timings, as suggested, for example, by the majority of CAN IDs having a repeat broadcast interval that was never seen to vary by more than 0.0007 seconds, and broadcast rates of 10 or more per second. This suggests that timing anomalies caused by either injected packets or by dropped or delayed broadcasts, might be detectable. The regularity observed in the two cars studied here supports the timing regularity observed by other researchers (e.g. [54, 85]).

Simulation of attacks by altering existing CAN logs to portray packet injection or dropped packets is adopted here because of the lack of any published, labelled attack data sets, in addition to the barriers of staging attacks on real cars. Simulated attack data in which packets have been individually inserted have also been used by Marchetti and Stabili [47], and Kang and Kang [38]. For this thesis the times of the injected packets are fabricated in a manner that ensures they do not impinge on legitimate packets. The dropped record simulation is generated in a manner similar to Taylor *et al.* [76], with the target CAN ID's records removed from the log without further alteration.

Chapter 6

Detecting Timing Anomalies - Results

This chapter presents the results from testing the timing anomaly detection methods' abilities to detect the random attack points generated for each CAN ID, as described in Chapter 5. The implications of the results are discussed in Chapter 9. For the experiment, the CAN logs were processed in fixed-time windows, with the detection using statistics from the broadcasts within the window, as described in Chapter 4.

6.0.1 Timing Anomalies: Assessing Detection Accuracy

The methods assessed here process the data-flow into fixed-size, non-overlapping windows, and detect change and variation based on the time intervals between broadcasts of matching CAN IDs. This means, though, that a positive detection might indicate an attack-simulation fabricated change in packet broadcast times in either the current time window, or in a previous window. This is shown by Figure 6.1, which shows that for a dropped or injected packet, the changes in time interval might only be picked up in a subsequent window if that is where the next broadcast packet resides. Thus, for the experiment a window flagged as a detection was considered a true positive detection if: a) an altered message was within the current window; or, b) an altered message was in the previous window, which had been flagged as a non-detection.

Likewise, a failure to detect a fabricated change in the current window (i.e. a false negative) could be determined if no detection flag was raised for both the current window and the next window. Thus, results were classified as follows:

- True Positive: Positive flag, and either the attack altered the current window, or the attack altered the previous window, which had a negative flag. (Otherwise the positive flag was classified as False Positive.)

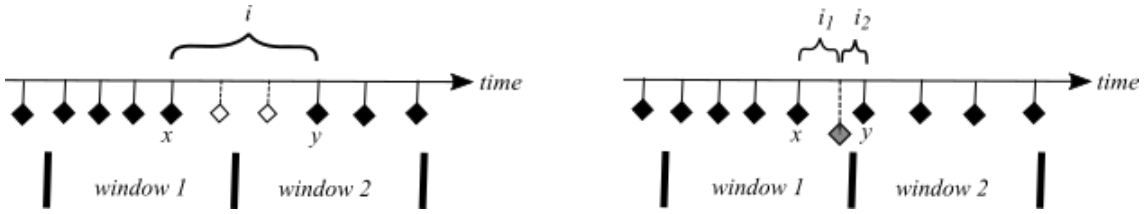


Fig. 6.1 Potential mismatches between packet manipulations and analysis windows. Left: packets are dropped between packet broadcast x and packet broadcast y. Although the first packet was dropped during window 1, the increased inter-packet time (i) will not be detected until packet y is broadcast. Right: a packet is injected between x and y. Depending on the interval sizes (i_1 and i_2) an anomaly might be detected at the injection (window 1) or at packet y (window 2).

- True Negative: Negative flag, and no attack alterations to current window or previous window. (Otherwise the negative flag was classified as False Negative.)

Analyses of the classifier performances drew on calculations described in [74, 22]:

Accuracy is the proportion of responses correctly classified:

$$\frac{TP+TN}{TP+FP+TN+FN}$$

Sensitivity (or Recall) is the ability to detect actual occurrences:

$$\frac{TP}{TP+FN}$$

Specificity is how specific the test is to the condition being assessed:

$$\frac{TN}{FP+TN}$$

Accuracy may mislead since it masks the relative proportions of the positive and negative elements in the testing dataset. A classifier that makes only negative responses could achieve a high accuracy score if the underlying test set contained a high proportion of negative instances. Likewise for a classifier that makes only positive responses on a dataset that contained a high proportion of positive instances. Therefore sensitivity and specificity are also included in the results analysis.

6.0.2 Timing Anomalies: Window Size

To ensure multiple observations for deriving the statistics for the detection-method thresholds, a window size of 2 seconds was initially selected for the detection window. This would ensure that in both cars, there could be expected to be at least 20 packets (or observations) in each window for the majority of CAN IDs, with many having 100 or more expected. It did mean, though, that there would be four CAN IDs in Car A that would have an expected count of 10 or less.

6.1 Timing Anomalies: Results and Discussion

Initial testing compared detection rates for adjustments to the error rates for Z-score and ARIMA, and threshold for the supervised method, in order to find optimal values as measured by accuracy scores. For this the initial testing was restricted to only the five highest priority CAN IDs in each car. These had a high broadcast rate, with very little variation in timing in Car A, though the broadcast intervals were slightly more erratic in Car B (Figures 5.1 to 5.4). Since these are the highest priority, it is likely that they would also carry important information, potentially making them a high target for attack.

6.1.1 Timing Anomaly Results for High Priority CAN IDs

Figures 6.2, 6.3 and 6.4 show the sensitivity, specificity and accuracy scores for detection at various error rates (for ARIMA and Z-score) and thresholds (supervised method), for Car A's five highest priority CAN IDs. The scores for Car B show a similar pattern to those of Car A, and are presented in Appendix B. Tables 6.1, 6.2, 6.3, and 6.4 list Car A and Car B scores and detection rates for those tests at the highest accuracy levels. For both cars the highest accuracy scores were achieved with an error rate of 9 for ARIMA and Z-score, and a threshold of 0.003 for the supervised methods. Perhaps predictably, given the broadcast pattern of Car B, the scores for that car are slightly lower than those of Car A.

In Car A, the sensitivity scores are maximum at these accuracies, reflecting the ability to capture all true positives. However, sensitivity was less high in Car B, for ARIMA and Z-score. The specificities are generally high, indicating an ability to avoid false negatives. The specificity, though, was slightly lower in ARIMA, indicating that its ability to distinguish

Table 6.1 Car A: Scores for the three methods at the highest accuracy points for 5 highest priority CAN IDs.

Method	Attack Type	Accuracy	Sensitivity	Specificity
ARIMA	Injection	0.9866	1.0000	0.9863
ARIMA	Dropped	0.9919	1.0000	0.9918
Z-score	Injection	0.9978	1.0000	0.9977
Z-score	Dropped	0.9982	1.0000	0.9982
Supervised	Injection	1.0000	1.0000	1.0000
Supervised	Dropped	1.0000	1.0000	1.0000

Table 6.2 Car B: Scores for the three methods at the highest accuracy points for 5 highest priority CAN IDs.

Method	Attack Type	Accuracy	Sensitivity	Specificity
ARIMA	Injection	0.9690	0.8000	0.9729
ARIMA	Dropped	0.9626	0.7800	0.9667
Z-score	Injection	0.9906	0.8000	0.9950
Z-score	Dropped	0.9902	0.7800	0.9950
Supervised	Injection	1.0000	1.0000	1.0000
Supervised	Dropped	1.0000	1.0000	1.0000

between attacks and non-attacks was less successful at these points. In Car A at the highest accuracy point, the ARIMA method, like the other methods, detected all 100 injections (sensitivity 1.0000), but incorrectly classified 60 of 4370 non-injection windows (specificity 0.9863). In Car B, 80 injections were detected (sensitivity 0.8000), with 119 out of 4390 non-injection windows being incorrectly classified (specificity 0.9729).

Accuracy scores are heavily influenced by the proportion of negative (non-attack) windows in the test data. Consequently, the detection response might not be optimal at the highest accuracy point. Figures 6.2 to 6.4 (Car A) and B.1 to B.3 (Car B) show the sensitivity, specificity and accuracy scores for injection detection, dropped record detection, and both detections combined. The results for injection detection and dropped record detection were both similar for the ARIMA and Z-score methods, suggesting that each method performed comparably. The sensitivity for all methods fell, often rapidly, when the optimal error rate had been surpassed. Of course, the speed of the sensitivity fall is dependent on the granularity of the error rates tested, as well as on the tested range. For example, a tested range of only 8 to 10 split into fractions, might have shown a gentler slope. Even so, the results demonstrate the importance of a correctly chosen error rate.

Table 6.3 Car A: False positives and false negatives while monitoring 5 highest priority CAN IDs.

Method	Attack Type	False Positives	False Negatives
ARIMA	Injection	60	40
ARIMA	Dropped	36	0
Z-score	Injection	10	0
Z-score	Dropped	8	0
Supervised	Injection	0	0
Supervised	Dropped	0	0

Table 6.4 Car B: False positives and false negatives while monitoring 5 highest priority CAN IDs.

Method	Attack Type	False Positives	False Negatives
ARIMA	Injection	119	20
ARIMA	Dropped	146	22
Z-score	Injection	22	20
Z-score	Dropped	22	22
Supervised	Injection	0	0
Supervised	Dropped	0	0

The specificity across all methods increased with threshold increases, before plateauing. Thus a positive response was more likely to be correct at the higher threshold levels. However, given the fall in sensitivity, the methods become less likely to detect an attack, hence leading to a fall in the number of responses. This illustrates how selecting the correct threshold value is a trade-off between sensitivity and specificity.

Receiver Operating Characteristic (ROC) curves (Figure 6.5) are similar for all the classifiers, and the closeness of the curve to the top-left shows they performed well on this task at the optimum threshold rates.

6.1.2 Results for All CAN IDs

Using the same error rates and threshold values as for the 5 highest priority CAN IDs, the tests were extended to include detection on all the CAN IDs. Figures 6.6 and B.5 show the combined dropped record and injection results of these tests (the results from each test were very similar so are not shown individually). The test set for Car A comprised 1040 attacks, and 45,448 non-attacked windows; and for Car B, 560 attacks, 24,584 non-attacked windows. Since these sets are approximately 10x and 5x larger than the top 5 ID test sets for Car A

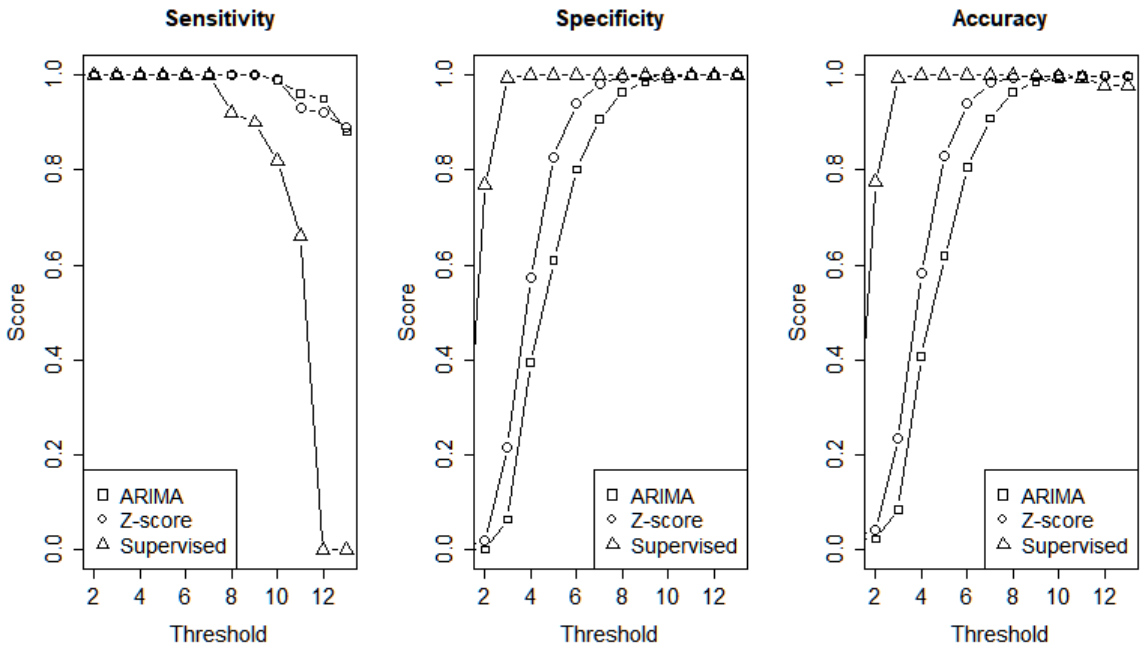


Fig. 6.2 Car A: Injection detection for the five highest priority CAN IDs: sensitivity, specificity and accuracy. (Supervised method used a 0.0001 to 0.05 threshold.)

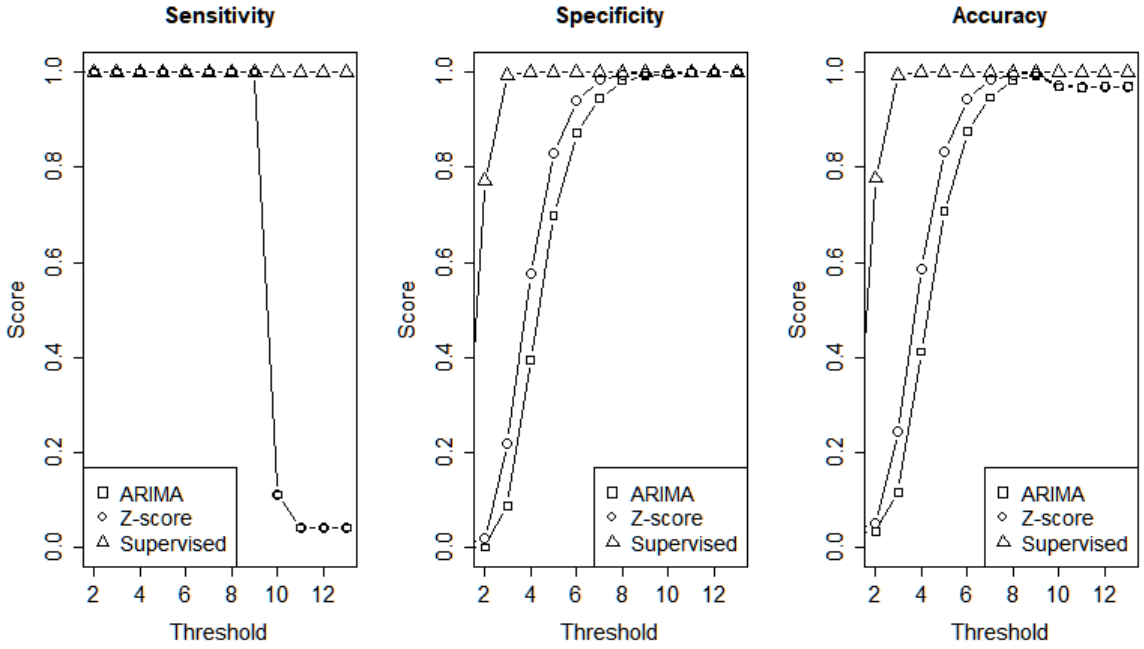


Fig. 6.3 Car A: Dropped record detection for the five highest priority CAN IDs: sensitivity, specificity and accuracy scores. (Supervised method used a 0.0001 to 0.05 threshold.)

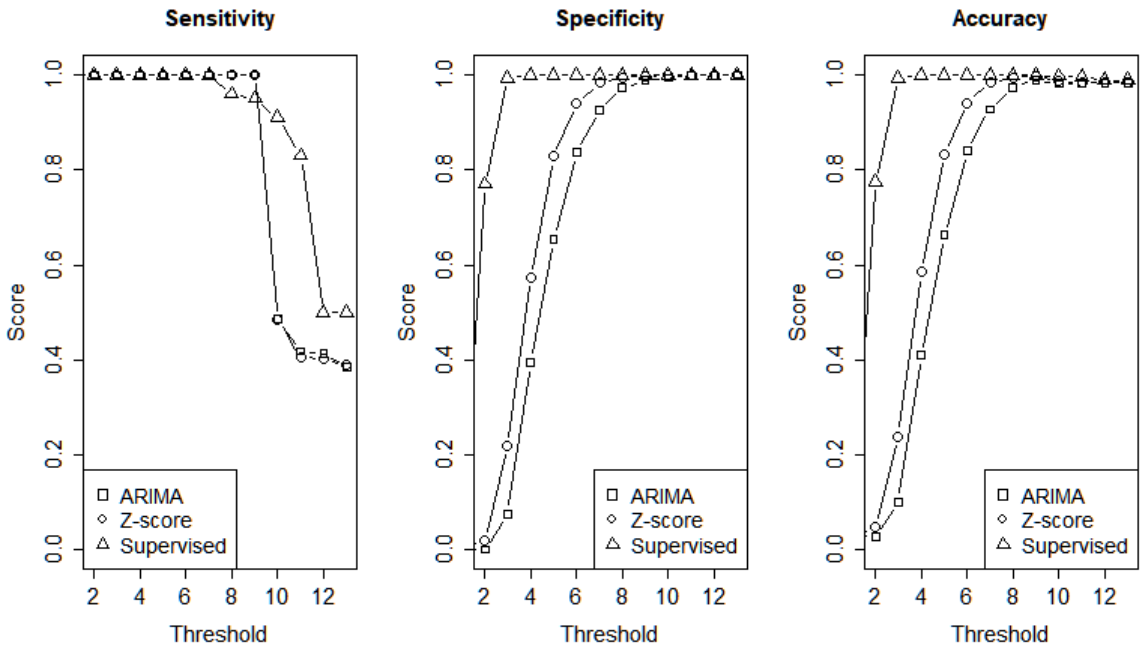


Fig. 6.4 Car A: Combined detection scores for the five highest priority CAN IDs: sensitivity, specificity and accuracy scores. (Supervised method used a 0.0001 to 0.05 threshold.)

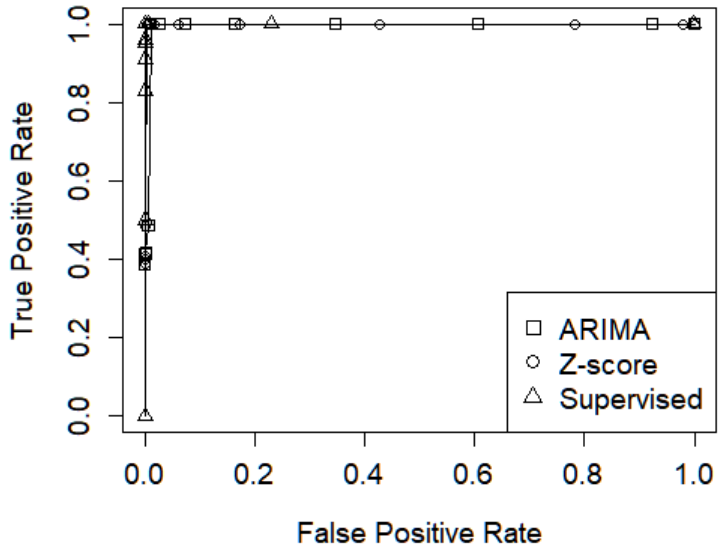


Fig. 6.5 Car A: ROC curves for combined detection in the five highest priority CAN IDs, adjusting error rate (ARIMA and Z-score) and threshold (Supervised).

and Car B respectively, we might expect the false positives and false negatives to rise by these amounts. However, in both Car A and Car B the sensitivity of the ARIMA and Z-score methods was reduced, thus these did not issue proportionately more positive responses. The supervised method showed a reduced specificity, especially at lower thresholds compared with its results for the top five CAN IDs. This was especially noticeable in Car A (compare Figure 6.6 with Figure 6.4) hence it saw a rise in false negatives, failing to detect attacks. Its sensitivity, though, remained similar, hence it saw a large rise in false positives.

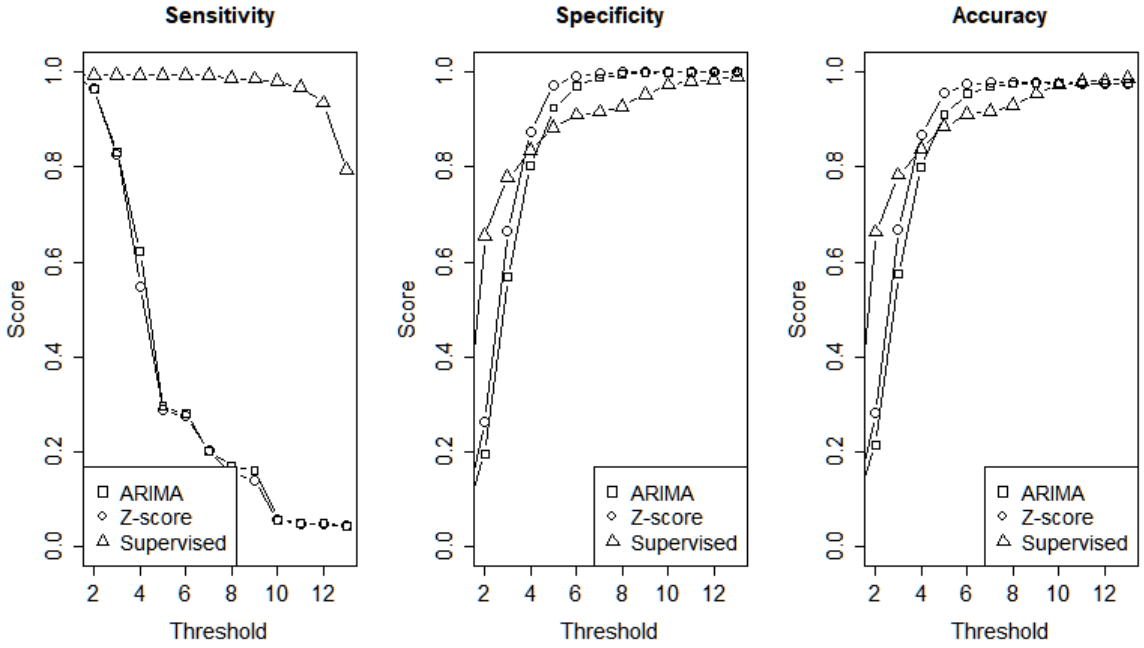


Fig. 6.6 Car A: Combined detection scores across all CAN IDs: sensitivity, specificity and accuracy scores. (Supervised method used a 0.0001 to 0.05 threshold.)

Table 6.5 Car A: Scores for the three methods for all CAN IDs.

Method	Attack Type	Accuracy	Sensitivity	Specificity
ARIMA	Injection	0.9804	0.2202	0.9978
ARIMA	Dropped	0.9793	0.1327	0.9986
Z-score	Injection	0.9812	0.1769	0.9996
Z-score	Dropped	0.9804	0.1326	0.9998
Supervised	Injection	0.8838	1.000	0.8811
Supervised	Dropped	0.8850	0.9846	0.8827

Table 6.6 Car B: Scores for the three methods for all CAN IDs.

Method	Attack Type	Accuracy	Sensitivity	Specificity
ARIMA	Injection	0.9770	0.5268	0.9872
ARIMA	Dropped	0.9714	0.3946	0.9846
Z-score	Injection	0.9862	0.4821	0.9977
Z-score	Dropped	0.9842	0.3929	0.9977
Supervised	Injection	0.9905	1.0000	0.9903
Supervised	Dropped	0.9910	1.0000	0.9908

Table 6.7 Car A: False positives and false negatives while monitoring all CAN IDs.

Method	Attack Type	False Positives	False Negatives
ARIMA	Injection	101	811
ARIMA	Dropped	61	902
Z-score	Injection	16	856
Z-score	Dropped	10	902
Supervised	Injection	5404	0
Supervised	Dropped	5332	16

These results might reflect many of the lower priority IDs having a less stable range of broadcast intervals, thus increasing the variance; which the unsupervised methods accommodated leading to them making proportionately fewer decisions. In contrast, the fixed variance assumptions of the supervised method ensured it carried on making a similar proportion of decisions, which became less correct. Thus, at the error rate 9 and threshold 0.003, the supervised method made 12800 positive responses, 16% of which were correct, whereas the ARIMA method made only 529 positive responses, 69% of which were correct, and the Z-score method made 348 positive responses, of which 93% were correct. Figure 6.7 shows the number of false positives for the intrusion detection plotted against time gap standard deviation for each CAN ID in Car A. There is a cluster (lower left) that showed low deviation in their broadcast times and low false positives. For the remaining CAN IDs, with more variance in their broadcasting, the false positives are higher. In contrast, the same data for Car B, plotted in Figure 6.8, shows that Car B has fewer CAN IDs with such high variance in their broadcasting and few that generated high numbers of false positives .

