Anomaly-based detection of malicious activity in in-vehicle networks

Adrian Taylor

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
University of Ottawa

Anomaly-based detection of malicious activity in in-vehicle networks

Adrian Taylor

Abstract

Modern automobiles have been proven vulnerable to hacking by security researchers. By exploiting vulnerabilities in the car's external interfaces, attackers can access a car's controller area network (CAN) bus and cause malicious effects. We seek to detect these attacks on the bus as a last line of defence against automotive cyber attacks. The CAN bus standard defines a low-level message structure, upon which manufacturers layer their own proprietary command protocols; attacks must similarly be tailored for their target. This variability makes intrusion detection methods difficult to apply to the automotive CAN bus. Nevertheless, the bus traffic is generated by machines; thus we hypothesize that it can be characterized with machine learning, and that attacks produce anomalous traffic. Our goals are to show that anomaly detection trained without understanding of the message contents can detect attacks, and to create a framework for understanding how the characteristics of a novel attack can be used to predict its detectability.

We developed a model that describes attacks based on their effect on bus traffic, informed by a review of published material on car hacking in combination with analysis of CAN traffic from a 2012 Subaru Impreza. The model specifies three high-level categories of effects: attacks that insert foreign packets, attacks that affect packet timing, and attacks that only modify data within packets. Foreign packet attacks are trivially detectable. For timing-based anomalies, we developed features suitable for one-class classification methods. For packet stream data word anomalies, we adapted recurrent neural networks and multivariate Markov model methods to sequence anomaly detection and compared their performance. We conducted experiments to evaluate our

detection methods with special attention to the trade-off between precision and recall, given that a practical system requires a very low false alarm rate. The methods were evaluated by synthesizing anomalies within each attack category, parameterized to adjust their covertness. We generalize from the results to enable prediction of detection rates for new attacks using these methods.

Acknowledgements

Of course I want to thank my advisors. Thanks to Nathalie Japkowicz for taking me on midway through this PhD. In our regular meetings she always had helpful, honest, and direct feedback that put me back on course when I was heading down rabbit holes. She guided me through narrowing my focus and turning ideas into products, and her feedback on my drafts always cut to the heart of what the document needed. I am also grateful to my co-advisor, Sylvain Leblanc for his feedback on my writing, and for keeping my work grounded in the security community. Dr. Leblanc, along with Scott Knight and Ron Smith at RMC, and Peter Mason at DRDC, were instrumental in helping me find a topic that suited both my employer's and my own research interests. Thanks also to Emil Petriu, who shepherded me through the early years and requirements of my PhD. His enthusiasm and strategic advice was of great help as I developed my thesis topic, even after circumstances necessitated a transition to new advisors. And I appreciate the time my committee put into evaluations, and their feedback on the work.

I owe a great debt to the managers at Defence Research & Development Canada who allowed me make this research a part of my day job; thanks in particular to Julie Tremblay-Lutter, Paul Hubbard, Julie Tremblay, Peter Mason, and Kathryn Perrett for their approvals of time and financial support over the years. Also I am grateful for the help of my colleague Francois Bernier. My many discussions with Francois were both enjoyable and helped to shape this work. Additionally, his team's technical

support was instrumental in getting me started with data gathering. Thanks also to Tricia Willink, and all my other colleagues at DRDC Ottawa, for their advice and moral support.

My friends and family provided a great deal of support over the years. I greatly enjoyed the conversations I had with my office mates Colin Bellinger and Shiven Sharma. Hearing about their work and discussing my ideas with them was great fun, and seeing how they completed their dissertations was inspirational as I hit my final stretch. Thanks to my boys Jackson, Luke, and Eric, for all the fun outside of work, and understanding why I was busy sometimes. Thanks also to Bob and Joy for all the help that allowed me to get things done, and for the moral support. Thanks as well to the rest of my family and friends who put up with a very long process, and were there to celebrate with me when it was done.

Most importantly I want to thank my wife Sue, for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automobiles have evolved from purely mechanical devices to connected computing platforms. Like any computer, these embedded control systems have security vulnerabilities. Historically the computers in cars were isolated from the outside world, so their security was of little concern. But in the last decade cars have become ever more connected to the outside world, through cellular, wireless networking, and other digital connections, exposing them to hacking. Hackers and security researchers have since demonstrated many attacks allowing for control of cyber physical systems in the car, through virtually every connection to the vehicle. Attacks work by leveraging weaknesses in the chain of connections from the outside world to the Electronic Control Units (ECUs) that control the physical parts of the car. ECUs communicate internally over the Controller Area Network (CAN) bus, a control system networking standard designed for vehicles and industrial control systems. Once a hacker has obtained access to a device connected to the CAN bus, control messages can be sent to cause specific effects. ECUs typically accept any properly formatted messages without authentication, making it relatively easy to control the vehicle in this manner [1, 2]. However, any attack will modify CAN bus traffic, and such modifications can be detected.

The detection of attack signatures and anomalies in other kinds of network traffic has a long history [3] and has been well-studied [4]. There are two broad approaches to intrusion detection: signature based, and anomaly based, where anomaly-based approaches include both behaviour-based and specification-based methods. Signature detection is the most common method employed in production networks; it requires knowledge of how specific attacks manifest within network protocol traffic. Anomaly detection methods attempt to characterize normal traffic and detect deviations from normality. This characterization can be programmed or learned [5]. Programmed methods, also known as whitelisting, use rules defining permitted behaviour to identify rule-breaking that is presumed to be associated with attacks. They require knowledge of a protocol's intended use. The learned anomaly detection approach uses examples of normal data to automatically characterize what is allowed. Learned anomaly detection has historically been difficult to use because it is difficult to identify attacks without generating a large number of false alarms. Typically human operators must evaluate anomaly detection-based alarms to investigate their cause. In practice, all these approaches can be combined to account for each others' shortfalls.

CAN bus traffic has constraints that favour learning-based anomaly detection over the other methods. CAN bus traffic is proprietary to each vehicle and model year, and so is difficult to characterize at the protocol level without inside knowledge of the specific vehicle. This proprietary nature of the protocol also makes it difficult to devise signatures for attacks, because attacks must be customized for each year and model of vehicle, and as such will change frequently. However CAN bus traffic is machine-generated, and more predictable than typical network traffic. This predictability suggests learned anomaly detection may be an appropriate fit for the CAN bus domain. Additionally, learning-based anomaly detection can work without knowing the communication protocols at all, so long as the method of characterizing

normal traffic is capable of learning entirely from examples. Finally, anomaly detection methods can detect novel attacks; this is an important feature when *every* attack is novel.

In this chapter, we first refine our problem statement and outline our proposed approach. Next we summarize our contributions, and provide an outline for the rest of this dissertation.

1.1 Problem statement

Our goal is to detect cyber attacks in cars when they appear on the CAN bus. More fully, we seek to fully characterize the problem of detecting attacks on the automotive CAN bus, devise methods for performing such detection, and evaluate the effectiveness of those methods. To reach these goal, we must answer the following questions:

- **What information is available.** What information can be obtained about traffic on the automotive CAN bus? What features can be derived that are useful for detecting attacks?
- **How can we generalize attacks?** How do attacks manifest on the CAN bus? How can we describe them in a systematic way, so that we can evaluate methods for detecting them without having actual examples of specific attack traffic? We must categorize attacks signatures observed given bus/data characteristics.
- **What anomaly detection algorithms are suitable?** Given the available information from the CAN bus and knowledge of attack signatures, what algorithms are suitable for detecting attacks, and how well do they perform?

Next we describe each of these questions in more detail, and explain why they are difficult to answer.

1.1.1 Information available

We must justify the assumptions that underly our decision to use anomaly detection to detect attacks. Our assumptions supporting anomaly detection as an approach are:

1. Normal traffic is routine and predictable, so outliers will be nonexistent or very rare.
2. Cyber attack traffic will be anomalous.
3. Anomalies resulting from attacks can be distinguished from other sources.

We cannot unequivocally verify all of these assumptions. We address the first two points in Chapter 2. The third is more difficult to confirm, but the trials in Chapter 5 provide some evidence that normal traffic does not contain the same kinds of anomalies produced by attacks.

We are further constrained in gathering information in that we do not know the application-layer protocol used by our test vehicle. Every car uses the same low-level CAN bus protocol, but the control traffic targeted by hackers is at a higher application layer. The application layer is where the control words and meaning of packets are defined. It is this application layer protocol that is specific to each model and year of automobile. While knowledge of the application protocol could be helpful for an anomaly detector, one that does not require such knowledge would be of greater value. Therefore we constrain the anomaly detector to be ignorant of this protocol. This constraint makes it easier to adapt the detector for any model of automobile.

Thus we must analyze CAN bus traffic data to determine what features are possible to extract without knowing its meaning. We must also decide, based on the data characteristics, what kinds of methods are appropriate for anomaly detection. We address all these questions in Section 2.3.

1.1.2 Generalizing attacks

In order to create test traffic for our methods, we have to understand how attacks work and how they manifest on the CAN bus. We must generalize these effects, and reproduce them by modifying our data. We propose an attack framework in Section 2.5 that characterizes any possible effect we have found in reviewing the literature on attack traffic. These effects can be implemented without knowledge of a specific protocol. Any actual attack can be characterized as an instance of specific parameters within the framework.

1.1.3 Detecting anomalies

Given the data available, and understanding of the attack framework, we must devise methods for detecting examples of attacks. This is challenging because CAN bus traffic has a time component as well as a high-dimensional digital sequence component. We solve this problem by proposing a two-part anomaly detector. The first part identifies time-based anomalies, and the second part detects data sequence anomalies. By splitting the problem into two parts, we can build specialized detectors that each focus on one aspect of anomalous behaviour. The detectors are described in Chapter 4, and evaluated in Chapter 5.

1.2 Related work

Previous approaches to detecting attacks on the CAN bus have mainly been based on timing information. CAN packets are normally transmitted at a regular frequency, motivating frequency detection as a defence against most attacks [6, 7]; such methods have been shown by us and others to be very effective at detecting inserted or missing packets [8–10].

A different time-based method uses clock skew to authenticate ECUs [11]. Many

attacks originate from an unusual source: either a foreign ECU, or a compromised one that normally sends different messages. An ECU sending malicious packets can be identified because its clock skew is different from the legitimate ECU. However this method could be defeated by an adversary able to match clock skews, and it cannot detect attacks when the compromised ECU is the normal emitter of the IDs of the malicious packets.

Fewer methods have investigated detecting attacks within packet data. Packet message entropy was used to detect insertion attacks [12], but the method was not evaluated against attacks that affect only the message contents of a packet stream. Another approach automatically classifies fields in CAN messages and measures valid ranges based on previous data [13]. However it also was not evaluated with any attack scenarios. We have previously demonstrated an early version of detecting anomalies within message traffic [14]. The detection methods presented here expand upon that work by considering complete CAN bus traffic instead of individual data streams; additionally we have developed a more complete attack framework for evaluating the methods. To our knowledge, ours is the first implementation of packet data sequence data anomaly detection for automotive CAN bus traffic.

The approach we develop here includes frequency-based and data-sequence-based anomaly detection methods. Authentication is a useful method that is complementary to our approach. Data field classification is an example of a protocol-based method that is also complementary to our approach. Field classification is however limited in that it can only detect anomalies in individual words, and not anomalies that are contextual within a sequence of words, for example, turning the wheel quickly while the car is driving at high speed. We do however make use of data field classification to generate anomalies within the attack framework.

1.3 Contributions

We have made the following contributions:

- Attack framework. Through thorough review of the literature on automobile hacking, we have generalized the features of different attacks in terms of how they manifest on the CAN bus. We can map any given attack to a parameterized simulation in our attack framework.
- Anomaly detection methods. With an understanding of CAN data and the attack framework, we identified features suitable for detecting attacks, and adapted anomaly detection methods suitable for working with those features. Traditional methods for sequence anomaly detection have been applied to much simpler problems; the nature and volume of traffic on the CAN bus creates significant implementation challenges. We have overcome these challenges to produce useful detectors that are practically implementable, for detecting both packet frequency and data sequence anomalies.
- Attack detection prediction. We have exhaustively evaluated our anomaly detection methods over ranges of simulated attacks to determine the limits of detection performance. Any future attacks can be characterized according to the attack framework parameters, and its detectability predicted for any given combination of parameters.

1.4 Roadmap

The structure of this document is as follows. Chapter 2 addresses the first two questions in our problem statement: what are the characteristics of CAN bus data and examples of attacks, and develops the attack framework. The third question is addressed in the remaining chapters. Chapter 3 reviews methods for anomaly detection and justifies the choices made for algorithms selected for detecting CAN bus attacks.

Chapter 4 describes the selected methods and how they were applied to CAN bus anomaly detection in detail. Chapter 5 describes experiments using simulated data to validate this approach, and results on sensitivity of detection methods to the parameters of the attack framework. Chapter 6 contains conclusions and avenues for future work.

Chapter 2

CAN data and attack framework

2.1 Introduction

In this chapter we describe traffic on the CAN bus, how it has been hacked, and propose a framework for describing those hacks. We begin with a detailed description of the CAN bus, reviewing how it is designed, followed by analysis of bus traffic from a vehicle. Next, we describe how car hacking works, and how it manifests on the CAN bus. Finally, we create a parameterized framework describing each attack effect, and describe how we use that framework to simulate attack signatures with real CAN bus data. The framework and attack simulations create the data used for later experiments.

2.2 The Controller Area Network Bus

In cars and trucks, the embedded computers that control the vehicle communicate with each other over a network. Cars contain embedded computers for controlling physical systems like the engine, anti-lock brakes, cruise control, etc. These computers are called electronic control units, or ECUs. Early vehicles wired these systems to each other and to the user interface controls (e.g. steering wheel, gas pedal) with dedicated

lines. Over time, it became more efficient to connect all the devices to a single shared bus. The Controller Area Network (CAN) bus is a an interface standard used in many vehicles worldwide for this purpose.

Most cars have two CAN buses. One is high-speed, on the order of 500 kbps, and is used primarily for systems controlling the car's movement, such as the engine, steering, and brakes. The second bus is low-speed, on the order of 125 kbps, and is used primarily for cabin features in the car, such as the radio, door locks and window controls, etc. The two buses can communicate through a gateway ECU. They are segregated to allow for more overall communications capacity, and to a lesser extent for security.

In North American vehicles, the CAN bus can be accessed through an on board diagnostics (OBD) port conforming to a specification known as OBD-II. Part of the specification requires that the ODB-II port is located within two feet of the steering wheel. The OBD-II port connector has, among other features, pins exposing the high-speed CAN bus. Some manufacturers also provide access to the low-speed CAN bus through optional pins; the Ford Explorer offers such a connection, but our Subaru Impreza does not.

2.2.1 CAN bus packets

Data is transmitted on the CAN bus in packets. The bus is serial and asynchronous; devices on the bus do not synchronize their clocks. Arbitration is handled by bus interface devices detecting whether the bus is busy or free, using a priority mechanism described below. The CAN bus is a broadcast medium. Without any routing mechanism, it is the responsibility of the receiver to determine which messages are of interest. Most CAN bus hardware interfaces offer filtering capabilities to limit the packets its owner receives.

The CAN standard defines a low-level packet structure illustrated in Figure 2.1.

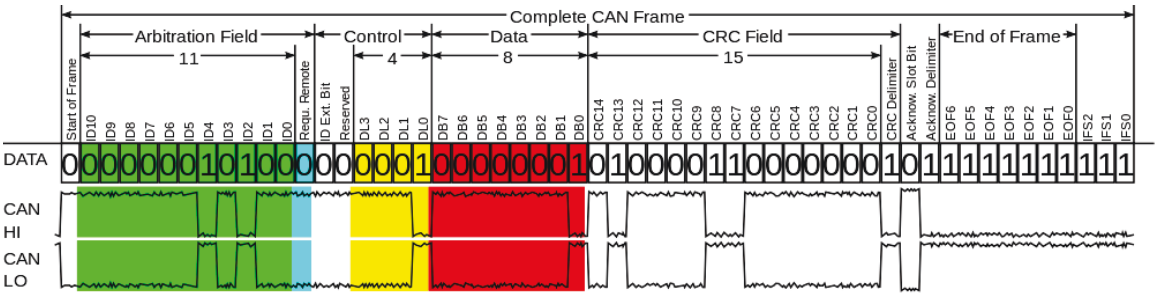


Figure 2.1: CAN bus message format. "CAN-Bus-frame in base format without stuff-bits" by Endres licensed under Creative Commons Attribution-ShareAlike 3.0 Unported, `https://en.wikipedia.org/wiki/CAN_bus`.

How these packets are used is up to the system designer; additional protocols can be defined within the data contents of the packets, or the low-level CAN packets can be used directly. The CAN bus is a control network so it is designed for speed, reliability, and robustness.

Many of the fields in the packet are used for by the CAN interface hardware. We are concerned only with the application-specific contents of the message: the ID and data fields. The ID is used to identify the packet, and the data field contains the information used by packet consumers.

2.2.1.1 Packet ID field and bus priority

The first bits in a packet comprise the ID of the message. IDs may be 11 or 29 bits long, depending on the mode used. The ID is normally used to identify the message, but it also controls bus priority. Lower ID values have higher priority. If an ECU begins transmitting a packet and senses that another ECU is driving the bus low, the first ECU will yield the bus because the second has a higher priority. Managing the transmission schedules of all the devices sharing a bus is the job of the engineers designing the car, and there is a host of tools and methods for making these systems work well together [15].

2.2.1.2 Packet data field

The existence of a particular packet at a given time can in itself be informative, but most packets also contain a data payload. The protocol used to transmit information using these payloads is specific to each vehicle. These data protocols are not publicly available; manufacturers sell access to the data dictionaries at their discretion. Furthermore, the protocols can change between models and even model years. Security researchers and car hacking hobbyists typically resort to reverse-engineering the protocols by observing relationships between traffic and the vehicle's parameters and behaviour [16].

In this work we assume no knowledge of the true meaning of messages. Nevertheless it is instructive to look at some examples to understand the kind of information that is transmitted. On the high speed bus, examples of parameters are the steering wheel angle, transmission gear, and engine RPMs [6]. Depending on the nature of the message, these values might indicate the status or be control messages. For example, a cruise control system must send accelerator commands to maintain a set speed. Such messages are fundamental to causing the cyber-physical effects created by car hacks.

2.3 CAN Data Analysis

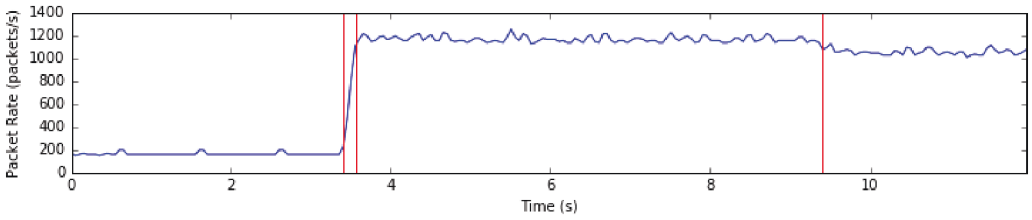
To gain further insight into the nature of CAN bus traffic, we analyzed packet traces from a live CAN bus. We captured data from two vehicles: a small amount of data from a 2011 Ford Explorer, and a large amount of high-speed bus data from a 2012 Subaru Impreza. The data was captured from a USB-to-CAN device connected to the OBD-II port of each vehicle. On the Explorer, both high and low speed buses were exposed. On the Subaru, only the high speed bus was available on the OBD-II port. We captured approximately 30 minutes of bus traffic from the Explorer, and 23

hours of bus traffic while driving the Impreza. The Explorer data was recorded during five short drives; in each one, the car was started, driven through the same route at low speed, and returned to the starting point. The Impreza data was collected during morning and evening commutes. Each drive was approximately 20 minutes long, and included a mix of driving in traffic, on side streets, and on highways. We focus on the Impreza for analysis (and later, experiments). Features of the Explorer buses are discussed as an indication of similarity and differences between manufacturers.

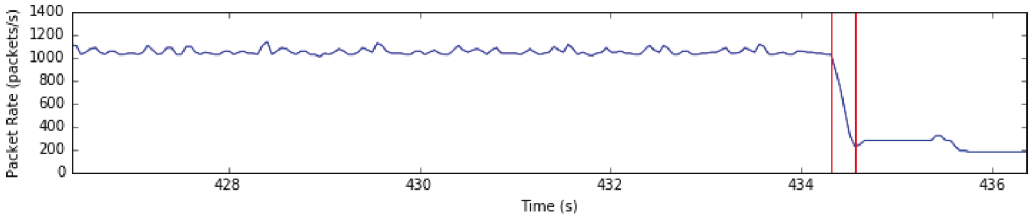
2.3.1 Packet rate

In both vehicles, the bus traffic rate before the car is started and while it is running is significantly different. Additionally, after the vehicle is turned off, a subset of IDs continue to appear for some time. Figure 2.2 shows a per-second rolling packet count at the beginning and end of a test drive on the Ford Explorer. The segments where the packet data transitions to and from the running period are marked. The Impreza high speed bus exhibited similar behaviour. For the remainder of the analysis and experiments we focus exclusively on data from the middle running period of each vehicle.

Table 2.1 summarizes information about the IDs and overall data rate on all three buses. This data is drawn from the period where the cars are running and their bus packet rates have stabilized. The overall packet rate is very consistent during this running period for all the buses. The two high speed buses run at the same rate, so the Impreza has fewer packets and is slightly more variable than the Explorer. The low speed bus runs at a quarter of the rate of the other buses, and has a commensurately lower packet rate. All the bus packet rates are very consistent, though the Impreza is slightly less so than the Explorer buses. The Impreza has slightly more than half as many IDs as the the Explorer. There is clearly room for variation in how buses are used by different engineering teams. Nevertheless, both high speed buses have packet



(a) Beginning.



(b) End

Figure 2.2: High speed bus rolling packet rate (packets/s). The rate was calculated over a 0.1s rolling window.

rates of the same order; it is likely that the design constraints on modern automobiles in turn constrain CAN bus usage to be similar for different manufacturers.

The main conclusion from this data is that the overall packet rates are fixed under normal circumstances. Next we look at the rates of individual IDs.

2.3.2 Packet periods

Almost every ID on the buses was observed to be periodic with a fixed frequency. Table 2.2 shows the specific periods observed on the different buses, and the number of IDs sharing those periods. These periods were in nearly all cases fixed over all data observed. Occasional deviations were observed for some IDs; these were composed of pairs of late and then early packets. It is not clear whether these events are occurring on the bus or are errors introduced in the data capture hardware or software. However without additional bus capture hardware we have no choice but to treat these events as examples of normal bus behaviour. The Impreza bus IDs have a smaller range of periods than the other buses, and its periods are even divisors of 10. The consistency of packet frequencies is important for anomaly detection, because it suggests that

Table 2.1: Statistics about overall packet counts on the Impreza, and Explorer CAN buses.

Packet Count Statistic (packets/s)	Impreza High Speed	Explorer High Speed	Explorer Low Speed
mean	813.0	1054.3	151.9
min	779.0	1049.0	145.0
25%	813.0	1053.0	151.0
50%	813.0	1054.0	152.0
75%	813.0	1056.0	153.0
max	822.0	1059.0	158.0
N_{IDs}	20	34	39

rate deviations of any particular ID can be detected with high confidence.

2.3.3 Data field analysis

We next turn our attention to the data fields payloads of packets, which we will also refer to as data words. Almost every ID contains 64-bit data words. How much of this possible word space is actually used by each ID? At the observed packet rates on the Impreza high speed bus, there is enough space in 64 bits to produce a novel word for each packet for the entire lifetime of the car (and were it to run continuously, for trillions of years). Furthermore, without going to the trouble of reverse-engineering the meaning of packets, can we identify patterns in bit usage that indicate something about packet structure? Next we answer both these questions.

2.3.3.1 Data symbol dictionary size

The way the data packets are used by each ID has an impact on how they can be characterized. Part of our motivation is to assess the suitability of anomaly detection methods that work with sequences of symbols drawn from a finite alphabets. Are sequences of data words drawn from a finite alphabet, or are they always changing?

A simple way to evaluate their variability is to count the number of unique words

Table 2.2: Number of IDs with each period. N/A IDs were only observed once per capture. Each period was calculated during high speed traffic only.

Period (s)	Impreza High Speed	Explorer High Speed	Explorer Low Speed
0.08	0	1	0
0.01	3	3	0
0.016	0	2	0
0.02	8	7	0
0.03	0	0	2
0.05	5	4	1
0.1	1	6	3
0.5	1	2	1
0.54	0	0	6
1	2	9	23
5	0	0	3
N/A	0	0	8
Total	20	34	47

observed for each ID. Table 2.3 shows the number and percentage of unique words observed over all the Impreza data captures for each ID. The number of unique words is highly variable for the different IDs, ranging from exactly one unique word to every possible word being unique. Given that this data contains sensor readings we expect it to be noisy and highly variable. However only one ID produces a novel word for every packet, and most are well below the full possible rate. Nevertheless, the number of words encountered in the available data suggests that finite-dictionary symbol methods will be difficult to apply to this problem for all IDs.

The table also reports how many bits were used by each ID. Even though most IDs transmit a full 64 bits, many of these bits were found to be constant over the entire data set.

2.3.3.2 Data symbol dictionary rate of growth

Counting unique words provides some information about data word usage, but not how this usage evolves over time. If the unique words shown above are all seen in

Table 2.3: Total and unique word counts on the Impreza high speed bus over nearly 24 hours of driving.

ID	Period (s)	N_{bits}	N_{packets}	N_{unique}	$\%_{\text{unique}}$
002	0.01	30	8519638	74787	0.8778
0D0	0.02	63	4258458	1467647	34.4643
0D1	0.02	18	4258532	26962	0.6331
0D2	0.02	16	4258448	35	0.0008
0D3	0.02	27	4258435	33573	0.7884
0D4	0.02	48	4258422	1410096	33.1131
140	0.01	53	8517960	3911659	45.9225
141	0.01	46	8517937	4879407	57.2839
144	0.02	24	4258954	15808	0.3712
152	0.02	19	4258160	681	0.0160
154	1.0	5	87125	12	0.0138
156	0.02	9	4258499	65	0.0015
282	0.05	32	1703417	279187	16.3898
360	0.05	45	1703595	1174020	68.9143
361	0.05	31	1703590	839256	49.2640
370	0.05	27	1702792	78864	4.6315
372	0.1	6	851809	32	0.0038
374	1.0	6	85362	11	0.0129
376	0.05	0	1703283	1	0.0001
660	0.5	30	170331	170331	100.0000

the first few hours of driving, we might reconsider finite-dictionary methods under the assumption that no new words will appear after some initial observation period. Figure 2.3 shows the rate of new words per packet for eight of the high-speed IDs (these particular IDs are identified as having the most variable data among the high-frequency IDs; details explaining this assessment are given in Section 2.3.3.4). The rate of new word growth is continuous throughout the duration of the data captures, suggesting it will continue in the future.

