Master's Dissertation

Intrusion Detection in the CAN bus using Long Short-Term Memory Networks

Araya Kibrom Desta

September 13, 2019

Graduate School of Information Science
Nara Institute of Science and Technology

A Master's Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Master of ENGINEERING

Araya Kibrom Desta

Thesis Committee:

Professor Kazutoshi Fujikawa	(Supervisor)
Professor Yuichi Hayashi	(Co-supervisor)
Associate Professor Ismail Arai	(Co-supervisor)

Intrusion Detection in the CAN bus using Long Short-Term Memory Networks*

Araya Kibrom Desta

Abstract

Nowadays, vehicles are equipped with multiple Electronic Control Units (ECUs) each of which communicates with one another using a specification called Controller Area Network (CAN). CAN provide its own share of benefits in modernizing automobiles, but it also brought along a security issue to the automotive industry. CAN bus does not have any mechanism for encrypting or authenticating CAN payloads. As a countermeasure against these drawbacks, we have experimented on identifying intrusions in the CAN bus using Long Short-Term Memory Networks (LSTM). LSTM networks are trained to predict forthcoming payloads and related attributes by looking at information that has already appeared in the CAN bus at some instant in time. The predicted values are compared with actual values, that are either sent by an intruder or the benign ones. Depending on how close the prediction is with the actual payload, we managed to effectively identify anomalies in an acceptable accuracy rate, up to 98%. Furthermore, we also experimented on an intrusion detection system (IDS) based on the sequence of arbitration IDs of the packets. The trained network learns about the sequence of packet IDs to predict a forth coming packet ID. This IDS is similar with the previous one except an anomaly signal is generated by comparing only the predicted ID and actual ID. We have tested our methods with a variety of attacks on the CAN bus and demonstrated how effective our detection methods are.

Keywords:

CAN bus, LSTM, In-vehicle Networks, Automotive Security, Anomaly Detection

*Master's Dissertation, Graduate School of Information Science, Nara Institute of Science and Technology, September 13, 2019.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Kazutoshi Fujikawa for the continuous support of my Masters study and related research, for his patience, motivation, and immense knowledge. Besides I would also like to thank Ass. Prof. Ismail Arai for all the constructive comments he gives us during the weekly meetings. His guidance helped me in all the time of research and writing of this thesis. Also I am grateful to the professors in our Laboratory for the immense support they extended so as for me to be steered in the right direction.

I also thank my fellow labmates for the stimulating discussions, and for all the fun we have had in the last two years. In particular, I am grateful to my labmate Shuji Ohira for the support he provided in collecting data for the research.

Last but not the least, I would like to thank my family: my parents and to my brothers and sister for supporting me spiritually throughout writing this thesis and my life in general.

Author
Araya Kibrom Desta

1 Introduction

Background

In the old days, automobiles used to be a mechanical device with no networking capability and a few electronic devices. Later in time, the number of Electronic Control Units (ECUs) inside automobiles have increased to some extent. But still, there was not any network that connects these ECUs. A wiring harness was being used to connect each of these devices until controller area network was introduced by BOSCH [22] as a multi-master, message broadcast system that specifies a maximum signaling rate of 1 megabit per second. As of the CAN bus release, the number of ECUs in vehicles started to increase due to the simplicity of inter-networking. Nowadays, automobiles have up to 100 ECUs [11], each of which communicates using the controller area network (CAN), which is the de-facto standard of most present-day vehicles.

CAN was introduced to the automotive industry for the sole purpose of enhancing the driving experience. Before CAN was invented, ECUs used to communicate with one another by a point to point wiring system. After the introduction of the CAN bus to vehicles, the wiring harness between ECUs has reduced to less and simple wiring, which in return saves the overall cost and time. Additionally, the specification calls for high immunity to electrical interference and the ability to self-diagnose and repair data errors. These features have led to CAN's popularity in a variety of industries including building automation, medical, and manufacturing [3].

A modern vehicle integrates a heterogeneous network of ditributed ECUs . The ECUs communicate over different buses and protocols ranging from the low speed bus Local Interconnect Network (LIN) over Control Area Network (CAN) to fast buses like FlexRay or Ethernet. The various buses are interconnected by gateways, creating a fully connected network [23]. For this research, we are mainly focusing on intrusion detection in the CAN bus that is used for much of vehicle communications [16].