ROC curves for the combined injection detection and dropped record detection scores are shown in Figures 6.9 and B.6. As suggested by the results already discussed, the methods performed worse with all the CAN IDs included, and thus the curves are flatter. The curve of

Table 6.8 Car B: False positives and false negatives while monitoring all CAN IDs.

Method	Attack Type	False Positives	False Negatives
ARIMA	Injection	314	265
ARIMA	Dropped	381	339
Z-score	Injection	56	290
Z-score	Dropped	56	340
Supervised	Injection	239	0
Supervised	Dropped	226	0

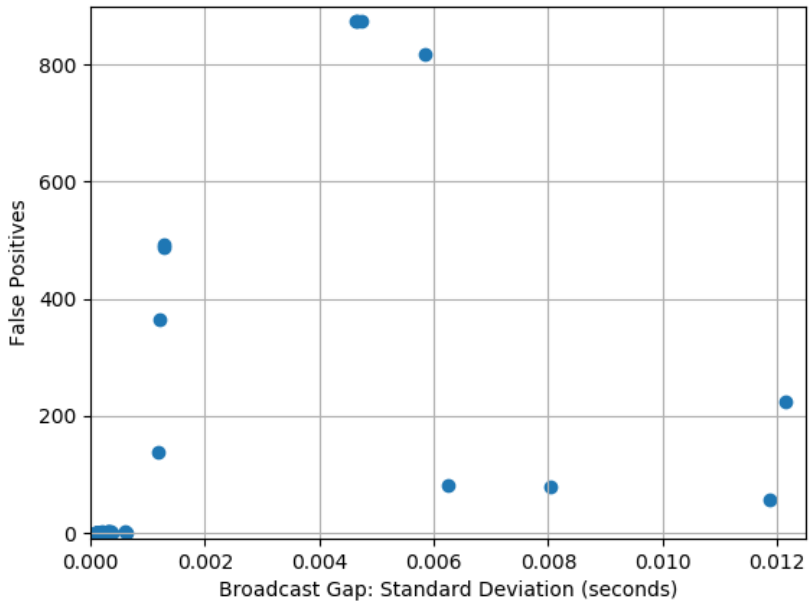


Fig. 6.7 Car A: Intrusion detection false positives by time gap standard deviation for individual CAN IDs.

the supervised method suggests its potential for the highest detection, but it would need the threshold adjusting.

6.1.3 Results for Window-size Variation

Varying the window size was investigated using the CAN data from Car A. Keeping the window size small is likely to be preferred since it would minimise the time between the attack and its detection. However, it was found that reducing the window size to 1 second produced "insufficient observations" errors with the auto.arima function when the 1 second reflash attack data was processed. Although the windows for the testing and for the reflash

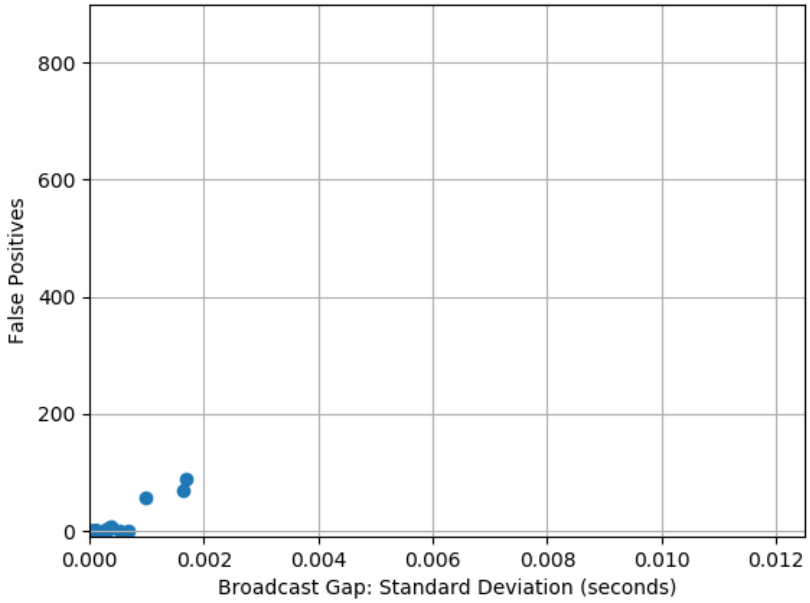


Fig. 6.8 Car B: Intrusion detection false positives by time gap standard deviation for individual CAN IDs.

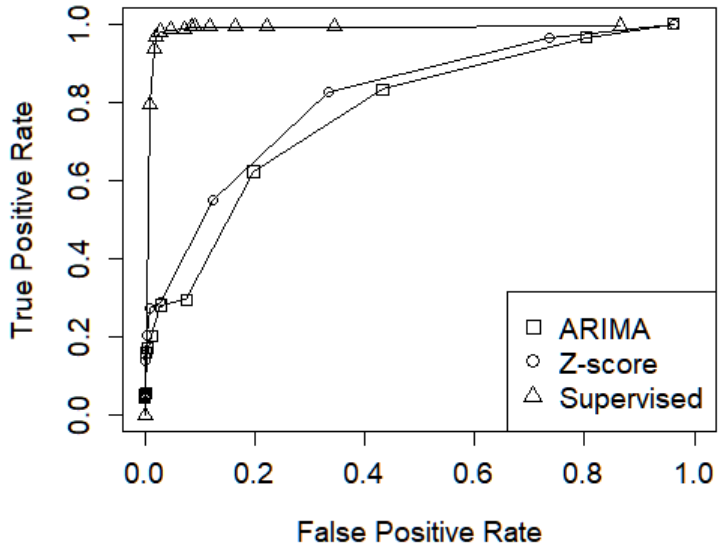


Fig. 6.9 Car A: ROC curves for combined detection across in all CAN IDs, adjusting error rate (ARIMA and Z-score) and threshold (Supervised).

attack simulation were not overlaid, it is likely that the error resulted from some of the tested windows having insufficient or no packets for the auto.arima process, although no minimum sample size is discussed in the auto.arima specification. Such errors might be easily avoided with simple clauses added to the detection code to bypass the detection when insufficient observations are present, and perhaps count the window as a suspected attack. However, first we would need to determine the precise circumstances that generate the error, such as how few observations are needed to trigger it.

Increasing the window size, with the threshold kept constant, had little effect on specificity and accuracy, but did increase sensitivity in the ARIMA and Z-score methods (Figure 6.10). This reflected an increase in true positives and false positives, hence might be of little benefit if false positives are to be avoided.

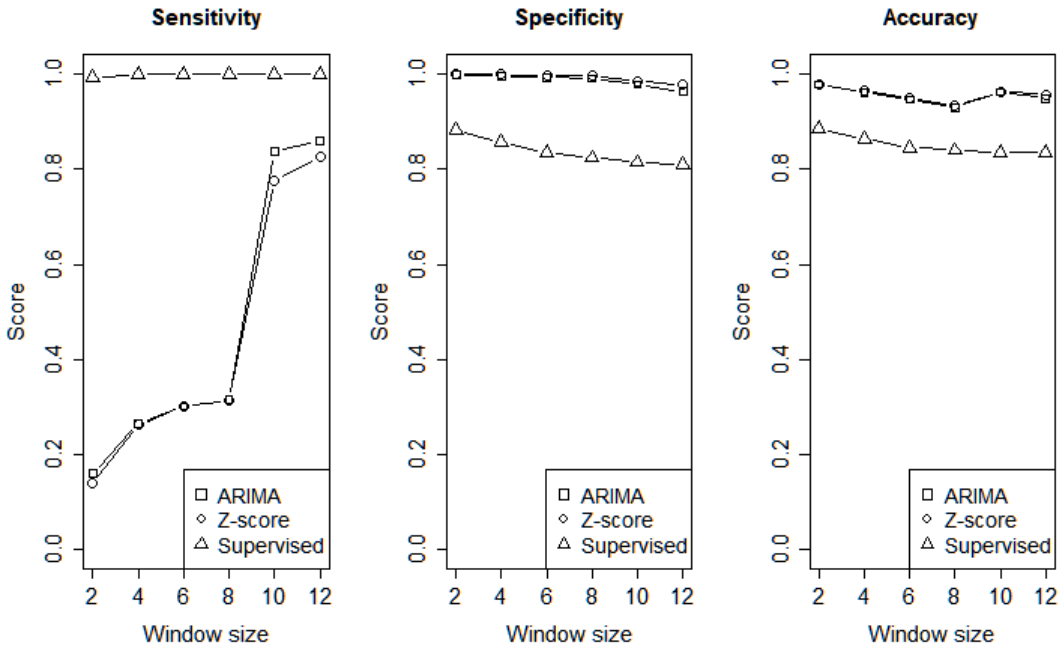


Fig. 6.10 Window size impact on combined injection detection and dropped record detection across all CAN IDs, Car A: sensitivity, specificity and accuracy scores.

6.2 Method Overheads

Prior to implementation, and during future development, the technical viability of any proposed method would need assessing. This would depend on the hardware and software that would be used for in-car implementation. Preliminary assessment of the method overheads was carried out on the set-up used for these experiments (i.e. a medium spec laptop computer),

rather than a possible in-vehicle system. Even so, the performance characteristics are worth noting to inform viability.

Process timing data was recorded for three CAN IDs at the beginning, middle and end of the CAN ID list when arranged by priority; thus testing a range of broadcast rates.

For the Z-score and Supervised methods, for each CAN ID each window was assessed in under 0.016 seconds. The ARIMA method also made the prediction in each window in under 0.016 seconds per CAN ID. However, on top of this must be added the training, or "fit", time for the ARIMA process. Mean times for this in the three CAN IDs were 0.31, 0.41 and 0.17 seconds per window, though the maximum times observed were 1.41, 2.20 and 0.39 seconds.

Given that these timings are for processing each 2 second window per CAN ID, and Car A had over 50 CAN IDs, the ARIMA timings could be problematical for a system covering the whole CAN ID range, analysing in real-time and being retrained for each new window, even if a processing unit that had a performance comparable to the laptop used in this experiment were available.

6.3 Conclusion

Analysis of the timing anomaly detection results drew on accuracy, sensitivity, specificity and response curve calculations generic across research fields and described in [74, 22]. To these were added the criteria to assess the detection decision, accounting for the window processing derived for parsing the packet flow. For the injection and dropped packet simulations tested, across the three methods assessed (ARIMA, Z-Score and supervised mean comparison) the detection rates were worse when all CAN IDs were tested, in contrast to testing only the highest priority CAN IDs. ARIMA and Z-Score suffered disproportionately higher rates of false negatives in particular, whereas the supervised method showed a high rise in false positives. Potential reasons for this were assessed from considering the packet broadcast profiles. Altering the length of the window used for capturing the CAN flow had little effect on detection accuracy, although program errors emerged if it was made too small. The observed performance overheads of the methods were also assessed, showing that processing time could be problematical, especially for an ARIMA system assessing all CAN IDs in real-time. The implications for the results are considered in Chapter 9.

Chapter 7

Detecting Payload Anomalies - Experiment Design

This chapter discusses the methods developed for identifying suitable packet clusters for the payload anomaly detection, and presents the experimental design adopted for the subsequent payload anomaly detection experiments. The results from those experiments are presented in Chapter 8.

The anomaly detection algorithm development and experiments used data captured from two unmodified popular UK small family cars (henceforth referred to as Car 1 and Car 2) made by different manufacturers. CAN logs were captured onto a laptop using a Kvaser Leaf Pro HS v2 OBD-II reader attached to the car's OBD port. The subsequent processing, analysis and experiments were developed in Python 3.6 running a standard laptop.

7.1 Preliminary Analysis and Cluster Identification

As stated in Chapter 4, the methods tested for the data anomaly detection require quantitative data, such as from sensor readings, rather than category data, and detect anomalies by comparing data fields that might be associated. Thus the identification of anomalies in one field is sought by comparing the values in that field against the values in others. Pre-analysis therefore involved profiling the data to try to identify likely sensor fields, identified by having a high count of unique values and data that showed increasing and decreasing trends. Algorithms were developed to: a) identify the sensor data fields, including those represented by combined two-byte values; and, b) group the identified sensor fields into functional clusters.

To determine the algorithms and profile the data, more than eight hours of driving was analysed from Car 1, totalling more than 42 million records. The developed algorithms were then also tested on Car 2, providing an indication of their cross-model validity. The journeys captured on Car 1 spanned a range of conditions, seasons, and lengths. Some of the recordings were made with the car static and specific functions operated in isolation to see if it was possible to identify the CAN messages associated with each function by monitoring field changes. The CAN IDs used by the car, their data field value profiles, and the packet intervals, were analysed across the captured records.

7.1.1 Identifying Concatenated Sensor Fields

Markovitz and Wool [48] classify CAN data fields into four categories: constant (fields with just one value); multi-value (containing a small number of unique values); counter (consecutive, uniformly increasing), and sensor (containing a large number of unique values describing a noisy sine). CAN data packets can have up to eight fields. However, those data fields might be composite, with the data value represented by the concatenation of the hex or binary representation of two or more fields.

From analysis of the CAN logs captured for this thesis, it was observed that concatenated sensor fields presented a pattern showing a field with a high number of unique values, adjacent to a field with only a few unique values. However, this pattern might be coincidental, so could not be used as the sole indication that fields should be concatenated. More reliably, such data fields could be determined by observing whether the values in a field incremented or decremented by one when the next data field in the packet was a sensor field that incremented gradually towards 255 and then plummeted towards 0, or fell gradually towards 0 and then jumped towards 255 (Figure 7.1). Whilst such fields were determined initially by visual inspections, an algorithm was produced to automate this process (Algorithm 2).

The algorithm is repeated for all combinations of adjacent data fields for a CAN ID (i.e. Field1 and Field2, Field2 and Field3, Field3 and Field2, Field2 and Field1, etc), and counts instances where the unit change of FieldA corresponds to previous, same-direction, changes in FieldB followed by FieldB suddenly changing in the other direction by an amount ("jump" in the Algorithm) large enough to suggest it has entered the next value-cycle. Such occurrences increase the likelihood that FieldA and FieldB are composite. Conversely, changes in FieldA that do not correspond to such a cyclical pattern in FieldB, decreases the likelihood. Each such occurrence is counted to generate likelihood and unlikelihood scores accordingly. Although an algorithm for parsing the CAN message was proposed in [48], it operated at the bit level, and was configured only for CAN packets that employed the full 64 bit (8 byte) data allocation. Moreover, although that algorithm attempted to distinguish

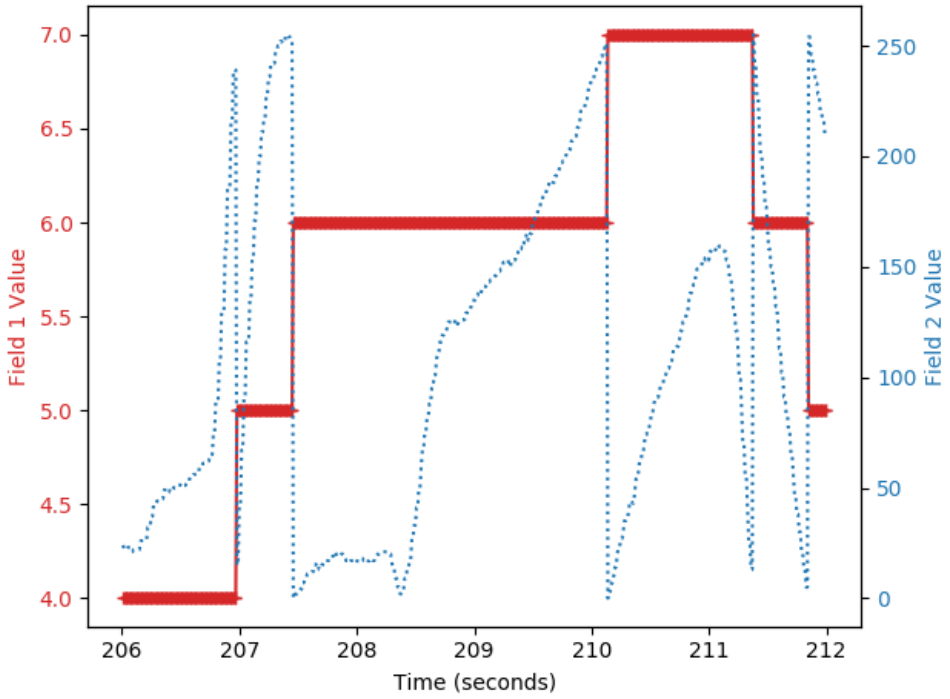


Fig. 7.1 Two data fields in a CAN packet show a pattern that suggests they are meant to be combined to represent a single reading from a sensor. When field 2 reaches the top or bottom of the value capacity suggested by its bit size, a corresponding unitary change is seen in field 1. Field 2 shows also a propensity to jump between its peak and near zero corresponding to the changes in Field 1.

Algorithm 2 Determine concatenated fields for a CAN ID

```
𝕏 ← all records in time order for CAN ID(x)
jump ← 100
for all data fields in 𝕏 do
  fieldA, fieldB ← next adjacent data field pair
  for all rows in 𝕏 do
    if fieldA^row(value) = fieldA^row−1(value) + 1 then
      if fieldB^row−1(value) - fieldB^row(value) >jump
      and fieldB^row−1(value) >= fieldB^row−2(value)
       then
        likely ← likely + 1
      else if fieldB^row(value) <= fieldB^row−1(value) then
        unlikely ← unlikely + 1
      end if
    else if fieldA^row(value) = fieldA^row−1(value) - 1 then
      if fieldB^row(value) - fieldB^row−1(value) >jump
      and fieldB^row−1(value) <= fieldB^row−2(value)
       then
        likely ← likely + 1
      else if fieldB^row−1(value) >= fieldB^row(value) then
        unlikely ← unlikely + 1
      end if
    end if
  end for
  if likely - unlikely >threshold then {our threshold = 1}
    fieldAB ← concatenate(fieldA(hex),fieldB(hex))
  end if
end for
```

```
CAN ID: 80
DATA1      24
DATA2     256
DATA3      54
DATA4     256
DATA5       1
DATA6       2
DATA7     256
DATA8       1
dtype: int64
```

```
CAN_ID: 80
Fields: DATA1 DATA2
Likely2: 249
Unlikely2: 0

CAN_ID: 80
Fields: DATA3 DATA4
Likely2: 518
Unlikely2: 0
```

Fig. 7.2 Example output from the composite data field detection algorithm. The algorithm shows both the number of unique values for each field (left of image), and the likelihood score of adjacent fields being connected (right of image). The pattern for the two identified composite fields (DATA1 and DATA2, and DATA3 and DATA4) was typical for all composite fields - a very high number of unique values (e.g. DATA2 or DATA4), following a field with a few unique values (e.g. DATA1 or DATA3).

between single value fields and category fields, it did not attempt to distinguish between sensor and counter data. Thus, their algorithm was unsuitable for the requirements of the processes envisaged here.

The algorithm devised for this thesis was implemented in Python and initially tested against Car 1. Fields were accepted as being potentially concatenated when the likelihood score was higher than the unlikelihood score. For validation, a visual inspection was carried out on these, and this also suggested they were concatenated sensor fields. The observed pattern for fields identified as composite was typical of that shown in Figure 7.2, i.e. two fields next to each other, the second with a high number of unique values. Visual inspection of the remaining fields did not suggest any missed concatenations, although it is acknowledged that the absence of evidence does not necessarily represent evidence of absence. The process was repeated on Car 2, and again, the fields flagged with a positive likelihood to unlikelihood ratio were checked through visual inspection, which also suggested they were all concatenated.

The algorithm does not pick up single fields that might be sensor readings. However, from studying the number of unique values and their pattern, some of these were suggested from Car 1, and included in the subsequent clustering process. Also, the algorithm does not compare non-adjacent fields, or check if sensor fields might be composite from three or more fields. These checks could be added to the algorithm, however the visual inspection of the data revealed no observed instances that suggested composite fields made of three data fields (i.e. at least 2 adjacent fields with high numbers of unique values).

7.1.2 Identifying Associated Fields

Having identified sensor fields, we need to group these into clusters, each of which contain the sensor fields pertaining to associated functions. For the anomaly detection process, the car function that the fields pertain to does not need to be known. What matters is that the fields are clustered into a group with a correlation that would enable the detection of anomalies within its elements. However, determining the function would provide a validation that the cluster detecting algorithm has worked, as well as help to determine which clusters might be targets for specific attacks. Although the lack of dictionary sharing by the manufacturers means that the mapping of CAN packet IDs to ECUs, or to car functions, is not available, estimation can be made from the analysis of CAN traffic combined with journey records captured on video and verbal commentary. In Car 1 it was possible, for example, to determine fields that change with the application of the accelerator, or with changes in speed regardless of the accelerator position, or as brakes are applied.

To support the manual functional analysis, a hierarchical cluster analysis was undertaken of the fields identified as sensor data. Pearson's correlation coefficient was used as a distance measurement to cluster the data fields according to the data field correlations. Implementation was in Python using the pandas .corr() function for the correlations [63], and .distance, .linkage and .fcluster functions in the SciPy package to generate the hierarchical clustering [68]. Fig 7.3 shows a plot of the resulting cluster matrix. The CAN packet ID is shown by the field's prefix, and the data field positions are shown by the D suffix. Composite fields have two D suffix elements. Thus, for example, data field 17C_D3D4 is a composite of the D3 and D4 fields from a packet with CAN ID 17C.

For this study, fields were considered as being in a related cluster where they had a correlation of 0.75 or greater with the other fields in the cluster. The two large darker clusters towards the top left and centre of the chart comprise fields that the manual analysis suggested were concerned with the accelerator and speed, respectively. These seemed to change as the accelerator was operated, or the car's speed changed, but seemed unaffected by other events. Of course, other than from conducting detailed and comprehensive reverse engineering, or from acquiring the CAN dictionary for the car, it is very difficult to determine what precisely the fields are measuring. They might be capturing something only very tentatively linked, such as fuel efficiency or a component pressure reading. Even so, the potential use of such correlated fields for determining contextual anomalies holds, no matter the nature of the link. Towards the lower right of the chart are fields for which the functionality could not be determined. Many of the fields show positive correlations, but the fields for packet ID 191 show a negative correlation with the other packets, and a particularly strong negative correlation with the cluster of packets that seemed to be concerned with the speed of the car.

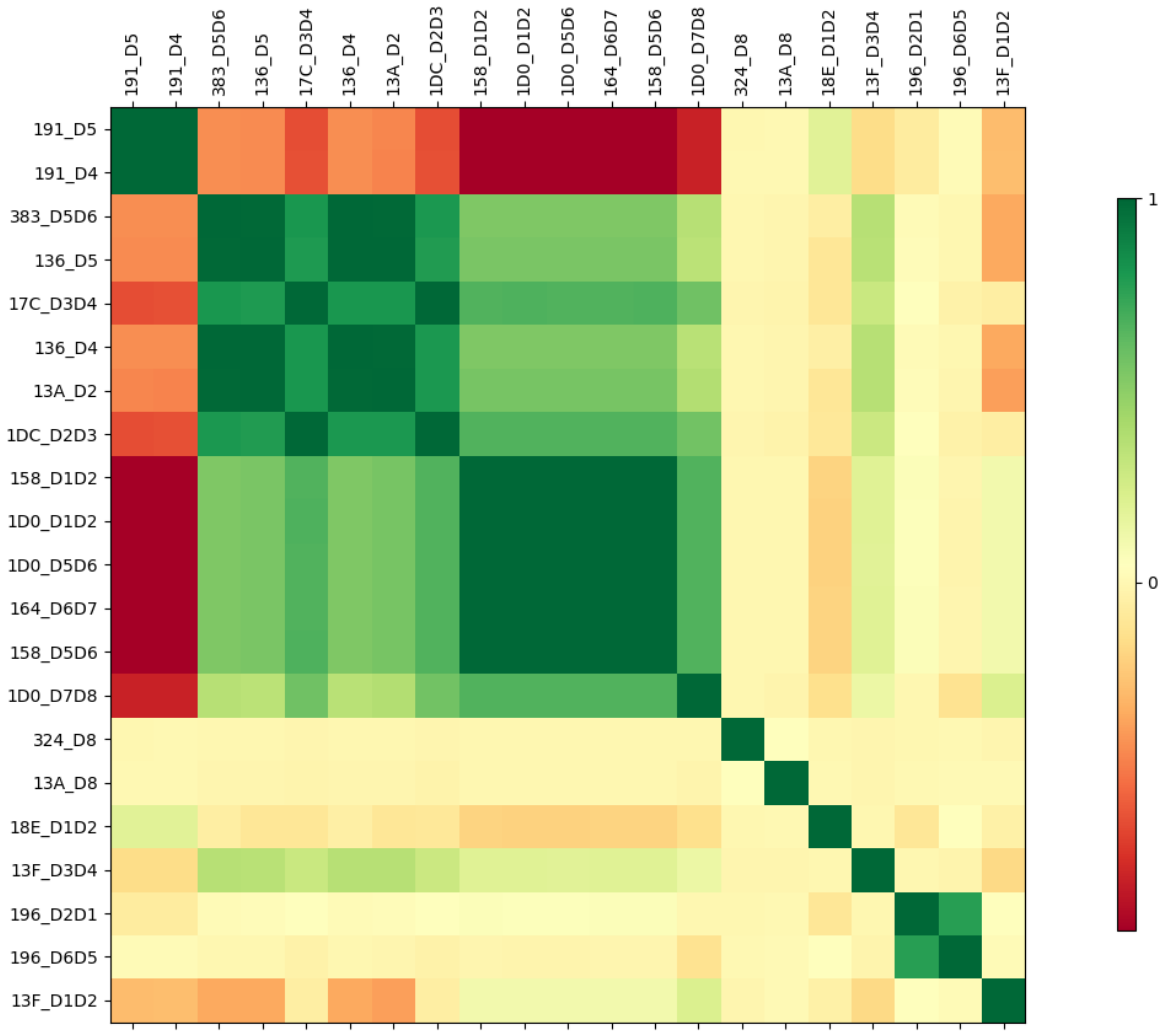


Fig. 7.3 Hierarchically clustered correlation matrix plot for sensor fields, Car 1. The two dark cluster blocks on the diagonal seems to correspond to the individual aspects (accelerator and speed) identified whilst monitoring the CAN data during driving.