2.3.3.3 Data word fields

Finally we investigate how data words are used by each ID. Reverse engineering the meaning of fields within each word is possible but time-consuming. Fortunately Markovitz and Wool have published an algorithm designed to automate the discovery of field types in CAN bus data [13]. The algorithm works by considering every possible subset of bits as an individual field. Each candidate field is awarded a score based on its length and the amount of variation for the field in the data history. A greedy search algorithm chooses the combination of candidate fields that covers the full 64 bits of data with the highest combination of scores. It is impossible to know if the algorithm's output is correct, but it appears to produce a reasonable guess for field divisions.

The algorithm classifies possible fields as one of counter/sensor, constant, and multi-value. Constant fields are always the same value. Multi-value fields are long enough to potentially have a large number of values, but take on only a select few of them (e.g. an 8 bit field that takes on only 4 values out of a possible 256). The actual maximum number of possible values is configurable. Counter/sensor fields are those that do not qualify as constant or multi-value, i.e. they take many values. Additionally, we added a step to detect obvious counters within sensor fields. To detect counters, for each candidate field we calculate every packet-to-packet difference over the data

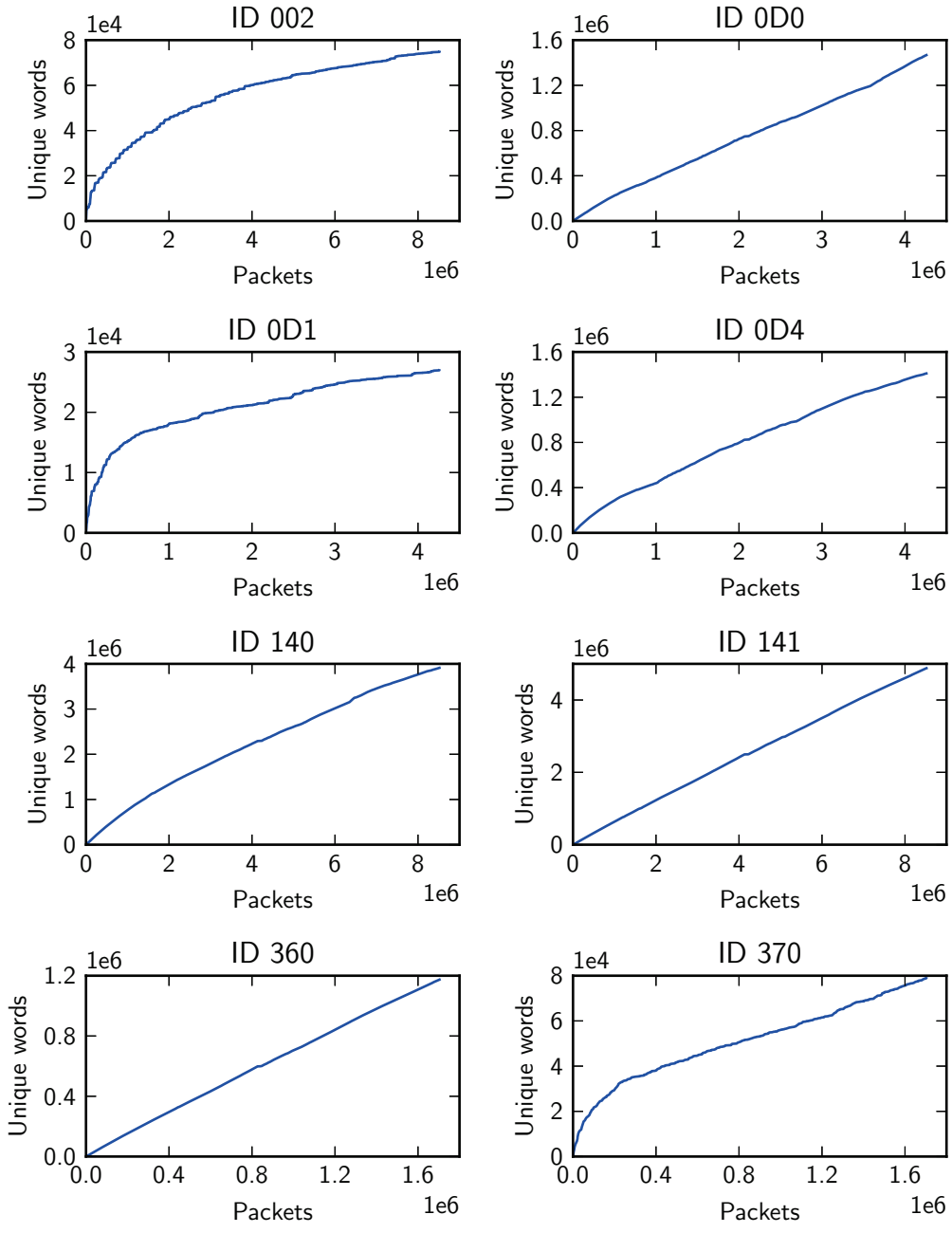


Figure 2.3: Unique data word growth over packets for selected high-frequency, high-data IDs.

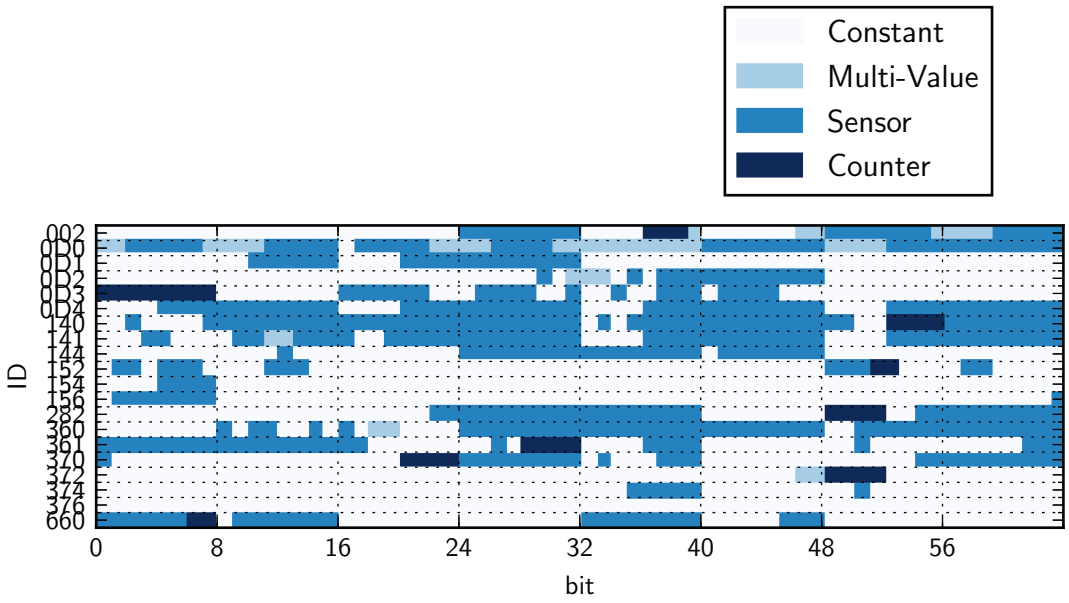


Figure 2.4: Fields identified by the Markovitz and Wool algorithm in Impreza high speed data.

history. Any fields that only ever increment by one or overflow back to zero are classified as counters.

We implemented the algorithm and applied it to the Impreza bus data. Its output is illustrated in Figure 2.4. First we note that there a great many unused bits, as was also noted in Table 2.3. Unused bits can be trivially monitored for changes, and otherwise excluded from the input to data anomaly detectors. Of the used bits, a small number are counters and multi-value fields. The majority of used bits are classified as sensor fields; these take many values and do not follow a simple pattern.

2.3.3.4 High, medium, and low variability sensor fields

We performed additional analysis on the sensor fields to attempt to further understand their behaviour. For each sensor field identified above, the number of observed unique values was counted. The unique value tallies revealed three patterns of usage. Low-variability fields are defined as those producing produce under 100 unique values. Medium-variability fields were defined as those producing between 100 and 500 unique

values. High-variability fields were defined as those producing more than 500 unique values; the minimum number was in fact over 2000. Tables showing which fields qualified for each designation, as well as field minimum, maximum, and number of unique values, are shown in Section A. We can draw no further conclusions about the nature of these fields. However we will return to the low, medium, and high variability categories of fields when designing the experiments in Chapter 5.

2.3.4 Data analysis summary

In summary, we have discovered the following information that will inform the attack framework.

- Each ID has a fixed frequency of occurrence on the bus.
- Many bits in each data sequence are constant, and we can exclude from data anomaly detection models to improve efficiency.
- The symbol alphabet for each data sequence is effectively infinite, restricting the kinds of models we can use for data anomaly detection.
- We have identified and categorized fields within each ID's data payload traffic with the aid of an automated discovery algorithm. This description of the field structure is used for simulating attacks for later experiments.

2.4 Background on car hacking

To understand and characterize the car hacking threat, we conducted an exhaustive literature review of car hacking research. The review Included papers from both academic security researchers (e.g. Checkoway et al [1]) and what information was available from hacking enthusiasts (e.g. Illera [17]. From each source we extracted a summary of each specific attack that was described, and characterized it in terms of how it affected traffic on the CAN bus. A complete summary of the attacks affecting

normal data traffic review is provided in Appendix B. In this section we summarize these findings in terms of the overall approach to car hacking, in order to inform the development of the attack framework below.

There are three steps to gaining unauthorized control of a vehicle [1, 7];

1. Obtain access through an external interface.
2. Obtain the means to communicate with the CAN bus.
3. Send messages that cause the desired effect.

The attack framework is entirely focused on the CAN bus, which is related to the effects in step 3. However, we review the entire process in order to provide context for the overall problem.

2.4.1 Hacking step 1: obtain access

The first step in an attack is obtaining access to the vehicle. Modern vehicles have many access points, providing an equal number of opportunities for attackers. Access can be obtained with a direct physical connection, via short range wireless channels, and long range wireless channels. Next we give examples of each type.

2.4.1.1 Physical access

Physical access to the vehicle is the simplest method; the OBD-II port can be used to inject traffic directly into the bus. The popularity of DIY single board computers has made this very inexpensive – a home-made $20 device is capable of injecting CAN packets [18]. Other avenues include counterfeit or compromised aftermarket or repair parts, or compromised diagnostic equipment at the repair shop [1]. Koscher et al even demonstrated an attack using a specially crafted WMA music file played on the in-car CD player [19]. The USB and iPod ports provide other entry points with potential vulnerabilities. Using these methods, multiple teams have demonstrated overriding

security controls to reflash ECUs [6,19], providing the opportunity to inject or monitor CAN traffic without leaving any physical traces.

2.4.1.2 Short range wireless access

Modern cars have many short-range wireless connections. Almost ubiquitous are: bluetooth (for connecting audio devices and cell phones to the radio), wireless tire pressure monitoring systems (TPMS), and remote keyless entry. TPMS and remote keyless entry have very limited functionality and so have limited utility as vectors for infiltrating the CAN bus. Wi-fi connections, intended for connecting passengers to the Internet through cellular links, are becoming more common. Bluetooth and WiFi have been shown to be vulnerable to attack. Checkoway et al demonstrated installing malware via a Bluetooth-connected device. They then showed several methods for connecting through the bluetooth interface: with malware installed on an already-paired Android phone, and with a method they developed for unauthorized pairing. Valasek and Miller demonstrated unauthorized access to the WiFi on a 2015 Jeep Cherokee exploiting a weakness in its password generation protocol [7]. Once connected, they were able to install malware and exploit the CAN bus.

Car sensors can also be used as access points. Roufa et al reverse engineered a wireless tire pressure sensor, and was able to exfiltrate private data as well as spoof it with erroneous values [20]. Shoukry et al demonstrated an attack on an anti-lock braking system by inserting a physical device near the brake sensor [21].

2.4.1.3 Long range wireless access

Many cars now have long range wireless channels. A simple example is the FM radio; modern radios can decode digital information from an FM signal containing, for example, song titles. This channel is not suitable for entry to the vehicle's systems but can be used as a triggering mechanism for already-embedded malware. Telemat-

ics systems such as GM's OnStar service use built-in cellular radios to communicate with the car wherever there is cell service. Newer vehicles can be constantly connected to the Internet via these channels. This connection was exploited by Miller and Valasek when they hacked a Jeep Cherokee remotely [2]. Years before, Checkoway et al developed a method for bypassing their test vehicle's authentication measures and installing malware on the telematics unit [1]. Aftermarket telematics units, designed to plug in to ODB-II ports and communicate over cellular channels for fleet tracking or insurance purposes, have also been shown to be vulnerable to attack [22].

2.4.2 Hacking step 2: communicate with CAN bus

Once the attack connects to the vehicle, they must find a way to transmit messages on the CAN bus. This requires control of a device connected directly to the bus. Most of the methods discussed in the previous section culminate in executing arbitrary code on a device in the car. ECUs are generally designed to be reflashed with new code (e.g. by mechanics) to accommodate updates and bug fixes. Such reflashing procedures have been subverted in order to install unauthorized malware [6, 16, 19]. Reflashing can occur on the CAN bus or via other means. If it is performed on the bus, then CAN packets will be visible to an observer. If it is performed via side channels, e.g. the USB port of a radio, the process will be impossible to detect from the CAN bus.

In other cases, the external compromised component is not directly connected to the CAN bus, but communicates with it through an intermediary connection. In the Jeep head unit, separate chips are responsible for the telematics and CAN bus interfaces. An application programming interface allowed for a limited amount of interaction with the CAN bus from the telematics unit. In this case the attackers were able to reprogram the CAN controller enabling it to send arbitrary messages [2]. Bypassing authentication to reflash ECUs is a complex process and we do not review it

in detail here. It is sufficient to note that researchers have demonstrated the subversion of almost every ECU in a vehicle [1]. Therefore, we may assume that it is possible that any given ECU may be compromised.

2.4.3 Hacking step 3: interact with CAN bus to cause cyber-physical effects

Once bus access is obtained, the attacker can produce various effects by injecting packets on the bus. A simple attack floods the network with high-priority packets; other ECUs cannot communicate resulting in a denial of service attack. The response of the other ECUs to an interruption of expected data or the inability to publish is unpredictable but generally destructive [6]. Other approaches are more subtle and interesting and we restrict further discussion to non-catastrophic bus attacks. Hackers have used two kinds of CAN packets to cause effects: diagnostic and normal messages. Diagnostic packets are used for maintenance and are not normally seen while driving. Other effects have been demonstrated using packets crafted to appear as legitimate normal traffic, but containing malicious instructions.

Diagnostic packets. Sending diagnostic or test packets outside of a service session is an obvious example of messages inappropriate for the vehicle state. Diagnostic packets are supposed to be protected by an authentication mechanism. This procedure validates that the tool sending the commands possesses a shared secret key. However this protection has been bypassed with a variety of methods [6] [1] [19]. Furthermore, while modules should not respond to these commands while the car is driving, they often do respond. Some examples include:

- An Engine Control Unit: accepted "disable CAN communications" and reflash commands while driving [19].

- Engage brakes, disable brakes (at low speed), kill the engine (any speed), reprogram ECUs, destroy the engine by redlining it [6].
- With the car stopped: lock/unlock doors, control blower, lights, seat belt motors [6].

Normal packets. Normal packets can also be used maliciously, e.g. to cause effects that distract the driver or affect her ability to control the vehicle. Examples include:

- Switch off headlights, setting off alarms, rolling windows up and down, accessing anti-lock brake or emergency brake systems [18].
- Open windows: rogue ECU sends packets to open the window, pressing the switch to raise it will stop its rise but not restore it [23].
- Suppress warning lights when unauthorized entry triggers car alarm by sending packets to turn them off continuously [23].
- Display incorrect speed and RPM, increase odometer reading, incorrect warning lights, apply the brakes, and direct a lane-assist feature to turn the steering wheel up to five degrees [6].

Some of these attacks depend on the hardware used in the car. For example applying the brakes may depend on the presence of an automated collision-avoidance system; in a simpler car, the the brakes might be hardwired to the brake pedal. A complete list of normal attacks found in reviewing the literature is presented in Appendix B.

We do not create specific attacks in this work. Developing an attack for a vehicle requires a great deal of expertise and time. Rather our goal is to distill the characteristics of these attacks into a framework we can use to simulate attack signatures to evaluate our detection methods.

2.4.4 Car hacking analysis summary

The important points to draw from this section are as follows: First, attacks are accomplished through the introduction of new packets, the absence of expected packets, or the modification of data of normal packets. Unlike some other kinds of network traffic, the addition or absence of packets is itself anomalous. New packets may also contain data that is unusual in the operating context.

Second, attack packets using an ID i can come from the device normally responsible for that ID, a different device on the bus, or a new device introduced to the bus by an attacker. This second distinction is important to note because it is possible to identify the normal transmitter of a kind of packet and detect when it has changed [11]. We focus on methods can detect attacks regardless of the source of the packet.

2.5 Attack framework

We present a framework describing and parameterizing the kinds of effects on CAN bus traffic produced by the attacks above. The effects of an attack will be a composition of frequency effects and data effects. Frequency effects are in the context of all known IDs being periodic with fixed frequencies. They are defined as insertions of extra packets, or the erasure of expected packets. Data effects are most broadly defined by the data contents of messages being in some way unusual for the operating context of the vehicle. Next we describe on the the frequency and data effects in more detail.

2.5.1 Frequency effects

The frequencies of normal packets are very consistent. So anomalies in terms of frequencies will involve additional packets, or missing packets that were expected. Thus

packet frequency effects are described entirely as insertions or erasures.

2.5.1.1 Insertions: extra packets

The majority of attacks involve inserted packets with specific IDs and data, designed to cause some cyber-physical effect. Insertions can be quantified with four basic parameters: the insertion packet ID i, its normal period t_i, the insertion multiplier rate r, and the duration of the event d. In addition, the data contents of the inserted messages must be valid and will be designed to cause some effect. However here we focus entirely on packet frequency effects. If the specific inserted data is important for a particular use of the framework, the insertion can be combined with one of the data modifications of Section 2.5.2. The inserted packet period is $\hat{t}_i = t_i/r$. The total number of inserted packets will be approximately $d/\hat{t}_i = rd/(t_i)$

Adding additional packets can have secondary effects. If the new packets have higher priority than other normal packets, and their insertion rate is high enough, they can preempt those normal packets. Thus we may also see other IDs with missing packets when the malicious packets are inserted. The exact effects will depend on the priority of the insertion ID and the other bus traffic.

2.5.1.2 Erasures: missing packets

Some attacks can manifest as the absence of packets that should arrive at regular intervals. For example, an attacker can suppress an ECU by intentionally interfering with it, then replace its messages using a rogue device on the bus. Alternatively a short period of missing messages may indicate that an ECU has been compromised internally.

Missing packets can be quantified with just two parameters: the missing packet ID i, and the duration of its absence d. At the start of the event, all expected packets of ID i are missing for duration d. If the period of the affected packet is t_i, approximately

d/t_i total packets will be missed. After d seconds, the normal packet stream resumes, possibly at a different phase than before. The new packet stream may originate from an imposter, or it may be the original ECU reasserting its message stream.

2.5.2 Data: Altering packet data contents

The second main signature of attacks is a change in the data sequence of some iD. This unusual data may be sent within additional packets on the bus, or it could be contained in an otherwise normal packet stream, for example if a compromised ECU is triggered externally to change from sending benign to malicious packets. Because inserted packets have obvious signatures on the CAN bus, here we focus on pure data changes that do not involve extra packets. These data-only attacks may qualify as mimicry attacks [24], in that they have identical packet timing to normal traffic. The only indication of an attack is that the data values, which are valid, are nonetheless dangerous for the current operating context of the vehicle.

Attacks generally work by setting control values within data fields to cause malicious effects. These can be single values, such as setting the power steering angle to a maximum, or may need to be a sequence of values in order to have an effect. Alternatively, entire packet sequences can be replayed to cause effects due to their being out of context. In either case the malicious data must result in valid messages, otherwise it will have no effect. Consequently, the nature of the attack will be apparent only given the context of the vehicle's operation. For example switching the transmission to reverse is not by itself a problem, unless the vehicle is travelling at 100 kph.

We define two broad types of data modifications: data replays, and field modifications. In a data replay, the data field sequence for some ID is replaced with an out of context sequence. For field modifications, the sequence of values in some subfield of an ID's data is modified in one of various ways. It is challenging to describe these changes in a generic way because they are specific to the meaning of each field in an

particular automobile. Nevertheless, although not all data modifications are attacks, all attacks are composed of data modifications of the types above. Next we describe both types of effects and their parameters.

2.5.2.1 Data replay

In a replay attack, the attacker records data traffic known to be the cause of some outcome. Taking control of the relevant ID, the attacker replays the data with the goal of causing the historical outcome. The attacker need not understand how the traffic works, making this a potentially very simple way to cause an effect. We assume the attacker has taken control of an ECU that normally emits the packet data being replayed, or is able to silence that ECU and replace its traffic with the replay. The only indication of the attack is that the data sequence of the ID being replayed has changed from one context to another.

Data replay sequences are parameterized only by the ID i. At a trigger point, the data sequence for ID i is replaced by a valid sequence from another point in time. The replaced data is a legitimate subsequence, but incongruous with preceding data sequence, as well as the ensuing traffic from the rest of the IDs.

2.5.2.2 Data field modifications

Any other data modifications are classified as field changes. The fields are those identified by the algorithm in Section 2.3.3.3 above. These kinds of modifications describe attacks that set values within data fields to cause effects.

As we argued above, it is impossible to describe valid field modifications in a generic way. However all field modifications can be described by the following parameters. Given an ID i, a field within that ID's data protocol f, for a duration d seconds that field is modified in one of the following ways:

- Constant: the field is set to a constant value, such as the maximum or minimum

possible value, or to an arbitrary constant.

- Sequence: the field is set to a sequence, for example a replayed sequence of field values, or an arbitrary sequence.

Any field modification can be described in this way.

2.5.3 Attack framework summary

The attack framework describes how attacks manifest on the CAN bus. The frequency effects of packet insertions and deletions describe the majority of published attacks at the packet level. The data effects describe how data values can be changed in attack traffic in as much detail as possible while still being a generic description. This framework is used to simulate attacks in the later experiments; details for how each effect is simulated are given in Section 5.1.2.

2.6 Summary

In this chapter we have described how the CAN bus works, how attacks on automobiles are implemented, and how we model the signatures of the attacks on the bus in a parameterized fashion. Armed with this knowledge of the normal working environment, and how attacks affect this environment, we next develop methods for detecting attacks on the CAN bus.

Chapter 3

Relevant background

3.1 Introduction

In this chapter we review methods for anomaly detection in sequences and evaluate their utility for CAN bus anomaly detection, concluding with methods for evaluating and comparing detectors. First we define anomaly detection in sequences, and define the CAN bus anomaly detection problem in those terms as informed by the attack framework. This definition divides the problem into two parts. We review methods applicable to both parts, and argue for which methods are selected for evaluation. Finally we describe different ways methods can be evaluated.

3.2 Anomaly detection in sequences

An anomaly detector learns normal behaviour from training data and tries to identify deviations from that behaviour in test data. This is different from classification machine learners, which learn examples of all classes in their training data and learn to distinguish between them. Anomaly detection has the advantage of in theory being able to detect any kind of non-normal behaviour, where a classifier can only distinguish the classes it was trained upon. The challenge with anomaly detection is in

maintaining a low false positive rate while still detecting anomalies of interest.

Anomaly detection in sequences is a specific variant of anomaly detection. Sequence anomaly detection is different from point anomaly detection in that anomalies may be contextual in nature. That is, a single element in a sequence may not be anomalous except in relation to it's preceding steps in the sequence.

In this section we review sequence anomaly detection definitions and categories, and which of those apply to the automotive CAN bus.

3.2.1 Sequence definition

A sequence is defined as an ordered collection of symbols $S = s_1, s_2, ...$ drawn from a finite alphabet A of size $|A|$. In multivariate sequences, s_i may be a vector of symbols, each drawn from their own alphabet. If the sequence originates from a time-based process, time is dropped. In most cases the sampling times are uniform. The sequence may be finite or infinite (e.g. sampled from an ongoing process).

On the CAN bus, the sequence of packet data payloads associated with a given ID can be viewed as a multivariate binary sequence. For each separate ID, packets arrive at a steady time interval, so time may be dropped without losing information.

If the ID contains 64-bit-long words, each symbol is a length-64 vector with symbols drawn from $(0, 1)$. This sequence could also be represented as a univariate sequence with symbols drawn from some subset of the integers between 0 and 2^{64}. As we showed in Section 2.3.3.1, for most IDs this is a very large subset, making the multivariate binary interpretation more efficient.

3.2.2 Sequence anomaly detector definitions

An anomaly detector has the following aspects (based on Ma and Perkins' definition [25]):

- the knowledge base, a set of training data $\boldsymbol{T}$, which is comprised of sequences $\boldsymbol{S}$ composed of symbols from an alphabet $\boldsymbol{A}$
- a model $\boldsymbol{M}$ that represents our knowledge about the training data
- a measure for comparing new sequences to the model, $\boldsymbol{D}$
- a threshold for deciding on normality

The knowledge base is the only information available to the anomaly detector to define what is normal. It may contain only normal data, in which case the problem is termed semi-supervised. If it contains unlabelled data that could include anomalies then the problem is called unsupervised. The semi-supervised case, where training data includes only normal examples, and the goal is to find anomalies in new examples, is also called novelty detection. If in the unsupervised case the problem is to find anomalies within the unlabelled training data, the problem is also called outlier detection.