Contributions

In order to avoid the security risks in the in-vehicle network, in this paper, we experimented on identifying intrusions in the CAN bus by using neural networks. And inspired by Long Short-Term Memory network's (LSTM) design to overcome short term memory problems even in case of noisy, in-

compressible input sequences. That is achieved by an efficient, gradient-based algorithm for an architecture enforcing constant (thus, neither exploding nor vanishing) error flow through internal states of special units. We have proposed an IDS using LSTM (LSTM-IDS) with the following contributions:

(1) We proposed LSTM-IDS with a suitable architecture of layers that is capable of learning to identify intrusions. We further studied how the architecture of layers have a big effect on the prediction capability of the neural network. The conventional method proposed by Japkowicz et el. [26] uses the bits in a packet to identify anomalies but they didn't take into consideration the meaning of each packet. Our methodology further reverse engineers the packets to learn about the meaning of each bit in a packet and this helps in improving the intrusion detection from an average detection of 33% to 98% for the first car's arbitration ID 0C000027.

(2) We fabricated different types of attacks to test if the trained network is competent enough to be used as an intrusion detection method. Our experimental results show that with the right combination of parameters, LSTM is suited for intrusion detection in in-vehicle networks.

(3) We also compared the detection performances for most arbitration IDs in the can bus with the conventional method that uses different attributes and network architecture to accomplish the task. Our proposed method significantly outperforms the conventional method. Moreover, we discussed how this LSTM-IDS can be implemented on a real vehicle.

(4) Additionally, we implemented an IDS based on the analysis of ID sequences. This detection system focuses on predicting an arbitration ID that will appear in future by using the last 20 IDs. This method leaves none of the packets behind unlike the LSTM-IDS which can't be implemented for some of the arbitration IDs unless we have enough data to go on with.

2 Attacking the CAN bus

2.1 CAN Specification

CAN network is an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus. The specification calls for high immunity to electrical interference and the ability to self-diagnose and repair data errors. These features have led to CAN's popularity in a variety of industries including building automation, medical, and manufacturing.

A controller area network (CAN) is ideally suited to the many high-level industrial protocols embracing CAN and ISO-11898:2003 as their physical layer. Its cost, performance, and upgradeability provide for tremendous flexibility in system design. In a CAN network, many short messages like temperature or RPM are broadcast to the entire network, which provides for data consistency in every node of the system [4].

The CAN communication protocol is a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP). CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. CD+AMP means that collisions are resolved through a bit-wise arbitration, based on a pre-programmed priority of each message in the identifier field of a message. The higher priority identifier always wins bus access. That is, the last logic high in the identifier keeps on transmitting because it is the highest priority. Since every node on a bus takes part in writing every bit "as it is being written," an arbitrating node knows if it placed the logic-high bit on the bus.

Figure 2.1: CAN Packet

In our IDS, we are mainly focusing on using the arbitration ID and the data field of CAN packet as it is depicted in Figure 2.1. CAN has two standards, the first standard with 11-bit identifier, provides for signaling rates from 125 kbps to 1 Mbps. This standard was later amended with the "extended" 29-bit identifier. The standard 11-bit identifier field provides for 2^{11} , or 2048 different message identifiers, whereas the extended 29-bit identifier provides

for 2^{29} , or 537 million identifiers. Both of the standards are almost similar, except for some control and identifier fields. ECU's bus access is event-driven and takes place randomly. If two nodes try to occupy the bus simultaneously, access is implemented with a nondestructive, bit-wise arbitration, CAN_ID in Figure 2.1, the node winning arbitration just continues on with the message without the message being destroyed or corrupted by another node. The lower the binary message identifier number, the higher its priority. An identifier consisting entirely of zeros is the highest priority message on a network since it holds the bus dominant the longest. And the data part in, Data Figure 2.1, contains information that is disseminated to the ECUs depending on the driver's actions. Our proposed method uses these two parts of the packet and the time the packet appears in the CAN bus to train LSTM-IDS that can identify anomaly messages.

2.2 Security Issues of the CAN bus

Even though CAN provides its share of advantages in modernizing the automotive industry, it also brought along a new security issue to vehicles. CAN is vulnerable to various types of attacks due to its security weakness [9]. An attacker has two ways of gaining access to the internals of in-vehicle network [10]. The first is physical access. An attacker, let it be a mechanic or a person who rents a car, can insert a malicious component into a car's internal network via the ubiquitous OBD-II port. Th attacker may leave the malicious malware in one of the vehicles connected ECU's to create an intended attack on the target vehicle. The second is through exploiting the numerous wireless interfaces implemented in the modern automobile to remotely gain access to the networks.