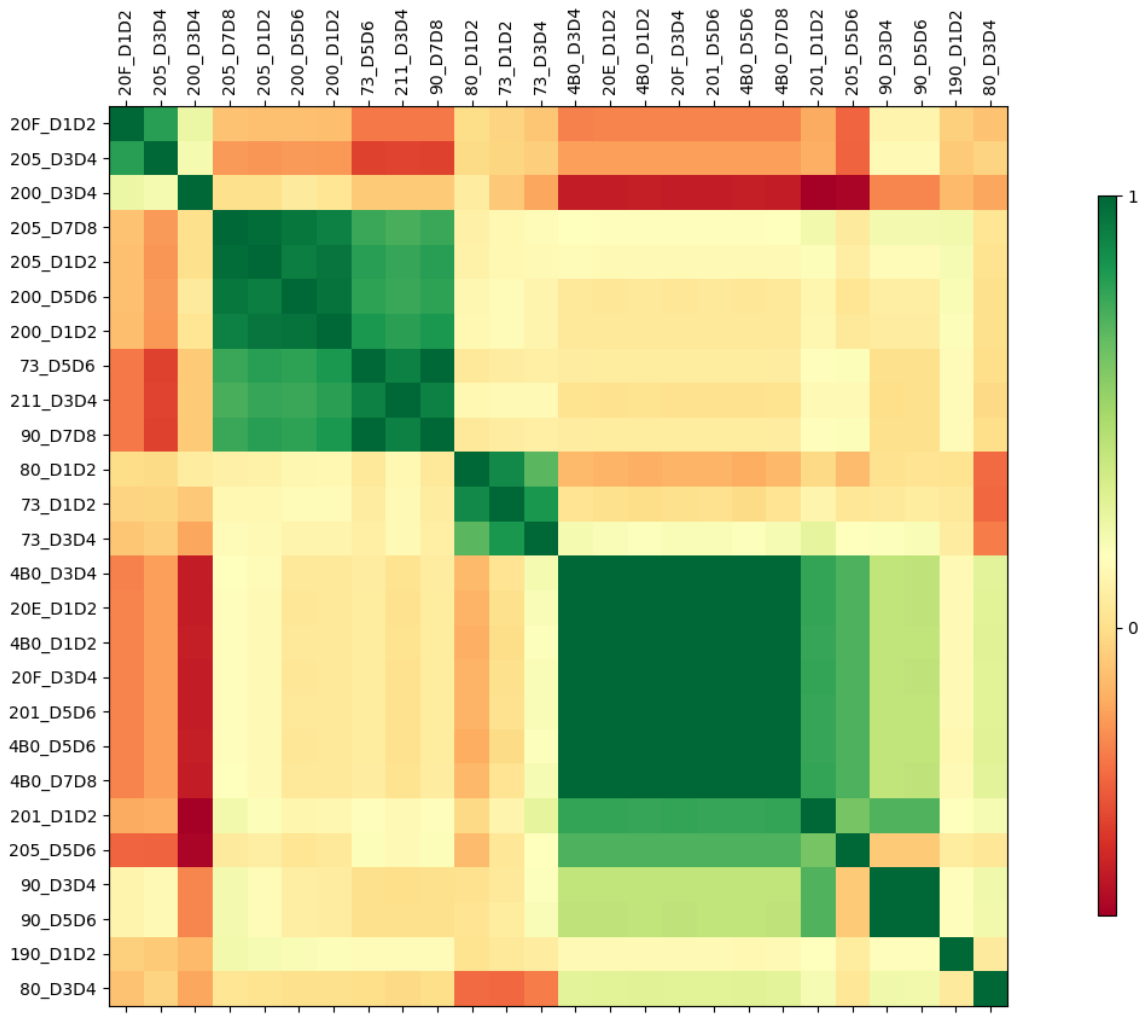


Fig. 7.4 Hierarchically clustered correlation matrix plot for sensor fields for Car 2.

The process was repeated with Car 2. Clusters for that car were also clearly identified (Figure 7.4), suggesting data fields that might describe associated functions. However, that car was not available long enough for a detailed functional analysis to be undertaken, so the functionality related to the clusters cannot be speculated. As with Car 1, two Car 2 clusters of fields having 0.75 correlation with each other were selected for the subsequent testing.

The clusters from both cars that were used for the experiments are shown in Table 7.1. The value distributions of the fields are shown in Appendix C, Figures C.1 to C.4. They show a mixture of distributions, and a few have a high instance of the lowest value, including values of 0 for Car 1. Since it is not known how the data is derived, it is difficult to speculate on the cause of the distributions. However, it is possible that the values at the range-origin for some fields represent occasions when a function (such as brakes or accelerator) was not active.

7.2 Experiment Data Generation and Evaluation Criteria

Having identified sensor fields, including those with values derived from composite fields, and determined clusters of associated fields, the CAN log flow can now be parsed into a format suitable for the anomaly detection. Snapshot records were created in the manner discussed in Chapter 4, with each snapshot record being created upon the broadcast of one of the CAN packets in the cluster list and comprising the most recently broadcasted values for the data fields in the cluster. Snapshot records were created for each of the main clusters identified by the clustering algorithm (i.e. Acceleration and Speed clusters for Car 1, and Cluster A and Cluster B for Car 2).

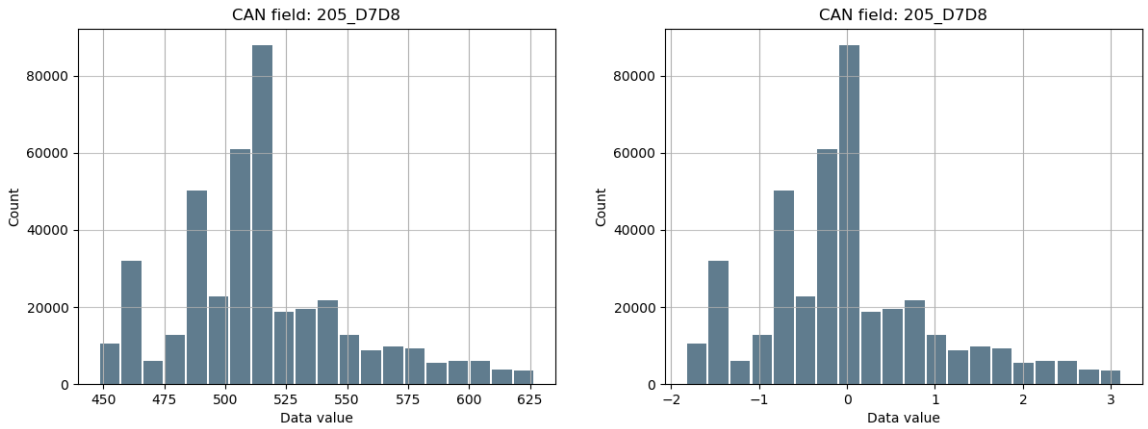
7.2.1 Experimental Data Sets

Each experimental data set comprised 8000 of the snapshot records for one of the clusters. The data set records were selected at random from the snapshot logs generated for a complete journey, with each snapshot being generated as described in Section 4.3.2. For each experimental data set, 6000 snapshots were assigned to a training set, 1000 for an unmanipulated testing set, and 1000 for an attack testing set. The testing sets determined the level of false positives and true negatives, whilst the attack sets, which were manipulated to mimic attacks (as discussed below), determined the level of true positives and false negatives.

The snaphsots from each experimental data set were randomly selected into the training, test and attack data sets using Scikit-learn train_test_split function [67]. Random sampling was chosen to deduce the likelihood of any systematic biases that might be caused by any

Table 7.1 Clusters used in the anomaly detection experiments. In Car 1 the cluster names indicate the function the cluster seemed to be associated with. A detailed analysis of car functions was not carried out in Car 2, so the cluster names are not given functional associations.

Car	Cluster	Data Field
Car 1	Accelerator	1DC_D2D3 136_D5 13A_D2 17C_D3D4 136_D4 383_D5D6
	Speed	1D0_D7D8 164_D6D7 158_D1D2 158_D5D6 1D0_D1D2 1D0_D5D6 191_D4 191_D5
Car 2	Cluster A	4B0_D1D2 4B0_D3D4 4B0_D5D6 4B0_D7D8 20E_D1D2 201_D5D6 20F_D3D4
	Cluster B	73_D5D6 90_D7D8 200_D1D2 200_D5D6 205_D1D2 205_D7D8 211_D3D4



(a) Frequency distribution of raw values.

(b) Frequency distribution post scaling.

Fig. 7.5 Data for the fields was scaled using the Scikit-learn StandardScaler, which retains the distribution shape. This example shows field 205_D7D8's value distribution before scaling (a), and post scaling (b).

trends or seasonality in the logs. Whilst selecting data at regular sampling points might ensure coverage across the full journey, it could produce a biased sample if, for example, the sampling corresponded to the activation of some time defined, cyclical functioning within the car. Although a random sample might not, in itself, be representative of the full population profile, the repetition of the testing during parameter selection, as well as the focus on the trends seen across repeating the experiment with different attack generation approaches and across journeys (discussed in Chapter 8) reduces the likely impact of any unrepresentative sample.

All sets were scaled using Scikit-learn unit variance StandardScaler function [67], retaining the scaling parameters obtained from processing the corresponding training set. The StandardScaler function standardises the data by first subtracting the mean value (giving the values a zero mean), and then dividing by the variance, thus giving the distribution unit variance. As shown in Figure 7.5 standardisation retains the distribution shape of the data, unlike scalers such as Min-Max which can crush values in data sets with outliers [27].

7.2.2 Attack Manipulation

The fields in the attack sets were manipulated to replicate the effects of attacks discussed in Chapter 2. These data sets were fed to the classifiers multiple times, with one field manipulated each time and the remaining fields being left unchanged. Two variations of

attack manipulation were used, with the classifiers being tested using attack data generated by each method.

In the first method, the data values for the attacked field were multiplied by a specific multiplier. Runs were repeated so that each data field was tested with every multiplier value. Thus, in a cluster with 7 data fields, the attack experiment would be replicated 63 times (7 * number of manipulation values). The nine multiplier values were 0.1, 0.5, 0.8, 0.9, 1.1, 1.2, 1.5, 2.0 and 3.0. This therefore tests attacks in which the attacker uses values that are multiples of the original value; for example, take the existing value for field x and rebroadcast it increased by 10%. Here the opportunistic attacker would be unconcerned with the construction of the original data value.

Provisional analysis of the fields from each car suggested that some fields had a range with an origin that did not start at zero; as is shown, for example, in Figure 7.5a. In cases where the range is narrow, yet there is a high range-origin, the outliers produced by the above manipulation method would be extreme; so they would be simple to detect, and would probably be rejected by any reading node in any case.

The second method first scaled the attack data set using the scaling parameters fitted from the training data set, and then applied the data manipulation to the attacked field by adding an offset to the result. The offsets were -1, -0.5, -0.3, -0.2, -0.1, 0.1, 0.2, 0.3, 0.5 and 1, thus increasing or decreasing the scaled data by specific magnitudes. This method produces attack data that, across fields, is more uniform in its magnitude, irrespective of the permitted range-origin for individual fields. Such a manipulation would be more representative of attacks in which the attacker had more knowledge of the data construction and meaning.

The manipulation ranges for both methods described above enable classifier performance to be assessed across a broad range of manipulations. Some of the demonstrated attacks have involved making subtle changes to the data field values (e.g. [52, 76]), thus the manipulation ranges included values that would induce very minor changes in the data field, as well as changes that are more extreme without being obviously beyond likely recognition limits.

7.2.3 Evaluation Definitions

During testing, the snapshot records in the test and attack data sets were fed individually to each classifier, which had been trained using the corresponding training data sets. The classifier then classified each snapshot according to its decision as to whether the snapshot was an anomaly (i.e. a snapshot containing an attacked field) or normal (i.e. a snapshot that contained no attacked fields). For the Compound Classifier (CC) an anomaly corresponded to instances that fell outside all hyperspheres. For the other Local Outlier Factor (LOF) and

One Class Support Vector Machine (OCSVM) classifiers, the decision was as reported by the software implementation function.

The analysis employed the following definitions: True Positive (TP) an attacked snapshot identified as such by the classifier; False Negative (FN) an attacked snapshot that was not identified as such by the classifier; False Positive (FP) a normal snapshot classified as having been attacked; True Negative (TN) a normal snapshot classified as such. F_β scores [6] were used to provide a summarisation of these, taking β=0.5 so as to prioritise precision above recall (Equation 7.1).

$$F_\beta = \frac{(1+\beta^2) \times TP}{(1+\beta^2) \times TP + \beta^2 \times FN + FP} \tag{7.1}$$

In the results tables discussed in Chapter 8 only the FP and FN values are presented. TN and TP values can be calculated by subtracting, respectively, FP and FN values from 1000 (the sample sizes).

7.2.4 Inter-Journey Assessment

The objectives of the initial experiments were to determine optimal parameters for the classifiers, assess the viability of the methods on different vehicles, compare the classifiers performance, and gauge their potential effectiveness. The experiments were performed using training, test and attack data sets from within the same journey. Although they provide an indication of the results that might be achieved, a greater challenge is determining the training that would be needed for the classifiers to perform at their optimal detection in the long term - spanning multiple journeys over a representative deployment time-span.

To provide indicative assessment of this, subsequent testing was undertaken in which the trained fit estimators from the classifiers and the scaling were saved and applied to testing across journeys. That is, the test and attack data sets were scaled and classified using classifiers trained on data that was not from their journey. In these experiments, the scaler fit and classifier fit were saved and reapplied using the Scikit-learn *joblib* functions [67] for the LOF and OCSVM, and the locate sizes and locations for the Compound Classifier. These experiments are discussed in Section 8.3.

7.3 Conclusion

This chapter has presented the algorithms devised for identifying the sensor data fields (including concatenated ones) and allocating these into clusters of likely associated functionality, thus allowing anomaly detection based on their interaction. The chapter also presented the

design and evaluation criteria used for testing the one-class anomaly detection methods discussed in Chapter 4 (the results are presented in Chapter 8).

An existing method proposed for identifying some field types [49] was inappropriate for this thesis since it did not distinguish sensor (continuous) data from other types, and was devised only for packets that used all eight bytes of data. The algorithm devised for this thesis determines concatenated sensor fields based on the data-value flow between fields, and was tested against a manual analysis of the data from two cars. Clustering was achieved using correlation matrices, and, again, was tested against a functional analysis of the packet characteristics observed during operations on the cars.

Data for the training and testing required the presentation of CAN data as snapshots of the most recent broadcast values for fields in the analysis cluster. The testing data sets comprised normal CAN records and records in which the data values in the sensor fields were manipulated by varying magnitudes, thus enabling comparison of attacks that might entail subtle value manipulations, as well as those employing more extreme value changes.

The evaluation definitions were presented, including the F_β scores defined in [6] that present a representation of the accuracy weighted according to the priority of recall versus precision.

Chapter 8

Detecting Payload Anomalies - Results

This chapter presents the results from the payload anomaly detection experiments described in Chapter 7. The implications of those results are discussed in Chapter 9.

8.1 Results and Discussion

All classifiers were trained and tested using data sets constructed as discussed in Chapter 7. Initial experimentation (Section 8.1.1) was conducted using the data sets from a journey from Car 1 to determine classifier parameter configurations that gave optimal all-round performance. Because of the large volume of results, these initial evaluations were restricted to examining only the averaged $F_\beta(0.5)$ scores for each test parameter permutation. Thus for each training parameter permutation, the average $F_\beta(0.5)$ score was taken across all the attack multipliers for all the fields in the cluster. More detailed analyses (Sections 8.2 and 8.3) were then undertaken to explore the performance of the optimised classifiers against the data sets from Car 1 and Car 2. That testing compared the optimised classifiers across a variety of attack and training/testing permutations.

8.1.1 Determining Optimal Parameter Values

The Local Outlier Factor (LOF) was tested with parameter permutations comprising k-nearest neighbours (KNN) between 3 and 100, and contamination between 0.01 and 0.12. For the accelerator data set, the highest average $F_\beta(0.5)$ score, 0.8722, was achieved with KNN=12 and contamination=0.06. For the speed data set, the highest average $F_\beta(0.5)$ score, 0.9376, was achieved with KNN=5 and contamination=0.04. The number of false positives at these settings from the 1000 undoctored records are shown in Table 8.1.

The One Class Support Vector Machine (OCSVM) was tested with permutations comprising Nu between 0.001 and 0.7 and gamma between 1e-07 and 0.9. For the accelerator data set, the OCSVM achieved a highest average $F_\beta(0.5)$ score, 0.7748, with Nu = 0.015, and gamma = 0.5. For the speed data set the highest average $F_\beta(0.5)$ score, 0.8332, was also achieved with Nu = 0.015, and gamma = 0.5.

Table 8.1 False positive recorded for Car 1 with classifiers individually optimised to the data sets (1000 record test data set).

	Accelerator	Speed
LOF	54	25
CC	100	167
OCSVM	14	12

The Compound Classifier (CC) was tested with parameter values that were recommended by its author [3, 4] scaled for the standardisation process (i.e. hypersphere size adjustment of 0.0001 * *the number of dimensions*; and hypersphere movement of 0.05). Other parameter values were tried, but these slowed the training, and produced no improvement in classification performance. The CCs were trained until they captured 97% of the training instances for the Car 1 Speed cluster, and 96% for the Accelerator cluster. The proportion of training instances captured by the hyperspheres plateaued at these rates, and trying to achieve higher rates produced no improvement in capture after many minutes extra training. The trained CCs replaced the 6000 training data points with 21 hyperspheres for the Accelerator cluster, and 8 hyperspheres for the Speed cluster. Test instances were then classified as normal if they fell inside any hypersphere, and as anomalies if they fell outside of all. The average CC $F_\beta(0.5)$ score was 0.6510 for the Accelerator cluster, and 0.6276 for the Speed cluster.

All subsequent tests for Car 1 and Car 2 were conducted with LOF and OCSVM parameters fixed midway between the observed optimal range, at KNN=9, contamination=0.05 for the LOF, and Nu = 0.015, gamma = 0.5 for the OCSVM.

8.2 Baseline Results

The initial testing of the optimised classifiers was conducted with the training data set, test data set, and attack data set all from within the same journeys. For both Car 1 and Car 2, the selected journey was approximately 11 minutes long. This testing presented a gauge of the potential ability of the classifiers. Although using training and test data from the same

journey clearly does not give an indication of the capability of a trained classifier to detect across journeys (which was subsequently tested, see Section 8.3), it does give an indication of the performance that might be achieved, without yet considering challenges of interpreting intra-journey results or determining suitable training regimes for long-term detection.

Appendix D presents, for each car, the breakdown of false positives, false negatives and $F_\beta(0.5)$ scores for the classifiers when detecting the attack data across the attack manipulations applied prior to the data scaling. Appendix E presents these scores for the attacks generated by applying the off-sets to the pre-scaled data.

8.2.1 Baseline Results - Car 1

The results for Car 1 are summarised below in tables 8.2 to 8.13, which present the numbers of false positives, as well as the $F_\beta(0.5)$ score averages by cluster and data field tested.

Table 8.2 Mean $F_\beta(0.5)$ scores by field: Car 1 Accelerator cluster (1000 record test data set). Attacks generated by applying multipliers prior to scaling.

	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6	All
LOF	0.9376	0.7227	0.7442	0.9391	0.8929	0.8634	0.8500
CC	0.7923	0.5399	0.5468	0.7922	0.6155	0.6190	0.6510
OCSVM	0.9292	0.6421	0.6527	0.9286	0.7461	0.7503	0.7748

Table 8.3 Mean $F_\beta(0.5)$ scores by field: Car 1 Speed cluster (1000 record test data set). Attacks generated by applying multipliers prior to scaling.

	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5	All
LOF	0.9448	0.8827	0.8839	0.8841	0.8817	0.8817	0.9514	0.9529	0.9079
CC	0.6319	0.6382	0.6377	0.6378	0.6379	0.6376	0.6000	0.5995	0.6276
OCSVM	0.7991	0.8209	0.8209	0.8212	0.8209	0.8205	0.8823	0.8801	0.8332

Table 8.4 False positives recorded for Car 1 Speed cluster (1000 record test data set). Attacks generated by applying multipliers prior to scaling.

	Accelerator	Speed
LOF	59	57
CC	100	167
OCSVM	14	12

Table 8.5 Mean $F_\beta(0.5)$ scores by field: Car 1 Accelerator cluster (1000 record test data set). Attacks generated by applying offsets after scaling.

	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6	All
LOF	0.8772	0.8779	0.8633	0.8793	0.8580	0.8375	0.8655
CC	0.5790	0.5811	0.5821	0.5794	0.5751	0.5745	0.5785
OCSVM	0.6435	0.6679	0.6684	0.6451	0.6804	0.6888	0.6657

Table 8.6 Mean $F_\beta(0.5)$ scores by field: Car 1 Speed cluster (1000 record test data set). Attacks generated by applying offsets after scaling.

	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5	All
LOF	0.9606	0.9619	0.9619	0.9620	0.9620	0.9621	0.9581	0.9571	0.9606
CC	0.4848	0.4971	0.4970	0.4971	0.4967	0.4975	0.5122	0.5137	0.4995
OCSVM	0.7626	0.8192	0.8194	0.8194	0.8205	0.8203	0.8857	0.8459	0.8241

Table 8.7 False positives recorded for Car 1 (1000 record test data set). Attacks generated by applying offsets after scaling.

	Accelerator	Speed
LOF	43	48
CC	86	185
OCSVM	22	10

From the $F_\beta(0.5)$ scores in Tables 8.2, 8.3, 8.5 and 8.6 it can be seen that the highest $F_\beta(0.5)$ scores overall, and averaged within fields, were obtained by the LOF, followed by the OCSVM; with the CC being the least successful. The OCSVM showed the lower rate of false positives (Tables 8.4 and 8.7); although tended to generate more false negatives compared with the LOF (as seen in the tables in Appendix D and E), hence achieved a lower $F_\beta(0.5)$ score.

Although, looking at the unscaled data attacks suggests the detection was far less successful in fields 136_D5, 13A_D2 of the Accelerator cluster (Table 8.2), interpretation of this data presents a difficulty due to the attack method adopted. In these two fields, almost 25% of values were zero and hence would not have been changed by the attack method. In the scaled data attack (Table 8.5), the zero values were converted to other values by the scaling, before being manipulated by attack offset. Hence those fields appear to yield better detection

in those attacks. The dilemmas posed by this data characteristic and the problems it presents to the data analysis, as well as the options to overcome these, are discussed in Chapter 9.