The measure for comparing new sequences to the model is specific to the model. However in most cases it is a single number. The threshold must be chosen to balance the false alarm rate against the detection rate, a decision that depends on factors specific to an application. These questions are not in the direct scope of our investigation, so we will take the approach of comparing methods in ways that are independent of the specific choice of the threshold.

3.2.3 Sequence Anomalies

Sequence anomalies are subsequences that are surprising [26]. We assume the anomaly is confined to a finite subsequence, or n-gram. A subsequence anomaly can be a foreign symbol, a foreign n-gram, or a rare n-gram. An anomaly could also be a normal n-gram appearing with a different frequency than previously seen [27]. Depending on the kind of data, n can be the length of the entire sequence.

On the CAN bus, we are interested in the anomalies defined by the attack frame-

work of Section 2.5. These are characterized by added, missing, or modified packets within a single ID's symbol stream. Most of the anomalies defined by the framework have a finite duration.

3.2.4 Sequence anomaly problem types

There are many variations on the kind of sequence and problem. A breakdown following Chandola's categories [27] is as follows:

- Sequence-based: Entire sequences are compared to one another. These are typically on the order of 1000s of symbols. An example is comparing aircraft landing control sequences to each other to identify pilot errors [28].
- Subsequence-based: test for anomalous subsequences within a longer sequence. This is also called discord detection [29].
- Pattern frequency: Identify n-grams occurring with unusual frequencies in one or more sequences.
- Online: sequences are tested one symbol at a time, and the goal is to identify anomalies as soon as they are apparent. This is applicable to both semi-supervised and unsupervised cases.
- Feature-based: Instead of evaluating sequences or subsequences directly, a different approach is to derive features from the sequence. These may include symbol rates or other statistical measure of the sequence. This approach is particularly popular with computer network traffic, as the volume of data has historically been intractable to process directly.

These problems can be framed as novelty detection or outlier detection, depending on whether there is normal labelled training data. Furthermore these categories are necessarily not exclusive of each other; e.g. credit card transaction fraud detection is an online, subsequence-based, discord detection problem. Detectors for one of these

problem definitions can usually be adapted to another type. However literature in a particular problem domain may be focused on addressing the specific challenges of that domain. Next we describe precisely which of these describes the CAN bus anomaly detection problem.

3.2.5 CAN bus anomaly detection problem definition

The overall goal is to detect attacks in a car, when they happen, while it is being driven. More precisely, the problem is detecting anomalous subsequences on the CAN bus while the car is being driven. We assume we have a large corpus of known normal data to train the detectors, so our problem is a semi-supervised anomaly detection problem, i.e. a novelty detection problem. From the data analysis in Chapter 2, part of the challenge with this problem is distinguishing the natural novelty in the data from anomalies corresponding to attacks.

For practical reasons, a production automotive anomaly detector must also work online, so that anomalies can be detected as soon as they manifest. However we will not simulate this aspect of the problem directly. All of the methods we evaluate can be adapted for online operation without substantial modification.

To use sequence methods with CAN bus packet data, the packet stream must be transformed into features or symbol sequences. The raw CAN bus data consists of a list of (ID, data payload) pairs indexed with timestamps. The anomaly detection problem is specifically focused on finding anomalies related to attacks, which we explored in detail with the attack framework of Section 2.5. The attack framework defines two broad classes of anomalies: packet-frequency effects, and data effects. We now propose separate methods of anomaly detection for each of these classes.

Anomalies where packets are added or removed are a packet frequency detection problem. To detect these anomalies, the CAN bus data capture is transformed into a feature vector that captures information about the timing of all IDs. Methods

appropriate for detecting anomalies given this feature vector are reviewed in the next section.

Data effects are changes in the bits of a data sequence of an ID. Assuming that any frequency-based anomalies will be detected using the feature vector above, the input to a data anomaly detector can be assumed to a fixed-frequency data sequence, so that the time index can be removed. Consequently the only information required for data anomaly detection is the set of data word sequences associated with each ID. Each IDs sequence can be evaluated independently, at the expense of possibly missing relationships between each IDs sequence. Treating the IDs as independent does however simplify detector design. The best representation for these sequences is as multivariate binary sequences. As was shown in Section 2.3.3, the data fields of most IDs are too varied to allow for a finite alphabet of symbol values. Thus it is most efficient to consider the sequence to be a multivariate binary one rather than a sequence of symbols. The evaluation of appropriate methods for data anomaly detection below is informed by this restriction.

3.3 Methods for detecting packet frequency anomalies

As we showed in Section 2.3.2, CAN bus traffic rates while driving are very consistent. This is true of the two vehicles whose data we have reviewed in detail, and has been noted in descriptions of traffic from other vehicles [6]. This fact, combined with the knowledge that many attacks are implemented by inserting extra packets at a high frequency, suggests a frequency-monitoring approach would be effective for detecting attack patterns. This approach was explicitly suggested by Valasek and Miller for detecting their attacks [6]. The pair later built a proof of concept detector [7], but they did not report on its performance. Our interest is in evaluating the limits of such

an approach, i.e., measuring its performance as a function of detecting parameterized deviations from normal traffic according to the attack model of Section 2.5.

To detect packet frequency changes, we measure historical frequencies and compare them with current ones. Here we adapt an approach for tracking this information in industrial control systems [30]. This approach captures timing information in feature vectors called flows. Next we describe how flows may be adapted for CAN data, and then describe some kinds of anomaly detectors that are well-suited to making decisions on the flows.

3.3.1 CAN traffic flows

In network traffic analysis, measurements about traffic between two endpoints can be described by features collectively known as a flow. For example a flow might contain the number and frequency of packets, the quantity of data exchanged, and the duration of the connection. We use a similar approach to characterize CAN bus traffic. Unlike network traffic, on the CAN bus all traffic is broadcast, the endpoints are unknown, and packets are sent on a fixed schedule that ends only when the vehicle is turned off. So the basic concept requires some modifications.

Instead of capturing all the information about a transaction between two endpoints, our CAN bus flow captures measurements over a fixed-length time window. We choose a window length t_w seconds, and for each packet ID capture the following features:

- i: The packet ID
- n: the number of packets in the flow
- μ_{t_i}: the average time difference between successive packets, i.e. the sample packet period.
- $\sigma^2_{t_i}$: the variance of the time difference between successive packets, i.e. the packet period sample variance.

We also investigated measuring the Hamming distance between successive data words but found this to be ineffective for detecting any kinds of anomalies [8]. The flow window moves in time to generate a flow vector every step time t_s; if $t_s < t_w$, the windows overlap. The packet frequency anomaly detectors then output an anomaly signal for each flow window.

3.3.2 Anomaly detectors for flows

Flow methods have been applied to machine to machine traffic such as Supervisory Control and Data Acquisition (SCADA) systems. Valdes and Cheung show how flows can be adapted to online systems with incremental updates [30] Simple flow statistics include as the average and variance of the number of data bytes in the payload and interarrival times. An anomaly signal is produced at east update, based on the root-mean-squared average of how far these two average values deviate from historical averages, scaled by the variance. On a global time interval, all flows are moved to a database of historical flows. These records are used to calculate the online score for new flows. The system works well for detecting changes in the communications patterns, and adapts to slowly varying changes in the inputs. We adapt this system for CAN bus flow vectors in Section 4.2.1.

Any model that detects point anomalies in fixed-length feature vectors can be applied to the frequency flow vectors. For example, the One Class Support Vector Machine (OCSVM) has been successfully applied to many different problems [31]. The OCSVM maps data to a high dimensional space and estimates a hyperplane maximally separating the data from the origin.

Multinomial Naïve Bayes is a popular approach that uses the most direct feature vector, a count of each symbol within a window over the sequence. A popular and effective method for text classification [32], it has been applied in other sequence domains [33–36]. Though this feature representation removes all information about

the order of symbols, it has proved relatively effective. A one-class variation of this approach showed little degradation as compared to the multiclass version, and a one-class SVM performed even better on a user command line masquerade detection [37].

Many different approaches have been tried for novelty detection [38]. A short list of methods includes neural network autoencoders, Parzen density estimate, nearest neighbour methods, modified decision trees, and others [39]. Point change detection can also be applied to this kind of problem [40]. Because OCSVMs have widely-available and high-quality software implementations, and often outperform other methods, we apply them to CAN bus flows as described in Section 4.2.2 and do not evaluate additional methods.

3.4 Methods for detecting data sequence anomalies

Next we discuss methods for detecting anomalies in individual ID packet data sequences. The input here is the sequence of 64-bit data words associated with a given ID. Thus candidate methods must be able to handle tens of binary sequences simultaneously. Methods for detecting anomalies in sequences vary widely depending on the type of detection problem and the nature of the data. The CAN bus data for a single ID is a multivariate binary sequence that is for all purposes infinitely long, without fixed start and end points, and without any obvious periodicity. Many methods are difficult to apply under these constraints. We focus our attention on three broad approaches: distance methods, Markov models, and recurrent neural networks.

3.4.1 Distance methods

Distance methods are based on comparing a given subsequence to a database of known normal subsequences using some distance measure. The distance may be uniqueness,

such as with STIDE which detects foreign n-grams [41]. Other options include pseudo Hamming distance, or string-matching distance measures such as the longest common subsequence [42]. Sequence alignment methods use string matching algorithms to compare long sequences. The goal of such an algorithm is to measure the distance between (or similarity of) two strings. Examples include the longest common subsequence [28] or the Smith-Waterman local alignment algorithm [43, 44]. These algorithms are intuitive, have low sensitivity to minor changes in sequences, and can handle long sequences of different lengths. However they are computationally expensive to compute, especially if they are used to compare a test sequence to a large database. There is a fundamental assumption for these methods that their sequences are made up of a finite number of symbols. The structure of CAN data sequences was shown in Section 2.3.3 to violate this assumption, making these methods unsuitable for direct application to our problem.

It is possible that with some adaptations distance methods could be used. Part of the problem is in maintaining a knowledge base of previously seen subsequences, in that such a database would grow continuously. If instead the structure of normal messages could be represented in a compressed way, the problem might be more tractable. Clustering is one such approach that can improve overall computational performance without sacrificing much accuracy [45]. But there is no avoiding the additional complication this introduces to the overall algorithm, and uncertainty as to whether performance deficits are due to the distance measure or the clustering algorithm.

A different distance measure uses a compression algorithm such as Lempel-Ziv to characterize normal sequences. These algorithms find common n-grams and replace them with shorter symbols. The idea is to compare the compression ratio of the test sequence compressed by itself vs. compressed with the training data. If the test data is similar to the training data is will compress better because it will be comprised of

similar n-grams. This idea has been tested on various data sets, but was not particularly effective [33, 46], so we did not evaluate it. Nevertheless, due to the simplicity of the algorithm, it could be a candidate for future comparisons with other methods.

Due to these deficits we do not pursue distance methods for detecting CAN bus data anomalies.

3.4.2 Markov models

Markov models are probabalistic methods, where measured transition probabilities are used to build a model of the process generating the sequence [33, 42, 47–49]. The anomaly score is derived from the probability that the model would have generated the sequence under test. Variations on this model can allow for variable length histories, or allow wildcards in the history. Markov models have been found to outperform n-gram methods, although not by much [47]. However they appear to be applicable to CAN bus data sequences.

One challenge is that Markov models are normally applied to low-dimensional sequences. Their parameters grow exponentially with the number of symbols and time history they use, making them difficult to apply to higher-dimensional sequences such as the CAN bus data sequences. However Ching et al have developed methods for making them tractable for multivariate, multistep sequences [50]. We describe how this adapted version is applied to CAN bus data in Section 4.3.4.

Hidden Markov Models (HMMs) allow for more complexity than Markov models with the addition of hidden states [41]. HMMs have been evaluated on many data sets [28, 42, 45]. The HMM is not well-suited to online problems or long sequences, although methods exist to overcome these limitations [51–53]. HMM models sometimes perform marginally better than simpler methods, but they are complex to configure and computationally expensive to use, leading many to question their value [27]. Furthermore they are not natively suited to high-dimensional problems such as ours. We

were unable to find a HMM implementation that could handle the dimensionality and volume of our data, and thus did not evaluate them.

3.4.3 Recurrent neural networks

RNNs are neural networks with connections from both the input and previous states. The feedback connections allow them to respond to input and hidden state values for arbitrary lengths of time. This structure makes RNNs a natural fit for time streams and variable-length sequence inputs. However, simple RNNs have a reputation for being difficult to train and have historically not performed well. Recent innovations like long short term memory (LSTM) units [54, 55], Hessian-free optimization for training [56], regularization methods [57, 58], and more powerful computers and bigger data sets have enabled RNNs to deliver state of the art performance in speech recognition and related fields [59–62]. They have also achieved competitive performance in natural language processing and language parsing [63–65]. RNNs can learn long-term associations [61, 66] and complex relationships [64]. They can be applied to many different kinds of data. Inputs can be symbolic (e.g. letters or words), real (e.g. digitized sound recordings), or multidimensional (e.g. pictures or video). Outputs can be a decision (classifiers, e.g. softmax), another sequence, a fixed length vector representation, or a real-valued signal. RNNs have also been applied to online learning problems, e.g. handwriting recognition [60].

LSTM-based RNNs have recently been shown to be effective for anomaly detection in standard time series test sets [67], electrocardiography [68], and aircraft telemetry [69]. Each of these systems trains a RNN to predict the next symbol in the sequence. Anomalies are flagged when the error between real and predicted values is high. When the series is inherently unpredictable, the LSTM encoder-decoder can produce superior results to the LSTM predictor [70]. Other variations used energy-based models [71] and variational inference [72], although it is not clear under what

circumstances one might be superior to another.

Modern RNNs architectures are well-suited to anomaly detection on CAN data. They are capable of processing the raw bits of CAN message sequences, and require no domain knowledge about the meaning of the sequences. RNNs can also scale to make sense of large amounts of data; indeed, large data sets are part of the reason they have become successful in recent years. They also have a great deal of support in various open-source libraries that simplify what would otherwise be a difficult model to develop, making it easy to leverage recent advancements in the field. Because of these strengths and the suitability of RNN methods for predicting CAN bus data sequences, we evaluate RNNs for this purpose. Details are given in Section 4.3.3.

3.5 Decision thresholds and evaluation methods

Every anomaly detector, for either type of CAN bus anomaly problem, generates a score based on its input. If the anomaly detector is effective, the score for anomalous inputs will be significantly different than it is for normal inputs. Choosing a threshold for this score allows for a decision on whether the input is normal or anomalous.

A detector can be evaluated by feeding in examples of normal and anomalous inputs and analyzing its output scores. Given a threshold, measures of performance can be calculated, such as the true positive rate, false positive rate, precision, and recall. These quantities depend on kinds of decisions that can be made by a detector:

- True positive (tp): anomaly correctly identified
- False positive (fp): normal data incorrectly labelled anomalous
- True negative (tn): normal data correctly labelled normal
- False negative (fn): anomalous data incorrectly labelled normal

The true and false positive rates are calculated by counting the instances of each labelled test case and reporting the proportion of true and false positive examples.

True positive and false positive rates sum to 1, as do true negative and false negative rates, so reporting just the true and false positive rates gives a complete picture of the overall results. Scores can also be evaluated without a specific threshold using the receiver operating characteristic (ROC). Next we review the decision-related scores, threshold setting, and using ROC to evaluate performance.

3.5.1 Precision, recall, and F-score

Precision and recall are quantities defined to highlight particular facets of performance. Precision is the proportion of identified anomaly data that is a true anomaly, defined as $\frac{tp}{tp+fp}$. A precision of 1.0 corresponds to a false positive rate of zero; 0.5 to both the true positive and false positive rates being 1.0, a case where no discrimination is occurring at all; 0.0 corresponds to a zero true positive rate with no indication of the false positive rate. Recall is the proportion of ratio of identified anomaly data to actual anomaly data, defined as $\frac{tp}{tp+fn}$. Recall ranges from 0.0, where no anomalies are identified, to 1.0 where all anomalies were identified.

Precision and recall are often combined into a weighted mean called the F_β score. The F_β score is defined as:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

The F_β score is designed to attach β times the weight to precision as to recall.

3.5.2 Setting thresholds

We have so far avoided the question of how to set a threshold. In practice, the choice of threshold depends on what false alarm rate can be tolerated. For example, if an alarm will result in some action by a human operator, the true and false positive rates must be balanced such that the number of generated alerts can be processed

by the number of people available to perform the work. Whether a large number of false alarms can be tolerated will depend on the importance of finding those rare true positive events. In a vehicle there are multiple possible modes of operation. At one extreme, noting that there are no security specialists within the vehicle waiting to respond to alerts, we could set thresholds to limit the false alarm rate to essentially zero. Though this may limit the precision of the detector, it is better than nothing if the alternative is be victim to every security event. Alternatively, if alerts are sent by an Internet-connected car in real time to the manufacturer's security monitoring centre for further analysis, a higher false alarm rate could be tolerated. The nature of the anomaly is also a factor. Events that are potentially life-threatening should be acted on immediately. Other anomalies may indicate that tampering has occurred, and could serve as a warning and indication to investigate the integrity of the vehicle.

Solving these issues is beyond the scope of this work, because they depend on factors that may vary with the degree to which an automobile is connected to a network, such as what remedies are available when anomalies are detected. Other decisions require more knowledge about the nature of each packet ID than is available to us. Instead, our goal is to evaluate anomaly detection methods independently of these factors. Consequently, when a threshold is required, we can take the approach of choosing the threshold that optimizes the f_β score over some validation data. If we are most concerned with minimizing false alarms, we can choose a small β; for example, $\beta = 0.1$ selects for a tenfold extra weighting on the precision of the detector.

3.5.3 Receiver operating characteristic and area under curve

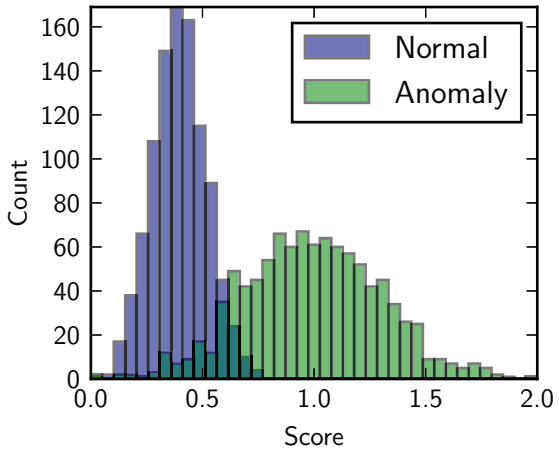
It is possible to compare methods without choosing a threshold by using the ROC and area under curve (AUC). The ROC is a plot of the true vs. false positive rate as the threshold is varied from the minimum to maximum score for the test data. The area under the ROC curve is known as the AUC, which summarizes the information in the

ROC curve with a single number. For an ideal anomaly detector, the true positive rate will be 1 and the false positive rate will be zero for some threshold, leading to a step-shaped curve with an AUC of 1.0. This case corresponds to the distribution of output values for the normal and anomaly classes being completely disjoint. In reality this is rarely the case; a more typical distribution is shown Figure 3.1a. The corresponding ROC curve is illustrated in Figure 3.1b. As the threshold varies over the overlapping output region. the true positive rate will climb from zero to one as the false positive rate climbs from zero to one. If the distributions overlap completely for the two classes, the ROC curve will be a straight line drawn from the bottom left-hand corner of the graph to the top right-hand corner, with an AUC of 0.5.

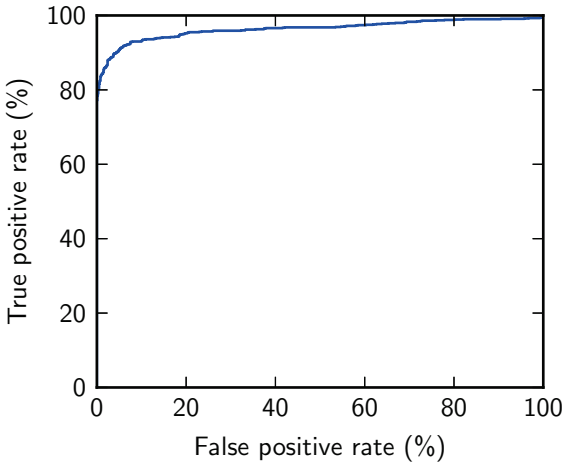
The AUC is convenient for comparing many cases, but it can obscure important information. For example, if the design goal is to have the highest true positive rate at a set false positive rate, the best solution may not have the best overall AUC. Whether this is a problem depends on the application.

3.6 Summary

In this chapter, we defined sequence anomaly detection in general, and specified what subsets of the problem are applicable to the automotive CAN bus. Given those characteristics, we defined two kinds of anomaly detection problems, the frequency-based anomaly problem and the multivariate binary sequence anomaly detection problem. For each of those problems, we reviewed relevant anomaly detection methods and selected several for evaluation. Finally we described how these methods will be evaluated. In the next chapter, we provide details about the selected methods and how they are employed on CAN bus data.



(a) A histogram showing normal and anomaly scores for some distribution.



(b) The ROC derived from Figure 3.1a.

Figure 3.1: Example score histogram with associated ROC plot.

Chapter 4

Anomaly detection methods

4.1 Introduction

This chapter contains details on the methods selected in Chapter 3 for detecting anomalies on CAN bus data. There we defined two broad classes of detectors: those that operate on flows, and those that operate on individual data sequences. The structure of the complete detector is illustrated in Figure 4.1. Each arm of the detector is designed to detect different kinds of anomalies. The flow information detector is sensitive to frequency changes in the input, but is completely blind to anomalies within the data fields of packets; data information is stripped from its input. Conversely, the data sequence detector is sensitive to data changes, but is blind to timing-related anomalies; all timing information is removed from its input. The figure shows our concept for a production system. However we evaluate the two kinds of detectors separately in Chapter 5.

In the rest of this chapter, we describe the candidate methods and overall structure of each of each type in detail. We first describe the frequency methods, followed by the data methods. For the data methods, we provide additional details on how the outputs from each ID's anomaly detector are combined to make a judgement for the

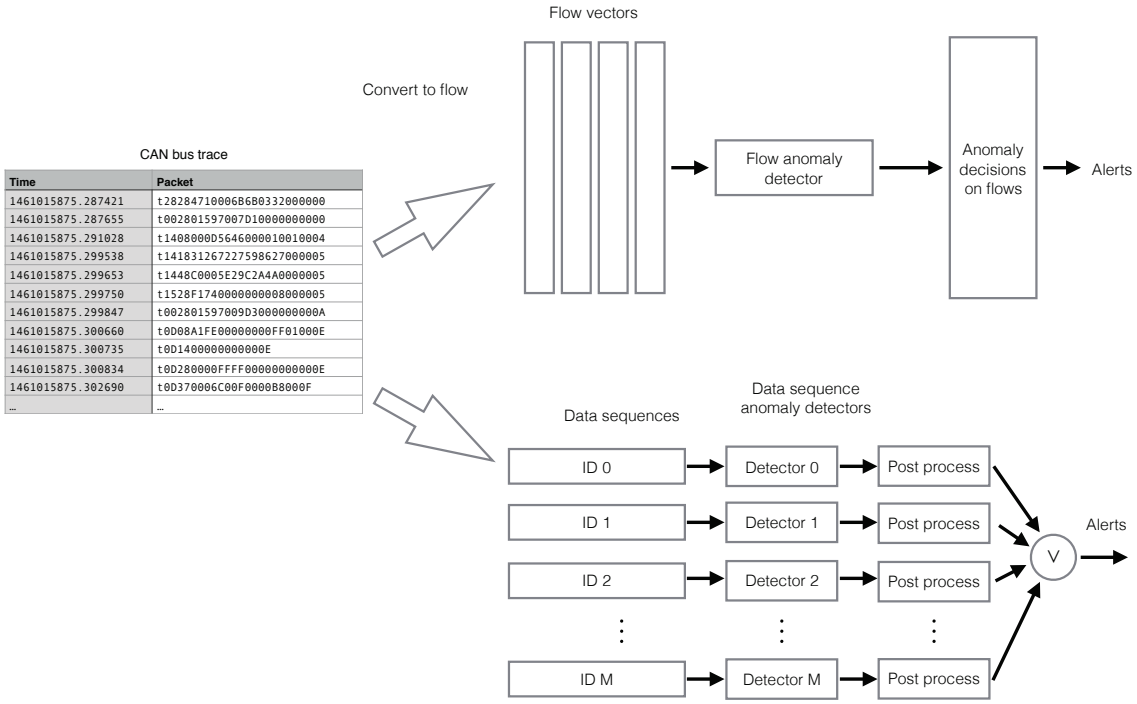


Figure 4.1: Anomaly detector pipeline. Time/frequency features are processed independently from data sequences.

entire input.

4.2 Frequency anomaly detection methods

We compare two methods for detecting anomalies using flows, following on the background research of Section 3.3.2. The first is a modified version of the t-test employed by Valdes and Cheung [30]. The second is a One-Class Support Vector Machine (OCSVM). We found as reported in the following chapter (Section 5.2) that both of these perform very well for our test cases, and so did not evaluate additional methods.

4.2.1 T-test model

The t-test model evaluates a flow by comparing each ID i 's flow mean period μ_{t_i} with its historical mean period t_{Hi} using Student's T-test:

$$T_{t_i} = \frac{\mu_{t_i} - \mu_{t_{H_i}}}{\sqrt{\frac{\sigma_{t_i}^2}{n}}} \tag{4.1}$$

Here n is the number of samples in the flow, and σ_{t_i} is the flow sample variance. Historical means are calculated from a training data set. Scores are normalized over all the IDs to be in the range $[0, 1]$ using the training data. Normalization ensures that IDs with more variable packet timing do not dominate anomaly signals originating from other IDs. Note this scaling step was not performed in our previous work [8].

This procedure produces a vector of scores, one for each ID for the window. The vector is multiplied by negative one so that normal scores are higher (i.e. closer to zero) than anomalous scores. The window score is the maximum normalized t-test value over all the IDs in the flow. If the input generates of multiple flow windows, we take the overall score for the entire input to be the minimum score over all the windows.

4.2.2 One-class support vector machine

To compare with the t-test approach, we also trained a one-class support vector machine (OCSVM) to classify flows. The OCSVM learns a decision function for a single class from training data by learning a class boundary in a transformed feature space of the input [31]. The class boundary is composed of transformed examples from the training data. It can then be used to classify new data-producings in or out of the training class. The scikit-learn Python library [73] OCSVM implementation was used for experiments in Chapter 5.