Once attackers get the access to the internal networks of the CAN bus, they might try to take the advantage of security issues in the CAN bus for their advantage. CAN bus use a broadcast type of communication method, whenever a message appears in the CAN bus it is accessible by all its connected nodes. This nature of the CAN bus enabled attackers to easily snoop on all communications or send packets to any other node on the network, which later on can help the attackers to reverse engineer the packets. Whenever two competing nodes want to use CAN bus resource, CAN bus handles the priority of the nodes using message arbitration ID. A node with a lower arbitration ID will be given a higher priority to use the CAN bus resources. This property of the CAN bus makes it extremely vulnerable to a denial-of-service attack. An attacker can continuously inject a packet with a lowest arbitration ID to the CAN bus, and this causes all other packets to be rejected from using the CAN bus. CAN bus also has no mechanism for encrypting or authenticating CAN packets which makes it easier for attackers to snoop on all connection nodes of the network.

By exploiting these security vulnerabilities of the CAN bus many security researchers have been able to control critical vehicle components. Miller and Valasek [15] illustrated on how they can control critical car components by remotely injecting packets to the target vehicle's CAN bus. Other security researchers have also demonstrated on how to remotely attack Tesla motors [12]. They discovered multiple security vulnerabilities and successfully implemented remote, aka none physical contact, control on Tesla Model S in both Parking and Driving Mode.

2.3 Attack Model

Dumping a CAN packet from a vehicle would show us a list of arbitration IDs and their respective data indexed by the time stamp. Our first LSTM-IDS model is trained with these three variables and one more variable, number of messages. Number of messages is acquired by counting the number of messages in a specific time window. To test the effectiveness of our trained network, we have simulated a variety of attacks on the CAN bus. Most of the attacks we have created consider the fact that packets are either removed or added deviating the sequence of the packets. We have fabricated five types of attacks that adversaries would try to inject to the CAN bus. With these types of attacks, we have tested the prediction accuracy of our trained network. These types of attacks are listed below

Fuzzy Attack - The case which adversaries implement a bunch of packets to be injected in the CAN bus for the purpose of learning how different ECUs behave. To simulate this specific type of attack we have created a program that generate a packet with a size 8 bytes or less, depending on the data length code (DLC) of each arbitration ID, and inserted in a random time location. In particular, our fuzzy attack generator randomly selects a bit 1 or 0 to create one packet with the size of a targeted arbitration ID's DLC.

Insertion Attack - This attack is simulated by randomly selecting a packet from the list of packets dumped from each vehicle to be inserted in a different time location. The time for this attack messages is assigned by calculating the average arrival time from two of the neighboring messages.

Drop attack - The case in which attackers drop some payloads from a targeted ECU preventing the messages from being processed or used by a receiving ECU. This attack is simulated by removing two packets in 1 second time window. Hence causing an unexpected situation in the in-vehicle networks.

Impersonation Attack - Impersonation attack is an attack by which adversaries drop some messages and send a fabricated replacement message

to get control of critical car components. Our impersonation attack is same as insertion attack except a message has to be dropped from the CAN bus and it will be replaced by a randomly selected packet.

Denial of Service Attack (DoS) - An attack meant to force a targeted ECU to continuously back off from using CAN resource, making it inaccessible from participating in the network. DoS attacks can be accomplished in vehicles by flooding CAN with a message integrated with lowest arbitration ID. To simulate this attack we have delayed the arrival time of messages by up to 5ms.

The main goal of this research would be for the trained network to identify any of the above attacks by looking at information that has already appeared in the bus. The trained network has to differentiate which of the packet sequences contain anomalies and which of these sequences are benign.

And for the case of the second LSTM-IDS, we only use the ID of the packets to identify anomalies in the CAN bus. This method can only identify attacks that either remove a packet from or add a packet to the CAN bus changing the normal flow of the packets. Therefore, for this method we are mainly experimenting on the packet ID prediction capability of the LSTM.