As might be expected, the detection rate across all CAN fields fell as the magnitude of attack manipulation decreases (the central rows in the tables in Appendix D and E). The LOF achieved fewer false negatives and a higher $F_\beta(0.5)$ score at lower magnitudes of manipulation compared with the OCSVM, but the lower false positive score of the OCSVM gained it a higher $F_\beta(0.5)$ score at the higher manipulations; even though there does not appear to be a large reduction in its false negative scores, compared to the LOF, at these magnitudes. For example, on the Accelerator cluster (Tables D.3 and D.5), with a 1.1 multiplication manipulation, the LOF achieved an average false negative rate of 466, compared with 854 for the OCSVM, giving it a mean $F_\beta(0.5)$ of 0.7601 when its 59 false positive score is considered; compared with 0.3752 for the OCSVM, with the false positive score of 14. However, with a 3.0 manipulation, the LOF achieved an average false negative rate of 146, compared with 141 for the OCSVM, giving it a mean $F_\beta(0.5)$ of 0.9076, compared with 0.9450 for the OCSVM.

For the attacks generated prior to scaling (Appendix D), the detection seems more successful on some fields compared with others. The fields with the higher number of false negatives also had data spread over higher ranges (see Figures C.1 and C.2). Thus, in the attacks generated by applying the offsets to the already scaled data, there is less variation in detection across fields (Appendix E).

8.2.2 Baseline Results - Car 2

The following results were obtained from Car 2, a 2012 small family car of a different manufacturer to Car 1. As discussed in Chapter 7, two ID clusters were chosen for testing. A detailed functional testing of Car 2 was not possible, so there is more uncertainty as to what the clusters might be associated with compared to Car 1. Therefore the clusters are referred to merely as Cluster A and Cluster B. The same classifier parameters were used as for Car 1. Training the CC plateaued for Cluster B at 95% training success rate, while Cluster A was trained to 97%.

Table 8.8 Mean $F_\beta(0.5)$ scores by field: Car 2 Cluster A (1000 record test data set). Attacks generated by applying multipliers prior to scaling.

	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4	All
LOF	0.9682	0.9682	0.9682	0.9682	0.9682	0.8996	0.9682	0.9584
CC	0.7952	0.7952	0.7952	0.7952	0.7952	0.6719	0.7952	0.7776
OCSVM	0.9819	0.9819	0.9819	0.9819	0.9819	0.8764	0.9819	0.9669

Table 8.9 Mean $F_\beta(0.5)$ scores by field: Car 2 Cluster B (1000 record test data set). Attacks generated by applying multipliers prior to scaling.

	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4	All
LOF	0.9601	0.9601	0.9601	0.9601	0.9601	0.9581	0.9601	0.9598
CC	0.9266	0.9266	0.9266	0.9266	0.9266	0.8501	0.9266	0.9157
OCSVM	0.9789	0.9789	0.9789	0.9789	0.9789	0.9788	0.9789	0.9788

Table 8.10 False positives recorded for Car 2 (1000 record test data set). Attacks generated by applying multipliers prior to scaling.

	Cluster A	Cluster B
LOF	41	52
CC	322	99
OCSVM	23	27

Table 8.11 Mean $F_\beta(0.5)$ scores by field: Car 2 Cluster A (1000 record test data set). Attacks generated by applying offsets after scaling.

	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4	All
LOF	0.9635	0.9629	0.9632	0.9635	0.9635	0.9635	0.9635	0.9634
CC	0.6006	0.6003	0.6006	0.5995	0.5998	0.5984	0.6000	0.5999
OCSVM	0.9300	0.9176	0.9315	0.9304	0.9303	0.9310	0.9304	0.9288

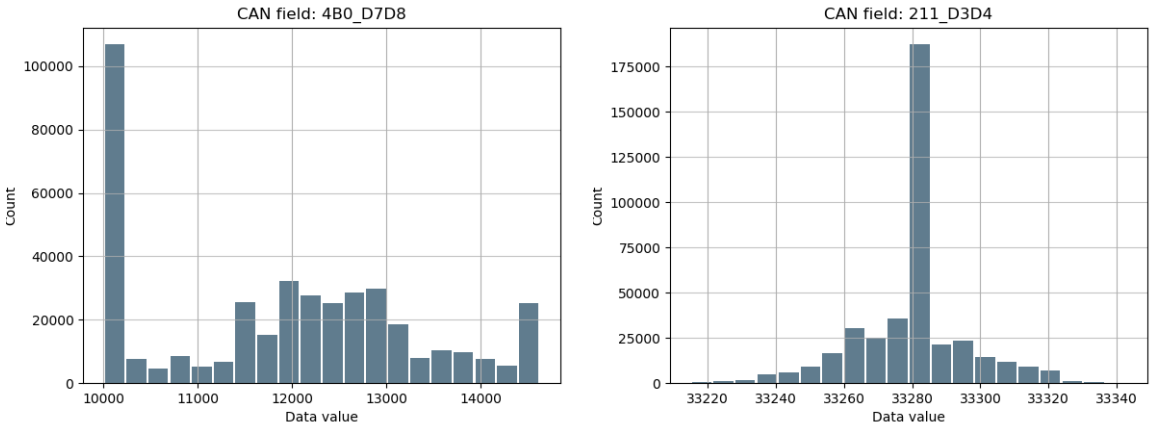
Table 8.12 Mean $F_\beta(0.5)$ scores by field: Car 2 Cluster B (1000 record test data set). Attacks generated by applying offsets after scaling.

	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4	All
LOF	0.7255	0.7356	0.7554	0.7397	0.7352	0.7222	0.3747	0.6840
CC	0.2787	0.2794	0.2834	0.2801	0.2851	0.2828	0.2772	0.2810
OCSVM	0.3375	0.3342	0.3494	0.3653	0.3871	0.3881	0.2220	0.3405

Table 8.13 False positives recorded for Car 2 (1000 record test data set). Attacks generated by applying offsets after scaling.

	Cluster A	Cluster B
LOF	47	42
CC	316	86
OCSVM	7	16

The $F_\beta(0.5)$ scores for the attacks generated prior to scaling (Tables 8.8 and 8.9) are higher than those for Car 1 (Tables 8.2 and 8.3). Tables D.17 to D.19, and D.23 to D.25, show there were no false negative detections for most of the fields. However, many of the fields in Car 2 had a range with a high origin (Figure 8.1). As explained in Chapter 7, generating attack data by simply multiplying the original value in such data will result in outliers that are proportionally further from the mean, hence being easier to to detect.



(a) Example sensor field from Cluster A.

(b) Example sensor field from Cluster B.

Fig. 8.1 Distribution of sensor field values that is typical of those observed in Car 2. The distribution shows a relatively narrow range, with the origin at a high value.

Interestingly, the OCSVM scored a slightly higher $F_\beta(0.5)$ average score on this data compared with the LOF, even though the LOF scored the higher on both Car 1 attack modes as well as on the pre-scaled attack mode on Car 2. Generating the attacks using the scaled data seems to have presented an especially difficult detection challenge for the Cluster B, since all classifiers achieving their lowest average $F_\beta(0.5)$ score here, with the OCSVM and CC scoring especially low. Although, similar to the other clusters, the false negatives peak at the lower magnitude manipulations, they remain higher across the manipulation range for this cluster.

8.3 Classification Across Journeys

The capability of the trained classifiers to detect across different journeys was explored using journeys captured from Car 1.

The rows in Tables 8.14 to 8.19 show the journey used for the training data sets. The columns show the two journeys used for the test and attack data sets (Journeys A and D). Journey A was 11 minutes, C was 82 minutes, and D was 8 minutes. B is a composite training data set, made by combining the data sets from journey A with the data set from an additional 8 minute journey, E. Because of the length of journey C, it was not possible to generate snapshot records for the complete journey due to the time taken to process such a large log file into a text-output snapshot file. Therefore six 100 second samples were extracted at equal intervals, including the start and the end of the journey. All journeys included a mixture of road types and traffic situations. The two test data journeys, A and D, were recorded a few days apart.

As with the other payload anomaly detection testing, the training data comprised 8000 instances from a random sample from the entire data file. For these experiments the 1000 record test and attack files were fixed, so all classifiers were tested against exactly the same records.

Perhaps unsurprisingly, training on the same data set (i.e. journey) often gave the better detection scores. This can be seen in the highest $F_\beta(0.5)$ scores in nine of the twelve columns of data across the Tables. However a deviation to this is seen in the OCSVM for the Speed cluster (Table 8.19), where the training using journey A gave a higher score for detection on journey D than the OCSVM classifier trained using journey D. Likewise, for both the Accelerator cluster and the Speed cluster , the CC achieved a higher $F_\beta(0.5)$ score for journey A using training data from other journeys (Tables 8.15 and 8.18).

Expanding the training data range by combining data from two journeys (A + E), produced mixed results. For the LOF, it resulted in a higher score when detecting from the non-included journey D (Tables 8.14 and 8.14), although for the OCSVM the performance fell in this situation (Tables 8.16 and 8.19).

Whilst the sample of results is small, and cannot be tested for statistical significance, it does highlight that training based on a particular journey cannot be assumed to produce similar detection results across other journeys. Moreover, increasing the range of training data by adding additional journeys, or choosing a longer journey, need not improve the classifier's effectiveness.

Table 8.14 LOF performance across journeys for the Accelerator cluster. Cells show False Positive, Average False Negative and $F_\beta(0.5)$.

		Journey used for Test/Attack Data	
		A	D
Journey used for Training Data	A	35 297 0.8465	423 275 0.6296
	B (A+E)	40 297 0.8393	412 200 0.6709
	C	257 288 0.7066	446 263 0.6292
	D	152 639 0.5415	57 562 0.6838

Table 8.15 CC performance across journeys for the Accelerator cluster. Cells show False Positive, Average False Negative and $F_\beta(0.5)$.

		Journey used for Test/Attack Data	
		A	D
Journey used for Training Data	A	195 555 0.5536	477 384 0.5539
	B (A+E)	193 457 0.6421	597 284 0.5653
	C	402 421 0.5726	330 524 0.5305
	D	505 255 0.6104	175 486 0.6067

Table 8.16 OCSVM performance across journeys for the Accelerator cluster. Cells show False Positive, Average False Negative and $F_\beta(0.5)$.

		Journey used for Test/Attack Data	
		A	D
Journey used for Training Data	A	17 458 0.7502	524 211 0.6219
	B (A+E)	18 465 0.7343	538 214 0.6147
	C	184 374 0.6985	157 417 0.6761
	D	496 195 0.6405	17 380 0.7891

Table 8.17 LOF performance across journeys for the Speed cluster. Cells show False Positive, Average False Negative and $F_\beta(0.5)$.

		Journey used for Test/Attack Data	
		A	D
Journey used for Training Data	A	56 208 0.8977	336 258 0.6924
	B (A+E)	58 212 0.8940	177 252 0.7910
	C	322 260 0.6970	404 298 0.6395
	D	700 138 0.5932	69 223 0.8801

Table 8.18 CC performance across journeys for the Speed cluster. Cells show False Positive, Average False Negative and $F_\beta(0.5)$.

		Journey used for Test/Attack Data	
		A	D
Journey used for Training Data	A	175 574 0.5792	295 466 0.5948
	B (A+E)	231 674 0.4701	335 533 0.5356
	C	426 282 0.6298	612 178 0.6027
	D	907 12 0.5756	465 58 0.7080

Table 8.19 OCSVM performance across journeys for the Speed cluster. Cells show False Positive, Average False Negative and $F_\beta(0.5)$.

		Journey used for Test/Attack Data	
		A	D
Journey used for Training Data	A	14 462 0.7772	34 435 0.7856
	B (A+E)	15 486 0.7482	23 465 0.7610
	C	98 457 0.7148	133 405 0.7227
	D	590 150 0.6241	355 216 0.6829

8.4 Method Overheads

As with the timing anomaly experiments discussed in Chapter 6, data on processing was collected to inform on the viability of the methods.

The maximum times taken by the trained classifiers to assess any of the 1000 snapshot test-runs are shown in Table 8.20. Added to these times should be the time needed to apply the scaling, which was found to be approximately 0.001 seconds per test run.

Table 8.20 Maximum time (seconds) for the trained classifiers to assess the 1000 snapshot test data sets.

	Car 1		Car 2	
	Accelerator	Speed	Cluster A	Cluster B
LOF	0.01199	0.02398	0.01799	0.020989
OCSVM	0.00301	0.00400	0.00400	0.004011
CC	2.02783	2.35264	3.31107	3.877810

The maximum time for the OCSVM and the LOF to process the 1000 record snapshots, suggests the potential for them to process data faster than the broadcast rate. For example, Car 1 showed a mean time between packet broadcasts of 0.0007 seconds, with a minimum observed gap of 0.00002, and broadcast at an average rate of 1432 packets per second. For Car 2 the mean time between packet broadcasts was 0.0006 seconds, with a minimum observed gap of 0.00001, and broadcast at an average rate of 1652 packets per second. However, classifier processing time does not include the time taken to update the snapshot record, and the same caveats presented in Chapter 6, for example the hardware used here compared with an in-vehicle implementation, apply to interpreting these results.

The methods would require the persistent storage of data for the scaling and for the decision surface. Using the Scikit-learn *joblib* functions [67] for the scaling information and the decision surface for both the LOF and the OCSVM, produced a scaler file of 1KB for each classifier, with a maximum 1850KB file for the LOF decision surface, and 12KB file for the OCSVM. For the CC the decision surface requires the storage of the coordinates and size of each hypersphere, resulting in a 1KB file in this experiment.

The training time would not impact the detection performance, since training would likely be done at system deployment and not repeated. Even so, times for training using the 6000 snapshot records were recorded. For the LOF training times for the complete 6000 record file ranged between 0.07 and 0.13 seconds, while for the OCSVM they were marginally

quicker, ranging between 0.05 and 0.08 seconds. Training for the CC took far longer. For most instances, the CC training took between 40 minutes and one hour, though almost three hours was required for one of the clusters. Training the CC required resource-intensive, nested, nearest neighbour operations. It was, though, coded entirely by myself, so is unlikely to have been tuned to its maximum programmable efficiency, whereas the LOF and OCSVM are implemented from dedicated, universally tested libraries.

8.5 Conclusion

The highest $F_\beta(0.5)$ scores overall, and averaged within the fields were obtained by the LOF, followed by the OCSVM; with the CC being least successful. However, the results showed that the OCSVM tended to generate the fewest false positives; though generated more false negatives compared with the LOF, especially at the higher magnitudes of data manipulation. Even at the lowest rates (less than 50 per 1000 records for the LOF), the number of false positives would be problematical. Likewise, the false negatives would imply the possibility of many missed attacks, especially at lower magnitudes of manipulation of the sensor data. Car 2 showed worst results compared with Car 1, especially with fabrications at smaller magnitudes.

Results from comparing journeys used for the training data showed, perhaps unsurprisingly, that training using data from one journey tended to produce worse results for detection across other journeys. However, this was shown to not always be the case. In addition, attempting to improve detection by providing a training set that uses composite journeys, or longer journeys, was not shown to be successful in this study. The implications for the results are discussed in Chapter 9.

Chapter 9

Discussion and Implications

This chapter discusses the implications of the results and considers the limitations of the research and how these might be resolved. The gaps in current CAN IDS research which this thesis has attempted to cover are summarised in Table 9.1. Section 9.1 discusses the timing anomaly detection, and Section 9.2 discusses the payload anomaly detection. Section 9.3 discusses factors that hold implications for both.

9.1 Timing Anomaly Detection

For the timing anomaly detection, the supervised method (comparing the mean broadcast time interval against a fixed threshold) was the simplest approach and gave the better detection scores with the five highest priority CAN IDs, giving complete accuracy with no false detection in both cars tested. On Car B, with CAN IDs that tended to be broadcast with persistent regularity, this method proved the most accurate, both for the highest priority IDs, and when all IDs were included in the detection. In Car A, however, in which lower priority CAN IDs showed more variation in their repeat broadcast times, the vulnerability of using a fixed threshold predefined according to other CAN IDs, became apparent, resulting in a large increase in false positives. The ROC curve of this method, however, was closest to the top-left of the chart, and suggests that improvements could have been made by adjusting the threshold (Figure 6.9).

The supervised method's testing undertaken here used a fixed threshold that was applied to all CAN IDs. An option for improved detection rates might be for individual thresholds to be determined for each packet, as suggested by the broadcast rate characteristics discussed in 5.1.1. The storage requirements for this would be minimal (at most one threshold value per CAN ID), although the process of capturing these would need considering. In the absence of a CAN dictionary for the car, determining the thresholds would present a challenge, but might

Table 9.1 Gaps in CAN IDS research addressed in this thesis.

Challenge	Consequential gap in CAN IDS proposals	Contribution from this thesis
CAN data dictionaries not published and specific to car models.	Methods proposed that require knowledge of CAN dictionary and specific to car: Studnia *et al.* [75], Ling and Feng [45], Song *et al.* [73], Gmiden *et al.* [28], Lee *et al.* [44], Marchetti and Stabili [47], Wasicek and Weimerskirch [89].	Timing anomaly detection and payload anomaly detection methods proposed that are agnostic to the car model and do not require the CAN dictionary specification.
Generating and maintaining attack signatures.	Requirement of maintaining attack signatures, or using attack data for training: Studnia *et al.* [75], Marchetti and Stabili [47], Martinelli *et al.* [49], Kang and Kang [38].	One class novelty detection methods that do not assume the availability of attack data or attack signatures.
Attacks might not affect both timing and payload.	Methods requiring timing and payload anomaly combinations: Taylor *et al.* [76].	Methods consider timing anomalies and payload anomalies independently.
CAN broadcast traffic difficult to predict across journeys or over the lifetime of the car.	Methods requiring immutable broadcast rates during detection lifetimes: Ling and Feng [45], Cho and Shin [17], Song *et al.* [73], Gmiden *et al.* [28], Marchetti and Stabili [47].	Window approach and detection methods that allows for packet rate variations across journeys, as well as controlling for naturally occurring trends and seasonality (ARIMA).
Evaluating how the detection will work with unpredicted attack models.	Payload anomaly testing has not investigated the potential range of the detection: Wasicek and Weimerskirch [89], Taylor *et al.* [77], Studnia *et al.* [75]	Testing using stratified payload manipulation to determine interactions between the magnitude of the manipulation and the response of the detection method.

be amenable to a simple analysis and training process. Although this method showed short computational times, it might be beneficial to reduce these further. Therefore, an additional extension of this research would be to determine if the packet broadcasts are consistent enough to merit timing anomaly detection across multiple journeys based purely on each broadcast interval, without reverting to mean calculations or window approaches.

ARIMA and Z-score did not quite match the highest scores achieved by the supervised method, but their scores did not fall as much when all packets for Car A were tested. Though ARIMA offers potential to counter effects of seasonality and natural trends, it is unclear from the data captured whether this would be required for the CAN broadcasts. Seasonal variation was not noticeable here, and has not been reported by other studies that have included analysis of CAN packet timings, (e.g. [52, 54, 76]). A study analysing behaviour long-term might determine whether it is observed; though, of course, not detecting it does not negate its potential existence.

In these tests, ARIMA scored slightly worse than the Z-score method, particularly by generating slightly higher numbers of false positives. Similar to the supervised method, detection accuracy was reduced when all CAN IDs were included in the analysis. Here ARIMA and Z-score showed reduced sensitivity, resulting in an increase in false negatives. There are though, parameters and factors that might be tweaked to optimize performance, so further research might improve performance in either method.

As with the supervised method, one of the cars (Car A) seems to have presented the greater problem for ARIMA and Z-score, with the lower priority CAN IDs reducing the sensitivity of detection, thus eliciting more false negatives. As shown in Figures 5.2, 5.4, 6.7 and 6.8, the variation in packet broadcast timings for individual CAN IDs are often slight, but not unimportant with regard to anomaly detection accuracy. Clearly this is an aspect that would need more investigation, with a detailed profiling of broadcast patterns across a range of cars. This is also an aspect that would benefit from consultation with automotive engineers, assuming manufacturers could be persuaded to divulge information about broadcast timings. However, even if such information were made available, Taylor *et al's* [76] observation that the behaviour of ECUs might not be fully known, and Checkoway *et al's* [14] reminder that many ECUs are programmed by suppliers rather than the car manufacturer, calls to question how much credence could be placed on it.

The research also highlights the importance of ensuring the optimal threshold or error rate, whatever method is adopted. In particular, the sensitivity of the detection methods falls rapidly if these are set too high, leading to missed detections; whereas specificity is reduced if these are low, leading to false positives. Whilst neither error is desirable, the preference of

one over the other needs deciding. As discussed in Chapter 4, for a CAN attack detection system false positives might be considered more disruptive than false negatives.

9.2 Payload Anomaly Detection

The results for the payload anomaly detection show that the detection by the Local Outlier Factor (LOF) and One Class Support Vector Machine (OCSVM) out-performed the Compound Classifier (CC) across all tests. The LOF achieved the fewer false negatives, and this was especially pronounced on some fields. However, it also achieved a slightly higher false positive rate on some of the tests compared with the OCSVM, so did not always achieve a better $F_\beta(0.5)$ score.

As with the timing anomaly detection, for payload anomaly detection the reduction of false positives would likely be a high priority since these would present a distraction to the driver, or prompt unnecessary ameliorative action. Given that CAN broadcast rates typically exceed 1000 per second, the number of false positives per second achieved (Tables 6.3 and 6.4) would clearly be problematical.

9.2.1 Payload Clustering

The clustering algorithm identified clusters of probable sensor data fields which tallied with clustering the data using manual data analysis and visual inspection. There are though, limitations of this approach, which are discussed in the Conclusion. Even so, clustering the fields using correlations, as demonstrated here, is an aspect that might be automated for system development without needing the underlying data dictionary for the car. In addition, a training period comprising a few predefined, and isolated, driving tasks could be staged to determine likely functional associations, should this be deemed useful.

9.2.2 Payload Profile Variations

One potential problem highlighted by including the second car was the different approaches that seem to be taken in deriving sensor reading values, as observed in the differences in the spread and range of the data. This makes direct comparison of results difficult. Car 1 showed many data ranges with the origin starting at zero (as shown in the Figures in Appendix C), whereas the data captured from Car 2 had most sensor fields with the origin starting at a high number, often in the thousands, with a narrow range of values. As was highlighted from initial runs of the classifier experiments, the first data manipulation method adopted (multiplication of the legitimate value by a specific percent), resulted in simulated attack

values that are far beyond the observed legitimate range for Car 2. Compare, for example, 100*1.5 versus 10100*1.5. Thus the results from the initial testing of Car 2, showed detection rates that were high and similar for some fields and multiplier manipulations; whereas the the results from generating attacks by manipulating scaled sensor data resulted in reduced detection, but more uniform scores across fields.

Here lies a dilemma for the experiment and evaluation: whether it should be assumed that the attacker or their method, would cause the value to change by an amount related to the absolute value, or whether the value would be altered by an amount relative to the range or information the field represents. Perhaps, depending on the goals, or the configuration of attack equipment, or the *modus operandi*, either is possible. Miller and Valasek [51–53], for example, conducted extensive analysis and reverse engineering to formulate their attacks and were able to manipulate data (such as speed) to precise, known values. More naive attacks, opportunist attacks, or attacks employing fuzzing, might not consider such precision or require knowledge of the data derivation.

9.2.3 Attack Data Ranges

Another aspect that the data simulation methods did not consider are the constraints placed on payload data in the CAN. In an actual vehicle the data field values are constrained by the packet size, so some values generated by the attack manipulation methods used here might not be permittable on the CAN. Also, ECUs are programmed to act on values within a permitted range, thus some of the values generated in the experiment would be deemed meaningless and be ignored. However, accounting for this would add complexity to both the test implementation, and to the interpretation of the results; and probably would not alter the conclusions drawn. Moreover, implementing it would imply detailed knowledge of the field-value derivations, which, as discussed earlier, is not a constraint intended for a generic anomaly detection system.

9.3 Common Implications

This section discusses factors that need considering for both tests. These concern common factors that might improve detection, and assessing the performance for in-vehicle deployment.

9.3.1 Improving The Detection

The results presented in Chapters 6 and 8 show that although detection of both timing anomalies and payload anomalies was achieved, the rates of false positives and false negatives were high and might result in multiple incorrect detections per second. There are a few options that might be considered for potential detection improvement, some of which were provisionally explored in the experimentation.