The OCSVM has several parameter choices: the kernel function, which defines

the feature space transformation, any kernel parameters, and a parameter ν that affects the decision boundary and how many training samples are used to define it. We used default values: the radial basis function kernel, which has a parameter γ controlling its size; γ was set to 0.1. We set ν to a value that would limit the number of support vectors to approximately 1000. This choice means a separate ν would be chosen for models with different numbers of training vectors, which is true when the same training set is converted into flows with different window lengths.

4.2.2.1 OCSVM inputs and outputs

The OCSVM is trained on normal flow examples. First the training data is converted into flows with given window length and step size parameters. The flow vectors must then be transformed before being input to the OCSVM. The IDs are omitted, and the mean periods, variances, and packet counts are concatenated to produce a single feature vector. The vector is normalized using the training data to scale all values between zero and one. Test vectors are also scaled using the training data scaler values.

The flow window score is the output of the decision function calculated by the OCSVM. This function measures the distance of a test input from the learned decision boundary. This results in a single score for each flow window. Lower (negative) scores are more anomalous; positive scores are inside the decision boundary of the model. As with the T-test model, for an input made up of multiple flow windows, the output is the maximum score over all the windows.

4.3 Data sequence anomaly detection

The second class of models detect anomalies within the data sequences of each ID. Each ID's data sequence is different, so a model is trained separately for each one.

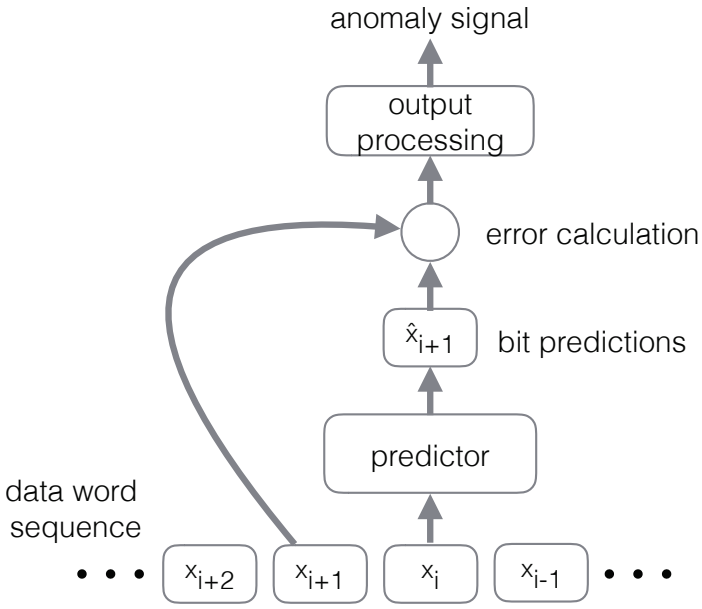


Figure 4.2: Generic system diagram for all sequence anomaly detectors. For a complete detector, a specific sequence predictor, error type, and output processing method must be chosen.

Every model uses the same approach to detect anomalies, as illustrated in Figure 4.2. The model first learns to predict the next data word based on training data. The predictions are compared to the actual word values using an error measure. We evaluated two measures: log loss, and absolute error (these are both defined below). The error values for each bit in a sequence are the basis for an anomaly decision. Once the errors are calculated, they are processed and combined to generate a single score for the input. This post-processing steps are basically the same for each model, although some parameters are optimized for each model separately.

We begin by describing how the data payload words are processed for input to the detector models, followed by a description of the error measures. Next we describe the prediction models: a recurrent neural network, a multivariate, multistep Markov chain model, and a straw-man model that predicts the previous value. Finally we describe methods for combining the bit error outputs into a single anomaly score for a given input.

4.3.1 Data sequence input format

The data sequence models directly process the sequence of packet data bits from a single ID. The input at each step in the sequence are the bits from the packet's data field. The only processing of those bits is to remove those that are constant, as identified Section 2.3.3.3. The input to the detector model is thus a binary matrix of shape (number of packets) $\times$ (number of non-constant bits).

In the experiments described in Chapter 5, the number of packets ranges from tens to a thousand examples. In a production system a method would need to be devised to either feed data into a detector continuously or with discrete windows.

4.3.2 Data sequence error signals

The output of the data sequence models are errors between the predicted and actual bits of the next packet. For each bit in an input word, the detector model outputs a prediction for the corresponding bit in the next word. This prediction is between 0 and 1 and can be interpreted as a probability of that bit's value.

We evaluate this prediction with two kinds of bit errors: absolute error and log loss. The absolute error is simply the absolute value of the difference between the bit prediction and the true value of the next bit. The binary log loss between the an actual and predicted bit is defined as:

$$L(\hat{b}_k, b_k) = -\Big(b_k \log\left(\hat{b}_k + \epsilon\right) + (1 - b_k) \log\left(1 - \hat{b}_k + \epsilon\right)\Big)$$

where b_k is the k^{th} bit in the target y_i, $\hat{b}_k$ is the k^{th} bit's predicted value by the model at step i, and ϵ is a fixed value that caps the maximum loss. Fig. 4.3 plots the log loss for a target of 0.0 (the plot is symmetrical for a target of 1.0). For the experiments in Chapter 5 $\epsilon = 10^{-15}$, which translate to a maximum bit loss of approximately 35. The log loss function is low for incorrect but middling predictions, but very high for

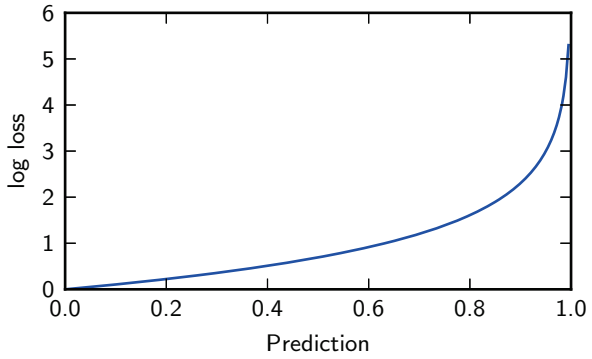


Figure 4.3: Log loss for target of 0.

confident but incorrect predictions.

4.3.3 Recurrent neural network sequence predictor

We train neural networks to predict the next packet data values. Their errors are used as a signal for detecting anomalies in the sequence. We evaluated variants of recurrent neural networks using Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRU) for predicting CAN data sequences. The basic RNN is a neural network with feedback (Fig. 4.4). The output h_t is a function of both the input x_t and the previous output h_{t-1}:

$$h_t = f(Wx_t + Uh_{t-1} + b)$$

where W, U are weight matrices, b is a bias term, and f is a nonlinear transformation (e.g. the sigmoid or tanh function). In this way RNNs make use of information from both the past and the present. However they are difficult to train. Neural networks are trained with backpropagation, a process of adjusting the weights according to the derivative of the output errors. The feedback loop in RNNs cause errors to shrink or grow exponentially, destroying the information in the backpropagation signal. LSTM units and GRUs were invented to overcome these problems; we review both types

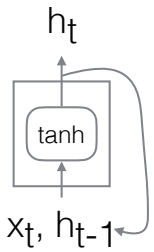


Figure 4.4: Basic RNN cell. The weight multiplications are omitted.

below. Then we explain the architectures of the networks, how they were developed, and how the networks are trained for predicting CAN bus anomalies.

4.3.3.1 Long Short Term Memory units

The LSTM unit was designed to solve the gradient problems described above with additional structure within each recurrent unit [54]. This structure is in the form of explicit input and forget gates. Figure 4.5 illustrates the LSTM cell. The input and forget gates depend on both the input x_t and previous step's output h_t, and control how much of the internal state depends on the new input and the previous state.

$$\begin{aligned} i_t =& \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ f_t =& \sigma(W_f x_t + U_f h_{t-1} + b_f) \end{aligned}$$

Here W_i, U_i, W_f, U_f are learned weights, and b_i and b_f are learned bias terms. The sigmoid function σ is typically used for the gate terms to scale their output between 0 and 1. The input and forget signals (i_t and f_t determine the new cell state c_t as a linear combination of the previous internal state c_{t-1} and new candidate internal state $\tilde{c}_t$:

$$\begin{aligned} \tilde{c}_t =& \tanh(W_c x_t + U_c h_{t-1} + b_c) \\ c_t =& f_t c_{t-1} + i\tilde{c}_t \end{aligned}$$

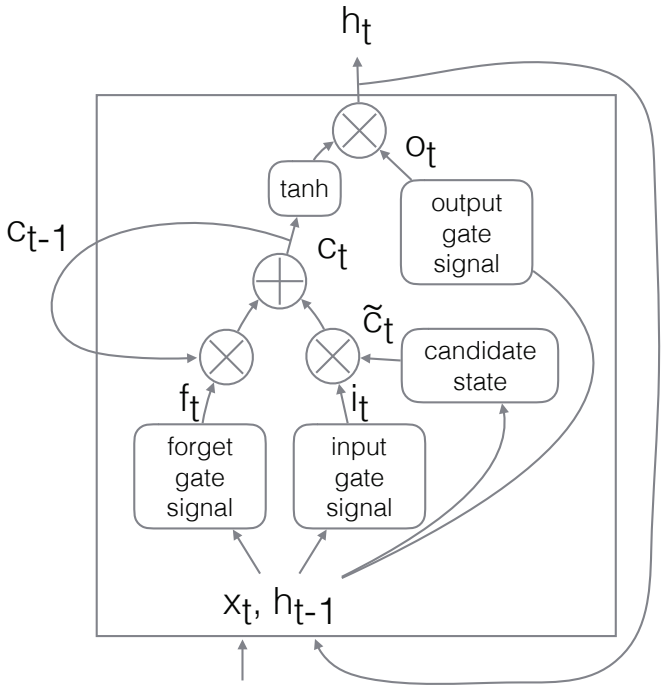


Figure 4.5: Basic LSTM unit. The LSTM adds additional connections designed to facilitate forgetting and remembering the past. The weight multiplications have been omitted for clarity in emphasizing the information flows.

where again W_c and U_c are weight matrices and b_c is the bias. The output is also controlled by a gate function with its own weights W_o, U_o and bias b_o:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \tanh(c_t)$$

The design and evaluation of LSTM variants is an active area of research; we chose the standard structure because it has been proven to work well over a range of problem domains [74].

4.3.3.2 Gated Recurrent Units

The GRU is another type of RNN, slightly simpler than the LSTM unit, but designed to solve the same problems [75]. The GRU equations are:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\tilde{h}_t &= tanh(W x_t + U(r_t \circ h_{t-1})) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t
\end{aligned}
$$

where $\circ$ denotes element-wise multiplication. Working backwards through these equations:

- h_t is the output and also hidden state of the cell. The update gate z_t controls how much it is influenced by the previous hidden state and the new candidate hidden state.
- $\tilde{h}_t$ is the new candidate hidden state of the cell. It is a function of the input and previous state. The degree to which the previous state influences the new state is controlled by the reset gate r_t.
- r_t is the reset gate.
- z_t is the update gate.

The GRU simplifies the overall structure of the LSTM and shifts the responsibilities for gating from forget, input, and output gates to the reset and update gates. Performance comparisons between the two approaches have been inconclusive and appear to depend on the specifics of each problem [55]. Therefore we decided to evaluate GRU in addition to the LSTM units.

4.3.3.3 Neural network architecture and training

One downside of neural networks is that they have a large number of parameters. The number of layers, number of hidden units, unit type, learning algorithm and its parameters, etc., all must be chosen and can have a large impact on performance. Our approach to choosing these parameters was to employ a random search algorithm [76]. The idea with random search is that for a given problem, some parameters will not make much difference, and others will have a large effect on performance. Randomly sampling this space can identify promising regions much more quickly than grid search, which will consume a great many resources retreading ineffective regions of the space. Additionally, random search is easy to implement.

We started with an architecture inspired by one that proved successful in natural language processing [77], There, the high-dimensional input space of words was mapped to a much lower-dimensional vector space that encoded words as vectors of real numbers using a multiple non-recurrent embedding layers before being fed into the recurrent layers. We similarly mapped our binary data packet words into a real vector with n_h hidden layers before they were processed by n_r recurrent layers, where both were varied from 0 to 3. Each hidden layer had m_h neurons, and the recurrent layers had m_r neurons, where m_h and m_r were varied between 32, 64, 128, 256, 512, and 1024 units. The output layer is linear; it converts the final LSTM layer outputs to a vector the same shape as the input. The type of recurrent unit could be an LSTM unit or a GRU. We also varied learning algorithms between stochastic gradient descent or Adam, an adaptive method for optimizing neural networks [78]. The networks were regularized with or without dropout, a method that turns of inputs randomly that has been found to make networks more robust [79]. The length of input sequences used for training was also varied, as one of 5, 10, 20, or 40 steps. Finally, an option to employ batch normalization was selected for all layers; this is a recent technique that normalizes the output of each layer and has been found to

improve training convergence [80]. The random parameter search was performed using a single ID. Each ID proved to have different typical log loss scores, making it difficult to randomize this aspect of the trials because comparisons were impossible using this measure. One way to improve performance may be to optimize each ID's model parameters separately.

We performed a random search over these parameters with training data, using the average log loss for the training set as an error measure. For every combination of parameters, the input was a sequence of data bits from a single ID, and the output was the log loss between the input bits and the future bits of the following word. The log loss over a separate validation set of data was used to control and evaluate the training performance. Training with a parameter set was stopped when validation performance failed to improve after three complete training data passes in a row. The validation log loss shows how well the network learned to predict a data sequence it had not seen before, which may or may not be well-correlated with its ability to detect particular kinds of anomalies.

The random search yielded a number of local minima for parameters. Initial searches found a relatively large LSTM network was most effective [14]. This network had two hidden layers with 128 units each, and two LSTM layers with 512 units each, trained on 20 steps of sequence data at a time. Later searches with an expanded parameter set identified a much smaller network, with no linear layers, and a single 256-unit GRU recurrent layer trained with 40-step inputs. We decided to evaluate both variants small and large, with both GRU and LSTM units, on the anomaly test data. The smaller networks are significantly faster to train and run approximately twice as fast as the large networks, sot they are preferable if they can provide equal performance. Each of these four variants was trained on data from all the ID used for the data sequence anomaly detection trials, using the validation data here to stop training once validation performance failed to improve three times in a row.

4.3.4 Multivariate Markov chain

We implemented a multistep, multivariate Markov chain for comparison with the RNN predictor. Markov chains are difficult to apply to high dimensional problems such as ours, because they rely on parameters that grow exponentially in time and with the number of sequences being modelled. However, Ching et al have developed methods for making them tractable [50]. We implemented their method for application to the CAN bus data inputs as described below.

4.3.4.1 Multivariate Markov chain formulation

We view an ID's packet data sequence as a collection of binary sequences. Define $x_r^{(k)}$ to be the state vector of the kth sequence at time r. Because our sequences are binary, the possible states are just 0 and 1. So x_r is $e_0 = (1,0)^T$ or $e_1 = (0,1)^T$.

Markov chain model assumes the relationship:

$$x_{r+1}^j = \sum_{k=1}^{s} \sum_{h=1}^{n} \lambda_{jk}^{(h)} P_h^{(jk)} x_{r-h+1}^{(k)}, \quad \text{for } j = 0, 1, \ldots, s-1, r = n-1, n, \ldots$$

where

$$\lambda_{jk}^{(h)} \geq, \quad 1 \leq j, k \leq s, \quad 1 \leq h \leq n, \quad \text{and} \quad \sum_{k=1}^{s} \sum_{h=1}^{n} \lambda_{jk} = 1, \quad \text{for } j = 0, 1, \ldots, s-1.$$

The terms are defined as:

- $P_h^{(jk)}$ is the h^{th} step transition probability matrix from sequence k at time $t = r - h + 1$ to sequence j at time $t = r + 1$.
- $\lambda_{jk}^{(h)}$ is the weight on the term.
- $x_{r+1}^{(k)}$ is the state probability distribution of kth sequence at step $r + 1$.

An n-step history of x is required for each of the s sequences.

For convenience, define a concatenated history vector of states as:

$$X_{r+1} = ((x_r^{(j)})^T, (x_{r-1}^{(j)})^T, \ldots, (x_{r-n+1}^{(j)})^T)^T \quad \text{for} \quad j = 0, 1, \ldots s-1$$

Then we can write the transition matrix all together. But because we have included the history state vector in the new state, the transition matrix Q is sparse. We are only depending on a weighted combination of previous states for the top of X_{r+1}, and the rest of X_{r+1} is copied from X_r. The overall equation can be written in matrix form as:

$$X_{r+1} = \begin{pmatrix} X_{r+1}^{(0)} \\ X_{r+1}^{(1)} \\ \vdots \\ X_{r+1}^{(s-1)} \end{pmatrix} = \begin{pmatrix} B^{(00)} & B^{(01)} & \ldots & B^{(0(s-1))} \\ B^{(10)} & B^{(11)} & \ldots & B^{(1(s-1))} \\ \vdots & \vdots & \vdots & \vdots \\ B^{((s-1)0)} & B^{((s-1)1)} & \ldots & B^{((s-1)(s-1))} \end{pmatrix} \begin{pmatrix} X_r^{(0)} \\ X_r^{(1)} \\ \vdots \\ X_r^{(s-1)} \end{pmatrix} = QX_r$$

Where

$$B^{(ii)} = \begin{pmatrix} \lambda_{ii}^{(1)} P_1^{(ii)} & \lambda_{ii}^{(2)} P_2^{(01)} & \ldots & \lambda_{ii}^{(n-1)} P_{n-1}^{(ii)} & \lambda_{ii}^{(n)} P_n^{(ii)} \\ I & 0 & \ldots & 0 & 0 \\ 0 & I & \ldots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & I & 0 \end{pmatrix}$$

$P_r^{(jk)}$ is a transition matrix from two to two states, so its dimension is $(2x2)$. $X_{n+1} = \begin{bmatrix} X_{n+1}^{(0)} \\ X_{n+1}^{(1)} \\ \vdots \\ X_{n+1}^{(s-1)} \end{bmatrix} \begin{bmatrix} \lambda_{00} P^{(00)} & \lambda_{01} P^{(01)} & \ldots & \lambda_{0s} P^{(0s)} \\ \lambda_{10} P^{(10)} & \lambda_{11} P^{(11)} & \ldots & \lambda_{1s} P^{(1s)} \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_{s-1,0} P^{(s-1,0)} & \lambda_{s-1,1} P^{(s-1,1)} & \ldots & \lambda_{s-1,s-1} P^{(s-1,s-1)} \end{bmatrix} \begin{bmatrix} X_n^{(0)} \\ X_n^{(1)} \\ \vdots \\ x_n^{(s-1)} \end{bmatrix} = QX_n$

To estimate the transition matrices, we need all the sequence-to-sequence transition matrices $P_r^{(jk)}$ as well as the weights λ. The $P_r^{(jk)}$ matrices are straightforward to

calculate: we simply count transitions to find the transition frequencies, then normalize to find the probabilities. The λ_{jk} estimates are more difficult. Chiang's solution is to treat the approximation $Q\hat{x} \approx \hat{x}$ as true equality. The resulting system of equations can be solved with linear programming; details of this solution are given in [50].

4.3.4.2 Multivariate Markov model formulation and training

We implemented this Markov model and applied it to CAN bus sequences using a history of one and two steps. The methods above were sufficient to enable these short histories, but creating models with longer histories was impossible with our implementation because of computer memory limits.

As with the RNN predictors, we train a new Markov model for each ID's data sequence. Both models were trained by counting transition frequencies over the training set, and applying the procedure outlined above to obtain transition probability matrices.

The output for a Markov prediction is the probable value of each bit. As with the RNN, both the log loss and absolute error measures were calculated for all output bits. These outputs were post-processed in the same was as the other data sequence models.

4.3.5 Guess previous symbol

Finally, we compare the above models against one that simply guesses the next value of each ID's data sequence to be the previous value. This test case ensures that, word to word, the sequences are not trivial in nature. If guessing the previous word performed well, it would indicate that the data sequences are slow to change. If this was true, it would be sufficient to detect anomalies by registering any unusually large numbers of bit changes. The guess model output was processed in the same way as the other models.

4.3.6 Making anomaly scores and decisions from data model outputs

Each model above generates an error score for every input bit. We are evaluating N IDs, each with some number of data words $m_i, i = 0, \ldots, N$. Each word has a number of bits b_i. So $\sum_{i=0}^{N} m_i b_i$ values must be combined into a single decision. This sum can amount to thousands of values for even a short test input. For our detector to be practical, a single scalar value or decision must be derived from all of the bit losses for all the words and IDs within a time window. In our previous work, the data sequence anomaly detectors were evaluated separately for each ID [14]. The overall score was taken to be the maximum output bit loss over the entire sequence. However if we wish to evaluate a suite of detectors operating on the complete CAN bus, we must make a decision for all of their outputs simultaneously.

Combining each model's output into a single decision is challenging. For a single ID, bit errors may vary across the data fields. The variance may be even greater for different IDs. Our approach to combining scores is first to define a serious of post-processing steps that convert a single ID's bit scores over time into a single output score. The goal of these post-processing steps is to produce a score that gives the best discrimination between anomalies and normal data. Given individual single scores, we can either take the maximum of them all, or choose individual thresholds and alarm for the complete system when any individual ID alarm is triggered. The signals could be combined in other ways across IDs, but we only consider the above two methods here.

4.3.6.1 Options for model output processing

In our previous work [14], the maximum bit loss was chosen as a decision metric. This metric was found to perform the best for the anomaly types evaluated there. The experiments in the next chapter contain additional types of data sequence anomalies,

so we evaluate additional alternatives. For simplicity the outputs from the models are processed in stages: first the bit scores are converted into a value for each data word, then the word scores into a single value for the ID and complete input sequence. If a decision is required, a threshold must also be chosen. So there are potentially four choices to be made:

- Error type: which possible model output to use as a basis for decisions (log loss or absolute error).
- Word output model: the method for deriving a single score from a vector of bit scores (maximum, scaled maximum, OCSVM, Gaussian distribution)
- Word score combination method: the method for combining the sequence of word scores into a single score for the input sequence (mean, rolling window mean, max, log sum).
- Threshold: how to choose a threshold for each IDs combined word score output.

Next we explain the reasoning behind the options evaluated for each of these steps, and how we chose the specific combination of steps for each model and ID.

4.3.6.2 Error type

Log loss was initially used because it is the most appropriate error function for training neural networks with binary data [81]. We evaluated absolute error as a linear method to contrast with log loss.

4.3.6.3 Word output model

For the word output model, we consider four different methods: maximum, scaled maximum, OCSVM, and Gaussian.

The maximum method simply chooses the highest value in the vector. We hypothesize that the maximum loss will be more sensitive to short disturbances, but will also be sensitive to outliers in normal data.

The scaled maximum is designed to reduce the effect of more highly varying bits for a given ID on the output decision. Bit loss ranges on normal data can vary between IDs, and between the bits within each IDs data word. If particular bit indices are prone to large normal loss values, they can hide meaningful signals from other bits. Scaling the output bit losses to a fixed range can mitigate this problem. The scaling values are calculated from the minimum and maximum outputs of the model over the training data. The same scaling values are then used when evaluating new test data. The word score is the maximum value from the scaled output.

Using an OCSVM is a relatively simple method to combine the bit scores within a word, in contrast to taking the single largest value as with the above methods. An OCSVM is trained on the scaled normal validation bit error outputs for each ID. The word score is the decision function output from the OCSVM. For each model and error type, an OCSVM was trained on the scaled normal validation outputs for each ID. The OCSVM was configured to use a radial basis function kernel with $\gamma = 0.1$ and ν set to maintain approximately 100 support vectors The word decision is the OCSVM decision function output.

The Gaussian method also combines bit measurements, by fitting a multivariate Gaussian to the validation set outputs. This approach was used in a LSTM-based multivariate sequence anomaly detector by Malhotra et al [67], so we evaluated it here. The score for each word is the probability density function value of the fitted Gaussian for the prediction errors.

4.3.6.4 Combining word scores

Once a score is developed for each word, it must be combined into a single score for the full input sequence. The challenge in combining word scores is balancing the sensitivity of the detector to anomalies with its resistance to noisy signals in normal data. We evaluate four methods of combining word decisions: taking the maximum,

the mean, the log sum, or a windowed mean. The log sum is defined as the average of the natural log of each value in the sequence. The combined word score is the score for the entire input sequence of the given ID. The rolling window mean calculates the average score over a sliding 0.1s time window, and returns the maximum such score over the test sequence.

The output score can be used for the evaluation of an individual ID model, e.g. using AUC. If the word scores for each ID are combined to a single anomaly score for the sequence, e.g. taking their maximum, the model and output processing steps can also be evaluated using ROC curves and AUC. Combining scores in a simple way such as this requires that all IDs use the same post-processing steps; otherwise their scores will be at different scales. If instead the decision for each ID is to be optimized, an alarm threshold must be chosen.

4.3.6.5 Choosing the alarm threshold

Finally, a threshold must be chosen for each model and ID. A good way to choose a detection threshold is to evaluate a model with test data, and choose a threshold that yields an acceptable balance of precision and recall, as was discussed in Section 3.5.2. We follow the approach of Malhotra et al [67], and choose thresholds to maximize the f_β score for each model and ID, with $\beta = 0.1$. This choice for β emphasizes precision performance approximately ten times more than recall.

These thresholds must be chosen using data containing examples of both normal and anomalous data. A separate threshold is chosen for each combination of model, ID, error type, word decision score type, and word decision combination type. The complete model must then be evaluated with additional test data to obtain an estimate of true performance. Details on the test data used for this procedure are provided in the next chapter, in Section 5.1.

4.3.6.6 Selecting post-processing steps

It is impossible to evaluate any of the methods above without validation data containing examples of anomalies. In the next chapter, we describe how such validation data was created and used to select post-processing steps for each ID and in aggregate. The validation data is also used to choose a decision threshold according to the criteria described above.

4.4 Summary

In this chapter, we have described procedures for detecting packet frequency and data sequence anomalies. Packet frequency anomalies are detected using models trained on normal CAN bus flows. Data sequence anomalies are detected with multivariate sequence predictors. These predictors are also trained with normal data, in this case containing data subsequences. Post-processing steps that convert predictions to anomaly scores were also described. The post-processing steps are selected using validation test data containing both normal and anomaly examples. Next we show results from experiments on simulated anomaly data using these models.