3 Related Work

The security vulnerabilities of CAN bus can be improved by two methods. The first method is to use intrusion prevention techniques by tackling attacks before the packets arrive in the targeted host. CAN bus has no authentication field or any mechanism to identify the source of a message. As the solution for these drawbacks P. Mundhenk et al. [17] have proposed a way to use a light message authentication for secure automotive networks. In their research, the keys required for secure communication with symmetric cryptography, such as the Advanced Encryption Standard (AES) or CMACs, can be exchanged between ECUs. In G. Han et al. [7], a different approach to introducing MACs into FlexRay is presented. Besides that intrusions can also be prevented by encrypting CAN packets, Y. Wu et al. [28]. This method proposes a security protocol for the CAN system based on AES-128 encryption and HMAC function. Despite the fact that this research can provide a way to enhance CAN security, implementation of these methods in a real vehicle would be of a great risk due to the real-time communication requirements of in-vehicle networks.

The second approach for enhancing the security of the CAN bus is to implement intrusion detection mechanism in the CAN bus. When a packet appears in the CAN bus, together with other information, it contains timing information, data field, and arbitration ID. Researchers have been illustrating on using timing information to identify anomalies in the CAN bus. H. M. Song et al. [24] used time intervals of CAN messages for intrusion detection. Their method uses the timing information of each ID by taking the fact that there is a unique time interval for each arbitration ID because each ECU connected to CAN bus sends messages regularly. When a new message appears on the CAN bus, their (Intrusion Detection System) IDS checks the arbitration ID and computes the time interval from the arrival time of the latest messages. According to their method, an anomaly message's time interval is shorter or longer than normal range.

A. Taylor et al. [25] experimented on using the frequency of messages to identify an anomalous sequence of messages. Their anomaly detector works by calculating statistics about ongoing network traffic and comparing them with historical values. They also trained OCSVM detector that is trained using variables collected from the flow of the traffic. OCSVM was able to detect very short packet insertions with acceptable false alarm rates.

Kuwahara T. et al. [11] also used CAN message frequencies for in-vehicle network intrusion detection. This approach analyzes the normal behavior of CAN messages in terms of their message frequency. They used two features,

total counting feature and ID counting feature, which count the number of messages and available IDs in some time windows. Using these two features, they detected whether a message sequence is malicious by calculating the nearest neighbor distance between a test sequence and a normal message sequence, and perform a statistical test and compute the p-value. Then used the calculated value as a threshold for differentiating benign and anomalous message sequences.

It's clear to see that most of the aforementioned research heavily focus on using timing information of CAN messages to detect intrusions. Furthermore, this techniques can only be implemented to periodic CAN messages, but all the aforementioned methods fail to detect any anomaly sent with a non-periodic arbitration ID or a compromised ECU which attackers are capable of manipulating its timing information. This problem can be tackled by implementing an anomaly detection method that can monitor sequence of data flows than one that focuses on only timing information.

Marchetti et al. [13] have proposed an intrusion detection method that monitors the traces of arbitration IDs that appear in the CAN bus. From the analyses of different traces recorded from their test car, they identified some recurring patterns on the ID sequences. Every IDs of their test model is followed only by a subset of all the available IDs, thus limiting the admissible transitions between all IDs. From this result, they developed an algorithm that is able to detect anomalies in the ID sequences. During training phase they used CAN bus logs to create a data structure that contains all the possible transitions between consecutive IDs observed in legit CAN traffic traces that did not contain any attack which outputs a transition matrix. During detection phase, they analyze the transition matrix and the traffic traces to identify possible attacks. One drawback of this method is its incapability to detect impersonation attacks. To detect impersonation attacks we need to monitor the data, not alone the IDs.

Nanduri et al. [18] and Japkowicz et al. [26] have proposed a method that can solve this issue in aircraft and vehicles respectively. Both of these methods focus on identifying intrusion by training a neural network that is capable of predicting the next packet data values, and its errors are used as a signal for detecting anomalies in the sequence. These approaches can identify some of the attacks that can not be detected by the previously mentioned methods, but it can't detect any denial of service attacks. DoS attack is simulated by delaying some of the message. When messages are delayed the conventional method has no way of telling if any of the messages arrived are on time because it only considers the packet bits for the training. Furthermore, this method has a poor performance compared to our proposed method due to all the packet bit flipping patterns the LSTM has to learn to make a right prediction. Our work is an improvement to the conventional LSTM-IDS method which takes the advantage of both using the timing information and the bit prediction method to further improve the intrusion detection capability of LSTM network. The

drawback with training a single LSTM network for each arbitration falls on those IDs which appear in the CAN bus less frequently. To solve this issue we created a second IDS that only considers the arbitration IDs.