Refinements might be made to the method parameters. The parameters for both the timing and payload detection were initially tested and optimised using data from single journeys. The results were then taken to set the parameters for subsequent trials. Perhaps other journeys would have produced different optimal parameter values. However, given that both cars in the timing experiment provided optimal parameter scores that were identical (see Section 6.1.1), it is possible that parameter values chosen could not be improved. For the OCSVM used for the payload anomaly detection, there are alternative kernels that might be tested; although preliminary testing of these during development, together with the published recommendations for normally distributed data (e.g. [15, 67] suggested that the radial basis function (RBF) kernel used here is probably the most suitable for the CAN data set.

Improved scores might also be gained from making refinements to the process configuration. An example would be raising the correlation thresholds for determining the clusters in the payload detection. However, reducing the number of data fields in a cluster would lessen the breadth of coverage, which holds implications for both coverage across the car's full CAN set, and for the capture within each cluster. This would need exploring.

Although a window approach was used in the timing anomaly detection to facilitate ARIMA processing and mitigate any effects from natural trend or seasonality in the broadcast intervals, the necessity for this is unproven. As suggested in this study, a simple method of comparing successive packet intervals against the mean might suffice, provided sufficient uniformity of packet broadcast intervals can be proven. However, the implication from the poorer performance when the fixed parameters were applied across all CAN IDs, would be that gap-mean and variation thresholds might need to be established for each CAN ID, as discussed above. In a non-window approach, these might be calculated by including multiple journeys and multiple driving conditions to ensure sufficient natural variation is captured to reduce the likelihood of false positives.

For the payload anomaly detection, similar improvements might also be suggested for training the classifiers. However, the results from comparing different training sets (Section 8.3) show the complexities of this, and that simply considering more journeys in the training set need not improve the detection across journeys. Improvements might also be made

by adjusting the training data-set size, although determining the optimal size to enable generalisability without over-training, is an aspect that would require additional research.

9.3.2 Assessing In-Vehicle Performance

Additional work would be required to test these methods on a device suitable for implementing within the vehicle. Basic process times were recorded during the testing, and the performance of the algorithms observed here might inform the expectations of a production deployment, but it should be born in mind that the code was written to facilitate testing analysis, rather than for processing efficiency, and the architecture was different to that which might be expected in an in-vehicle deployment.

The timing detection was tested using R, while the payload anomaly detection was implemented in Python. These choices were made for reasons of library availability, integration and project collaboration, rather than a specific inherent requirement, though both software products could be run on micro controllers or board computers. Since this research was carried out, Auto ARIMA has been released for Python [72], and Python also offers a CAN library to facilitate packet reading and processing, [82], which might be useful. The implementation of a system in a mode suitable for in-vehicle running needs investigation. For the payload anomaly detection, some processes, such as identifying the data field profiles and clusters, and the classifier training, could be resource intensive and might not be be done *in situ*. Data capture and testing would be far simpler. The classification process would need the storing of the trained classifier and the most recent values for the included data fields, though this would be a low overhead. For the payload anomaly detection running in Python on a reasonably powerful laptop, the detection time for the CC was longer than for the LOF and OCSVM, stretching to two seconds or more for 1000 records. In comparison, the detection time was only fractions of seconds for the LOF and OCSVM, suggesting the potential to cope with the rate of packet broadcasts on the automotive CAN.

Chapter 10

Conclusion and Future Work

Having considered the results, their implications and possible improvements to the detection methods in the previous chapters, this final chapter considers the general conclusions that can be drawn from the research presented here. In particular, it assesses the extent to which the hypothesis and research questions have been addressed (Section 10.1), outlines the evidence for the original contributions (Section 10.2), and discusses where future research might focus (Section 10.3).

10.1 Evaluating the hypothesis and research questions

The underlying hypothesis for this research was that characteristics might be present in CAN packet traffic that allow CAN network attacks to be detected through anomalies. This raised research questions as to whether such characteristics could be identified; what detection methods might be adopted to detect such anomalies, and how the detection methods could be tested and validated. However, to progress these research questions, there are some important constraints that must be confronted. The lack of CAN dictionary sharing by manufacturers, together with the variety and legacy of car models, make it difficult to propose a system that could use knowledge of ECU specifications. Automotive cybersecurity is a nascent field and the nature of future attacks is difficult to predict. Finally, obtaining sample attack data is difficult due to many problems in staging example attacks.

The attack detection approach adopted was to model characteristics that have been shown to be altered across a broad range of attacks, rather than attempting to produce test data that represented a single, but verified, attack on a car. While the latter would be an important test for proof that the detection system could detect a real attack, the complexities of generating such attacks, and the difficulty in comparing results across cars and attack variations, suggested the approach taken here. For the payload anomaly detection, the test

model allowed the testing of the characteristics across simulated attack manipulations of varying magnitudes, enabling a comprehensive picture of detection capabilities. Such range-testing could also be extended to the timing manipulations, though this would require some adapting of the testing algorithms.

The experimental approaches taken provide an indication of the magnitude of changes caused by an attack that might be detected, and the detection methods are not constrained by having to understand the data derivation and meaning, thus are suitable for inclusion in detection systems that assume no knowledge of the car's CAN dictionary. Also, they provide a generic approach which is suitable for a range of attacks, including attacks that might not yet be envisaged.

The timing anomaly detection used time-defined window methods to detect changes in CAN activity resulting from attacks such as injection and reflash attacks. The methods compare each broadcast interval for CAN packets within a window against the averages for all packets with the same ID within that window. A simple comparison of the window-mean packet-interval against each time gap, showed high success in some situations; but also was shown to be more vulnerable to higher rates of error in other situations compared to ARIMA and Z-score methods. Whilst the mean comparison can give superior results, those are dependent on it being configured for the target system. Unsupervised approaches tested here (ARIMA and Z-score) offer solutions that are potentially less dependent on an analysis of the specific CAN.

The detection of anomalies in payload values was considered a two stage process, each stage coping with the constraint of not knowing the CAN dictionary for the cars. The first stage was the identification of correlated sensor fields, including those that might be composite amalgamations of packet fields. The algorithms developed were able to identify composite sensor fields and group these into clusters that matched detailed manual inspections of the data and functional analyses of the car operations. Testing, however, did not extend to investigate different correlation cutoffs for deciding the cluster members, which would impact subsequent detection rates. As mentioned in Chapter 7, identification of clusters does not, of itself, give any indication as to what functionality might be relevant to the cluster. Whilst knowing this is not vital to the subsequent detection process, it would be useful from a security perspective, helping to identify which clusters might be more prone to be attacked in certain situations. Such knowledge might be gained from performing a staged set of independent operations on the car (for example, pressing the accelerator with the car in neutral), perhaps extending this into a full analysis using a supervised clustering approach such as random forest classification.

Three field-value anomaly detection methods were chosen because of their ability to cope with one-class classification, i.e. detection of anomalies which are not represented in the training data sets. Of these methods, the One Class Support Vector Machine (OCSVM) and Local Outlier Factor (LOF) showed better detection and faster performance than the Compound Classifier (CC).

There are limitations to the study and the methods that need acknowledging. The payload detection process assessed here would only include a subset of the data cohort – the two clusters selected for testing comprised fewer than half of the sensor fields determined. Even so, it is clear from Figures 7.3 and 7.4 that a few other smaller clusters might also be considered. In addition, data for this experiment did not consider fields that might hold category data or other representations. These might also be fabricated in an attack.

The use of ranges starting at a high value presented a dilemma for attack simulation, due to the potentially increased effect on data field manipulation, as discussed in Section 9.2.2. There are some options that might allow the attack simulation to be more independent of the data origin, potentially facilitating test comparison across cars. The first option would be to not simulate attacks, but use actual, understood values instead. This would ensure the tested attack fields comprise properly proportioned values. However, the problems of staging attacks, or of discovering the CAN dictionary, hinder this approach. Moreover, such knowledge would be needed for all cars tested. Another option for testing (as demonstrated here) is to generate the attack data on the scaled data. This ensures all cars are tested with data having similar ranges. Potentially the standardisation process used in the experiments staged here, which centres the normalised data around 0, could leave some naturally occurring values standardised to 0 if their value coincides with the mean, thus they would not be changed if the attack-by-multiplying-the-original-value method is used. However, examining the data sets showed no records that held 0 values post standardisation.

For both strands of detection, the accuracy was below what would be acceptable in a production classifier. The levels of false positives (normal snapshots classified as attacks) would trigger a few unwarranted attack warnings per second. In addition, attacks on some CAN packets were poorly detected. It is probable that accuracy would be improved by adjusting parameters and conducting transformations on the data; though, of course, this would add to the complexity of the process. Intuitively we might suppose that clustering the CAN IDs into smaller, more rigorously chosen cohorts of higher correlation might improve the payload anomaly detection. But this would be at the cost of monitoring fewer IDs within each decision.

10.2 Summary of Contributions

This research devised, demonstrated and evaluated methods that might continue to be adapted and used in future research into CAN attack detection. Summarised here is the evidence for the original contributions presented in Section 1.4.

- *Methods for detecting cyber-attacks of the CAN*: This thesis evaluated methods new to this context, as discussed in Chapter 4. These focused on one-class training and detection, which is suitable where attack data is not available. This is appropriate for CAN attack detection systems because of factors such as: difficulties in generating attack data; the diversity of CAN dictionaries stemming from the vast range of car makes and models, and the nascent nature of automotive cyber-security and attack modelling. Methods were proposed and tested for timing anomaly detection and payload anomaly detection. The testing demonstrated the abilities of these to detect attack-representative data, and showed situations where the detection might be less successful.

- *Processes for transforming the CAN packet flow into structures suitable for analysis and detection*: In addition to testing the anomaly detection methods discussed above, the detection processes required transforming the packet flow into structures suitable for anomaly detection. A challenge here is the secretive nature of the CAN dictionary, meaning that characteristics, such as packet timings or the meaning of data values, are not disclosed by the manufacturers. The timing detection was tackled by using time-bound windows, within which analysis of each CAN ID's packet broadcasts could be compared (detailed in Chapter 5). For the payload anomaly detection, a process for automating the discovery of sensor fields was devised, including detecting fields that comprised concatenated data (detailed in Chapter 7). A method for clustering these fields into associated groups was also included. Like the anomaly detection methods, these were evaluated on cars of different makes, providing an gauge of validity.

- *Attack models assimilated according to CAN traffic implications*: An extensive review of CAN attack research was undertaken (Chapter 2), which informed the choice of attack data simulation. Whilst actual data from specific attacks would verify that the methods could capture actual attacks, which would certainly be a useful test, it would provide only a validation that would be restricted to a specific attack on a specific model of car. The approach adopted here was therefore to use data that might offer an indication of the detection capabilities across categories of attack, and across vehicles.

- *Empirical testing of attack detection considered as a stage to be automated, facilitating comprehensive testing*: The testing of the payload attacks has been staged in a manner that facilitates testing manipulations to individual data fields. The manipulations were performed across stepped magnitudes, providing a gauge as to the performance of the detection at different manipulation intensities (Chapter 8). This also facilitated the testing of different attack simulations, and training deployments. Similarly, the testing of the timing anomaly detection methods provided exploration of different window sizes and method parameters (Chapter 6).

10.3 Further Work

This thesis has determined processes and methods that might be adapted for CAN attack detection. As highlighted in Chapter 9, there is still much that needs exploring; in particular to improve the detection accuracy and validate the methods against representative attacks. Possible improvements and refinements of the individual methods and processes have been discussed previously, such as in Chapter 9. Discussed in this section are the more general suggestions for future work, which would be required regardless of the detection method tested.

- *Profiling and analysis of vehicles*: Profiling and analysis of CAN data across car models is needed to establish the full pattern of packet timings and payloads. In particular, establishing how consistent packet timings are, and whether they show any variation, trend or seasonality. Likewise, analysing the patterns of payload data adopted by the manufacturers. Such profiling would be useful across cars, but also across multiple journeys, scenarios, drivers, and driving conditions.

- *Acquiring test data*: Perhaps the biggest challenge still to be overcome is the production of valid test data. The problems of acquiring such data have been discussed a few times within the thesis. There exists, as yet, no standard set of labelled data akin to the iris classification data set, or similar, often used in machine learning demonstrations. The attack simulations generated in this thesis cannot embrace full CAN functionality, such as by replicating CAN arbitration and error confinement. In addition, the lack of any published CAN dictionaries compounds the problems of data understanding and interpretation. Including attacks in CAN simulators might provide a useful compromise, though the output will always be a simplification, run on a platform that is different to an actual vehicle.

- *Improving the detection accuracy*: There is much that could be explored to try to improve the detection accuracy. This would be especially warranted given the safety critical nature of driving, and the implications of detection errors causing incorrect response or distraction. The payload anomaly detection, for example, showed the challenges of detecting across journeys, and highlighted that incorporating additional journeys or longer journeys into the training data does not necessarily improve general detection performance. The size of the training sets also needs investigation, for example, to avoid under- and over-training. It is possible, though, that no single approach will suffice, and that multiple approaches, embracing many of the methods outlined above, will be needed. Thus multiple classifiers would form a consensus-ensemble of tests assessing a range of CAN properties.

References

[1] Alsmadi, I. and Xu, D. (2015). Security of Software Defined Networks: A survey. *Computers and Security*, 53:79–108.

[2] Batchelor, B. G. (1974). *Practical Approach to Pattern Classification*. Plenum Press.

[3] Batchelor, B. G. (1978). Classification and Data Analysis in Vector Spaces. In Batchelor, B. G., editor, *Pattern Recognition: Ideas in Practice*, chapter 4, pages 65–116. Springer US.

[4] Batchelor, B. G. (2012). Colour Recognition. In Batchelor, B. G., editor, *Machine Vision Handbook*, chapter 16, pages 665–694. Springer-Verlag, London.

[5] Bhattacharyya, D. and Kalita, J. (2013). *Network Anomaly Detection*. CRC Press.

[6] Bonaccorso, G. (2018). *Machine Learning Algorithms*. Packt Publishing, Birminghan, UK, 2nd edition.

[7] Bozdal, M. and Jennions, I. (2018). A Survey on CAN Bus Protocol: Attacks , Challenges , and Potential Solutions. In *IEEE International Conference on Computing, Electronics & Communications Engineering (iCCECE '18)*, University of Essex, Southend, UK.

[8] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). LOF: Identifying Density-Based Local Outliers. *Proceedings of the 2000 Acm Sigmod International Conference on Management of Data*, pages 1–12.

[9] Buczak, A. and Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, PP(99):1.

[10] CAN in Action (CiA) (2017). CAN data link layers in some detail. https://www.can-cia.org/can-knowledge/can/can-data-link-layers/.

[11] Carsten, P., Andel, T. R., Yampolskiy, M., and McDonald, J. T. (2015). In-Vehicle Networks: Attacks, Vulnerabilities, and Proposed Solutions. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*.

[12] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection. *ACM Computing Surveys*, 41(3):1–58.

[13] Cheah, M., Shaikh, S. A., Haas, O., and Ruddle, A. (2017). Towards a systematic security evaluation of the automotive Bluetooth interface. *Vehicular Communications*, 9:8–18.

[14] Checkoway, S., Mccoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., and Kohno, T. (2011). Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *20th USENIX Security Symposium*, volume August, pages 77–92, San Francisco. The USENIX Association.

[15] Chio, C. and Freeman, D. (2018). *Machine Learning and Security*. O'Reilly Media Inc., Sebastopol, CA, first edition.

[16] Cho, K.-T., Kim, Y., and Shin, K. G. (2018). Who Killed My Parked Car? http://arxiv.org/abs/1801.07741.

[17] Cho, K.-T. and Shin, K. G. (2016). Fingerprinting Electronic Control Units for Vehicle Intrusion Detection. In Holz, T. and Savage, S., editors, *Proceedings of the 25th USENIX Security Symposium*, pages 911–927, Austin, TX. USENIX Association.

[18] Cisco (2019). Snort. https://snort.org/.

[19] Di Natale, M., Zeng, H., Giusto, P., and Ghosal, A. (2012). *Understanding and Using the Controller Area Network Communication Protocol*. Springer, New York.

[20] Ericson, G. and Rohm, W. A. (2017). How to choose algorithms for Microsoft Azure Machine Learning. https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice.

[21] European Automobile Manufacturers Association (2019). The Automobile Industry Pocket Guide 2018-2019. Technical report, European Automobile Manufacturers Association. www.acea.be.

[22] Fawcett, T. (2003). ROC Graphs: Notes and Practical Considerations for Data Mining Researchers ROC Graphs : Notes and Practical Considerations for Data Mining Researchers. Technical report, Hewlett-Packard Company, Palo Alto.

[23] Foster, I. and Koscher, K. (2015). Exploring Controller Area Networks. *USENIX ;login:*, pages 6–11.

[24] Foster, I., Prudhomme, A., Koscher, K., and Savage, S. (2015). Fast and Vulnerable: A Story of Telematic Failures. *9th USENIX Workshop on Offensive Technologies (WOOT 15)*.

[25] Fröschle, S. and Stühring, A. (2017). Analyzing the capabilities of the CAN Attacker. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10492 LNCS:464–482.

[26] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1):18–28.

[27] Geron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media Inc., Sebastopol, CA.

[28] Gmiden, M., Gmiden, M. H., and Trabelsi, H. (2016). An intrusion detection method for securing in-vehicle CAN bus. *2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, pages 176–180.

[29] Goldstein, M., Goldstein, M., and Uchida, S. (2016). A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. *PLoS ONE*, (April):1–31.

[30] Groza, B. and Murvay, P. S. (2019). Efficient Intrusion Detection with Bloom Filtering in Controller Area Networks. *IEEE Transactions on Information Forensics and Security*, 14(4):1037–1051.

[31] Hoppe, T., Kiltz, S., and Dittmann, J. (2009). Applying Intrusion Detection to Automotive IT – Early Insights and Remaining Challenges. *Journal of Information Assurance and Security*, 4(May):226–235.

[32] Hoppe, T., Kiltz, S., and Dittmann, J. (2011). Security threats to automotive CAN networksPractical examples and selected short-term countermeasures. *Reliability Engineering and System Safety*, 96(1):11–25.

[33] Hyndman, R., Ihaka, R., Reid, D., and Shaub, D. (2019). Package ' forecast ': Forecasting Functions for Time Series and Linear Models. Technical report. https://cloud.r-project.org/web/packages/forecast/forecast.pdf.

[34] Hyndman, R. J. and Khandakar, Y. (2008). Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software*, 27(3):22.

[35] Ibrahim, D. (2011). *Controller Area Network Projects*. Elektor International Media.

[36] International Organization for Standardization (2015). ISO 11898-1:2015 Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling. Technical report. https://www.iso.org/standard/63648.html.

[37] Kalutarage, H. K., Shaikh, S. A., Wickramasinghe, I. P., Zhou, Q., and James, A. E. (2015). Detecting stealthy attacks: Efficient monitoring of suspicious activities on computer networks. *Computers and Electrical Engineering*, 47:327–344.

[38] Kang, M.-J. and Kang, J.-W. (2016). Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. *PLoS ONE*, 11(6).

[39] Khan, S. S. and Madden, M. G. (2010). A survey of recent trends in one class classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6206 LNAI:188–197.

[40] Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., Mccoy, D., Kantor, B., Anderson, D., Shacham, H., and Savage, S. (2010). Experimental Security Analysis of a Modern Automobile. In *IEEE Symposium on Security and Privacy*, pages 1–16, Oakland, CA.

[41] Kriesel, D. (2007). *A Brief Introduction to Neural Networks*. http://www.dkriesel.com.

[42] Kyaw, A. K., Chen, Y., and Joseph, J. (2016). Pi-IDS: Evaluation of open-source intrusion detection systems on Raspberry Pi 2. *2015 2nd International Conference on Information Security and Cyber Forensics, InfoSec 2015*, pages 165–170.

[43] Lee, H., Choi, K., Chung, K., Kim, J., and Yim, K. (2015). Fuzzing CAN packets into automobiles. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2015-April:817–821.

[44] Lee, H., Jeong, S. H., and Kim, H. K. (2017). OTIDS : A Novel Intrusion Detection System for In-vehicle Network by using Remote Frame. *Privacy, Security, and Trust 2017*.

[45] Ling, C. and Feng, D. (2012). An Algorithm for Detection of Malicious Messages on CAN Buses. In *National Conference on Information Technology and Computer Science*. Atlantis Press.

[46] Maglaras, L. A. (2015). A Novel Distributed Intrusion Detection System for Vehicular Ad Hoc Networks. *International Journal of Advanced Computer Science and Applications*, 6(4):1–6.

[47] Marchetti, M. and Stabili, D. (2017). Anomaly detection of CAN bus messages through analysis of ID sequences. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):1577–1583.

[48] Markovitz, M. and Wool, A. (2017). Field classification, modeling and anomaly detection in unknown CAN bus networks. *Vehicular Communications*, 9:43–52.

[49] Martinelli, F., Mercaldo, F., Nardone, V., and Santone, A. (2017). Car Hacking Identification through Fuzzy Logic Algorithms. In *IEEE International Conference on Fuzzy Systems*, Naples.

[50] Menendez, A. and Paillet, G. (2008). Fish inspection system using a parallel neural network chip and the image knowledge builder application. *Ai Magazine*, 29(1):21–28.

[51] Miller, C. and Valasek, C. (2014). A Survey of Remote Automotive Attack Surfaces. Technical report. http://illmatics.com/remoteattacksurfaces.pdf.

[52] Miller, C. and Valasek, C. (2016). CAN Message Injection: OG Dynamite Edition. Technical report. http://illmatics.com/canmessageinjection.pdf.

[53] Miller, C. and Valasek, C. (2018). Securing Self-Driving Cars - one company at a time. Technical Report August. http://illmatics.com/securing_self_driving_cars.pdf.

[54] Moore, M. R., Bridges, R. A., Combs, F. L., Starr, M. S., and Prowell, S. J. (2017). Modeling inter-signal arrival times for accurate detection of CAN bus signal injection attacks. *Proceedings of the 12th Annual Conference on Cyber and Information Security Research - CISRC '17*, (April):1–4.

[55] Murali, A. and Rao, M. (2005). A survey on Intrusion Detection Approaches. In *International Conference on Information and Communication Technologies: 2005*, pages 233–240, Karachi. IEEE.

[56] Muter, M. and Asaj, N. (2011). Entropy-based anomaly detection for in-vehicle networks. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):1110–1115.

[57] Müter, M., Groll, A., and Freiling, F. C. (2010). A structured approach to anomaly detection for in-vehicle networks. *2010 6th International Conference on Information Assurance and Security, IAS 2010*, pages 92–98.

[58] Naim, M. M., Chan, J., and Huneiti, A. M. (1994). Quantifying the learning curve of a vision inspection system. In *IEE Conference Publication*, number 398, pages 3–5.

[59] Narayanan, S. N., Mittal, S., and Joshi, A. (2016). OBD_SecureAlert: An Anomaly Detection System for Vehicles. *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6.

[60] Omar, S., Ngadi, A., and Jebur, H. H. (2013). Machine Learning Techniques for Anomaly Detection: An Overview. *International Journal of Computer Applications*, 79(2):975–8887.

[61] Patcha, A. and Park, J. M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470.

[62] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn Machine Learning in Python. *Journal ofMachine Learning Research*, 12:2825–2830.

[63] PyData.org (2018). Pandas 0.23.4 documentation. https://pandas.pydata.org/pandas-docs/stable/index.html.

[64] Sagong, S. U., Ying, X., Clark, A., Bushnell, L., and Poovendran, R. (2018). Cloaking the Clock: Emulating Clock Skew in Controller Area Networks. *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018*, pages 32–42.

[65] Scharring, K., Nash, S., and Wong, D. (2017). Connected and Autonomous Vehicles: Position Paper. Technical Report February, The Society of Motor Manufacturers and Traders Limited, London. www.smmt.co.uk.

[66] Schulze, S., Pukall, M., Saake, G., and Hoppe, T. (2009). On the need of data management in automotive systems. *BTW*, 144(C):217–226. http://wwwiti.cs.uni-magdeburg.de/~sanschul/papers/BTW2009autodama_final.pdf.

[67] Scikit-learn (2018). Scikit-learn web site. https://scikit-learn.org/stable/.

[68] SciPy.org (2018). SciPy 1.1.0 Reference. https://docs.scipy.org/doc/scipy/reference/index.html.

[69] Seclist.org (2015). Snort mailing list archives. https://seclists.org/snort/2015/q3/117.

[70] Shaikh, S. A. and Kalutarage, H. K. (2016). Effective network security monitoring: from attribution to target-centric monitoring. *Telecommunication Systems*, 62(1):167–178.

[71] Singh, J. and Nene, M. J. (2013). A Survey on Machine Learning Techniques for Intrusion Detection Systems. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(11):4349–4355.