Chapter 5

Experiments

In this chapter we describe and present results for experiments on detecting CAN bus anomalies. We synthesized anomaly examples using the methods described in Section 2.5. Each of the detection algorithms described in Chapter 4 was evaluated using this simulated data.

Ideally these experiments would be performed with actual attack data. Unfortunately such data is not available. Furthermore, if it were available, it would be only in small quantities. Attacks are expensive to create and must be customized towards a particular make and model of car. By contrast, our approach of simulating attacks allows for a more comprehensive evaluation of detector performance. Using the attack framework as a basis for simulating anomalies allows for creating large numbers of trials with an element of randomness. Results for particular simulation parameters allows for the prediction of performance for detecting attacks with similar characteristics. While there can be uncertainty in the signature of a specific attack, if this signature is represented by some of the simulated cases then we may predict similar detection rates when it is encountered in the wild.

This chapter is organized as follows. First we explain how the data captures were processed into test data. This includes how the anomalies described in the attack

framework are simulated, and how test cases were processed for evaluation by both classes of detector. The packet frequency experiment and data sequence anomaly test results are then presented separately.

5.1 Experiment data

Experiments were conducted using data captured from the Impreza high speed CAN bus. There was insufficient data available from the Explorer low and high speed buses to obtain significant results for experiments; we captured nearly 23 hours of Impreza data, but had only 25 minutes of Explorer data. We previously performed a version of the packet frequency anomaly experiments on Explorer data [8]. These yielded similar results to those presented in this chapter, providing some confidence that our methods will perform similarly with other vehicles.

In the rest of this section we explain how the data was prepared for experiments. First we explain preprocessing steps and how the data is split into different data sets for training, validation, and testing. Then we explain how anomalies were created based on the framework of Section 2.5. Finally we explain how the data was processed for input into the flow and data sequence models.

5.1.1 Data set division and preprocessing

The data in its original state is a collection of text files containing comma separated values with a timestamp, ID, and data fields. Each file corresponds to a single drive, in most cases about 20 minutes long. We divided these files into different sets:

- Training: s_{train}, 43 files. For training models.
- Validation A: s_{va}, 5 files, normal data. For model hyperparameter optimization where applicable.
- Validation B: s_{vb}, 10 files, split into normal and simulated anomaly examples.

- Test: s_{test}, 10 files, split into normal and simulated anomaly examples.

Three additional files were reserved for obtaining data used for insertion and data replay anomalies as described below. For all the files, the used data was drawn only from the periods when the engine was running (see Section 2.3). Training and validation data was used differently by each of the models; details on these uses were given in Chapter 4.

The second validation and test sets s_{vb} and t_{test} each were further processed into normal and anomaly cases in the following manner. First, the raw data from the files was split into non-overlapping 3-second-long segments, yielding about 3200 segments in total. 1000 of these segments were randomly selected without replacement to comprise the normal data subset. The remaining 2200 segments were used as a pool to generate anomalies. For each type of anomaly, 1000 segments were randomly selected from the pool without replacement and modified to become anomalous segments. However the same pool was used to create examples of each anomaly type. There was not enough data available to use a unique segment for each anomaly. The resulting division for sets s_{vb} and t_{test} thus looks like this, with 1000 3-second-long segments per case:

- Normal segments
- Anomaly cases:
  - Anomaly insertion case 1
  - Anomaly insertion case 2
  - etc.
  - Anomaly erase case 1
  - Anomaly erase case 2
  - etc.
  - Anomaly data cases
  - etc.

In the experiments below, each anomaly case is evaluated against the single normal case separately.

Anomaly test cases were generated for each of the attack classes developed in Section 2.5. We evaluate the packet frequency anomalies and detectors separately from the data field anomalies and detectors. Frequency anomalies types are *insert* and *erase*. Data anomaly types are *data field modification* and *replay*. Most anomaly types have additional parameters, so separate test cases were created for each combination of parameters. The parameter variation enables sensitivity tests on each detector. Next we describe the details of each type of test case.

5.1.2 Simulating anomalies

Anomalies were created by simulating the kinds of effects described by the attack framework, namely insertions, erasures, data replay attacks, and data field modifications. All anomalies are created by modifying one of the 3-second-long capture segments described above.

5.1.2.1 Simulation IDs

Packet frequency anomalies were only generated for higher-frequency IDs, defined as those with periods of 0.01 or 0.02s were modified. Eleven of the twenty Impreza IDs meet this condition. There main reason for restricting attention to these IDs is that is simplifies the generation of anomalies and the experiment design. Splitting data into 3-second windows makes good use of the available data, but the slower IDs would only appear a small number of times within this window, while the faster iDs appear 150 or 300 times. To create large numbers of cases with different parameters, it is helpful to be able to assume a large number of packets will be available. We also hypothesize that the higher speed IDs are more likely to be associated with cyber physical control systems, and therefore attacks, because these will require a higher update

rate than lower-priority IDs. However we cannot evaluate this hypothesis without more examples of decoded automotive CAN bus implementations. Nevertheless, since the principles of the detectors are the same at different time scales, we assume that performance will also be similar. Repeating all the experiments with slower IDs and longer data segments would complicate them significantly with little benefit.

Data sequence IDs used for simulation were a subset of the high-frequency IDs. Of the eleven high-frequency IDs, three made only trivial use of their data field. We define trivial use to mean that fewer than one percent of the data word values were unique (see Table 2.3. The remaining eight IDs all contained examples of low, medium, and high frequency fields as identified in Section 2.3.3.4.

5.1.2.2 Insertion anomalies

Insertion anomalies are simulated by adding extra packets to a portion of the segment as described in Section 2.5.1.1. The parameters for insertion anomalies are: the period t_i, insert rate r, and duration d, and the insertion ID i and its data source. The insertion rate r is the multiplier of the period for the inserted packets. The inserted packets then have a period of t_i/r, or frequency r/t_i. Test cases were created with r values of 1 and 2. The period parameter is the normal packet period t_i for the ID i that is inserted. For each segment, an ID i with the chosen period is randomly selected for insertion.

Packets are inserted starting at a time t_s and for a duration t_d. The start time within the segment was randomly chosen, but restricted to be after the first third of the segment, and early enough that packets could be inserted for the full duration. The inserted packets were drawn as a contiguous sequence from another data capture, reserved for this purpose. Drawing the packets in this way ensures that the inserted packets contain a valid data sequence, although it is likely to be inconsistent with the rest of the bus traffic in the segment.

The average number of packet insertions is also reported in the results; it is roughly $t_d/(t_i/r)$, but depends on the specifics of packet priorities in each capture segment and can therefore vary. As described in Section 2.5.1.1, inserted packets can cause existing packets to be dropped. For this reason we also report the average number of dropped packets in each insertion test case. Test cases were created with t_d set to 0.01, 0.02, 0.1, 0.2s.

New packets are inserted into the capture record starting at t_s seconds, every t_i/r seconds, until time t_s+d. If the bus is not free at those times, the new packet is queued for insertion at the next time the bus would have been free. If an inserted packet would have interfered with a packet in the recording, we change the existing packet's timestamp to the earliest possible time after the inserted packet had occupied the bus. Because these changes can cause a cascade of modifications, we use the following three rules to adjust all the timestamps in the capture as necessary:

1. If an inserted packet would have caused an existing packet to be delayed: the existing packet time stamp is advanced to when the bus is again available.
2. If multiple packets are delayed, they are put in a queue. When the simulated bus is free, packets in the queue are put on the bus according to normal priority rules.
3. If a packet is added to the queue, and an older packet with the same ID is present, the older one is discarded.

The bus utilization is low enough that for most insertion parameter combinations evaluated here, very few packets are dropped or have their timing substantially modified.

5.1.2.3 Erase anomalies

Erase anomalies are simulated by removing packets with a given ID from a segment as described in Section 2.5.1.2. The parameters for *erase* anomalies are: period t_i and

duration t_d. As with insert anomalies, the start time is chosen randomly for each segment, but restricted to be after the first third of the segment, but early enough to allow for the full requested duration.

The period parameter is the period t_i of ID i that is chosen for removal. As with the insertion case, packet erasures are evaluated for different ID periods separately. For each segment, an ID with the given period is randomly selected for erasure.

As with the insertion case, packets are erased starting at a randomly chosen start time t_s and for the duration t_d. The start time is again randomly chosen and restricted to be after the first third of the segment and early enough that $t_s + t_d$ is not after the end of the segment. The average number of packet deletions are reported for each test case; they are all approximately equal to t_d/t_i. Test cases were created with t_d set to 0.01, 0.02, 0.1, 0.2, and 0.3 seconds.

5.1.2.4 Replay anomalies

Replay anomalies, defined in Section 2.5.2.1, are created by replacing data for the given ID with a data subsequence captured at a different time. Replay anomalies create discontinuities in data sequences. This replay attack also provides a way to simulate an anomaly on the bus that is only anomalous in context. The new data sequence is guaranteed to be normal in all ways except that it is discontinuous with the previous data subsequence of the given ID.

The only parameter for the *replay* case is the list of IDs that are candidates for modification. We restrict chosen anomalies to be those that are high frequency (i.e. have a period of 0.01 or 0.02s), and that also have non-trivial data sequences. IDs with trivial data sequences are those with a very low percentage of unique values in Table 2.3. They are omitted because they tend to vary so little that the replay anomaly is not applicable to their data sequence.

As with the other anomaly cases, the start time for the anomaly t_s is chosen

randomly to be in between the first and third second of the segment. From t_s to the end of the segment, the data payloads of the identified ID are replaced with different data. The new data sequence is drawn from a random starting point in another capture file. The new data is thus composed of contiguous samples from the same ID.

5.1.2.5 Data field anomalies

Data field anomalies cases are created by modifying just the bits within a single field for an ID as defined in Section 2.5.2.2. For each ID, fields were identified using the algorithm of Section 2.3.3.3. The parameters for field anomalies are: the ID to modify i, field category, modification method, and duration d seconds.

Fields selected for modification are restricted to sensor fields. The field category is the categorization developed in Section 2.3.3.4: low, medium, or high variability. By making field variability a parameter, its effect on anomaly detectability can be measured. For a given anomaly case, we specify the field type, and then randomly select a field of that type from the tables in Section A. The selection is also restricted to fields within high-frequency IDs. Durations were set at 0.2, 0.5, 1, and 1.5 seconds. Separate test cases were created for each of these methods and for each duration.

The field modification method is one of the following:

- maximum: the field is set to its maximum possible value
- minimum: the field is set to its minimum possible value
- random constant: the field is set to a randomly chosen value
- random: for each affected packet in the sequence, the field is set to a new random value
- replay: the field is replaced with data from the same field captured at a different time. Unlike the data replay in the preceding section, only the field data is replaced. The rest of the data payload is left unchanged.

As with the other duration-limited cases, the start time t_s is randomly chosen,

starting no earlier than one third of the way into the segment, and early enough to accommodate the full duration. For a combination of parameters, a field is chosen randomly, and its data is replaced from t_s to $t_s + d$ according to the selected method.

5.1.3 Anomaly detector data processing

Each test segment, normal or anomalous, must be converted into a data format appropriate for the anomaly detectors. The packet frequency detectors process flow vectors, which are derived from timing information within a segment. The data sequence detectors process individual data payload sequences from each ID separately. Next we provide details on these transformations, first for the packet frequency methods, then for the data sequence methods.

5.1.3.1 Flow parameters and model training

The packet-frequency anomaly detectors all evaluate CAN bus flows as defined in Section 3.3.1. Flows have two parameters, window size t_w and step size t_s. The window size can affect the model performance, so we treat each combination of flow model and window length as a separate method in order to evaluate its effect. We expect longer windows to produce more reliable statistics, but a shorter window will produce a results faster, so it is worth exploring this relationship between window length and performance. We evaluated methods with windows lengths of 1.0, 0.5, and 0.2 seconds. With a step length $t_s = t_w/2$, this produced six, twelve, or twenty flows respectively for each three-second test segment.

The OCSVM and T-test models were trained using the full training set s_{train}. We used the scikit-learn OCSVM implementation [73]. Scikit-learn provides a Python-language interface for the LIBSVM project [82], a well-supported SVM library.

5.1.3.2 Data sequence preprocessing and model outputs

The data sequence models were all trained using the full training set. The Validation A set s_{va} was used by the recurrent neural network models for hyperparameter selection as well as detecting training convergence. The Validation B set s_{vb} was used by all the models to optimize the post-processing steps and decision thresholds as described in Section 4.3.6. The performance results reported below are on the Test set s_{test}.

5.2 Packet frequency experiment results

Here we present results on the packet frequency anomalies. Performance is reported in terms of AUC.

5.2.1 Insertions

Insertion cases are parameterized by the natural period of the ID, the multiplier to this period of the inserted packets, and the duration of the insertion. The point of these parameters is both to describe the insertion and determine which are most important for predicting detection accuracy. Results for all flow methods for IDs with period 0.01 are given in Table 5.1, and for period 0.02 in Table 5.2. In both tables the cases are sorted according to the total number of packets inserted.

For each native period (0.01 and 0.02s), the factor that predicts the confidence in detection is the actual number of inserted packets. For every detector method the AUC increases almost monotonically with the average number of added packets. The flow window is twice the length of the shortest insertion duration. Since the window steps are precisely this length, at least one of the flow windows will contain all the extra packets. Thus it is not surprising that the strength of the anomaly signal increases with the number of inserted packets.

In almost every test case and flow window size, the t-test method dominates the

Table 5.1: Insertion test AUC for flows with 0.01s natural period IDs, sorted by the mean number of added packets.

μ_{added}	insert period (s)	dur (s)	insert multiplier	OCSVM 0.2s	OCSVM 0.5s	OCSVM 1.0s	t-test 0.2s	t-test 0.5s	t-test 1.0s
1.455	0.010	0.01	1.0	0.4924	0.8957	0.9685	0.9049	0.8357	0.9055
2.442	0.010	0.02	1.0	0.5059	0.8974	0.9690	0.9914	0.9729	0.9877
2.454	0.005	0.01	2.0	0.5191	0.8958	0.9690	0.9945	0.9778	0.9901
4.387	0.005	0.02	2.0	0.6464	0.9016	0.9718	0.9989	0.9946	0.9997
10.289	0.010	0.10	1.0	0.9824	0.9911	0.9960	1.0000	1.0000	1.0000
19.989	0.005	0.10	2.0	0.9962	0.9965	0.9975	1.0000	1.0000	1.0000
20.074	0.010	0.20	1.0	0.9960	0.9964	0.9976	1.0000	1.0000	1.0000
39.696	0.005	0.20	2.0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Table 5.2: Insertion test AUC for flows with 0.02s natural period IDs, sorted by the mean number of added packets.

μ_{added}	period (s)	dur (s)	insert multiplier	OCSVM 0.2s	OCSVM 0.5s	OCSVM 1.0s	t-test 0.2s	t-test 0.5s	t-test 1.0s
0.990	0.02	0.01	1.0	0.4986	0.8957	0.9686	0.6605	0.9225	0.9865
1.465	0.02	0.02	1.0	0.5378	0.8966	0.9701	0.7811	0.9520	0.9933
1.479	0.02	0.01	2.0	0.5612	0.8991	0.9709	0.7759	0.9559	0.9949
2.477	0.02	0.02	2.0	0.6892	0.9157	0.9781	0.9280	0.9926	0.9997
5.361	0.02	0.10	1.0	0.9770	0.9924	0.9956	0.9986	0.9992	1.0000
10.311	0.02	0.20	1.0	0.9958	0.9976	0.9983	1.0000	1.0000	1.0000
10.350	0.02	0.10	2.0	0.9964	0.9976	0.9982	1.0000	1.0000	1.0000
20.138	0.02	0.20	2.0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

OCSVM performance. In our previous results, the opposite was true: the OCSVM was vastly superior to the t-test method [8]. The improvement here is due to the normalization of the t-test output; in the previous experiment, the lack of normalization meant IDs with larger timing variance dominated the decision making, sometimes obscuring anomalies with lower-variance IDs. The use of normalization here rectified this problem. Nevertheless it is surprising that a simple statistical test outperformed the robust OCSVM.

5.2.2 Erase results

Erase case results for flow methods are shown in Table 5.3. Here test cases for each natural ID period are given in the same table because there are fewer cases than for the insertion trials. Cases are sorted according to the number of dropped packets within each period class.

Again there is a trend for improved performance as the average number of dropped packets increases. However here the OCSVM is the superior model for all window sizes. It perfectly detects most cases involving more than ten dropped packets, even for a very short detection window. This is in contrast to the insertion cases, where the T-test methods outperformed the OCSVM.

5.2.3 Flow method conclusions

The t-test method with a 1s window was able to perfectly detect most of the insertion cases. The only disadvantage of a 1 second window is that it takes a full second to detect the anomaly. To detect anomalies faster, we lose discrimination power, as the shorter window detectors only match the longer window reliability with more added or missing packets. However the OCSVM methods were superior for the erase cases. Even single packet insertions and deletions can be detected with a long enough observation period. Attacks that affect at least 10 or 20 packets can be detected perfectly by one

Table 5.3: Erase test AUC results. Test cases are sorted by erased packet period, then number of dropped packets.

μ_{dropped}	period (s)	dur (s)	OCSVM 0.2s	OCSVM 0.5s	OCSVM 1.0s	t-test 0.2s	t-test 0.5s	t-test 1.0s
1.037	0.01	0.01	0.4842	0.8992	0.9688	0.6890	0.6269	0.7868
2.007	0.01	0.02	0.4884	0.8962	0.9685	0.7211	0.7240	0.9215
10.020	0.01	0.10	0.9961	0.9959	0.9960	0.7425	0.7656	0.9645
20.009	0.01	0.20	1.0000	1.0000	1.0000	0.7522	0.7658	0.9659
29.996	0.01	0.30	1.0000	1.0000	1.0000	0.7520	0.7658	0.9662
1.001	0.02	0.01	0.4893	0.8984	0.9694	0.5449	0.8069	0.9698
1.013	0.02	0.02	0.4975	0.8978	0.9692	0.5535	0.8134	0.9693
5.003	0.02	0.10	0.9963	0.9956	0.9959	0.5536	0.8252	0.9757
10.002	0.02	0.20	1.0000	1.0000	1.0000	0.5698	0.8249	0.9784
15.003	0.02	0.30	1.0000	1.0000	1.0000	0.5699	0.8224	0.9793

of the two methods. Practically, any attack that affects packet frequencies involves many more packets than 20, and often at a faster rate.

Because neither model dominates performance, judging which one is most useful is difficult. Most attacks involve insertions, although erasures may indicate tampering with an ECU, an event of serious concern. However a choice does not necessarily have to be made. The t-test method has very low computational overhead. The OCSVM takes more resources but with judicious parameter choices can also be run cheaply. Unless computational resources are very limited, the best approach would be to run all of the t-test methods simultaneously, along with the 0.2s window OCSVM. The majority of events will be detected quickly with the short window methods, and any that are missed will be detected by a longer method. For the The OCSVM with a 0.2s window detected all erase events as reliably as those with longer windows. We conclude that with these relatively simple methods, attacks affecting packet frequencies can be detected with high confidence.

5.3 Data modification results

The data modification cases include the replay case, and the five types of field modification. There is only one replay case, because it has no parameters. The field modifications are parametrized by field type (low, medium, or high variation) and duration of modification (0.2, 0.5, 1.0, or 1.5 seconds), for a total of 60 test cases.

There are two main goals in how we interpret the experiment results: comparing model performance, and evaluating the effect of the parameters. In addition we will evaluate the model performance on each ID under test individually. These three factors are evaluated primarily using AUC. We first determine the best post-processing step combination for each model using s_{vb}, and determine test results with that combination to address the above questions. We also use the validation set s_{vb} to choose decision thresholds for each model and ID, and evaluate performance under that threshold decision as an indication of how the existing system would perform in a vehicle.

We first describe the chosen post-processing results, then discuss performance on the two types of test cases.

5.3.1 Post-processing configuration

The output of the models above is a prediction for each input bit, which is converted into both a loss score and absolute error for each output bit. For each model, these outputs were processed to yield a single anomaly score for each ID within the test sequence following the methods described in Section 4.3.6. As described in Section 4.3.6.6, each model was evaluated on s_{vb} with every possible combination of post-processors. The top three post-processor configuration and corresponding AUC for each model is given in Table 5.4. Both the overall AUC and the AUC for each individual ID is given. First we note that for the RNN, simply taking the max value

continues to dominate as in our previous work [14]. Here the error and log loss measures give almost identical performance, with the error performing slightly better overall. The Markov models and Guess model all have the best performance with the combination of log loss, scaler, and max. However the Guess model has better performance than the Markov chains overall. The top-performing post-processing combination was selected for each model for evaluation on s_{test} below.

Looking at the individual ID scores, performance is generally significantly better than the overall AUC. The overall score in each case is close to its worst individual ID's AUC. Combining the detectors results in overall performance at the level of its weakest link.

In terms of ranking models, the RNNs are clearly the winners, with the large LSTM slightly leading all other models. The Markov models in overall terms are actually worse than simply guessing the previous value. However for some IDs they do perform significantly better, suggesting that they are not entirely without merit.

Table 5.4: AUC of the top three post-processing step combinations for each model over all the validation test cases. Both overall and per-ID AUCs are shown.

Model name	Error type	Output model	Score type	Overall	002	0D0	0D1	0D4	140	141	360	370
Guess previous	logloss	scaler	max	0.5878	0.5000	0.5290	0.6050	0.7586	0.5559	0.5000	0.6793	0.6649
Guess previous	logloss	scaler	window	0.5849	0.5000	0.5313	0.5792	0.7199	0.5560	0.5156	0.6793	0.6647
Guess previous	error	scaler	max	0.5740	0.5000	0.5288	0.6054	0.7574	0.5560	0.5000	0.6126	0.6118
Large LSTM	error	max	max	0.8274	0.9738	0.8023	0.9305	0.9706	0.9049	0.8955	0.8076	0.8070
Large LSTM	logloss	max	max	0.8273	0.9738	0.8023	0.9305	0.9705	0.9046	0.8955	0.8076	0.8070
Large LSTM	logloss	scaler	max	0.8205	0.9731	0.8011	0.9307	0.9629	0.9022	0.8955	0.8161	0.7994
Large GRU	error	max	max	0.8162	0.9682	0.7725	0.9304	0.9693	0.8993	0.8959	0.8052	0.8005
Large GRU	logloss	max	max	0.8159	0.9683	0.7725	0.9304	0.9693	0.8987	0.8959	0.8052	0.8005
Large GRU	logloss	scaler	max	0.8049	0.9678	0.7729	0.9299	0.9565	0.8974	0.8959	0.8154	0.7916
Small LSTM	error	max	max	0.8211	0.9555	0.8609	0.9274	0.9628	0.9016	0.8803	0.8216	0.7906
Small LSTM	logloss	max	max	0.8209	0.9555	0.8609	0.9274	0.9628	0.9015	0.8803	0.8216	0.7907
Small LSTM	logloss	scaler	max	0.8096	0.9556	0.8603	0.9289	0.9513	0.8980	0.8804	0.8229	0.7869
Small GRU	error	max	max	0.8190	0.9581	0.8603	0.9308	0.9662	0.9015	0.8847	0.8186	0.7777
Small GRU	logloss	max	max	0.8189	0.9581	0.8602	0.9307	0.9661	0.9016	0.8847	0.8186	0.7777
Small GRU	logloss	scaler	max	0.8091	0.9575	0.8603	0.9305	0.9530	0.8989	0.8849	0.8197	0.7678
Markov one step	logloss	scaler	max	0.5483	0.5119	0.5245	0.7715	0.7000	0.7022	0.5553	0.6502	0.6425
Markov one step	logloss	OCSVM	max	0.5438	0.5276	0.5548	0.7717	0.6539	0.7232	0.5393	0.6566	0.6281
Markov one step	logloss	scaler	window	0.5438	0.5191	0.5247	0.7574	0.6971	0.7143	0.5467	0.6437	0.6417
Markov two step	logloss	scaler	max	0.5414	0.5100	0.5266	0.5064	0.4991	0.6305	0.5470	0.6055	0.6060
Markov two step	logloss	scaler	window	0.5366	0.5048	0.5262	0.6454	0.6310	0.6340	0.5505	0.6090	0.6255
Markov two step	logloss	scaler	average	0.5333	0.4901	0.5541	0.7336	0.7022	0.6108	0.5536	0.5834	0.6377

Table 5.5: Post processing steps for the Large LSTM model

ID	Error type	Output model	Score type	Threshold	Precision	Recall	f_β
002	logloss	scaler	average	1.4243	0.9996	0.4840	0.9892
0D0	logloss	max	average	2.5763	0.9902	0.1619	0.9424
0D1	logloss	max	max	13.7482	0.9885	0.6866	0.9842
0D4	logloss	max	average	2.5589	0.9968	0.5925	0.9901
140	logloss	scaler	average	1.6726	0.9976	0.5641	0.9901
141	logloss	scaler	max	7.9265	0.9939	0.4476	0.9821
360	error	OCSVM	logsum	-12.900	0.9828	0.2411	0.9538
370	logloss	max	average	2.1141	0.9874	0.2091	0.9523

5.3.1.1 Post processing with threshold decisions

We also evaluate potential real-world performance given the information available from s_{vb} results. For each model, ID, and post-processing step combination, we found the threshold that maximized the f_β score. Then the combination with the highest f_β for each ID was chosen to evaluate s_{test} results. As an example, the post processing steps for each ID in the Large LSTM model is shown in Table 5.5. The choices for all the models are shown in Section C.1. The table also shows the validation test data precision, recall, and f_β scores for the individual IDs on the aggregated validation test cases.

5.3.1.2 Overall model performance with decisions

Table 5.6 shows the performance of all the models on all the test cases with the post processors and decision thresholds defined above. Results for individual test cases are given in Section C.3, but we discuss just the aggregate results here. The Large LSTM model is the overall winner, followed by the large GRU and the two smaller RNN models. The Guess previous model is predictably weak. The Markov models are in the middle, with the two step model beating the one step one in terms of f_β score. However, comparing the precision of each Markov model, the single step version has

Table 5.6: Overall data sequence test performance on all test cases.