[72] Smith, T. (2019). pmdarima: ARIMA estimators for Python. Technical report. https://www.alkaline-ml.com/pmdarima/.

[73] Song, H. M., Kim, H. R., and Kim, H. K. (2016). Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. *International Conference on Information Networking*, 2016-March:63–68.

[74] Streiner, D. L. and Cairney, J. (2007). What's Under the ROC? An Introduction to Receiver Operating Operating Characteristics Curves. *Canadian Journal of Psychiatry*, 52(2):121–128.

[75] Studnia, I., Alata, E., Nicomette, V., Ka, M., Studnia, I., Alata, E., Nicomette, V., Ka^, M., and A, Y. L. (2015). A language-based intrusion detection approach for automotive embedded network. In IEEE, editor, *21st IEEE Pacific Rim International Symposium on Dependable Computing*, Zhangjiajie, China.

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

[77] Taylor, A., Leblanc, S., and Japkowicz, N. (2016). Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks. *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 130–139.

[78] Tencent Keen Security Lab (2018). Experimental Security Assessment of BMW Cars: A Summary Report. Technical report. https://keenlab.tencent.com/en/.

[79] The Institution of Engineering and Technology (IET) and The Knowledge Transfer Network (KTN) (2015). Automotive Cyber Security: An IET/KTN Thought Leadership Review of risk perspectives for connected vehicles. Technical report. https://www.theiet.org/media/2309/iet-automotive-cyber-security-tlr-lr-1.pdf.

[80] The Open Information Security Foundation (2019). Suricata. https://suricata-ids.org/.

[81] Theissler, A. and Dear, I. (2012). Detecting anomalies in recordings from test drives based on a training set of normal instances. In *Proceedings of the IADIS International Conference Intelligent Systems and Agents and European Conference on Data Mining 2012*, number July, pages 124–132. IADIS Press.

[82] Thorne, B. (2019). Python-CAN: Controller Area Network interface module for Python. https://pypi.org/project/python-can/.

[83] Tuohy, S., Glavin, M., Hughes, C., Jones, E., Trivedi, M., and Kilmartin, L. (2015). Intra-Vehicle Networks : A Review. *IEEE Transactions on Intelligent Transport Systems*, 16(2):534–545.

[84] U.S. Department of Transportation / National Highway Traffic Safety Administration (2016). Federal Automated Vehicles Policy. Technical Report September, U.S. Department of Transportation.

[85] Valasek, C. and Miller, C. (2013). Adventures in Automotive Networks and Control Units. *Technical White Paper*, page 99. https://ioactive.com/resources/library/page/2/.

[86] Vector Informatik GmbH (2016a). Introduction to CAN. https://www.vector.com/gb/en-gb/know-how/technologies/networks/can/.

[87] Vector Informatik GmbH (2016b). Technical Papers on the Development of Embedded Electronics. Technical report.

[88] Vector Informatik GmbH (2018). CANoe: Product Information (V6.0.04/2018). Technical report.

[89] Wasicek, A. and Weimerskirch, A. (2015). Recognizing Manipulated Electronic Control Units. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, pages 1–8.

[90] Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 3rd edition.

[91] Woo, S., Jo, H. J., and Lee, D. H. (2015). A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):993–1006.

Appendix A

Compound Classifier

The Compound Classifier was devised by Batchelor [3, 4], and uses Euclidean distance and nearest neighbour analysis. It requires that data is numeric, and normalised to give each dimension parity.

Decision Making by Compound Classifier

Distance between instances can be used as an indication of their similarity. The Euclidean distance D between any two vectors $U = (U_1, \ldots, U_q)$ and $V = (V_1, \ldots, V_q)$ can be calculated as:

$$D(U,V) = \sqrt{\sum_{j=1}^{q} (U_j - V_j)^2} \tag{A.1}$$

Although distance-based nearest neighbour analysis might be used for deciding whether an instance falls into either of two classes, it can not be used in its pure form for one-class determination, and might be prone to over-fitting, where the classifier follows the training set boundary too tightly, losing the capacity to generalise to the population. Also, there are considerable overheads in searching the full dataset when making each prediction. Therefore the Compound Classifier reduces the decision surface into a set of circles (for a vector having two attributes), spheres (three attributes) or hyperspheres (four or more attributes), which between them cover the instances in the training set[1]. Decision making compares the coordinates of the target instance against the centre coordinates and radius of each sphere (Figure A.1). If the target instance falls within any sphere, it is classed as normal. If it falls outside all spheres, it is classed as an anomaly. Thus, the decision of the classifier is given

[1]The term "sphere" is used hence forth for convenience. Clearly the shape will depend on the dimensionality of the vector.

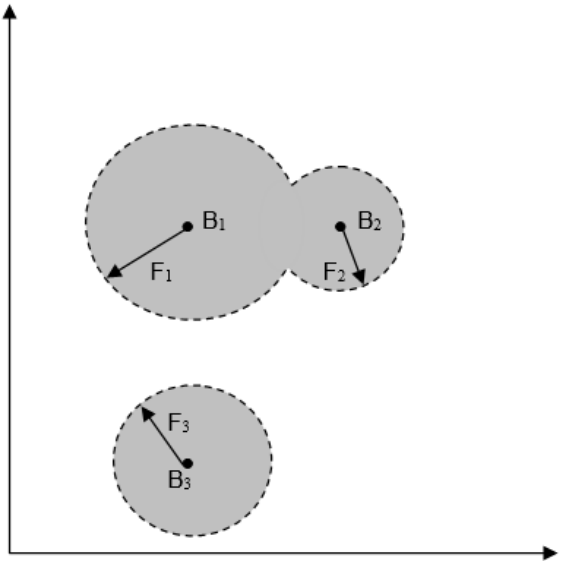


Fig. A.1 Decision surface of a Compound Classifier comprising three circles. B_i is the centre a circle and its radius is given by F_i. Normal instances are assumed to lie inside the classifier, anomalies lie outside. (Adapted from Batchelor [3].)

by:

$$M = sign\{max_{i=1...N}[F_i - D_2(B_i, X)]\} \tag{A.2}$$

Here N is the size of the set of spheres, F_{i} is the radius of the ith sphere and B_{i} is the centre of the ith sphere, $D_2()$ is the Euclidean distance as defined in (1), X is the target vector we wish to classify. X is determined as a normal class member if M is positive or zero, and an anomaly if M is negative.

Training the Compound Classifier

Training the Compound Classifier uses the above equations, and involves adjusting the size and location of each sphere. The training set is processed for a few iterations, with the training data shuffled between each to reduce the risk of systematic biases within the time sequence of the data. During training, for each instance X:

i) If M is negative, the nearest sphere is moved towards X and is made larger. All other spheres are made smaller.

ii) If M is positive, X lies within one or more spheres. Those spheres are moved towards X and made smaller. All other spheres are made smaller [4].

The values for the sphere movement and the size-change are kept small to enhance the accuracy of the classifier, and are suggested in [3, 4], together with the desired numbers of

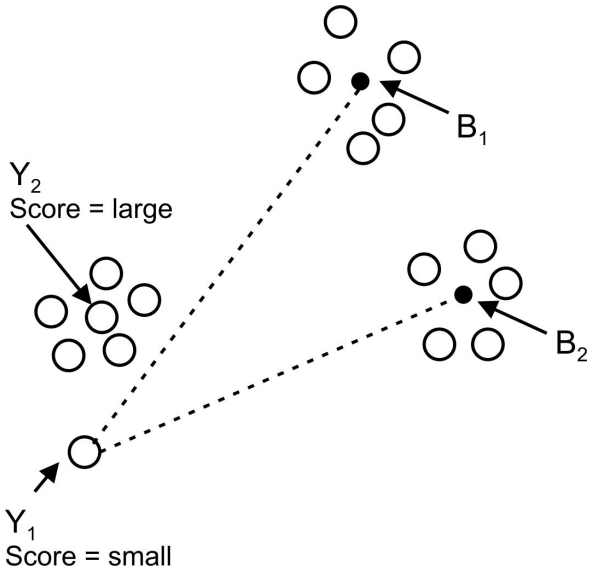


Fig. A.2 Maximindist: vector Y1 is furthest from sphere centres B1 and B2, so could be considered as the new sphere centre. However, scoring candidates by density estimate, as well as existing sphere-centre distance, favours Y2 as the new sphere centre.

training examples, training iterations, and initial spheres. Batchelor also devised processes for adding and pruning spheres to give efficient coverage of the training data set. As more spheres are added, the classification space could become messy, so methods are proposed to remove spheres nested inside other spheres, and divide large spheres that reduce the classifiers compactness and discernibility.

The parameters recommended by Batchelor [3] for data scaled between 0 and 100 are:

- Iterations between hypersphere addition: repeat until the equivalent of 10,000 vectors have been processed.
- Sphere size adjustment: expand by 0.01**nd*; shrink by (0.01**nd*)/*nh*, where *nd* is the number of dimensions and *nh* is the number of hyperspheres
- Sphere movement: 0.005
- Initial number of locates: *ni* / (10*(*nd*+1)), where *ni* is the number of data items and *nd* is the number of dimensions .

At the start of training, the initial sphere centres could use the location of any candidate instances. However, randomly assigned centres might not be optimal starting points. An alternative is to choose a subset of instances that are furthest apart, ensuring the initial spheres are spread throughout the data space. However, this might favour lone outliers, which again

might be suboptimal. Therefore, following analyses conducted by [58], the Compound Classifier adopts "Maximindist", which combines nearest neighbour measurements with probability density function estimates, to locate the initial sphere centres close to cluster centres (Figure A.2). The probability density is estimated as:

$$\frac{1}{{D_{NN}}^2} \tag{A.3}$$

Here D_{NN} is the distance to the nearest neighbour of the instance.

Appendix B

Timing Anomaly Detection Results for Car B

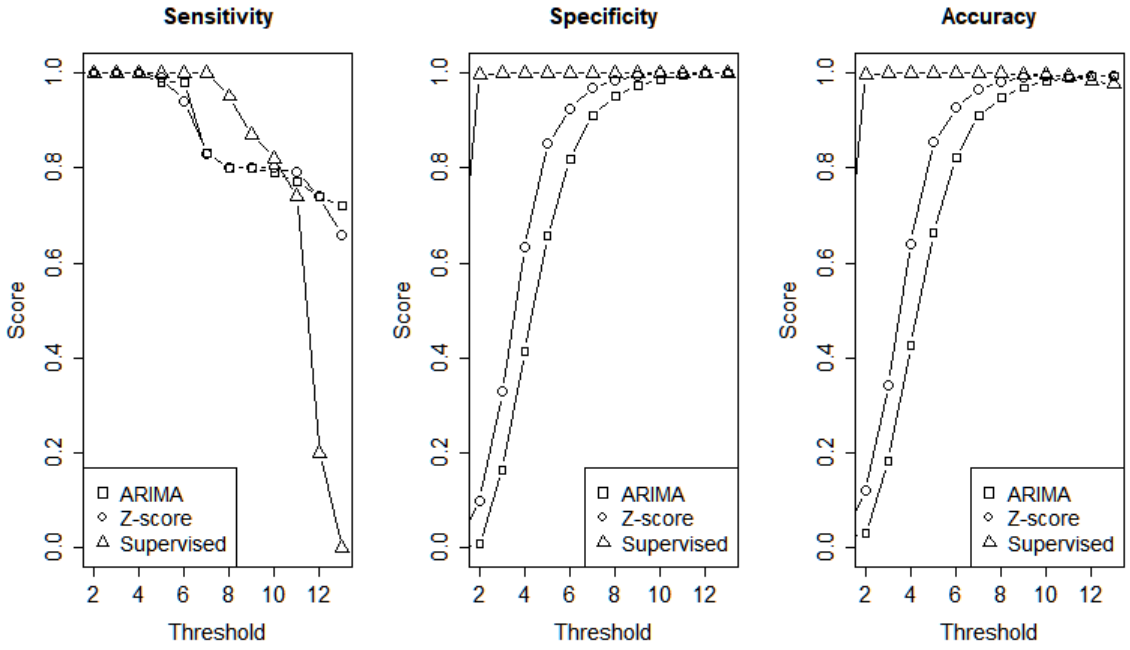


Fig. B.1 CarB: Injection detection for the five highest priority CAN IDs: sensitivity, specificity and accuracy. (Supervised method used a 0.0001 to 0.05 threshold.)

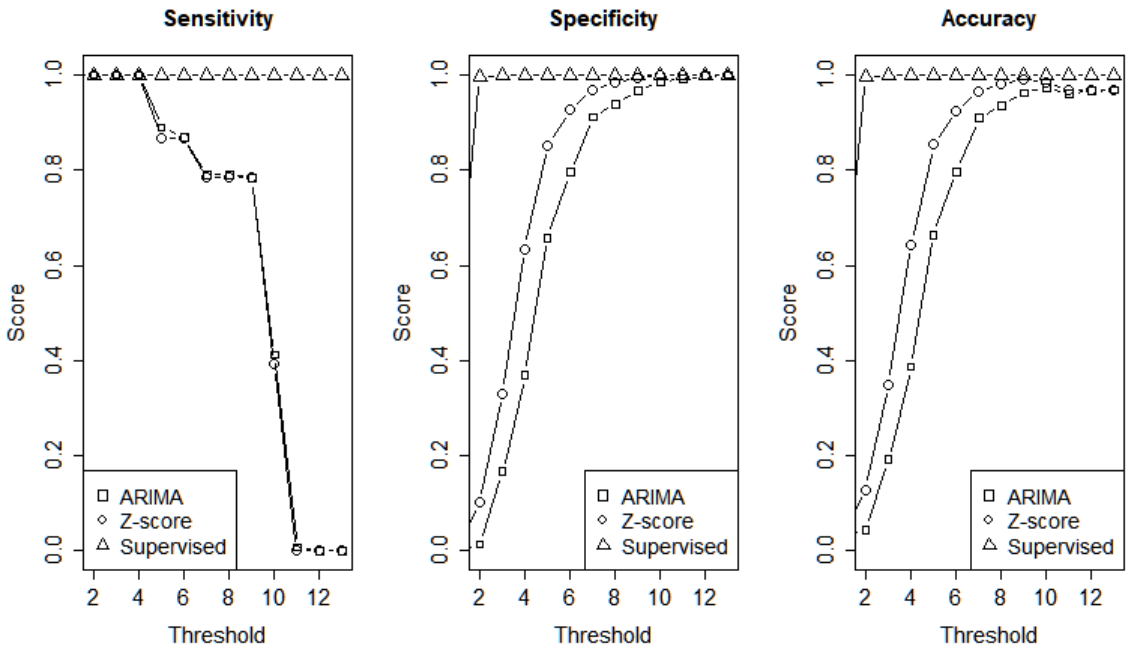


Fig. B.2 CarB: Dropped record detection for the five highest priority CAN IDs: sensitivity, specificity and accuracy scores. (Supervised method used a 0.0001 to 0.05 threshold.)

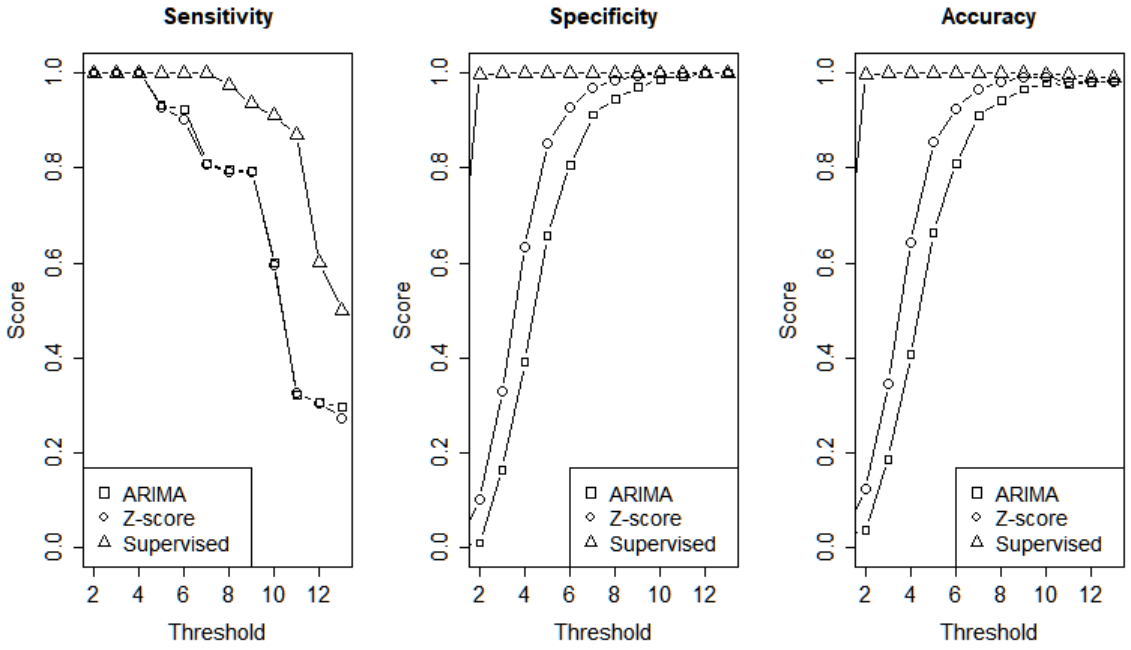


Fig. B.3 CarB: Combined detection scores for the five highest priority CAN IDs: sensitivity, specificity and accuracy scores. (Supervised method used a 0.0001 to 0.05 threshold.)

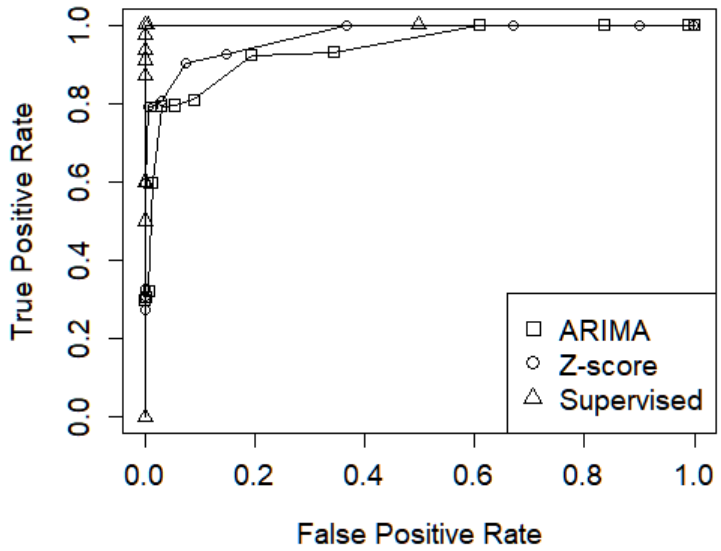


Fig. B.4 CarB: ROC curves for combined detection in the five highest priority CAN IDs, adjusting error rate (ARIMA and Z-score) and threshold (Supervised).

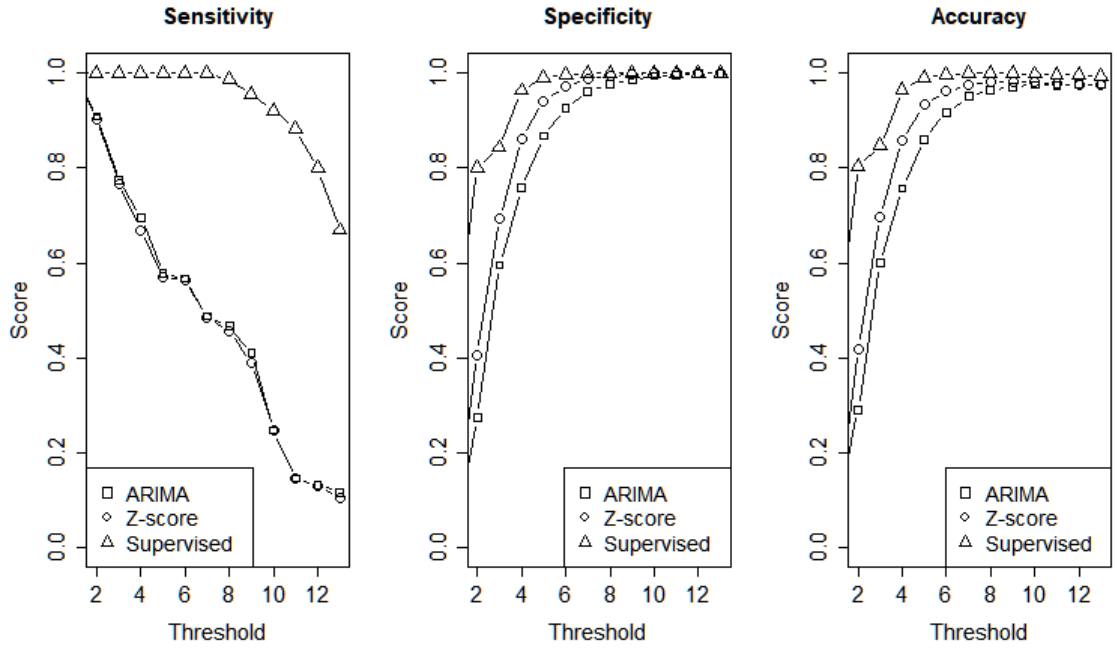


Fig. B.5 CarB: Combined detection scores across all CAN IDs: sensitivity, specificity and accuracy scores. (Supervised method used a 0.0001 to 0.05 threshold.)

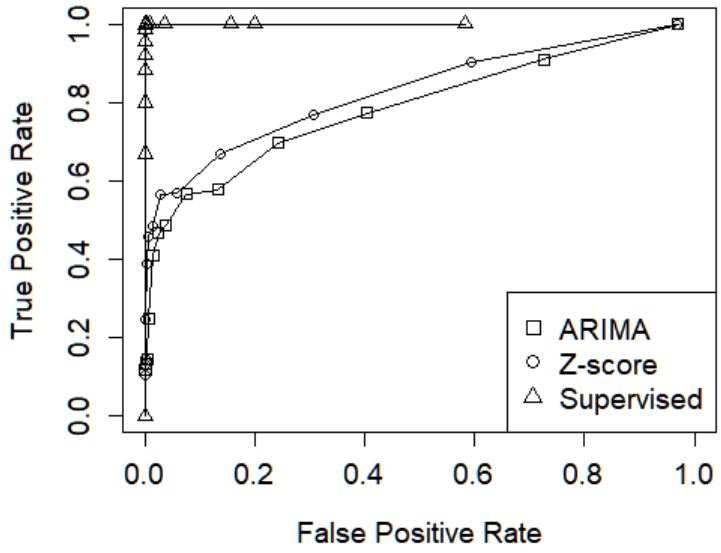


Fig. B.6 CarB: ROC curves for combined detection across in all CAN IDs, adjusting error rate (ARIMA and Z-score) and threshold (Supervised).

Appendix C

Field Profiles

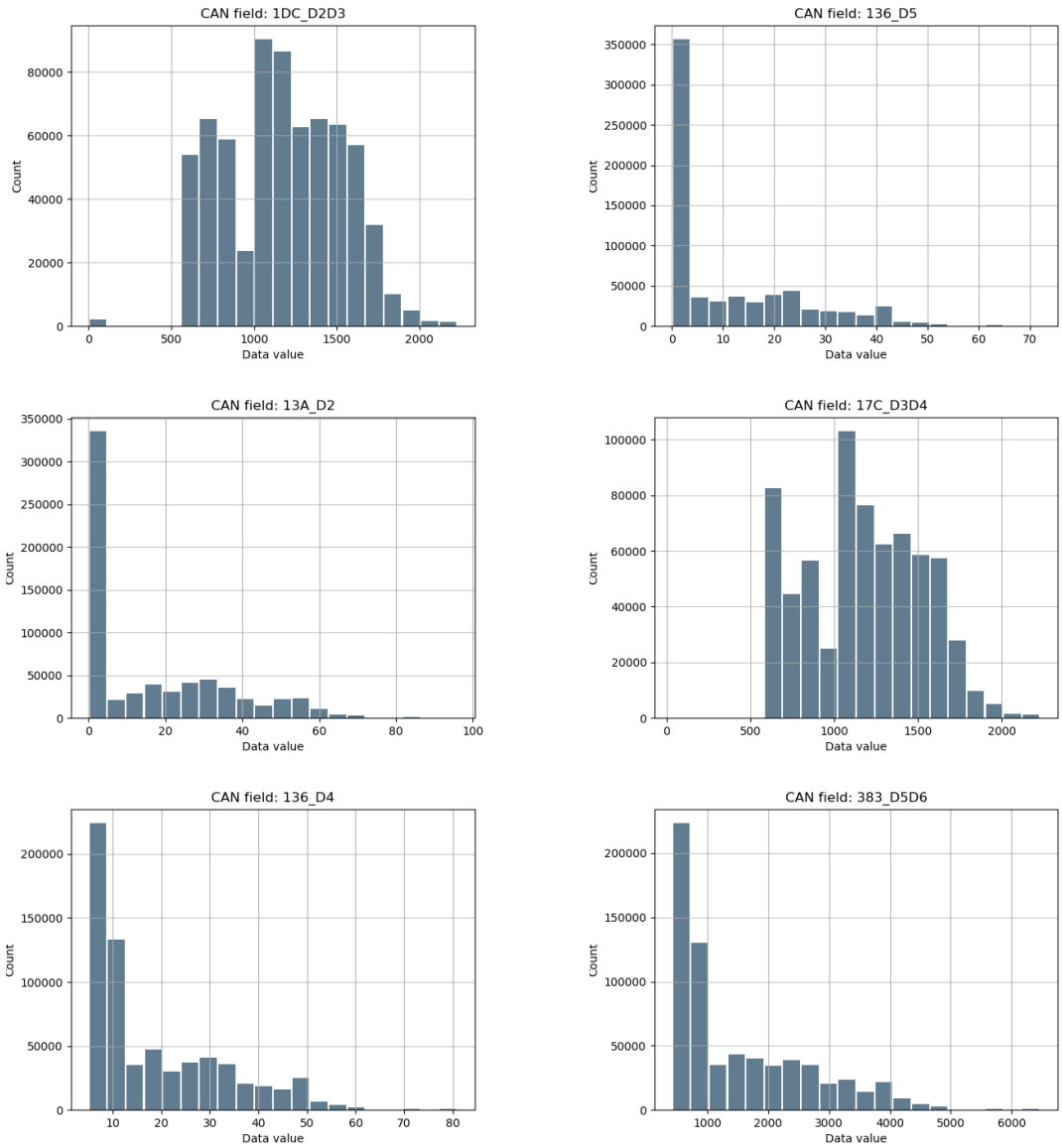


Fig. C.1 Field value distributions: Car 1 Accelerator cluster.

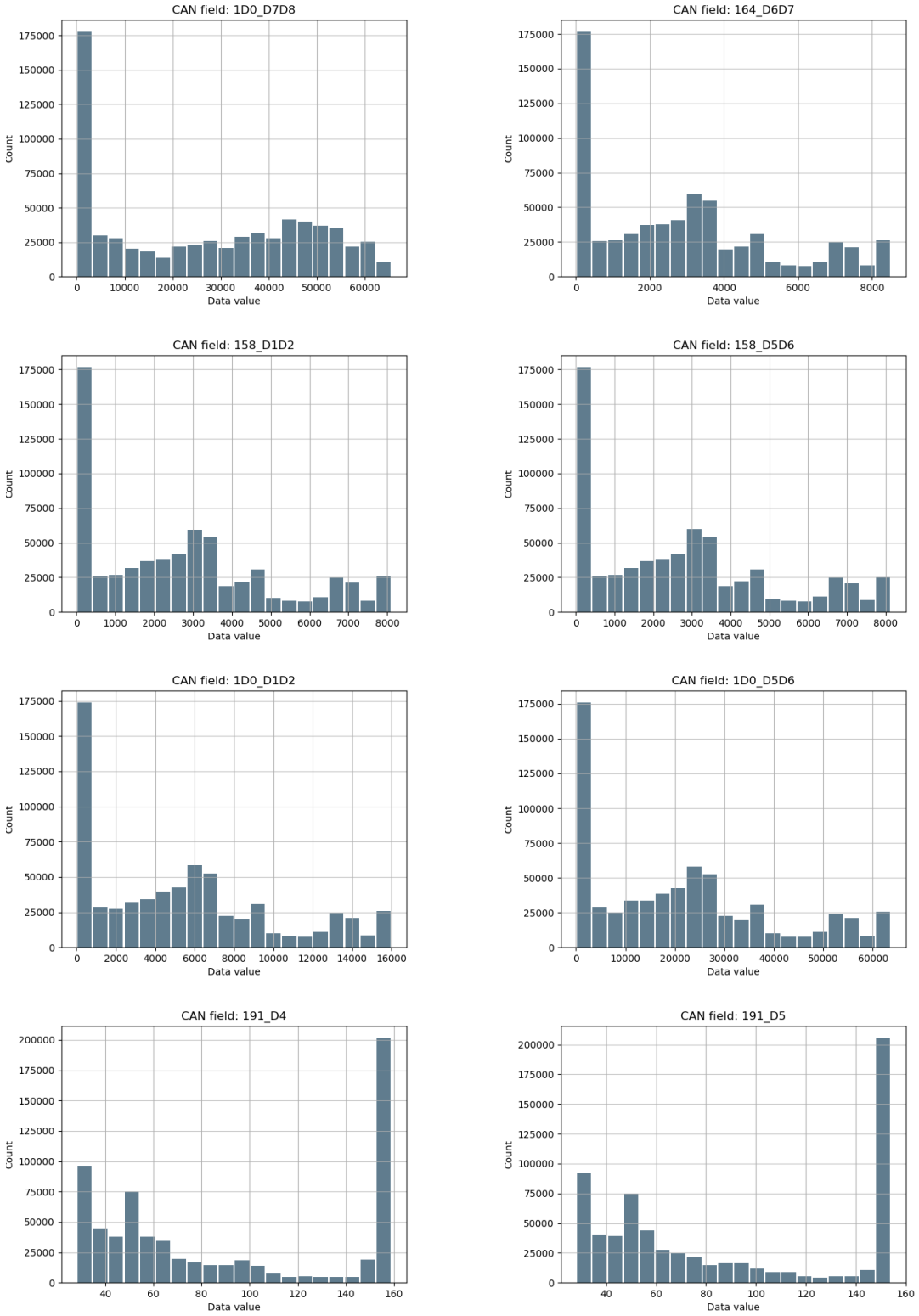


Fig. C.2 Field value distributions: Car 1 Speed cluster.

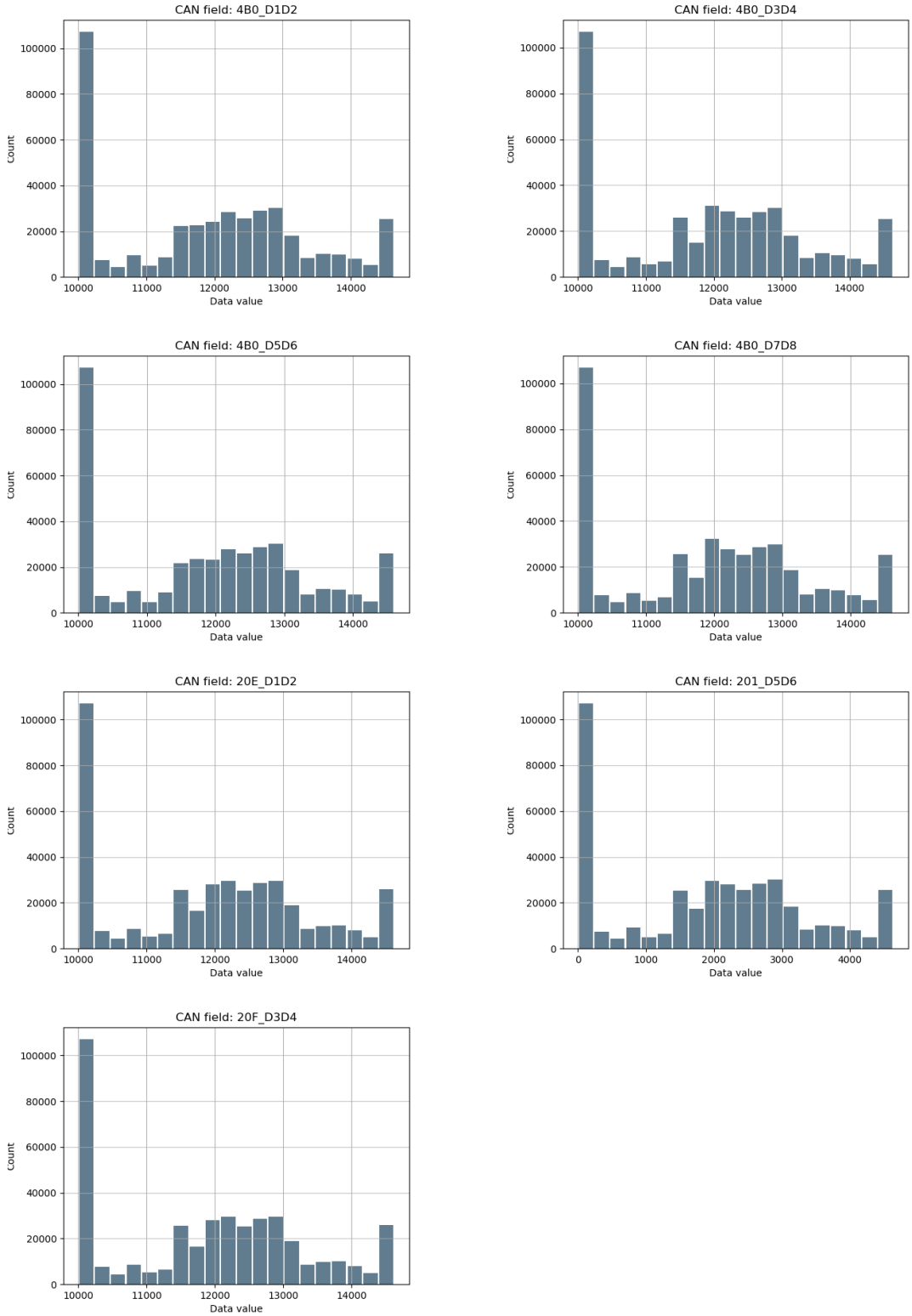


Fig. C.3 Field value distributions: Car 2 Cluster A.

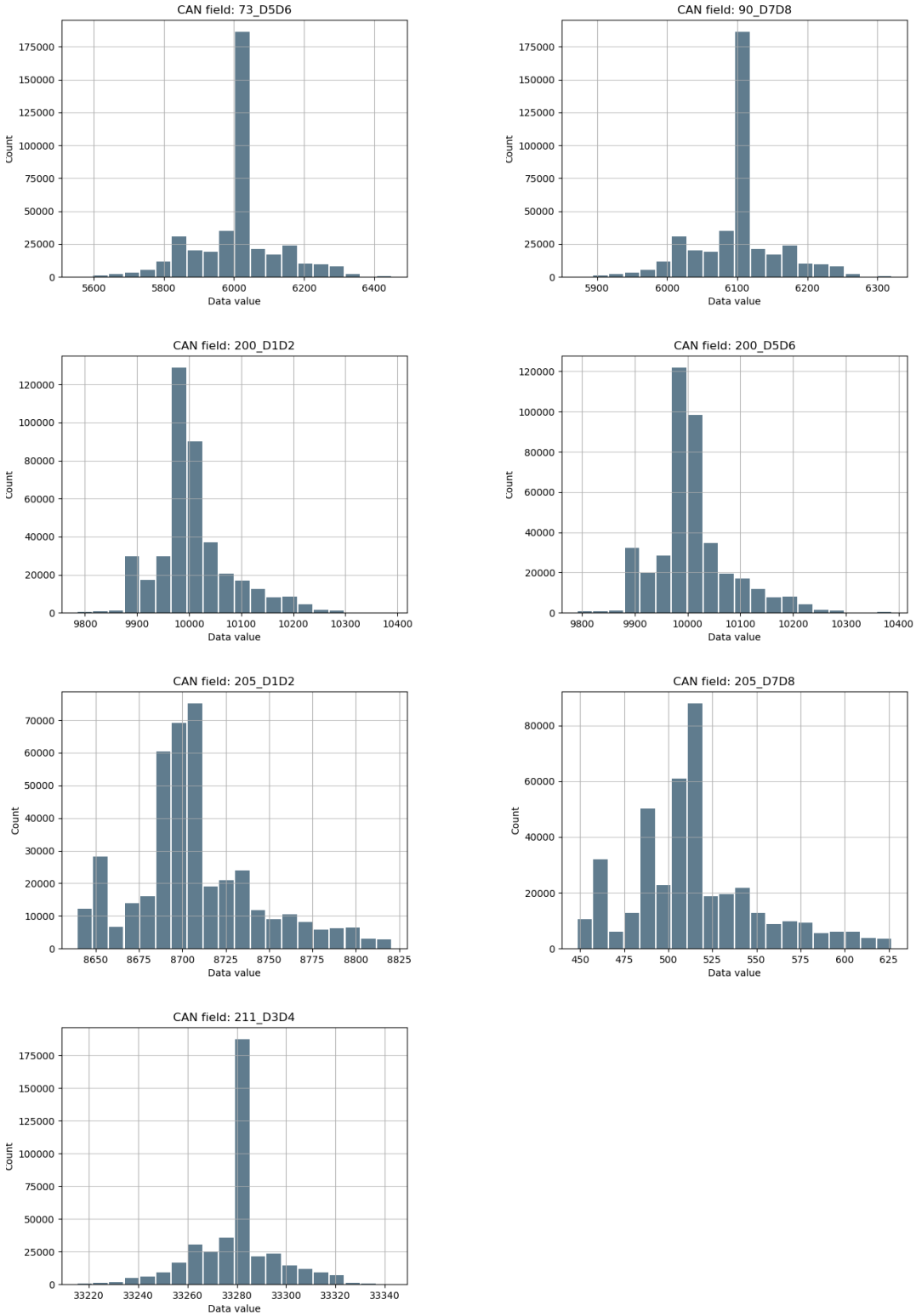


Fig. C.4 Field value distributions: Car 2 Cluster B.

Appendix D

Payload Anomaly Detection - Attacks Generated on Unchanged Data

The detection results in this Appendix are from attack simulations generated by multiplying the payload field values by 0.1, 0.5, 0.8, 0.9, 1.1, 1.2, 1.5, 2.0, 3.0 prior to standardising the data. The multiplications were applied sequentially to each field, with the remaining fields being unchained.

D.0.1 Car 1

Table D.1 False positives recorded for Car 1 with attacks generated on unscaled data (1000 record test data set).

	Accelerator	Speed
LOF	59	57
CC	100	167
OCSVM	14	12

Table D.2 $F_\beta(0.5)$ scores recorded for Car 1 with attacks generated on unscaled data (1000 record test data set).

	Accelerator	Speed
LOF	0.8500	0.9079
CC	0.6510	0.6276
OCSVM	0.7748	0.8332

Table D.3 Car 1 Accelerator packet cluster: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
0.1	0	469	433	0	9	88
0.5	62	476	458	49	54	167
0.8	87	630	597	77	219	368
0.9	183	754	702	173	446	557
1.1	206	713	695	199	427	575
1.2	67	625	575	60	332	397
1.5	4	477	459	4	223	147
2.0	0	440	433	0	115	76
3.0	0	423	407	0	0	43

Table D.4 Car 1 Accelerator packet cluster: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
0.1	0	451	455	0	195	180
0.5	0	591	563	0	451	448
0.8	266	873	868	265	841	839
0.9	828	901	901	828	893	887
1.1	795	898	898	795	894	898
1.2	417	861	870	419	834	833
1.5	0	623	572	0	473	453
2.0	0	456	440	0	249	238
3.0	0	443	437	0	1	0

Table D.5 Car 1 Accelerator packet cluster: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
0.1	0	449	448	0	0	0
0.5	0	489	482	0	195	193
0.8	8	760	738	8	632	675
0.9	508	951	952	503	887	895
1.1	668	953	950	676	941	934
1.2	260	751	731	264	691	660
1.5	0	490	456	0	437	421
2.0	0	450	428	0	201	62
3.0	0	429	419	0	0	0 0

Table D.6 Car 1 Speed packet cluster: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
0.1	20	236	229	229	239	239	0	0
0.5	22	237	235	233	241	241	0	0
0.8	58	275	274	274	276	276	3	1
0.9	119	310	309	308	311	310	62	64
1.1	111	308	307	306	312	313	97	63
1.2	53	277	275	274	277	278	26	5
1.5	21	238	233	233	243	243	0	0
2.0	18	235	229	229	239	239	0	0
3.0	12	235	229	229	239	239	0	0

Table D.7 Car 1 Speed packet cluster: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
0.1	280	304	314	314	312	310	162	161
0.5	367	545	545	545	545	545	442	456
0.8	652	558	557	558	559	559	839	837
0.9	828	800	800	797	799	800	849	848
1.1	847	815	815	817	816	816	845	846
1.2	682	559	560	560	558	558	822	825
1.5	427	520	520	520	521	521	435	426
2.0	253	339	340	338	335	339	97	99
3.0	243	260	260	260	260	261	0	0

Table D.8 Car 1 Speed packet cluster: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
0.1	126	295	295	295	295	295	0	0
0.5	267	308	308	308	308	308	48	48
0.8	435	492	490	495	505	497	520	560
0.9	848	754	757	753	754	753	663	668
1.1	761	729	728	727	723	729	675	647
1.2	582	589	586	586	587	591	503	536
1.5	427	306	307	307	307	307	106	88
2.0	381	292	292	292	292	292	0	0
3.0	337	266	266	268	267	267	0	0

Table D.9 Car 1 Accelerator packet cluster: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
0.1	0.9549	0.7902	0.8091	0.9549	0.9529	0.9337
0.5	0.9403	0.7863	0.7961	0.9435	0.9422	0.9118
0.8	0.9339	0.6811	0.7075	0.9365	0.8956	0.8395
0.9	0.9070	0.5541	0.6137	0.9100	0.8024	0.7364
1.1	0.8998	0.6019	0.6209	0.9020	0.8121	0.7238
1.2	0.9390	0.6853	0.7238	0.9408	0.8547	0.8265
1.5	0.9540	0.7858	0.7956	0.9540	0.8943	0.9176
2.0	0.9549	0.8055	0.8091	0.9549	0.9265	0.9367
3.0	0.9549	0.8141	0.8218	0.9549	0.9549	0.9449

Table D.10 Car 1 Accelerator packet cluster: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
0.1	0.9259	0.7633	0.7612	0.9259	0.8712	0.8761
0.5	0.9259	0.6736	0.6941	0.9259	0.7633	0.7650
0.8	0.8464	0.3328	0.3423	0.8468	0.3905	0.3938
0.9	0.4119	0.2756	0.2756	0.4119	0.2927	0.3051
1.1	0.4617	0.2821	0.2821	0.4617	0.2906	0.2821
1.2	0.7811	0.3553	0.3385	0.7801	0.4021	0.4038
1.5	0.9259	0.6482	0.6877	0.9259	0.7511	0.7623
2.0	0.9259	0.7606	0.7692	0.9259	0.8526	0.8566
3.0	0.9259	0.7676	0.7708	0.9259	0.9257	0.9259

Table D.11 Car 1 Accelerator packet cluster: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
0.1	0.9889	0.8451	0.8456	0.9889	0.9889	0.9889
0.5	0.9889	0.8242	0.8280	0.9889	0.9413	0.9419
0.8	0.9873	0.5952	0.6226	0.9873	0.7278	0.6897
0.9	0.8135	0.1957	0.1923	0.8164	0.3747	0.3557
1.1	0.6963	0.1889	0.1990	0.6888	0.2283	0.2500
1.2	0.9213	0.6067	0.6309	0.9200	0.6741	0.7036
1.5	0.9889	0.8236	0.8416	0.9889	0.8510	0.8585
2.0	0.9889	0.8446	0.8553	0.9889	0.9396	0.9755
3.0	0.9889	0.8548	0.8595	0.9889	0.9889	0.9889

Table D.12 Car 1 Speed packet cluster: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
0.1	0.9518	0.8917	0.8940	0.8940	0.8907	0.8907	0.9564	0.9564
0.5	0.9514	0.8914	0.8920	0.8927	0.8900	0.8900	0.9564	0.9564
0.8	0.9428	0.8781	0.8785	0.8785	0.8778	0.8778	0.9557	0.9562
0.9	0.9270	0.8651	0.8655	0.8659	0.8647	0.8651	0.9418	0.9413
1.1	0.9291	0.8659	0.8662	0.8666	0.8643	0.8639	0.9329	0.9415
1.2	0.9440	0.8774	0.8781	0.8785	0.8774	0.8771	0.9504	0.9553
1.5	0.9516	0.8910	0.8927	0.8927	0.8893	0.8893	0.9564	0.9564
2.0	0.9523	0.8920	0.8940	0.8940	0.8907	0.8907	0.9564	0.9564
3.0	0.9537	0.8920	0.8940	0.8940	0.8907	0.8907	0.9564	0.9564

Table D.13 Car 1 Speed packet cluster: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
0.1	0.7916	0.7817	0.7774	0.7774	0.7783	0.7791	0.8347	0.8350
0.5	0.7536	0.6522	0.6522	0.6522	0.6522	0.6522	0.7154	0.7076
0.8	0.5686	0.6432	0.6439	0.6432	0.6425	0.6425	0.3482	0.3513
0.9	0.3650	0.4052	0.4052	0.4093	0.4066	0.4052	0.3323	0.3339
1.1	0.3355	0.3841	0.3841	0.3812	0.3827	0.3827	0.3387	0.3371
1.2	0.5408	0.6425	0.6418	0.6418	0.6432	0.6432	0.3739	0.3695
1.5	0.7235	0.6689	0.6689	0.6689	0.6682	0.6682	0.7192	0.7240
2.0	0.8022	0.7665	0.7660	0.7669	0.7683	0.7665	0.8551	0.8545
3.0	0.8060	0.7995	0.7995	0.7995	0.7995	0.7991	0.8821	0.8821

Table D.14 Car 1 Speed packet cluster: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
0.1	0.9617	0.9113	0.9113	0.9113	0.9113	0.9113	0.9905	0.9905
0.5	0.9209	0.9067	0.9067	0.9067	0.9067	0.9067	0.9802	0.9802
0.8	0.8540	0.8247	0.8258	0.8230	0.8174	0.8219	0.8086	0.7835
0.9	0.4589	0.6053	0.6015	0.6066	0.6053	0.6066	0.7033	0.6987
1.1	0.5963	0.6356	0.6367	0.6379	0.6424	0.6356	0.6921	0.7175
1.2	0.7684	0.7634	0.7655	0.7655	0.7648	0.7619	0.8185	0.7989
1.5	0.8578	0.9074	0.9071	0.9071	0.9071	0.9071	0.9667	0.9710
2.0	0.8783	0.9124	0.9124	0.9124	0.9124	0.9124	0.9905	0.9905
3.0	0.8959	0.9212	0.9212	0.9205	0.9209	0.9209	0.9905	0.9905

D.0.2 Car 2

Table D.15 False positives recorded for Car 2 with attacks generated on unscaled data (1000 record test data set).

	Cluster A	Cluster B
LOF	41	52
CC	322	99
OCSVM	23	27

Table D.16 $F_\beta(0.5)$ scores recorded for Car 2 with attacks generated on unscaled data (1000 record test data set).