Model name	Precision	Recall	f_β score
Guess previous	0.9984	0.1147	0.9277
Small LSTM	0.9996	0.2501	0.9708
Large LSTM	0.9995	0.3509	0.9816
Small GRU	0.9992	0.2611	0.9720
Large GRU	0.9995	0.2851	0.9753
Markov one step	0.9983	0.1196	0.9306
Markov two step	0.9889	0.3773	0.9732

a full percentage point lead over the two step model, a highly significant difference given that all test cases are being compared to a single normal case. In the individual tests, metrics are calculated on a single test case versus the normal cases. Here all 61 test cases are evaluated versus the single normal case. The base rate of 61 to 1 anomaly to normal data inflates the precision in comparison to the individual test cases. Given this ratio and the choice of $\beta = 0.1$, small differences in precision cause large differences in outcome. The two-step Markov model wins in terms of f_β score, but at a cost of a significantly larger false alarm rate.

Even the best-performing model, the large LSTM, identifies only about a third of the anomaly cases, though with reasonably high precision. We are more concerned however with evaluating the parameters of the attack framework, which we describe next.

5.3.2 Replay results

The AUC scores for the test replay case is given in Table 5.7. There is a broad range in results, with the the large LSTM model fares the best, followed by the large GRU and smaller RNNs. The Markov models have no discriminative power for this test case. Replay results for individual IDs are much more favourable, as was seen in our previous work [14]. Precision, recall, and f_β scores for the model decision outputs are given in Table 5.8. This table shows how the aggregate performance also results in a

low precision and recall, despite good performance on some individual IDs.

Table 5.7: Replay test AUC in aggregate and for individual IDs.

Model name	Overall	002	0D0	0D1	0D4	140	141	360	370
Guess previous	0.5705	0.5000	0.4984	0.5345	0.5387	0.4984	0.5000	0.8365	0.7664
Small LSTM	0.8826	0.9766	0.9224	0.9916	0.9764	0.9983	0.9887	0.9436	0.9928
Large LSTM	0.9011	0.9887	0.9298	0.9897	0.9837	0.9980	0.9975	0.9345	0.9959
Small GRU	0.8792	0.9783	0.9259	0.9943	0.9795	0.9974	0.9908	0.9546	0.9896
Large GRU	0.8877	0.9935	0.9161	0.9911	0.9802	0.9961	0.9988	0.9390	0.9959
Markov one step	0.5335	0.5401	0.5032	0.6511	0.6654	0.6321	0.5943	0.6983	0.7039
Markov two step	0.5068	0.5191	0.5234	0.5058	0.5015	0.5095	0.5179	0.6253	0.5655

Table 5.8: Replay test results with decision threshold for all models.

Model name	Precision	Recall	f_β score
Guess previous	0.3889	0.0070	0.2525
Small LSTM	0.9178	0.0670	0.8153
Large LSTM	0.9405	0.1580	0.8965
Small GRU	0.7966	0.0470	0.6880
Large GRU	0.9344	0.1140	0.8723
Markov one step	0.4000	0.0080	0.2693
Markov two step	0.5122	0.2720	0.5078

5.3.3 Data field modification results

Figures 5.1 and 5.2 graph the AUC scores for all test cases and models. We show only the aggregate results over all IDs in order to keep the graphs manageable. These graphs show general trends, but are too compact to provide full details. The complete AUC results are provided in Section C.2. The decision-based results in terms of precision, recall, and f_β score are also given in Section C.3. Rather than discuss each case in detail, we describe general trends in the AUC graphs.

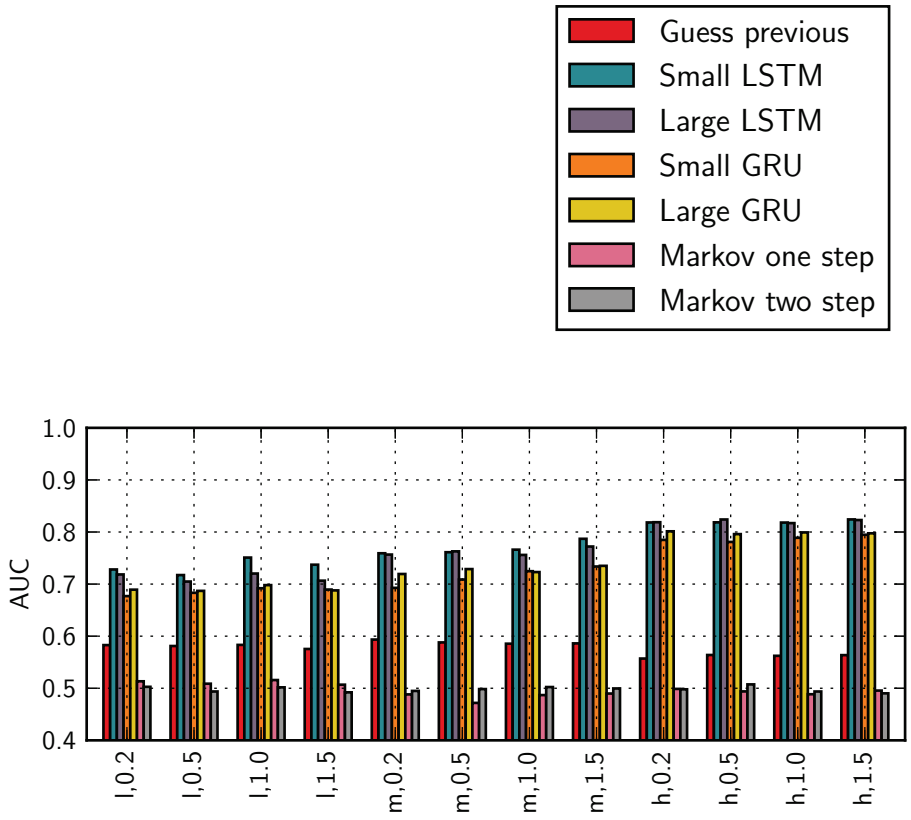
Three parameters describe each test case. The parameters and their motivation are:

- The duration of the modification: 0.2, 0.5, 1.0, or 1.5 seconds. This is meant to evaluate whether the duration of an effect plays a role in its detectability.
- The type of modification: min, max, constant (random), random, and replay. This parameter is meant to describe changes an attacker might make to cause an effect, e.g. setting a control value to a limit, or replaying data out of context.
- The class of field being modified: low, medium, or high variability. This is meant to discriminate tests between fields that display different characteristics.

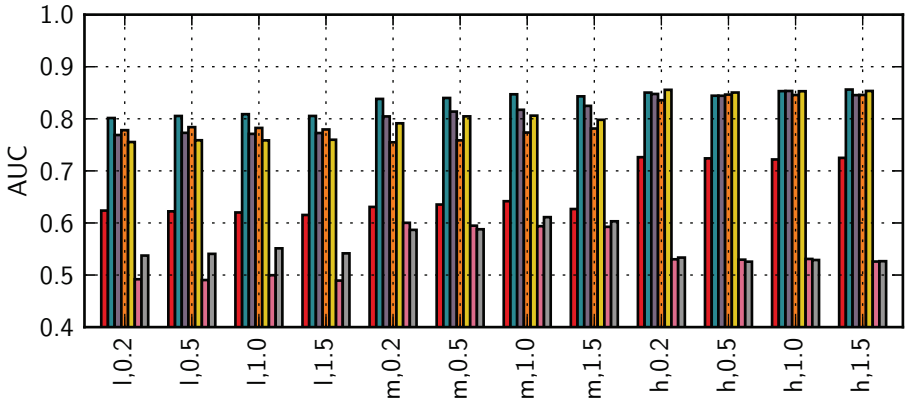
Additionally we wish to determine whether any of models are better suited to particular test cases, and if any dominate performance over all test cases. Figures 5.1 and 5.2 reveal some general trends in the field modification results.

5.3.3.1 Anomaly duration effects

It is evident from the bar plots that longer duration anomalies have higher scores across field categories and anomaly modification types. It is likely that longer-duration events provide more opportunities for high anomaly scores. The exception is for the low-variability fields, where the duration appears to have little to no effect in most cases. There is no obvious explanation for this behaviour; the models all have some

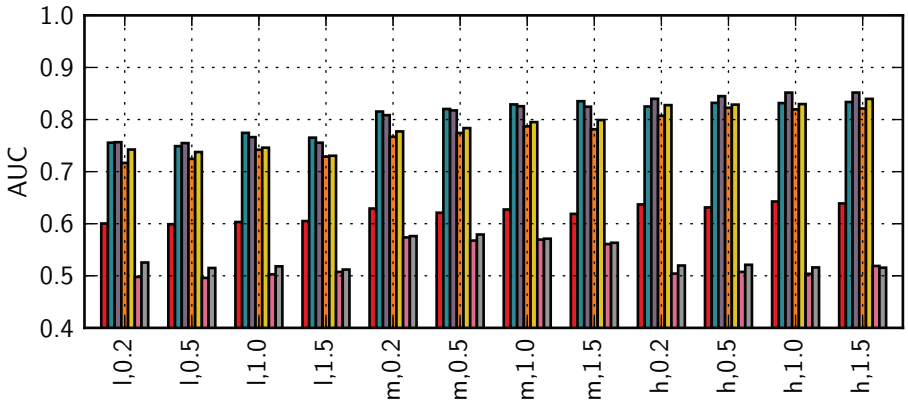


(a) Field minimum.

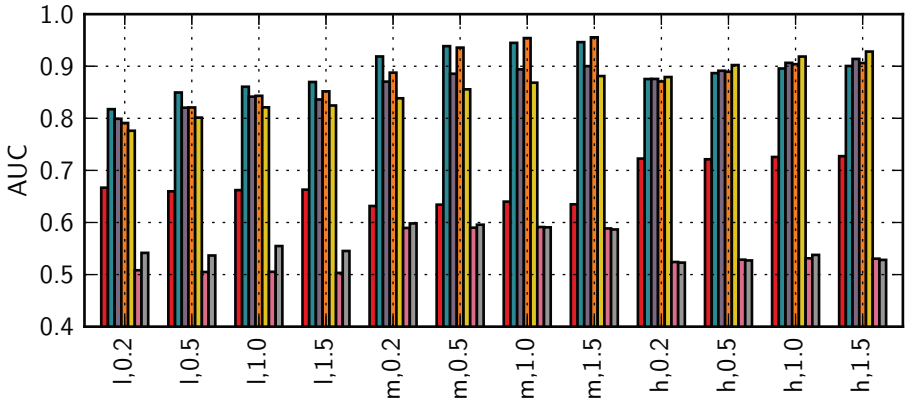


(b) Field maximum.

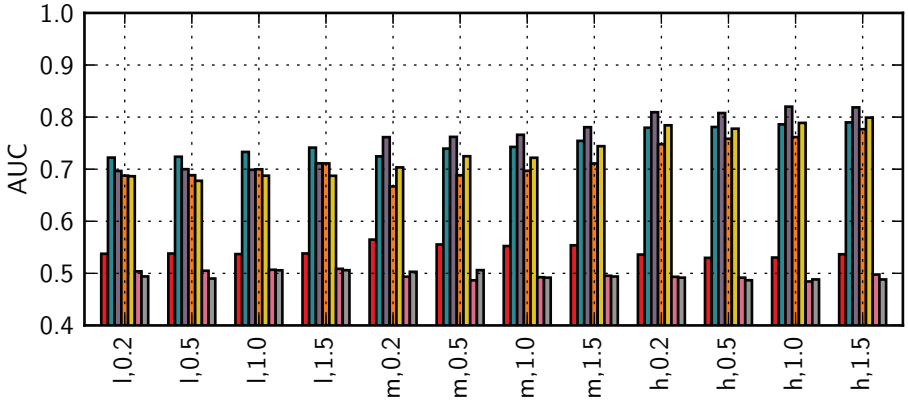
Figure 5.1: AUC score for data sequence field min and max cases.



(a) Random constant.



(b) Random values.



(c) Field replay.

Figure 5.2: AUC score for constant, random, and replay cases. Figure 5.1 contains the legend for these plots.

discriminatory power for low-variability fields, so it is not clear why longer durations do not improve performance.

5.3.3.2 Field modification type and field category effects

Some modification cases are much more detectable than others. Models generally perform well on the maximum, random constant, and random value cases, regardless of the field type. The minimum and replay cases have weaker performance overall. The replay case fills a field with data that is correct in a different context; if the field is mostly independent from the rest of the packet data, there may be little for the detector to identify except for the discontinuity between the normal data and the beginning of the replay data. For the minimum case, it may be that some fields are commonly set to zero for some period; however we did not investigate this hypothesis.

Performance on the replay and min field is not consistent over the different field categories. The low-variability fields in particular have the weakest performance by far. Since these are shorter fields, it may be that the detectors have fewer opportunities to identify problems. Investigating field-specific anomaly detection could be a valuable direction for future work.

5.3.4 Effect of ID, field type, and field modification

In the previous sections, results were aggregated for all IDs within each test case (with the exception of the replay test case). This allowed us to focus on the trends related to each test case parameter. However as we saw in the replay case, performance varied significantly between IDs. There is no easy way to review individual IDs for each model, so we select just the highest-performing one (the large LSTM) and review its results for each ID and test case separately.

The test AUC for each ID and every 1s duration test case, along with statistics about those scores, is shown in Table 5.9. We have already established a general

positive correlation between increasing duration and AUC, so we focus on a single duration case to avoid overcomplicating the discussion. The overall AUC is indicated by the final column. It is in most cases close to the lowest AUC of all the IDs; in some cases, it is even lower. The aggregate score hides these individual ID score variations. While the aggregate is useful for comparing performance over test cases and detector models, it is not a good predictor of overall performance, because we do not know which IDs and fields are really capable of being used in an attack.

Reviewing the performance over each ID individually reveals that there are two distinct classes of results. Five of the IDs (002, 0D1, 0D4, 140, and 141) have median AUC scores above 0.98. The remaining three (0D0, 360, and 370) have median scores around 0.85. While the higher scores are not perfect except in a few cases, the RNN methods are viable for these IDs and fields. The models would require additional improvements to be useful for the lower-scoring IDs. Understanding how these differences arise may require more knowledge about the meaning of each ID, or more sophisticated analysis of each ID's traffic, questions which will be important ones in future work. For now we can only conclude that the RNN methods are effective for a subset of the IDs that were evaluated.

The test case parameters have inconsistent effects between IDs. We note some trends that are generally consistent with those identified above. The low-variation fields are associated with most of the low scores for all the IDs. The medium-variation fields have more mixed results; for IDs 002 and 140, most cases have high AUC, while for other IDs they are less consistent. The high-variation fields have generally higher scores. The type of field modification appears to be only weakly linked to AUC in most cases. The min case tends to have lower performance than other cases, especially for the low-variation fields. Other cases have examples of both high and low scores, depending on the field type and ID.

Table 5.9: All 1s test case AUC results for the large LSTM model, for individual IDs and in aggregate. Entries with an AUC of "NaN" had no fields for the given ID and test case. The 'full' entry refers to the complete data word replay test case.

		AUC								
Field type	Test case	002	0D0	0D1	0D4	140	141	360	370	Overall
full	Replay	0.9887	0.9298	0.9897	0.9837	0.9980	0.9975	0.9345	0.9959	0.9011
low	Constant	0.9691	0.7492	0.9937	NaN	0.7064	0.7744	0.7174	0.7771	0.7043
	Field Replay	0.8489	0.6565	0.7639	NaN	0.5532	0.6257	0.5446	0.5757	0.5826
	Max	0.7400	0.7792	1.0000	NaN	0.8297	0.8711	0.8680	0.8604	0.7823
	Min	0.8511	0.5717	0.5830	NaN	0.5097	0.6363	0.5428	0.5699	0.5642
	Random	0.9992	0.8547	1.0000	NaN	0.8453	0.8809	0.9216	0.9457	0.8146
med	Constant	0.9947	0.8353	NaN	NaN	0.9933	NaN	NaN	0.9478	0.8744
	Field Replay	0.9946	0.6854	NaN	NaN	0.9380	NaN	NaN	0.7125	0.7796
	Max	0.9895	0.7417	NaN	NaN	0.9981	NaN	NaN	0.9215	0.8538
	Min	0.9848	0.6105	NaN	NaN	0.8659	NaN	NaN	0.5951	0.7514
	Random	0.9973	0.9255	NaN	NaN	0.9983	NaN	NaN	0.9881	0.9223
high	Constant	NaN	0.9566	0.9997	0.9934	0.9975	0.9867	0.9127	NaN	0.9065
	Field Replay	NaN	0.8746	0.9897	0.9833	0.9840	0.9803	0.8702	NaN	0.8778
	Max	NaN	0.8564	0.9999	0.9995	0.9963	0.9873	0.9703	NaN	0.9091
	Min	NaN	0.8393	0.9077	0.8860	0.9933	0.9911	0.8588	NaN	0.8608
	Random	NaN	0.9948	1.0000	1.0000	0.9981	0.9989	0.9973	NaN	0.9645
statistics	count	11	16	11	6	16	11	11	11	16
	mean	0.9416	0.8038	0.9298	0.9743	0.8878	0.8846	0.8307	0.8081	0.8156
	std	0.0875	0.1258	0.1357	0.0439	0.1633	0.1443	0.1595	0.1697	0.1169
	min	0.7400	0.5717	0.5830	0.8860	0.5097	0.6257	0.5428	0.5699	0.5642
	25%	0.9101	0.7277	0.9487	0.9834	0.8414	0.8227	0.7881	0.6538	0.7725
	50%	0.9887	0.8373	0.9937	0.9885	0.9886	0.9803	0.8702	0.8604	0.8573
	75%	0.9946	0.8873	1.0000	0.9979	0.9976	0.9892	0.9280	0.9467	0.9024
	max	0.9992	0.9948	1.0000	1.0000	0.9983	0.9989	0.9973	0.9959	0.9645

5.3.5 Data modification conclusions

Performance over the data modification cases was mixed. The RNN models, and in particular the Large LSTM model, was for some cases and IDs able to achieve very high levels of performance. However in other cases performance is much lower. Without knowing which cases are important, it is difficult to anticipate the effectiveness of this approach overall. However, the experiments have revealed useful trends in terms of the anomaly fields, types, and durations of anomalies that are consistent over different models. With more information about the nature of the normal traffic it may be possible to improve overall performance by focusing on high-risk IDs and fields.

5.4 Summary

We have shown that even very short frequency anomalies can be detected with high confidence. The t-test method performed the best overall, but a combination of both methods would provide the best coverage if computational resources are sufficient to support this choice. Results for data sequence anomalies are less consistent. Performance varies more across different IDs than with the parameters of the attack model. Nevertheless we drew some general conclusions about those parameter effects over all anomaly detectors and IDs. Whether this performance is sufficient to enable practical use depends on external factors. We discuss these and other issues next in the conclusions.

Chapter 6

Conclusion and future work

This thesis presents an approach to detect cyber attacks on the automotive CAN bus. Our approach to achieving this goal had a series of steps:

1. Analyzing CAN bus data;
2. Researching attacks in cars to understand how they manifest on the CAN bus as anomalies, and using this analysis to synthesize a parameterized framework describing these effects;
3. Framing the attack detection problem in a way suitable for sequence anomaly detection methods, and identifying specific methods for detecting attacks;
4. Designing experiments based on the attack framework; simulating attacks, and using these to evaluate the detection methods and predict how attack parameters affect their detectability.

In this section, we review the research findings, our contributions, and provide ideas for future work.

6.1 Review of research

6.1.1 CAN bus data

We characterized normal CAN traffic by reviewing data from two vehicles, a 2011 Ford Explorer, and a 2012 Subaru Impreza. We found that the high speed control bus was populated by a fixed number of IDs, each associated with a periodic packet sequence. The periods for each ID ranged from 10ms to 1s, and were found to be very consistent while the engine was running. Each ID's packet stream also contained a sequence of data words, though usage of the data field varied widely between IDs. Fields were identified within the data words using an automated methods; further analysis categorized these fields according to the variety of their contents. For the more highly-used IDs, we showed that the number of unique words employed by each ID grew in an apparently unbounded manner. This constant novelty restricts the types of anomaly detection methods capable of characterizing data sequences.

6.1.2 Attack framework

We also reviewed the literature on cyber attacks on vehicles. Hacking a car has three steps: obtain access to the vehicle electronics, obtain access to the CAN bus, and send commands on the bus to cause cyber-physical effects. We developed a framework describing just the consequences of the third step. The framework describes two categories of effects: those that affect the normal packet frequencies, and those that affect just the data words within packets. The packet frequency effects, in the context of highly periodic normal packets, involve just packet insertions or deletions. The data effects are divided into replay attacks, where an entire packet data sequence is replaced by one from an earlier time, or field effects, where the field identified in the data analysis are modified in various ways. All effects in the framework were parameterized according to the their nature, e.g. with duration, insertion rates, etc.

6.1.3 CAN bus anomaly detection

Based on the data analysis and attack model, we identified a separate approach to detect packet frequency and data anomalies. For packet frequency anomalies, we adapted the concept of flows to CAN bus traffic. CAN flows describe packet frequencies within a sliding window. While any single-point one-class classifier or anomaly detector would be suitable for anomaly detection with CAN flows, we chose two to evaluate: the T-test detector because of its history with similar kinds of data, and the OCSVM because of its history of effectiveness in a broad range of domains. For data sequence anomalies, the challenge is to characterize the large amounts of high-dimensional sequence data. While most sequence anomaly detectors were not well-suited to the task, we identified some prediction methods capable of characterizing the data. We chose four variants of RNNs as well as multistep, multivariate Markov models for evaluation, as well as a guessing method intended to establish a baseline level of minimum performance.

6.1.4 Performance evaluation and attack framework parameter effects

Methods were tested with simulated attacks. No real attack data of this type is generally available, and moreover, such data would be limited to a small number of examples designed for specific vehicles. Our approach is more broadly useful: by characterizing attacks with the attack framework, we can generate limitless anomalies with signatures corresponding to a range of attack effects. We used this approach to generate attack-like anomalies by modifying real data. The detection methods were evaluated on each anomaly type over a range of parameters to reveal broad patterns in the detectability of specific effects as well as the efficacy of different detection methods for individual packet IDs.

6.1.4.1 Packet frequency anomaly detection results

We showed that packet-frequency anomalies can be detected with high confidence. The t-test method perfectly detected all cases in which more than 10 packets were inserted with a detection window of 0.5s, and for more insertions, with a window of 0.2s. These rates are much lower than those of any published attacks. The limits of perfect detection were for longer windows. As few as three to five extra packets could be detected with a 1s window. Similarly, between 10 and 20 erased packets could be perfectly detected with a 0.2s window using the OCSVM method. There are fewer attacks that have the effect of removing packets from the bus, but for fast IDs, missing packets will be identified in time to respond to changes involving physical control systems. We argue that because these methods are computationally inexpensive to run, a combination of t-test and OCSVM detectors with multiple window sizes can provide both fast and reliable detection of most frequency-related events.

6.1.4.2 Data sequence anomaly detection results

The data methods involved many more parameters, so they are more difficult to summarize succinctly. Also their performance was more varied across these parameters. There were two classes of data sequence anomalies: replay attacks, and field modifications. We reviewed results in terms of both aggregate scores for each model to identify general trends, and then reviewed performance within individual IDs. The broadest trend in performance was for individual IDs. Of the eight IDs being evaluated, five had consistently high AUCs across the anomaly tests, and the remaining three had consistently lower AUCs.

Replay attacks are straightforward to evaluate because they have no parameters aside from the ID affected by the attack. The high-performing ID detectors had AUCs in the range of 0.99, with the low performers in in the range of 0.9-0.93. The higher scores provide a level of performance that makes practical detection of replay attacks

a possibility, while the lower scores would produce an unacceptable false alarm rate.

The data field modification test cases had several parameters, some having more pronounced effects than others. These parameters were the category of field being modified (low, medium, or high variability), the kind of modification (max, min, constant, random, or replay), and the duration of the effect. We identified several general patterns in the results as these parameters were varied. First, anomaly detectability was generally lower for fields characterized as low-variability than for medium- and high-variability fields. Second, the type of alteration had a significant effect. Although three of the five types of alteration were to set the field to a constant value, detectability increased from setting this value to the field minimum, to a random value, and to the field maximum. Performance losses compounded for the low-variability, field-minimum case. For the two cases where the replaced data changed, random insertions were easier to detect than replayed data. The difficulty in detecting field replays is not surprising because the replayed data is only anomalous with 1. respect to other fields in the data sequence, and 2. the data sequence preceding and following the anomaly. Finally, longer anomaly durations were generally correlated with better detection performance, with more pronounced effects on the more difficult cases.

Unlike for the packet-frequency case, it is not possible to give specific examples of how the data sequence anomaly experiments inform performance on published attacks. This is because the attack framework parameters require a description of the field being modified, and this information is not available for the published attacks we have seen. The point of this exercise was to explore where performance is good and bad based a range of factors without detailed knowledge of the data message meaning. We have shown that the RNN methods are capable of predicting some of the ID's data sequences and subfields within them highly reliably. Others are less reliable. The fact that some are better than other suggests that it is the nature of the fields themselves that is inherently unpredictable. If some fields are inherently unpredictable then our

approach cannot succeed without additional information, which could include simply knowing which IDs are most important to monitor.

The relative model performance revealed clear winners. The RNN models are all effective at detecting anomalies at rates much higher than the Markov models,. The Markov models performed not much better than the previous guess model. We have also shown that the large LSTM model is the most effective detector for the majority of test cases.

A detailed look at the large LSTM performance across different IDs showed how performance varied between the high and low-performing IDs. The low-performers had scores significantly higher than the guessing model, but still well below a minimum level for practically useful performance. For the high-performers, the same general trends with parameters described above held across the IDs. Most cases had AUCs within reach of practical performance (around 0.99). Cases that did not meet this standard consistently across high-performing IDs were generally for the low-variability fields, and the min and field replay cases for all field types.

We conclude that our approach is viable at this time for detecting attacks with signatures in a subset of the tested categories, and for a subset of the IDs in our system. These categories were mainly for medium- and high-variability fields and all of the field modification types, with some ID-dependent exceptions for the min and replay types that are ID-dependent. With appropriate thresholds we can detect this subset of the attacks with confidence.