	Cluster A	Cluster B
LOF	0.9584	0.9598
CC	0.7776	0.9157
OCSVM	0.9669	0.9788

Table D.17 Car 2, Cluster A: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
0.1	0	0	0	0	0	244	0
0.5	0	0	0	0	0	241	0
0.8	0	0	0	0	0	253	0
0.9	0	0	0	0	0	281	0
1.1	0	0	0	0	0	279	0
1.2	0	0	0	0	0	252	0
1.5	0	0	0	0	0	241	0
2.0	0	0	0	0	0	241	0
3.0	0	0	0	0	0	241	0

Table D.18 Car 2, Cluster A: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
0.1	0	0	0	0	0	24	0
0.5	0	0	0	0	0	60	0
0.8	0	0	0	0	0	526	0
0.9	0	0	0	0	0	636	0
1.1	0	0	0	0	0	641	0
1.2	0	0	0	0	0	496	0
1.5	0	0	0	0	0	55	0
2.0	0	0	0	0	0	17	0
3.0	0	0	0	0	0	7	0

Table D.19 Car 2, Cluster A: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
0.1	0	0	0	0	0	273	0
0.5	0	0	0	0	0	281	0
0.8	0	0	0	0	0	289	0
0.9	0	0	0	0	0	562	0
1.1	0	0	0	0	0	570	0
1.2	0	0	0	0	0	290	0
1.5	0	0	0	0	0	280	0
2.0	0	0	0	0	0	269	0
3.0	0	0	0	0	0	251	0

Table D.20 Car 2, Cluster A: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
0.1	0.9682	0.9682	0.9682	0.9682	0.9682	0.9026	0.9682
0.5	0.9682	0.9682	0.9682	0.9682	0.9682	0.9036	0.9682
0.8	0.9682	0.9682	0.9682	0.9682	0.9682	0.8996	0.9682
0.9	0.9682	0.9682	0.9682	0.9682	0.9682	0.8899	0.9682
1.1	0.9682	0.9682	0.9682	0.9682	0.9682	0.8906	0.9682
1.2	0.9682	0.9682	0.9682	0.9682	0.9682	0.8999	0.9682
1.5	0.9682	0.9682	0.9682	0.9682	0.9682	0.9036	0.9682
2.0	0.9682	0.9682	0.9682	0.9682	0.9682	0.9036	0.9682
3.0	0.9682	0.9682	0.9682	0.9682	0.9682	0.9036	0.9682

Table D.21 Car 2, Cluster A: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
0.1	0.7952	0.7952	0.7952	0.7952	0.7952	0.7881	0.7952
0.5	0.7952	0.7952	0.7952	0.7952	0.7952	0.7771	0.7952
0.8	0.7952	0.7952	0.7952	0.7952	0.7952	0.5664	0.7952
0.9	0.7952	0.7952	0.7952	0.7952	0.7952	0.4861	0.7952
1.1	0.7952	0.7952	0.7952	0.7952	0.7952	0.4820	0.7952
1.2	0.7952	0.7952	0.7952	0.7952	0.7952	0.5855	0.7952
1.5	0.7952	0.7952	0.7952	0.7952	0.7952	0.7787	0.7952
2.0	0.7952	0.7952	0.7952	0.7952	0.7952	0.7902	0.7952
3.0	0.7952	0.7952	0.7952	0.7952	0.7952	0.7931	0.7952

Table D.22 Car 2, Cluster A: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
0.1	0.9819	0.9819	0.9819	0.9819	0.9819	0.9087	0.9819
0.5	0.9819	0.9819	0.9819	0.9819	0.9819	0.9060	0.9819
0.8	0.9819	0.9819	0.9819	0.9819	0.9819	0.9032	0.9819
0.9	0.9819	0.9819	0.9819	0.9819	0.9819	0.7700	0.9819
1.1	0.9819	0.9819	0.9819	0.9819	0.9819	0.7646	0.9819
1.2	0.9819	0.9819	0.9819	0.9819	0.9819	0.9028	0.9819
1.5	0.9819	0.9819	0.9819	0.9819	0.9819	0.9063	0.9819
2.0	0.9819	0.9819	0.9819	0.9819	0.9819	0.9101	0.9819
3.0	0.9819	0.9819	0.9819	0.9819	0.9819	0.9161	0.9819

Table D.23 Car 2, Cluster B: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
0.1	0	0	0	0	0	0	0
0.5	0	0	0	0	0	0	0
0.8	0	0	0	0	0	0	0
0.9	0	0	0	0	0	44	0
1.1	0	0	0	0	0	31	0
1.2	0	0	0	0	0	0	0
1.5	0	0	0	0	0	0	0
2.0	0	0	0	0	0	0	0
3.0	0	0	0	0	0	0	0

Table D.24 Car 2, Cluster B: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
0.1	0	0	0	0	0	0	0
0.5	0	0	0	0	0	0	0
0.8	0	0	0	0	0	51	0
0.9	0	0	0	0	0	686	0
1.1	0	0	0	0	0	648	0
1.2	0	0	0	0	0	157	0
1.5	0	0	0	0	0	0	0
2.0	0	0	0	0	0	0	0
3.0	0	0	0	0	0	0	0

Table D.25 Car 2, Cluster B: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Multiplier	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
0.1	0	0	0	0	0	0	0
0.5	0	0	0	0	0	0	0
0.8	0	0	0	0	0	0	0
0.9	0	0	0	0	0	0	0
1.1	0	0	0	0	0	2	0
1.2	0	0	0	0	0	0	0
1.5	0	0	0	0	0	0	0
2.0	0	0	0	0	0	0	0
3.0	0	0	0	0	0	0	0

Table D.26 Car 2, Cluster B: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
0.1	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601
0.5	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601
0.8	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601
0.9	0.9601	0.9601	0.9601	0.9601	0.9601	0.9499	0.9601
1.1	0.9601	0.9601	0.9601	0.9601	0.9601	0.9530	0.9601
1.2	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601
1.5	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601
2.0	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601
3.0	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601	0.9601

Table D.27 Car 2, Cluster B: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
0.1	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266
0.5	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266
0.8	0.9266	0.9266	0.9266	0.9266	0.9266	0.9139	0.9266
0.9	0.9266	0.9266	0.9266	0.9266	0.9266	0.5920	0.9266
1.1	0.9266	0.9266	0.9266	0.9266	0.9266	0.6277	0.9266
1.2	0.9266	0.9266	0.9266	0.9266	0.9266	0.8840	0.9266
1.5	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266
2.0	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266
3.0	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266	0.9266

Table D.28 Car 2, Cluster B: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Multiplier	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
0.1	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789
0.5	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789
0.8	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789
0.9	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789
1.1	0.9789	0.9789	0.9789	0.9789	0.9789	0.9784	0.9789
1.2	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789
1.5	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789
2.0	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789
3.0	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789	0.9789

Appendix E

Payload Anomaly Detection - Attacks Generated Using Offset to Standardised Data

The detection results in this Appendix are from attack simulations generated by adding the offsets -1.0, -0.5, -0.3, -0.2, -0.1, 0.1, 0.2, 0.3, 0.5, 1.0 to the standardised data. The offsets were added sequentially to each field, with the remaining fields being unchained.

E.0.1 Car 1

Table E.1 False positives recorded for Car 1 with attacks generated on standardised data (1000 record test data set).

	Accelerator	Speed
LOF	43	48
CC	86	185
OCSVM	22	10

Table E.2 $F_\beta(0.5)$ scores recorded for Car 1 with with attacks generated on standardised data (1000 record test data set).

	Accelerator	Speed
LOF	0.8655	0.9607
CC	0.5785	0.4995
OCSVM	0.6657	0.8241

Table E.3 Car 1 Accelerator packet cluster attacks generated on scaled data: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
-1.0	105	46	29	88	19	195
-0.5	174	193	122	164	103	393
-0.3	293	413	323	294	291	432
-0.2	335	405	399	336	402	443
-0.1	489	454	436	501	530	515
0.1	501	454	442	476	629	537
0.2	396	410	400	399	575	490
0.3	343	298	394	342	409	407
0.5	232	149	412	206	87	331
1.0	14	29	345	16	24	158

Table E.4 Car 1 Accelerator packet cluster attacks generated on scaled data: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
-1.0	72	117	97	71	111	106
-0.5	457	416	384	456	408	414
-0.3	738	701	704	741	741	708
-0.2	877	876	876	878	884	882
-0.1	895	901	899	894	895	901
0.1	902	898	898	900	898	901
0.2	853	880	877	851	847	856
0.3	713	700	702	714	759	764
0.5	485	462	493	486	486	494
1.0	44	50	45	43	47	45

Table E.5 Car 1 Accelerator packet cluster attacks generated on scaled data: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
-1.0	0	0	0	0	0	0
-0.5	177	55	26	142	40	95
-0.3	584	469	469	567	527	531
-0.2	767	637	586	766	701	701
-0.1	877	808	764	878	847	876
0.1	955	982	979	954	968	968
0.2	918	966	961	917	939	914
0.3	848	813	861	850	723	613
0.5	320	103	271	338	37	107
1.0	0	0	0	0	0	0

Table E.6 Car 1 Speed packet cluster attacks generated on scaled data: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
-1.0	0	0	0	0	0	0	0	0
-0.5	0	0	0	0	0	0	0	9
-0.3	0	0	0	0	0	0	1	18
-0.2	1	1	1	1	1	1	7	45
-0.1	51	22	22	22	22	22	80	66
0.1	50	24	23	21	20	15	91	68
0.2	3	2	2	2	2	1	23	40
0.3	3	2	2	2	2	1	23	40
0.5	0	0	0	0	0	0	3	0
1.0	0	0	0	0	0	0	0	0

Table E.7 Car 1 Speed packet cluster attacks generated on scaled data: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
-1.0	197	190	196	195	185	183	127	125
-0.5	569	555	555	555	555	554	550	534
-0.3	800	776	776	777	777	777	786	778
-0.2	843	836	836	834	837	835	832	828
-0.1	854	852	852	853	852	852	851	850
0.1	854	855	854	854	854	854	848	848
0.2	848	836	836	837	835	835	826	824
0.3	810	790	792	791	792	789	776	777
0.5	551	532	532	532	533	533	551	555
1.0	551	532	532	532	533	533	551	555

Table E.8 Car 1 Speed packet cluster attacks generated on scaled data: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
-1.0	115	0	0	0	0	0	0	0
-0.5	245	4	3	3	2	2	70	86
-0.3	436	235	239	242	255	244	245	272
-0.2	533	531	531	529	526	525	457	484
-0.1	617	672	678	678	669	668	648	905
0.1	935	915	915	915	912	914	667	655
0.2	798	774	770	770	771	769	531	505
0.3	651	319	312	313	319	329	294	312
0.5	464	18	16	16	15	17	47	75
1.0	90	0	0	0	0	0	0	0

Table E.9 Car 1 Accelerator packet cluster attacks generated on scaled data: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
-1.0	0.9417	0.9563	0.9602	0.9461	0.9625	0.9164
-0.5	0.9227	0.9170	0.9372	0.9256	0.9422	0.8431
-0.3	0.8838	0.8338	0.8724	0.8834	0.8845	0.8246
-0.2	0.8677	0.8376	0.8403	0.8673	0.8389	0.8191
-0.1	0.7945	0.8135	0.8226	0.7876	0.7700	0.7792
0.1	0.7876	0.8135	0.8196	0.8017	0.6984	0.7655
0.2	0.8417	0.8352	0.8399	0.8403	0.7399	0.7939
0.3	0.8645	0.8819	0.8426	0.8649	0.8357	0.8366
0.5	0.9048	0.9299	0.8343	0.9131	0.9463	0.8693
1.0	0.9636	0.9602	0.8637	0.9632	0.9614	0.9273

Table E.10 Car 1 Accelerator packet cluster attacks generated on scaled data: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
-1.0	0.9177	0.9055	0.9110	0.9180	0.9071	0.9085
-0.5	0.7722	0.7935	0.8088	0.7727	0.7974	0.7945
-0.3	0.5477	0.5886	0.5854	0.5441	0.5441	0.5812
-0.2	0.3350	0.3370	0.3370	0.3330	0.3208	0.3249
-0.1	0.2976	0.2845	0.2889	0.2998	0.2976	0.2845
0.1	0.2823	0.2911	0.2911	0.2867	0.2911	0.2845
0.2	0.3804	0.3289	0.3350	0.3840	0.3911	0.3750
0.3	0.5758	0.5896	0.5875	0.5748	0.5221	0.5157
0.5	0.7565	0.7695	0.7518	0.7559	0.7559	0.7512
1.0	0.9249	0.9234	0.9247	0.9252	0.9242	0.9247

Table E.11 Car 1 Accelerator packet cluster attacks generated on scaled data: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6
-1.0	0.9827	0.9827	0.9827	0.9827	0.9827	0.9827
-0.5	0.9395	0.9706	0.9771	0.9491	0.9740	0.9611
-0.3	0.7558	0.8266	0.8266	0.7677	0.7936	0.7912
-0.2	0.5767	0.7146	0.7544	0.5781	0.6546	0.6546
-0.1	0.3892	0.5172	0.5807	0.3871	0.4500	0.3914
0.1	0.1774	0.0776	0.0896	0.1808	0.1316	0.1316
0.2	0.2895	0.1389	0.1568	0.2923	0.2290	0.3003
0.3	0.4481	0.5093	0.4227	0.4443	0.6307	0.7341
0.5	0.8929	0.9592	0.9103	0.8860	0.9747	0.9582
1.0	0.9827	0.9827	0.9827	0.9827	0.9827	0.9827

Table E.12 Car 1 Speed packet cluster attacks generated on scaled data: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
-1.0	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630
-0.5	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9610
-0.3	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9628	0.9590
-0.2	0.9628	0.9628	0.9628	0.9628	0.9628	0.9628	0.9615	0.9527
-0.1	0.9513	0.9581	0.9581	0.9581	0.9581	0.9581	0.9442	0.9476
0.1	0.9515	0.9576	0.9578	0.9583	0.9585	0.9597	0.9414	0.9472
0.2	0.9624	0.9626	0.9626	0.9626	0.9626	0.9628	0.9578	0.9539
0.3	0.9628	0.9630	0.9630	0.9630	0.9630	0.9630	0.9617	0.9610
0.5	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9624	0.9630
1.0	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630	0.9630

Table E.13 Car 1 Speed packet cluster attacks generated on scaled data: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
-1.0	0.8108	0.8133	0.8111	0.8115	0.8150	0.8157	0.8343	0.8349
-0.5	0.6221	0.6321	0.6321	0.6321	0.6321	0.6328	0.6356	0.6465
-0.3	0.3937	0.4249	0.4249	0.4236	0.4236	0.4236	0.4122	0.4224
-0.2	0.3315	0.3422	0.3422	0.3453	0.3407	0.3438	0.3483	0.3542
-0.1	0.3141	0.3173	0.3173	0.3157	0.3173	0.3173	0.3189	0.3205
0.1	0.3141	0.3125	0.3141	0.3141	0.3141	0.3141	0.3237	0.3237
0.2	0.3237	0.3422	0.3422	0.3407	0.3438	0.3438	0.3571	0.3601
0.3	0.3800	0.4070	0.4044	0.4057	0.4044	0.4083	0.4249	0.4236
0.5	0.6349	0.6478	0.6478	0.6478	0.6472	0.6472	0.6349	0.6321
1.0	0.7231	0.7316	0.7341	0.7346	0.7292	0.7287	0.8317	0.8195

Table E.14 Car 1 Speed packet cluster attacks generated on scaled data: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5
-1.0	0.9662	0.9921	0.9921	0.9921	0.9921	0.9921	0.9921	0.9921
-0.5	0.9298	0.9912	0.9914	0.9914	0.9917	0.9917	0.9769	0.9732
-0.3	0.8556	0.9329	0.9317	0.9307	0.9266	0.9301	0.9298	0.9211
-0.2	0.8030	0.8042	0.8042	0.8054	0.8072	0.8078	0.8453	0.8312
-0.1	0.7446	0.6973	0.6916	0.6916	0.7001	0.7010	0.7190	0.3345
0.1	0.2500	0.3080	0.3080	0.3080	0.3161	0.3107	0.7019	0.7128
0.2	0.5465	0.5813	0.5867	0.5867	0.5854	0.5881	0.8042	0.8195
0.3	0.7163	0.9046	0.9072	0.9068	0.9046	0.9009	0.9136	0.9072
0.5	0.8417	0.9883	0.9887	0.9887	0.9890	0.9885	0.9821	0.9757
1.0	0.9722	0.9921	0.9921	0.9921	0.9921	0.9921	0.9921	0.9921

E.0.2 Car 2

Table E.15 False positives recorded for Car 2 with attacks generated on standardised data (1000 record test data set).

	Cluster A	Cluster B
LOF	47	42
CC	316	86
OCSVM	7	16

Table E.16 $F_\beta(0.5)$ scores recorded for Car 2 with with attacks generated on standardised data (1000 record test data set).

	Cluster A	Cluster B
LOF	0.9634	0.6840
CC	0.5999	0.2810
OCSVM	0.9288	0.3405

Table E.17 Car 2, Cluster A packet attacks generated on scaled data: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
-1.0	0	0	0	0	0	0	0
-0.5	0	0	0	0	0	0	0
-0.3	0	0	0	0	0	0	0
-0.2	1	0	1	0	1	0	1
-0.1	5	9	16	5	9	5	9
0.1	4	10	8	6	2	4	2
0.2	0	8	0	1	0	1	0
0.3	0	11	0	0	0	0	0
0.5	0	0	0	0	0	0	0
1.0	0	0	0	0	0	0	0

Table E.18 Car 2, Cluster A packet attacks generated on scaled data: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
-1.0	0	0	0	0	0	0	0
-0.5	277	262	274	258	271	267	271
-0.3	548	546	548	545	548	550	548
-0.2	548	546	548	545	548	550	548
-0.1	678	687	682	688	680	684	680
0.1	693	692	694	693	694	698	694
0.2	619	625	617	627	621	631	621
0.3	530	524	530	527	534	539	534
0.5	288	279	285	286	286	279	287
1.0	0	0	0	0	0	0	0

Table E.19 Car 2, Cluster A packet attacks generated on scaled data: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
-1.0	0	0	0	0	0	0	0
-0.5	0	0	0	0	0	0	0
-0.3	72	74	81	85	77	74	77
-0.2	197	211	308	293	236	185	238
-0.1	498	459	525	504	520	482	521
0.1	717	746	676	710	707	725	705
0.2	255	498	164	131	192	220	192
0.3	45	61	45	53	49	59	49
0.5	0	0	0	0	0	0	0
1.0	0	0	0	0	0	0	0

Table E.20 Car 2, Cluster A packet attacks generated on scaled data: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
-1.0	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638
-0.5	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638
-0.3	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638
-0.2	0.9635	0.9638	0.9635	0.9638	0.9635	0.9638	0.9635
-0.1	0.9627	0.9618	0.9602	0.9627	0.9618	0.9627	0.9618
0.1	0.9629	0.9615	0.9620	0.9624	0.9633	0.9629	0.9633
0.2	0.9638	0.9620	0.9638	0.9635	0.9638	0.9635	0.9638
0.3	0.9638	0.9613	0.9638	0.9638	0.9638	0.9638	0.9638
0.5	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638
1.0	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638	0.9638

Table E.21 Car 2, Cluster A packet attacks generated on scaled data: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
-1.0	0.7982	0.7982	0.7982	0.7982	0.7982	0.7982	0.7982
-0.5	0.7011	0.7074	0.7024	0.7091	0.7037	0.7054	0.7037
-0.3	0.5550	0.5564	0.5550	0.5571	0.5550	0.5536	0.5550
-0.2	0.4941	0.4876	0.4949	0.4860	0.4901	0.4909	0.4917
-0.1	0.4533	0.4451	0.4497	0.4442	0.4515	0.4478	0.4515
0.1	0.4396	0.4405	0.4386	0.4396	0.4386	0.4349	0.4386
0.2	0.5029	0.4981	0.5045	0.4965	0.5013	0.4933	0.5013
0.3	0.5671	0.5710	0.5671	0.5691	0.5644	0.5611	0.5644
0.5	0.6964	0.7003	0.6977	0.6973	0.6973	0.7003	0.6968
1.0	0.7982	0.7982	0.7982	0.7982	0.7982	0.7982	0.7982

Table E.22 Car 2, Cluster A packet attacks generated on scaled data: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4
-1.0	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944
-0.5	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944
-0.3	0.9789	0.9784	0.9768	0.9759	0.9778	0.9784	0.9778
-0.2	0.9469	0.9429	0.9115	0.9168	0.9354	0.9503	0.9347
-0.1	0.8267	0.8474	0.8111	0.8234	0.8141	0.8355	0.8135
0.1	0.6551	0.6213	0.6971	0.6627	0.6659	0.6461	0.6680
0.2	0.9294	0.8267	0.9561	0.9647	0.9484	0.9402	0.9484
0.3	0.9849	0.9814	0.9849	0.9832	0.9841	0.9818	0.9841
0.5	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944
1.0	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944	0.9944

Table E.23 Car 2, Cluster B packet attacks generated on scaled data: LOF False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
-1.0	56	54	48	214	224	101	803
-0.5	406	406	309	441	432	372	940
-0.3	651	654	574	571	617	594	881
-0.2	767	756	673	677	695	711	862
-0.1	728	730	775	791	802	833	849
0.1	727	730	763	761	762	781	850
0.2	770	732	663	662	695	718	887
0.3	678	624	544	574	551	656	885
0.5	363	362	374	402	351	385	953
1.0	47	52	156	158	91	84	777

Table E.24 Car 2, Cluster B packet attacks generated on scaled data: CC False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
-1.0	826	824	796	808	797	800	839
-0.5	907	907	916	918	904	908	906
-0.3	920	919	925	926	916	915	922
-0.2	922	922	929	927	919	919	924
-0.1	926	926	927	928	924	926	925
0.1	931	931	928	927	932	931	929
0.2	926	926	923	922	928	927	929
0.3	922	922	915	919	926	929	920
0.5	914	914	904	907	911	914	906
1.0	914	914	904	907	911	914	906

Table E.25 Car 2, Cluster B packet attacks generated on scaled data: OCSVM False Negatives by CAN field manipulated. Actual number out of 1000 attack outliers generated per cell.

Offset	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
-1.0	59	68	164	255	73	181	766
-0.5	915	916	831	872	828	783	944
-0.3	965	967	934	951	929	939	966
-0.2	973	978	966	972	957	964	976
-0.1	978	979	975	980	974	978	982
0.1	978	978	981	978	978	978	979
0.2	972	971	970	969	968	964	972
0.3	956	955	954	947	945	943	964
0.5	890	892	909	809	850	838	931
1.0	15	14	435	84	95	120	836

Table E.26 Car 2, Cluster B packet attacks generated on scaled data: LOF $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
-1.0	0.9547	0.9552	0.9566	0.9114	0.9082	0.9435	0.5036
-0.5	0.8380	0.8380	0.8787	0.8211	0.8256	0.8533	0.2131
-0.3	0.6806	0.6779	0.7416	0.7438	0.7093	0.7271	0.3619
-0.2	0.5548	0.5690	0.6603	0.6565	0.6386	0.6218	0.4012
-0.1	0.6028	0.6005	0.5440	0.5215	0.5051	0.4548	0.4261
0.1	0.6040	0.6005	0.5600	0.5626	0.5613	0.5357	0.4242
0.2	0.5508	0.5982	0.6697	0.6706	0.6386	0.6141	0.3488
0.3	0.6555	0.7036	0.7620	0.7416	0.7574	0.6761	0.3532
0.5	0.8571	0.8575	0.8524	0.8399	0.8621	0.8476	0.1733
1.0	0.9568	0.9556	0.9287	0.9281	0.9461	0.9478	0.5413

Table E.27 Car 2, Cluster B packet attacks generated on scaled data: CC $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
-1.0	0.4265	0.4297	0.4722	0.4545	0.4708	0.4664	0.4049
-0.5	0.2710	0.2710	0.2500	0.2452	0.2778	0.2687	0.2733
-0.3	0.2404	0.2428	0.2281	0.2256	0.2500	0.2524	0.2355
-0.2	0.2355	0.2355	0.2181	0.2231	0.2428	0.2428	0.2306
-0.1	0.2256	0.2256	0.2231	0.2206	0.2306	0.2256	0.2281
0.1	0.2130	0.2130	0.2206	0.2231	0.2104	0.2130	0.2181
0.2	0.2256	0.2256	0.2331	0.2355	0.2206	0.2231	0.2181
0.3	0.2355	0.2355	0.2524	0.2428	0.2256	0.2181	0.2404
0.5	0.2547	0.2547	0.2778	0.2710	0.2618	0.2547	0.2733
1.0	0.4590	0.4605	0.4590	0.4590	0.4605	0.4635	0.4500

Table E.28 Car 2, Cluster B packet attacks generated on scaled data: OCSVM $F_\beta(0.5)$ scores by CAN field manipulated.

Offset	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4
-1.0	0.9745	0.9725	0.9483	0.9211	0.9713	0.9435	0.5850
-0.5	0.3027	0.3000	0.4856	0.4061	0.4909	0.5616	0.2174
-0.3	0.1453	0.1380	0.2485	0.1944	0.2634	0.2332	0.1417
-0.2	0.1152	0.0955	0.1417	0.1190	0.1739	0.1490	0.1034
-0.1	0.0955	0.0915	0.1074	0.0874	0.1113	0.0955	0.0792
0.1	0.0955	0.0955	0.0833	0.0955	0.0955	0.0955	0.0915
0.2	0.1190	0.1229	0.1267	0.1305	0.1342	0.1490	0.1190
0.3	0.1774	0.1809	0.1843	0.2077	0.2142	0.2206	0.1490
0.5	0.3657	0.3610	0.3186	0.5224	0.4507	0.4731	0.2575
1.0	0.9842	0.9844	0.8499	0.9687	0.9661	0.9599	0.4767