The focus of the experiments was not on specific attacks and overall performance, but rather ranges of anomaly types. In reality, only a small number of combinations of fields, IDs, and effects will correspond to meaningful attacks. The true real-world performance of our approach depends on whether the IDs of interest are the more or less predictable ones. To our knowledge, this work represents the first evaluation of anomaly detection methods in CAN bus data sequences over a parameterized range

of anomaly effects.

6.2 Review of contributions

This thesis proposes that cyber attack within automobiles can be characterized in a systematic way, that these attacks can be detected using an anomaly detection approach, and that we can predict the detectability of attacks based on parameters describing their characteristics. The attack framework presented in Chapter 2 fulfilled this goal, and furthermore informed the development of detectors designed aimed at identifying specific effects of those attacks. The methods presented in Chapter 4 were designed to fulfill the second goal, and were tailored to the specific effects on CAN bus traffic identified by the attack framework. The experiments in Chapter 5 demonstrated the limits for detecting the effects described by the attack framework, and identified trends in performance for individual IDs, attack types and parameters. We showed the limits of detectability of packet frequency changes, and described how easily a range of changes in the packet data sequences of individual IDs could be detected. These results could form the basis of a CAN bus attack detection module designed to safeguard automotive control systems from malicious attacks.

6.3 Future work

Anomaly and attack detection in cyber-physical systems is a very new field, and much work is yet to be done in safeguarding automobiles from cyber attacks. The packet-frequency anomaly detection performance demonstrated here is sufficiently precise for deployment in a consumer automobile. However we have not addressed the actions required in response to a detected anomaly. Such a response may require additional work in the design of individual components, such that a safe mode may be activated in response to a detected attack, allowing the vehicle to be safely parked. Coordinating

such a response may in fact be more complex than improving the anomaly detection performance, in that it requires the coordination of multiple control computers, any one of which may be assumed to be compromised.

Our data sequence approach presented here has limitations. It succeeded for some IDs but not for others. Foremost, it would be helpful to understand why this is the case. Additional analysis might reveal why some IDs were easier to predict by the RNNs than others, but answering this question may also require a better understanding of the meaning of the packet data. Our approach, which assumes no knowledge of the packet data protocol, may be fundamentally limited. Even simple high-level knowledge about which IDs perform particular functions could yield significant improvements, for example by limiting the number of IDs and fields that require monitoring. We could also investigate tracking individual field sequences with additional methods; for example, the low-variability fields may be amenable to sequence methods requiring small fixed symbol dictionaries. However, integrating the individual sequence detectors introduces other challenges, as evidenced by the drop in performance as individual ID detectors outputs are combined. Each monitored ID adds additional potential for false alarms.

One way to improve the combined assessment of multiple IDs may be extending the sequence prediction RNNs to learn all relevant ID data sequences simultaneously. The relationships between IDs could be helpful to identify outliers within a single data sequence. Alternatively, different ways of combining individual ID data model errors could yield additional improvements. The approach presented here derived anomaly scores from data words individually, then integrated those scores into an assessment for the full sequence. Methods could be applied that integrate error scores in both time and within words simultaneously, such as multichannel change detection [83].

Finally, to validate the attack framework, it would need to be tested with examples of specific attacks, along with knowledge of the data protocol of the target. It

would be prohibitively time-consuming to explore the parameter space defined by the framework. However checking individual points of interest in this space would help to calibrate its predictions. The approach we have presented to frame this problem, and predict how it can be solved, can guide the design and implementation of systems that will increase the safety of drivers and protect them from malicious cyber-attacks.

Appendix A

CAN data analysis details

This chapter contains details about the different kinds of fields identified in the Impreza packet data.

Table A.1: Low-variation fields identified in the Impreza data.

ID	start bit	length	μ_{diff}	σ_{diff}	min	max	n_{unique}
002	59	5	0.00	0.89	0	31	32
0D0	2	5	68.83	103.80	0	31	32
	11	5	40.04	88.16	0	31	32
	17	5	22.65	71.95	0	17	18
	26	4	10.21	48.12	0	15	16
0D1	10	6	18.15	61.00	0	52	50
140	2	1	6.57	40.33	0	1	2
	33	1	0.85	14.65	0	1	2
	35	1	0.02	2.00	0	1	2
141	3	2	0.95	15.47	0	2	3
	9	2	14.59	59.10	0	3	4
	13	4	67.16	109.84	0	15	15
	19	1	0.01	1.32	0	1	2
360	8	1	10.81	51.27	0	1	2
	10	2	5.52	37.04	0	2	3
	14	1	2.24	23.74	0	1	2
	16	1	20.84	69.71	0	1	2
	50	1	0.00	0.85	0	1	2
361	0	1	5.93	38.37	0	1	2
	26	1	0.00	0.39	0	1	2
	36	4	0.00	0.66	3	15	13
	50	1	0.00	0.00	0	1	2
	61	3	0.00	0.00	0	5	6
370	0	1	85.13	119.90	0	1	2
	33	1	0.08	4.63	0	1	2
	37	2	0.02	2.40	0	3	4
	39	1	0.00	0.91	0	1	2

Table A.2: Medium-variation fields identified in the Impreza data.

ID	start bit	length	μ_{diff}	σ_{diff}	min	max	n_{unique}
002	24	8	39.16	86.97	0	255	256
	48	7	0.16	3.20	0	127	128
0D0	40	8	1.87	14.97	0	255	256
140	24	8	23.78	73.86	0	255	256
	56	8	0.00	0.01	0	255	255
282	22	18	99460300.80	520982454.94	774766592	4278059008	410
	54	10	468.77	5521.34	1	65027	422
370	24	8	1.55	14.08	0	255	180
	54	10	0.83	14.49	0	65280	454

Table A.3: High-variation fields identified in the Impreza data.

ID	start bit	length	μ_{diff}	σ_{diff}	min	max	n_{unique}
0D0	52	12	2.29e+01	1.22e+03	0	65295	4096
0D1	20	12	2.42e+00	2.40e+01	0	65287	2228
0D4	4	12	1.56e+04	2.72e+04	0	65287	2218
	20	12	3.84e+03	1.49e+04	0	65287	2223
	36	12	1.03e+02	2.11e+03	0	65287	2220
	52	12	4.63e-02	3.34e+00	0	65287	2220
140	7	17	1.36e+09	1.99e+09	256	4294901760	12202
	36	14	7.00e+02	4.18e+03	0	65343	12771
141	20	12	8.04e+03	1.38e+04	0	65295	3886
	36	12	2.93e+01	3.58e+02	6	65293	2247
	52	12	1.34e-02	1.79e+00	6	65293	2194
360	24	24	2.15e+07	2.30e+08	657863936	2290663936	122299
	51	13	7.81e-01	2.18e+02	0	65305	6274
361	1	17	2.72e+08	1.02e+09	131072	4294770944	12118

Appendix B

Attacks reported in the literature

The following table lists attacks found in a variety of sources. Only attacks using normal packets are included; attacks making use of diagnostic packets are omitted.

Attack	Ref	Description	Data changes	Needs car state?	Bus level description	Complexity
Accelerator	[6]	Use the messages normally coming from the accelerator pedal to instruct the engine Internal Combustion Engine to accelerate. Needs to appear between CAN v1 and v2 bridges, i.e. not on ODB-II port. Also gas ICE must be engaged then disengaged. Attack lasts only a few seconds.	Updating	Independent	Crafted updating inject, independent of state	modified packet, repeated
Braking	[6]	The cruise control can accelerate and brake (it can detect if a car is ahead). The control packet can be used to force the car to brake. They could not figure out how to make it accelerate.	Updating	Independent	Crafted updating inject, independent of state	modified packet, repeated
Steering, Parking Assist, Toyota	[6]	Use the parking assist system to steer the car. Requires the car to think it's in reverse at low speed, so spoof/flood the bus with packets indicating that. Then the car gets messed up from the DOS, but you can jerkily control the wheel.	Updating	Independent	Crafted updating inject, independent of state	multiple packets, repeated
Steering, Parking Assist, Ford	[6]	The Parking Assist Module controls the Power Steering Control Module to turn the wheel to park the car. Use these control packets to control steering. Need to send rational series of adjustments that are absolute steering angles. PSCM ignores instructions above 5 mph. Needs to read the steering wheel position to start with the correct control values.	Updating	Read once	Crafted updating inject, based on state	read once, modified packet, repeated

Attack	Ref	Description	Data changes	Needs car state?	Bus level description	Complexity
Steering, Lane Keep Assist, Toyota	[6]	The Lane Keep Assist (LKA) function is designed to turn the wheel at arbitrary speeds. Direct this system to turn the wheel; must increment a counter so the packet changes each time. Maximum turn of about 5 degrees from center (requests to turn more are ignored)	Static	Independent	Static inject, independent of state	modified packet, repeated
Falsify odometer	[6]	Make the odometer increase. It expects the next value of the count, so the current value has to be read first.	Updating	Read once	Crafted updating inject, based on state	modified packet, repeated
Confuse onboard navigation	[6]	Spoof the on board navigation system. Read the wheel count, increment it and make the navigation system think you are moving. No video or other description for this one.	Updating	Read once	Crafted updating inject, based on state	read once, modified packet, repeated
Responsive speedometer inject	[19]	Set the speedometer to a function of the real speed, e.g. 15 kph less than actual. Needs to read the speed and then put out packets with the calculated false speed fast enough to override the real one.	Updating	Responsive	Crafted updating inject, responsive to evolving state	responsive, modified packet, repeated
Disable alarm	[23]	When the alarm goes off, packets are normally sent to trigger horn and turn lights on and off. Detect these packets and immediately send commands to undo whatever they did, e.g. turn lights back off.	Static	Responsive	Crafted updating inject, responsive to evolving state	responsive, modified packet, repeated

Attack	Ref	Description	Data changes	Needs car state?	Bus level description	Complexity
Lights out	[19]	Disable all lights by sending packets for each and every one to turn off.	Static	Independent	Static inject, independent of state	single packet
Kill bus	[6]	Flood the CAN network with priority 0 packets. Among other things, causes the power steering to fail so the car is very difficult to turn. If the car is off, it will not start.	Static	Independent	Static inject, independent of state	single packet, repeated
Door ajar	[6]	Every 2 seconds, door sensor sends packet with ID 03B1. Data all zero means door close, one bit difference means door ajar. Door ajar indicator lights on dash if you send the packet	Static	Independent	Static inject, independent of state	single packet, repeated
Control user functions	[19]	Falsify speedometer, increase radio volume, change radio display, change DIC display, unlock and lock, remote start, honk, kill engine.	Static	Independent	Static inject, independent of state	single packet, repeated
Speedometer	[6]	Set the speed and RPM displayed to the driver.	Static	Independent	Static inject, independent of state	single packet, repeated
Speedometer	[6]	Set the speedometer to an arbitrary reading.	Static	Independent	Static inject, independent of state	single packet, repeated
Control user functions	[10]	Inject packets spoofing traffic from tire pressure monitor, steering column button pushes, alerts. Vary timing of injects to test anomaly detection performance.	Static	Independent	Static inject, independent of state	single packet, repeated

Attack	Ref	Description	Data changes	Needs car state?	Bus level description	Complexity
Self-destruct	[19]	Display countdown, make clicks and horn honks, kill engine.	Static	Independent	Static inject, independent of state	multiple packets, repeated
Error inject	[19]	Targetted denial of service. Disable communications of individual components.	Static	Responsive	Static inject, responsive	responsive, low level, modified packet, repeated
Error inject	[17]	Watch for a particular packet; when it arrives, force bus to dominant state before it finishes transmitting. The sender detects and error and will try again. In the meantime, optionally send your own spoofed packet. After a particular number of errors, transmitting ECU will stop sending packets altogether.	Static	Responsive	Static inject, responsive	responsive, low level, modified packet, repeated
Triggered inject	[23]	When speed exceeds 200 kph, open the window. If the driver tries to close it, it is stuck. Injected packet is at same frequency as normal one.	Static	Read once	Static Inject, triggered	single packet, repeated
Counterfeit airbag	[23]	Airbag unit periodically communicates its fault-free status. Can replay these messages, and also emulate the startup check, so the airbag can be removed without the dash error lighting up. Diagnostic routines can also be emulated.	Updating	Responsive	Masquerading component	replay

Appendix C

Complete experimental results

This appendix contains the complete results for the experiments in Chapter 5.

C.1 Data post processing methods

The following table shows the post-processing configuration chosen be test validation for each data sequence anomaly detection model.

Model Name	ID	Error type	Output model	Score type	Threshold	Precision	Recall	F_β
	002	logloss	scaler	average	1.4243	0.9996	0.4840	0.9892
	0D0	logloss	max	average	2.5763	0.9902	0.1619	0.9424
	0D1	logloss	max	max	13.7482	0.9885	0.6866	0.9842
Large	0D4	logloss	max	average	2.5589	0.9968	0.5925	0.9901
LSTM	140	logloss	scaler	average	1.6726	0.9976	0.5641	0.9901
	141	logloss	scaler	max	7.9265	0.9939	0.4476	0.9821
	360	error	ocsvm	logsum	-12.8999	0.9828	0.2411	0.9538
	370	logloss	max	average	2.1141	0.9874	0.2091	0.9523
	002	logloss	ocsvm	logsum	-10.8129	0.9991	0.3996	0.9844
	0D0	logloss	ocsvm	average	2.9924	0.9868	0.1206	0.9213
	0D1	error	max	max	1.0000	1.0000	0.7017	0.9958
Large	0D4	logloss	max	max	14.2420	0.9951	0.5672	0.9878
GRU	140	logloss	ocsvm	average	5.7806	0.9974	0.4793	0.9868
	141	logloss	scaler	average	1.5041	1.0000	0.2397	0.9696
	360	error	ocsvm	logsum	-13.0325	0.9797	0.1817	0.9389

Model Name	ID	Error type	Output model	Score type	Threshold	Precision	Recall	F_β
	370	logloss	max	average	1.8796	0.9895	0.2395	0.9598
Small LSTM	002	logloss	scaler	average	1.9123	0.9760	0.1145	0.9083
	0D0	logloss	ocsvm	average	5.1396	1.0000	0.1037	0.9211
	0D1	logloss	scaler	max	18.0790	1.0000	0.4224	0.9866
	0D4	logloss	max	max	14.3334	0.9842	0.3657	0.9680
	140	logloss	ocsvm	average	3.7783	0.9974	0.4836	0.9870
	141	logloss	scaler	average	1.6170	1.0000	0.2422	0.9699
	360	error	ocsvm	average	1.4859	0.9676	0.2157	0.9353
	370	logloss	max	max	34.2344	0.9950	0.3433	0.9766
Small GRU	002	logloss	max	max	13.6491	0.9329	0.2456	0.9077
	0D0	logloss	ocsvm	average	3.4128	1.0000	0.1193	0.9319
	0D1	logloss	max	average	1.7579	1.0000	0.3932	0.9849
	0D4	logloss	max	average	3.5931	0.9932	0.2793	0.9687
	140	logloss	max	average	1.9603	0.9978	0.5562	0.9900
	141	logloss	scaler	average	1.6348	1.0000	0.2271	0.9674
	360	error	ocsvm	average	0.2240	0.9579	0.2261	0.9282
	370	logloss	max	average	2.2460	0.9874	0.3044	0.9660
Markov one step	002	logloss	ocsvm	max	32.7483	0.7927	0.0122	0.4856
	0D0	logloss	ocsvm	max	62.4865	0.7357	0.0151	0.4994
	0D1	logloss	ocsvm	max	68.4484	1.0000	0.1410	0.9431
	0D4	error	max	max	0.9999	0.5537	0.4582	0.5525
	140	logloss	scaler	max	35.3948	0.8019	0.1757	0.7746
	141	logloss	scaler	average	21.2300	1.0000	0.0087	0.4712
	360	error	max	max	1.0000	0.9544	0.1103	0.8872
	370	logloss	scaler	average	4584.1343	1.0000	0.1479	0.9460
Markov two step	002	error	gaussian	logsum	-36.8467	0.3778	0.0096	0.2737
	0D0	error	gaussian	logsum	-26.9340	0.6887	0.2864	0.6793
	0D1	error	ocsvm	max	-1.5515	1.0000	0.1294	0.9376
	0D4	error	gaussian	logsum	-24.8897	0.8790	0.2776	0.8606
	140	error	gaussian	logsum	-40.3791	1.0000	0.4661	0.9888
	141	error	ocsvm	logsum	-10.2333	0.2969	0.0400	0.2791
	360	error	ocsvm	max	1.0315	0.2581	0.3184	0.2586
	370	logloss	scaler	average	10253.2227	1.0000	0.1496	0.9467
Guess previous	002	error	ocsvm	max	3.0435	0.2130	0.0043	0.1441
	0D0	error	gaussian	logsum	-81.2230	0.9590	0.0616	0.8382
	0D1	error	ocsvm	max	0.9671	1.0000	0.1777	0.9562
	0D4	logloss	scaler	max	34.5388	0.7672	0.5040	0.7632
	140	logloss	scaler	average	17.7089	0.8857	0.1132	0.8297
	141	error	ocsvm	average	-2.8205	0.3623	0.0034	0.1776
	360	error	gaussian	logsum	-69.0574	0.9988	0.1258	0.9346

Model Name	ID	Error type	Output model	Score type	Threshold	Precision	Recall	F_β
	370	error	ocsvm	max	3.8337	1.0000	0.1336	0.9397

C.2 Data sequence anomaly field modification AUC results

The AUC scores for all models on all subfield modification cases are shown below.

Table C.2: All min field replacement test case AUC results.

		AUC			
		0.2	0.5	1.0	1.5
Model name	Field type				
Guess previous	low	0.5210	0.5171	0.5170	0.5182
	med	0.4999	0.5030	0.5005	0.4982
	high	0.5274	0.5338	0.5342	0.5328
Small LSTM	low	0.5903	0.5948	0.5753	0.5714
	med	0.7114	0.7186	0.7451	0.7404
	high	0.8409	0.8534	0.8539	0.8596
Large LSTM	low	0.5697	0.5747	0.5642	0.5418
	med	0.7193	0.7340	0.7514	0.7531
	high	0.8446	0.8628	0.8608	0.8650
Small GRU	low	0.5719	0.5734	0.5736	0.5679
	med	0.6948	0.7093	0.7332	0.7298
	high	0.8383	0.8556	0.8520	0.8570
Large GRU	low	0.5733	0.5602	0.5638	0.5420
	med	0.7012	0.7170	0.7204	0.7369
	high	0.8267	0.8432	0.8377	0.8441
Markov one step	low	0.4927	0.4946	0.4936	0.4712
	med	0.4980	0.4982	0.5000	0.4890
	high	0.4835	0.5003	0.4810	0.4966
Markov two step	low	0.4978	0.4969	0.4921	0.4729
	med	0.4959	0.5025	0.4940	0.4906
	high	0.4814	0.4949	0.4783	0.4930

Table C.3: All max field replacement test case AUC results.

		AUC			
		0.2	0.5	1.0	1.5
Model name	Field type				
Guess previous	low	0.6675	0.6718	0.6691	0.6704
	med	0.5672	0.5648	0.5666	0.5682
	high	0.7198	0.7139	0.7153	0.7131
Small LSTM	low	0.7884	0.7858	0.7968	0.7920
	med	0.8464	0.8595	0.8723	0.8789
	high	0.9282	0.9240	0.9308	0.9242
Large LSTM	low	0.7550	0.7542	0.7823	0.7642
	med	0.8233	0.8383	0.8538	0.8642
	high	0.9186	0.9091	0.9091	0.9092
Small GRU	low	0.7826	0.7781	0.7898	0.7921
	med	0.7941	0.8123	0.8244	0.8316
	high	0.9293	0.9265	0.9306	0.9309
Large GRU	low	0.7458	0.7479	0.7694	0.7627
	med	0.7998	0.8170	0.8311	0.8332
	high	0.9170	0.9154	0.9104	0.9101
Markov one step	low	0.5128	0.5171	0.5215	0.5191
	med	0.5792	0.5644	0.5736	0.5910
	high	0.5476	0.5538	0.5545	0.5518
Markov two step	low	0.5423	0.5551	0.5598	0.5530
	med	0.5716	0.5532	0.5680	0.5767
	high	0.5268	0.5357	0.5353	0.5279

Table C.4: All constant field replacement test case AUC results.

		AUC			
		0.2	0.5	1.0	1.5
Model name	Field type				
Guess previous	low	0.5993	0.5927	0.5980	0.5930
	med	0.5483	0.5531	0.5473	0.5546
	high	0.6202	0.6142	0.6243	0.6258
Small LSTM	low	0.7121	0.6981	0.7168	0.6996
	med	0.8474	0.8671	0.8651	0.8807
	high	0.8917	0.8979	0.9005	0.9003
Large LSTM	low	0.7081	0.6911	0.7043	0.6854
	med	0.8579	0.8701	0.8744	0.8812
	high	0.9010	0.8970	0.9065	0.9085
Small GRU	low	0.7140	0.6977	0.7088	0.6966
	med	0.8234	0.8449	0.8389	0.8527
	high	0.9010	0.9067	0.9101	0.9062
Large GRU	low	0.6892	0.6854	0.7110	0.6756
	med	0.8230	0.8442	0.8434	0.8467
	high	0.8908	0.8917	0.8938	0.8945
Markov one step	low	0.5083	0.4817	0.5065	0.5098
	med	0.5632	0.5672	0.5595	0.5674
	high	0.5314	0.5301	0.5221	0.5188
Markov two step	low	0.5301	0.4945	0.5278	0.5295
	med	0.5605	0.5581	0.5482	0.5595
	high	0.5162	0.5109	0.5062	0.5050

Table C.5: All random field replacement test case AUC results.

		AUC			
		0.2	0.5	1.0	1.5
Model name	Field type				
Guess previous	low	0.6872	0.6942	0.6941	0.6933
	med	0.5667	0.5670	0.5698	0.5637
	high	0.7174	0.7209	0.7179	0.7194
Small LSTM	low	0.8251	0.8403	0.8518	0.8543
	med	0.9540	0.9791	0.9849	0.9853
	high	0.9484	0.9609	0.9691	0.9733
Large LSTM	low	0.8004	0.8109	0.8146	0.8120
	med	0.9002	0.9148	0.9223	0.9283
	high	0.9404	0.9568	0.9645	0.9718
Small GRU	low	0.8172	0.8305	0.8474	0.8530
	med	0.9210	0.9625	0.9767	0.9781
	high	0.9438	0.9549	0.9625	0.9666
Large GRU	low	0.7856	0.7973	0.8093	0.8034
	med	0.8646	0.8804	0.8905	0.8920
	high	0.9445	0.9614	0.9724	0.9774
Markov one step	low	0.5163	0.5024	0.5152	0.5136
	med	0.5925	0.5842	0.5839	0.5848
	high	0.5640	0.5605	0.5636	0.5454
Markov two step	low	0.5536	0.5360	0.5532	0.5509
	med	0.5841	0.5738	0.5798	0.5777
	high	0.5433	0.5321	0.5342	0.5159

Table C.6: All field replay replacement test case AUC results.

		AUC			
		0.2	0.5	1.0	1.5
Model name	Field type				
Guess previous	low	0.5223	0.5216	0.5194	0.5212
	med	0.4930	0.4949	0.4930	0.4950
	high	0.5087	0.5103	0.5123	0.5082
Small LSTM	low	0.5870	0.6025	0.5950	0.6200
	med	0.7498	0.7668	0.7815	0.7918
	high	0.8463	0.8476	0.8488	0.8555
Large LSTM	low	0.5711	0.5864	0.5826	0.5871
	med	0.7726	0.7719	0.7796	0.8024
	high	0.8700	0.8748	0.8778	0.8830
Small GRU	low	0.5777	0.5982	0.5941	0.6053
	med	0.7426	0.7527	0.7643	0.7893
	high	0.8525	0.8567	0.8666	0.8666
Large GRU	low	0.5745	0.5846	0.5795	0.5903
	med	0.7462	0.7575	0.7626	0.7841
	high	0.8481	0.8549	0.8613	0.8576
Markov one step	low	0.4871	0.5037	0.4935	0.5103
	med	0.5010	0.4917	0.4819	0.5004
	high	0.4854	0.5086	0.4952	0.4899
Markov two step	low	0.4861	0.5026	0.4913	0.5086
	med	0.4996	0.4922	0.4796	0.4941
	high	0.4821	0.5060	0.4916	0.4880

C.3 Data sequence anomaly field modification decision-based results

The Precision, Recall, and F_β scores for all models on all subfield modification cases are shown below.

Table C.7: All min field replacement test case results.

Model name	Field type	Precision				Recall				F_β score			
		0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s
Guess previous	low	0.6207	0.5217	0.5600	0.5600	0.018	0.012	0.014	0.014	0.4662	0.3673	0.4040	0.4040
	med	0.4211	0.3529	0.3529	0.4211	0.008	0.006	0.006	0.008	0.2786	0.2244	0.2244	0.2786
	high	0.7442	0.7708	0.7556	0.7250	0.032	0.037	0.034	0.029	0.6098	0.6443	0.6244	0.5858
Small LSTM	low	0.8378	0.8333	0.7778	0.8235	0.031	0.030	0.021	0.028	0.6662	0.6587	0.5732	0.6427
	med	0.6667	0.9294	0.9645	0.9729	0.012	0.079	0.163	0.215	0.4329	0.8399	0.9197	0.9400
	high	0.9535	0.9686	0.9770	0.9787	0.123	0.185	0.255	0.276	0.8937	0.9296	0.9504	0.9547
Large LSTM	low	0.7143	0.7059	0.8361	0.8276	0.025	0.024	0.051	0.048	0.5611	0.5509	0.7255	0.7129
	med	0.7500	0.9580	0.9732	0.9751	0.030	0.228	0.363	0.391	0.6060	0.9285	0.9573	0.9609
	high	0.9611	0.9772	0.9803	0.9812	0.247	0.429	0.498	0.521	0.9343	0.9650	0.9710	0.9727
Small GRU	low	0.5714	0.4545	0.5862	0.6471	0.016	0.010	0.017	0.022	0.4253	0.3156	0.4403	0.5050
	med	0.8588	0.9290	0.9560	0.9597	0.073	0.157	0.261	0.286	0.7761	0.8859	0.9315	0.9379
	high	0.8095	0.9172	0.9481	0.9652	0.051	0.133	0.219	0.333	0.7056	0.8666	0.9178	0.9474
Large GRU	low	0.4286	0.6000	0.7037	0.7037	0.006	0.012	0.019	0.019	0.2525	0.4040	0.5186	0.5186
	med	0.5556	0.9610	0.9737	0.9755	0.010	0.197	0.296	0.318	0.3607	0.9254	0.9521	0.9559
	high	0.9370	0.9563	0.9690	0.9732	0.119	0.175	0.250	0.291	0.8773	0.9158	0.9422	0.9512
Markov one step	low	0.5200	0.3333	0.5200	0.4783	0.013	0.006	0.013	0.011	0.3751	0.2164	0.3751	0.3367
	med	0.5200	0.4286	0.5000	0.4545	0.013	0.009	0.012	0.010	0.3751	0.2932	0.3565	0.3156
	high	0.4545	0.4545	0.5200	0.4000	0.010	0.010	0.013	0.008	0.3156	0.3156	0.3751	0.2693
Markov two step	low	0.4971	0.5076	0.5221	0.5019	0.256	0.267	0.283	0.261	0.4925	0.5031	0.5178	0.4974
	med	0.5432	0.5654	0.5843	0.6003	0.308	0.337	0.364	0.389	0.5391	0.5617	0.5808	0.5971
	high	0.5239	0.5104	0.6219	0.6279	0.285	0.270	0.426	0.437	0.5196	0.5059	0.6191	0.6252

Table C.8: All max field replacement test case results.

Model name	Field type	Precision				Recall				F_β score			
		0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s
Guess previous	low	0.9035	0.9106	0.9098	0.9076	0.103	0.112	0.111	0.108	0.8390	0.8505	0.8493	0.8456
	med	0.9160	0.9127	0.9141	0.9127	0.120	0.115	0.117	0.115	0.8596	0.8540	0.8563	0.8540
	high	0.9722	0.9714	0.9717	0.9718	0.385	0.373	0.378	0.379	0.9578	0.9562	0.9568	0.9570
Small LSTM	low	0.9722	0.9801	0.9818	0.9831	0.210	0.295	0.323	0.350	0.9385	0.9580	0.9623	0.9658
	med	0.9647	0.9826	0.9846	0.9864	0.164	0.339	0.383	0.435	0.9202	0.9645	0.9695	0.9742
	high	0.9787	0.9818	0.9834	0.9844	0.276	0.323	0.356	0.378	0.9547	0.9623	0.9666	0.9690
Large LSTM	low	0.9593	0.9699	0.9730	0.9738	0.236	0.322	0.361	0.372	0.9311	0.9509	0.9570	0.9585
	med	0.9351	0.9742	0.9818	0.9832	0.144	0.378	0.540	0.585	0.8868	0.9592	0.9739	0.9766
	high	0.9721	0.9819	0.9840	0.9844	0.348	0.543	0.615	0.629	0.9551	0.9741	0.9782	0.9789
Small GRU	low	0.9522	0.9620	0.9650	0.9663	0.239	0.304	0.331	0.344	0.9249	0.9418	0.9471	0.9493
	med	0.9259	0.9678	0.9701	0.9732	0.150	0.361	0.390	0.435	0.8808	0.9520	0.9561	0.9614
	high	0.9277	0.9520	0.9724	0.9767	0.154	0.238	0.423	0.503	0.8837	0.9245	0.9601	0.9677
Large GRU	low	0.9553	0.9753	0.9788	0.9794	0.171	0.316	0.369	0.381	0.9138	0.9556	0.9630	0.9644
	med	0.7838	0.9794	0.9826	0.9841	0.029	0.381	0.452	0.495	0.6232	0.9644	0.9713	0.9746
	high	0.9775	0.9814	0.9822	0.9822	0.347	0.423	0.441	0.442	0.9602	0.9688	0.9704	0.9705
Markov one step	low	0.9161	0.9195	0.9241	0.9245	0.131	0.137	0.146	0.147	0.8648	0.8703	0.8777	0.8785
	med	0.9437	0.9397	0.9423	0.9406	0.201	0.187	0.196	0.190	0.9104	0.9037	0.9081	0.9052
	high	0.9650	0.9651	0.9657	0.9655	0.331	0.332	0.338	0.336	0.9471	0.9472	0.9483	0.9479
Markov two step	low	0.5580	0.5889	0.6028	0.6076	0.327	0.371	0.393	0.401	0.5541	0.5855	0.5996	0.6045
	med	0.6623	0.6709	0.6818	0.6906	0.508	0.528	0.555	0.578	0.6603	0.6691	0.6803	0.6892
	high	0.5705	0.5953	0.6758	0.6964	0.344	0.381	0.540	0.594	0.5668	0.5920	0.6742	0.6952

Table C.9: All constant field replacement test case results.

Model name	Field type	Precision				Recall				F_β score			
		0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s
Guess previous	low	0.8675	0.8608	0.8675	0.8690	0.072	0.068	0.072	0.073	0.7819	0.7717	0.7819	0.7844
	med	0.7755	0.7708	0.7800	0.8254	0.038	0.037	0.039	0.052	0.6505	0.6443	0.6565	0.7195
	high	0.9570	0.9528	0.9545	0.9570	0.245	0.222	0.231	0.245	0.9303	0.9227	0.9258	0.9303
Small LSTM	low	0.9589	0.9636	0.9674	0.9697	0.140	0.159	0.178	0.192	0.9064	0.9177	0.9267	0.9323
	med	0.9406	0.9786	0.9790	0.9815	0.095	0.274	0.280	0.319	0.8644	0.9543	0.9554	0.9618
	high	0.9634	0.9729	0.9755	0.9777	0.158	0.215	0.239	0.263	0.9171	0.9400	0.9466	0.9521
Large LSTM	low	0.9306	0.9479	0.9567	0.9569	0.134	0.182	0.221	0.222	0.8788	0.9100	0.9262	0.9265
	med	0.8958	0.9755	0.9822	0.9834	0.086	0.398	0.552	0.593	0.8194	0.9617	0.9747	0.9770
	high	0.9663	0.9781	0.9819	0.9823	0.287	0.447	0.544	0.556	0.9442	0.9667	0.9742	0.9749
Small GRU	low	0.9241	0.9259	0.9355	0.9385	0.146	0.150	0.174	0.183	0.8777	0.8808	0.8966	0.9016
	med	0.9273	0.9666	0.9706	0.9728	0.153	0.347	0.396	0.429	0.8830	0.9498	0.9568	0.9607
	high	0.8378	0.9216	0.9524	0.9649	0.062	0.141	0.240	0.330	0.7455	0.8737	0.9252	0.9469
Large GRU	low	0.9245	0.9556	0.9636	0.9638	0.098	0.172	0.212	0.213	0.8533	0.9143	0.9310	0.9313
	med	0.7500	0.9797	0.9832	0.9848	0.024	0.386	0.469	0.518	0.5771	0.9650	0.9727	0.9761
	high	0.9667	0.9760	0.9767	0.9783	0.232	0.326	0.336	0.360	0.9373	0.9572	0.9586	0.9619
Markov one step	low	0.8696	0.8537	0.8605	0.8835	0.080	0.070	0.074	0.091	0.7922	0.7685	0.7785	0.8134
	med	0.9077	0.9000	0.9084	0.9211	0.118	0.108	0.119	0.140	0.8513	0.8391	0.8524	0.8728
	high	0.9341	0.9255	0.9337	0.9362	0.170	0.149	0.169	0.176	0.8943	0.8801	0.8937	0.8978
Markov two step	low	0.5350	0.5316	0.5496	0.5527	0.298	0.294	0.316	0.320	0.5308	0.5274	0.5456	0.5487
	med	0.6174	0.6337	0.6403	0.6597	0.418	0.448	0.461	0.502	0.6145	0.6311	0.6378	0.6576
	high	0.5503	0.5795	0.6246	0.6491	0.317	0.357	0.431	0.479	0.5464	0.5760	0.6219	0.6468

Table C.10: All random field replacement test case results.

Model name	Field type	Precision				Recall				F_β score			
		0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s
Guess previous	low	0.9173	0.9173	0.9203	0.9209	0.122	0.122	0.127	0.128	0.8617	0.8617	0.8667	0.8677
	med	0.9027	0.9106	0.9185	0.9154	0.102	0.112	0.124	0.119	0.8376	0.8505	0.8637	0.8585
	high	0.9742	0.9753	0.9753	0.9756	0.416	0.435	0.434	0.440	0.9615	0.9635	0.9634	0.9640
Small LSTM	low	0.9698	0.9832	0.9881	0.9888	0.193	0.352	0.498	0.530	0.9327	0.9661	0.9786	0.9804
	med	0.9649	0.9861	0.9912	0.9924	0.165	0.427	0.679	0.779	0.9207	0.9735	0.9867	0.9897
	high	0.9847	0.9914	0.9925	0.9928	0.387	0.695	0.791	0.825	0.9699	0.9873	0.9900	0.9908
Large LSTM	low	0.9500	0.9734	0.9796	0.9806	0.190	0.366	0.481	0.506	0.9138	0.9577	0.9697	0.9716
	med	0.9254	0.9808	0.9868	0.9874	0.124	0.512	0.745	0.781	0.8697	0.9720	0.9836	0.9848
	high	0.9725	0.9860	0.9877	0.9877	0.354	0.706	0.800	0.804	0.9560	0.9822	0.9854	0.9855
Small GRU	low	0.9467	0.9677	0.9754	0.9769	0.213	0.359	0.475	0.507	0.9154	0.9517	0.9653	0.9680
	med	0.9612	0.9795	0.9842	0.9846	0.297	0.574	0.747	0.768	0.9403	0.9727	0.9811	0.9819
	high	0.9447	0.9771	0.9859	0.9862	0.205	0.513	0.840	0.860	0.9121	0.9685	0.9842	0.9848
Large GRU	low	0.9175	0.9767	0.9833	0.9845	0.089	0.336	0.472	0.508	0.8401	0.9586	0.9729	0.9754
	med	0.6923	0.9845	0.9890	0.9898	0.018	0.509	0.716	0.779	0.5050	0.9755	0.9852	0.9872
	high	0.9783	0.9890	0.9905	0.9905	0.360	0.718	0.832	0.837	0.9619	0.9853	0.9886	0.9887
Markov one step	low	0.9055	0.9200	0.9155	0.9189	0.115	0.138	0.130	0.136	0.8478	0.8711	0.8638	0.8694
	med	0.9487	0.9529	0.9565	0.9549	0.222	0.243	0.264	0.254	0.9189	0.9262	0.9323	0.9295
	high	0.9645	0.9662	0.9667	0.9658	0.326	0.343	0.348	0.339	0.9461	0.9491	0.9499	0.9484
Markov two step	low	0.5519	0.5527	0.5991	0.5978	0.319	0.320	0.387	0.385	0.5479	0.5487	0.5958	0.5946
	med	0.6326	0.6500	0.6601	0.6705	0.446	0.481	0.503	0.527	0.6300	0.6477	0.6581	0.6687
	high	0.5754	0.5740	0.5768	0.5889	0.351	0.349	0.353	0.371	0.5718	0.5704	0.5732	0.5855

Table C.11: All replay field replacement test case results.

Model name	Field type	Precision				Recall				F_β score			
		0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s	0.2s	0.5s	1.0s	1.5s
Guess previous	low	0.6452	0.6562	0.6857	0.6562	0.020	0.021	0.024	0.021	0.4927	0.5050	0.5387	0.5050
	med	0.4211	0.3889	0.3529	0.5000	0.008	0.007	0.006	0.011	0.2786	0.2525	0.2244	0.3472
	high	0.8036	0.7660	0.7885	0.7925	0.045	0.036	0.041	0.042	0.6886	0.6379	0.6679	0.6733
Small LSTM	low	0.7600	0.8235	0.8571	0.8750	0.019	0.028	0.036	0.042	0.5483	0.6427	0.6992	0.7314
	med	0.6250	0.9542	0.9704	0.9742	0.010	0.125	0.197	0.227	0.3885	0.8954	0.9341	0.9435
	high	0.9155	0.9406	0.9577	0.9552	0.065	0.095	0.136	0.128	0.8105	0.8644	0.9037	0.8978
Large LSTM	low	0.7872	0.7917	0.8039	0.8649	0.037	0.038	0.041	0.064	0.6556	0.6617	0.6789	0.7695
	med	0.7959	0.9609	0.9754	0.9769	0.039	0.246	0.397	0.423	0.6676	0.9341	0.9616	0.9644
	high	0.9500	0.9716	0.9782	0.9789	0.190	0.342	0.448	0.464	0.9138	0.9542	0.9668	0.9683
Small GRU	low	0.5556	0.6571	0.7447	0.7333	0.015	0.023	0.035	0.033	0.4095	0.5162	0.6202	0.6060
	med	0.8667	0.9372	0.9537	0.9585	0.078	0.179	0.247	0.277	0.7878	0.8995	0.9274	0.9357
	high	0.7073	0.8605	0.9178	0.9420	0.029	0.074	0.134	0.195	0.5743	0.7785	0.8676	0.9076
Large GRU	low	0.6364	0.6364	0.7778	0.8095	0.014	0.014	0.028	0.034	0.4419	0.4419	0.6148	0.6604
	med	0.5294	0.9652	0.9760	0.9780	0.009	0.222	0.325	0.356	0.3367	0.9343	0.9570	0.9614
	high	0.9452	0.9615	0.9646	0.9677	0.138	0.200	0.218	0.240	0.8935	0.9266	0.9330	0.9395
Markov one step	low	0.5000	0.5000	0.4783	0.4286	0.012	0.012	0.011	0.009	0.3565	0.3565	0.3367	0.2932
	med	0.5200	0.5200	0.6000	0.5862	0.013	0.013	0.018	0.017	0.3751	0.3751	0.4545	0.4403
	high	0.6000	0.5714	0.5714	0.4000	0.018	0.016	0.016	0.008	0.4545	0.4253	0.4253	0.2693
Markov two step	low	0.4830	0.5076	0.5265	0.5256	0.242	0.267	0.288	0.287	0.4783	0.5031	0.5222	0.5213
	med	0.5565	0.5775	0.5915	0.5829	0.325	0.354	0.375	0.362	0.5526	0.5739	0.5881	0.5794
	high	0.5256	0.5440	0.6052	0.6140	0.287	0.309	0.397	0.412	0.5213	0.5399	0.6021	0.6110

Bibliography

[1] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. 20th USENIX Security Symposium 2011*, (Berkeley, CA, USA), USENIX Association, 2011.

[2] C. Valasek and C. Miller, "Remote Exploitation of an Unaltered Passenger Vehicle," tech. rep., IOActive Labs Research, 2015.

[3] M. Roesch and others, "Snort: Lightweight Intrusion Detection for Networks.," in *LISA*, vol. 99, pp. 229–238, 1999.

[4] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, pp. 16–24, Jan. 2013.

[5] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," tech. rep., Technical report, 2000.

[6] C. Miller and C. Valasek, "Adventures in Automotive Networks and Control Units," tech. rep., IOActive Labs Research, Aug. 2013.

[7] C. Miller and C. Valasek, "A Survey of Remote Automotive Attack Surfaces," tech. rep., IOActive Labs Research, 2014.

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

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

[10] M. Brooks, "Anomaly Detection on Vehicle Networks," in *Presented at the 11th ESCAR Embedded Security in Cars Conference*, 2013.

[11] K.-T. Cho and K. G. Shin, "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection," in *Proc. 25th USENIX Security Symposium*, (Austin, TX, USA), Aug. 2016.

[12] M. Muter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Proc. 2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1110–1115, June 2011.

[13] M. Markovitz and A. Wool, "Field Classification, Modeling and Anomaly Detection in Unknown CAN Bus Networks," in *escar Europe 2015*, 2015.

[14] A. Taylor, S. Leblanc, and N. Japckowicz, "Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks," in *Proc. 2016 IEEE International Conference on Data Science and Advanced Analytics*, (Montreal, Canada), IEEE Computational Intelligence Society, Oct. 2016.

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

[16] C. Smith, *The Car Hacker's Handbook: A Guide for the Penetration Tester*. No Starch Press, Mar. 2016.

[17] A. G. Illera and J. V. Vidal, "Dude, WTF in my CAN!," in *Presented at Black Hat Asia 2014*, (Singapore), 2014.

[18] A. Greenberg, "This iPhone-Sized Device Can Hack A Car, Researchers Plan To Demonstrate," Mar. 2014.

[19] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," in *Proc. IEEE Symposium on Security and Privacy 2010*, pp. 447–462, 2010.

[20] R. M. Ishtiaq Roufa, H. Mustafaa, S. O. Travis Taylora, W. Xua, M. Gruteserb, W. Trappeb, and I. Seskarb, "Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study," in *19th USENIX Security Symposium, Washington DC*, pp. 11–13, 2010.

[21] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava, "Non-invasive Spoofing Attacks for Anti-lock Braking Systems," in *Cryptographic Hardware and Embedded Systems - CHES 2013* (G. Bertoni and J.-S. Coron, eds.), no. 8086 in Lecture Notes in Computer Science, pp. 55–72, Springer Berlin Heidelberg, Jan. 2013.

[22] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: a story of telematic failures," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[23] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks – Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, pp. 11–25, Jan. 2011.

[24] D. Wagner and P. Soto, "Mimicry Attacks on Host-based Intrusion Detection Systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, (New York, NY, USA), pp. 255–264, ACM, 2002.

[25] J. Ma and S. Perkins, "Online Novelty Detection on Temporal Sequences," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), pp. 613–618, ACM, 2003.

[26] R. Maxion and K. Tan, "Benchmarking anomaly-based detection systems," in *Proceedings International Conference on Dependable Systems and Networks, 2000. DSN 2000*, pp. 623–630, 2000.

[27] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection for Discrete Sequences: A Survey," *IEEE Trans. Knowl. Data Eng.*, vol. 24, pp. 823–839, May 2012.

[28] S. Budalakoti, A. Srivastava, and M. Otey, "Anomaly Detection and Diagnosis Algorithms for Discrete Symbol Sequences with Applications to Airline Safety," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 39, pp. 101–113, Jan. 2009.

[29] E. Keogh, J. Lin, and A. Fu, "HOT SAX: efficiently finding the most unusual time series subsequence," in *Proc. Fifth IEEE International Conference on Data Mining*, pp. 226–233, Nov. 2005.

[30] A. Valdes and S. Cheung, "Communication pattern anomaly detection in process control systems," in *IEEE Conference on Technologies for Homeland Security, 2009. HST '09*, pp. 22–29, 2009.

[31] B. Schlkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," *Neural Computation*, vol. 13, pp. 1443–1471, July 2001.

[32] A. McCallum, K. Nigam, and others, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48, Citeseer, 1998.

[33] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer Intrusion: Detecting Masquerades," *Statistical Science*, vol. 16, pp. 58–74, Feb. 2001.

[34] R. Maxion and T. Townsend, "Masquerade detection using truncated command lines," in *International Conference on Dependable Systems and Networks, 2002. DSN 2002. Proceedings*, pp. 219–228, 2002.

[35] R. Maxion, "Masquerade detection using enriched command lines," in *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings*, pp. 5–14, June 2003.

[36] R. Maxion and T. Townsend, "Masquerade detection augmented with error analysis," *IEEE Transactions on Reliability*, vol. 53, pp. 124–147, Mar. 2004.

[37] K. Wang and S. Stolfo, "One-Class Training for Masquerade Detection," in *Proc. ICDM Workshop on Data Mining for Computer Security*, (Melbourne, FL), Nov. 2003.

[38] M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, June 2014.

[39] S. S. Khan and M. G. Madden, "A Survey of Recent Trends in One Class Classification," in *Artificial Intelligence and Cognitive Science* (L. Coyle and J. Freyne,

eds.), Lecture Notes in Computer Science, pp. 188–197, Springer Berlin Heidelberg, Aug. 2009. DOI: 10.1007/978-3-642-17080-5_21.

[40] M. I. Baron, "Nonparametric adaptive change point estimation and on line detection: Nonparametric adaptive change point," *Sequential Analysis*, vol. 19, pp. 1–23, Jan. 2000.

[41] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy, 1999*, pp. 133–145, 1999.

[42] V. Chandola, V. Mithal, and V. Kumar, "Comparative Evaluation of Anomaly Detection Techniques for Sequence Data," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, (Washington, DC, USA), pp. 743–748, IEEE Computer Society, 2008.

[43] S. Coull, J. Branch, B. Szymanski, and E. Breimer, "Intrusion detection: a bioinformatics approach," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pp. 24–33, Dec. 2003.

[44] S. E. Coull and B. K. Szymanski, "Sequence alignment for masquerade detection," *Computational Statistics & Data Analysis*, vol. 52, pp. 4116–4131, Apr. 2008.

[45] T. Lane and C. E. Brodley, "An Empirical Study of Two Approaches to Sequence Learning for Anomaly Detection," *Machine Learning*, vol. 51, pp. 73–107, Apr. 2003.

[46] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards Parameter-free Data Mining," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, (New York, NY, USA), pp. 206–215, ACM, 2004.

[47] N. Ye, X. Li, Q. Chen, S. Emran, and M. Xu, "Probabilistic techniques for intrusion detection based on computer audit data," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 31, pp. 266–274, July 2001.

[48] R. Maxion and K. Tan, "Anomaly detection in embedded systems," *IEEE Trans. Comput.*, vol. 51, pp. 108–120, Feb. 2002.

[49] K. Tan and R. Maxion, ""Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector," in *2002 IEEE Symposium on Security and Privacy, 2002. Proceedings*, pp. 188–201, 2002.

[50] W.-K. Ching, M. K. Ng, and E. S. Fung, "Higher-order multivariate Markov chains and their applications," *Linear Algebra and its Applications*, vol. 428, pp. 492–507, Jan. 2008.

[51] E.-S. Platzer, J. Ngele, K.-H. Wehking, and J. Denzler, "HMM-Based Defect Localization in Wire Ropes A New Approach to Unusual Subsequence Recognition," in *Pattern Recognition* (J. Denzler, G. Notni, and H. Se, eds.), no. 5748 in Lecture Notes in Computer Science, pp. 442–451, Springer Berlin Heidelberg, Jan. 2009.

[52] G. Florez-Larrahondo, S. M. Bridges, and R. Vaughn, "Efficient Modeling of Discrete Events for Anomaly Detection Using Hidden Markov Models," in *Information Security* (J. Zhou, J. Lopez, R. H. Deng, and F. Bao, eds.), no. 3650 in Lecture Notes in Computer Science, pp. 506–514, Springer Berlin Heidelberg, Jan. 2005.

[53] K. Yamanishi and Y. Maruyama, "Dynamic Syslog Mining for Network Failure Monitoring," in *Proceedings of the Eleventh ACM SIGKDD International Con-*

ference on Knowledge Discovery in Data Mining, KDD '05, (New York, NY, USA), pp. 499–508, ACM, 2005.

[54] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[55] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv:1412.3555 [cs]*, Dec. 2014. arXiv: 1412.3555.

[56] J. Martens and I. Sutskever, "Learning Recurrent Neural Networks with Hessian-Free Optimization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (L. Getoor and T. Scheffer, eds.), ICML '11, (New York, NY, USA), pp. 1033–1040, ACM, June 2011.

[57] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *arXiv:1409.2329 [cs]*, Sept. 2014. arXiv: 1409.2329.

[58] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8624–8628, May 2013.

[59] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015. arXiv: 1404.7828.

[60] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Berlin; New York: Springer, 2012.

[61] A. Graves, "Generating Sequences With Recurrent Neural Networks," *arXiv:1308.0850 [cs]*, Aug. 2013. arXiv: 1308.0850.

[62] A. Graves and N. Jaitly, "Towards End-To-End Speech Recognition with Recurrent Neural Networks," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1764–1772, 2014.

[63] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *arXiv:1409.3215 [cs]*, Sept. 2014. arXiv: 1409.3215.

[64] W. Zaremba and I. Sutskever, "Learning to Execute," *arXiv:1410.4615 [cs]*, Oct. 2014. arXiv: 1410.4615.

[65] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," *arXiv:1409.1259 [cs, stat]*, Sept. 2014. arXiv: 1409.1259.

[66] M. Hermans and B. Schrauwen, "Training and Analysing Deep Recurrent Neural Networks," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 190–198, Curran Associates, Inc., 2013.

[67] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," in *Proc. 23rd European Symposium On Artificial Neural Networks, Computational Intelligence and Machine Learning*, (Bruges, Belgium), pp. 89–94, 2015.

[68] S. Chauhan and L. Vig, "Anomaly detection in ECG time signals via deep long short-term memory networks," in *Proc. IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015. 36678 2015*, pp. 1–7, Oct. 2015.

[69] A. Nanduri and L. Sherry, "Anomaly detection in aircraft data using Recurrent Neural Networks (RNN)," in *Proc. 2016 Integrated Communications Navigation and Surveillance*, pp. 5C2–1–5C2–8, Apr. 2016.

[70] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection," in *Presented at ICML2016 Anomaly Detection Workshop*, (New York, NY), July 2016.

[71] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep Structured Energy Based Models for Anomaly Detection," in *Proc. 33rd International Conference on Machine Learning*, pp. 1100–1109, 2016.

[72] M. Soelch, J. Bayer, M. Ludersdorfer, and P. van der Smagt, "Variational Inference for On-line Anomaly Detection in High-Dimensional Time Series," *arXiv:1602.07109 [cs, stat]*, Feb. 2016. arXiv: 1602.07109.

[73] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and . Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, p. 28252830, Oct. 2011.

[74] K. Greff, R. K. Srivastava, J. Koutnk, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–11, 2016.

[75] K. Cho, B. van Merrinboer, . Glehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[76] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[77] M. Sundermeyer, R. Schlter, and H. Ney, "LSTM Neural Networks for Language Modeling.," in *INTERSPEECH*, pp. 194–197, 2012.

[78] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[79] Y. Gal, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," *arXiv:1512.05287 [stat]*, Dec. 2015. arXiv: 1512.05287.

[80] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of The 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

[81] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, Aug. 2013.

[82] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 27:1–27:27, May 2011.

[83] M. Baron, "Asymptotically Pointwise Optimal Change Detection in Multiple Channels," *Sequential Analysis*, vol. 33, pp. 440–457, Oct. 2014.