4 IDS using LSTM

To identify anomalies sent with a non-periodic arbitration ID or a compromised ECU, the detection methods should track how the data is changing through time. Data sequence learning methods can be used to track data of payloads through time. A simple strategy for general sequence learning is to use neural networks. By training a neural network about a sequence of messages, it will be able to predict forthcoming payloads using information which has already appeared in the CAN bus.

We trained a neural network to predict subsequent CAN payload data using previously seen data sequences. To train the network we need to prepare training data, unfortunately, we couldn't find public data that can fit our network. Due to this situation, we had to prepare our own benign and anomalous payload sequences.

The conventional method in [26] can identify anomalies even if adversaries mimic the timing information of a compromised ECU. This is done by learning the flow of data to see if there are packets out of a correct sequence. Even though, this method can identify any fuzzy attack at 1.0 accuracy, it fails to detect delayed messages because it only uses the data portion of packets for anomaly detection. Messages can be delayed due to an insertion attack or a DoS attack. Our method further incorporates the arrival time of packets to see if each message is arriving in the CAN bus at the expected time. This helps us to further improve the detection accuracy and identify DoS attacks that can't be detected by the conventional method.

4.1 Long Short-Term Memory Networks - LSTM

We selected RNN for our research because we wanted to see if RNNs are capable of connecting previous information to the present task, such as using previous payloads might inform the understanding of the present payload [19]. But due to the long-term dependencies problem, RNNs are incapable of retaining information for a long time [1]. The basic problem is that gradients propagated over many stages tend to either vanish (most of the time) or explode(rarely, but with such damage to the optimization). Recurrent networks involve the composition of the same function multiple times, once per time step. These compositions can result in extremely nonlinear behavior [6]. Long Short-Term Memory Networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies [5]. The

main contribution of LSTM is the capability of using self-loops to produce paths where the gradient flow for long durations.

Instead of having a single neural network layer, LSTM has four layers (forget gate layer, input gate layer, candidate layer, and output layer), and each of these components is updated in every state according to Figure 4.1.

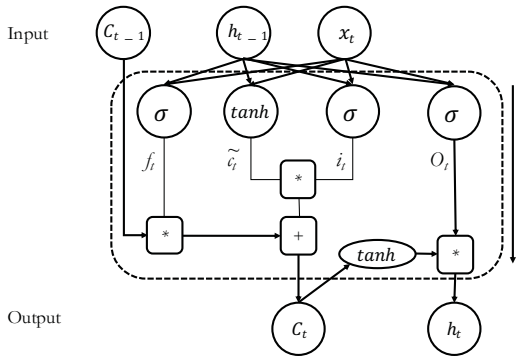


Figure 4.1: Diagram for LSTM at the time step t including all the layers

Initially, we have to decide the information that should pass from one state to the next state. In LSTM such decision is made by a sigmoid layer called the forget gate layer(f_t) 1 represents "completely keep this" and 0 represents "completely get rid of this." The function that evaluates this value is shown in Equation 4.1.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{4.1}$$

The information that will be stored in the cell state will be decided by using the input gate layer and candidate layer, Equation 4.2 and Equation 4.3 respectively. Next, the values from these two layers are combined to create an update to the state, Equation 4.4.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{4.2}$$

$$\widetilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{4.3}$$

$$C_t = f + t * C_{t-1} + i_t * \widetilde{C}_t \tag{4.4}$$

Finally, Equations 4.5 and 4.6 evaluate an output value and hidden state value that passes to the next layer respectively.

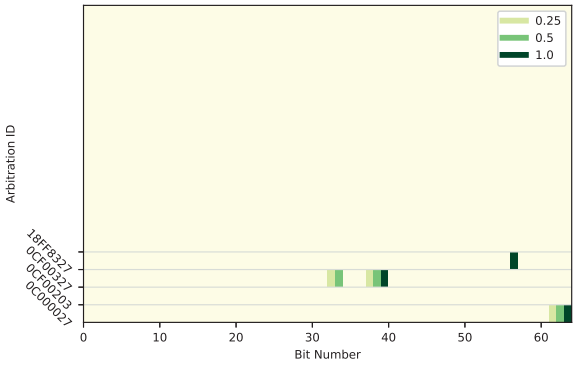
$$O_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{4.5}$$

$$h_t = O_t * \tanh(C_t) \tag{4.6}$$

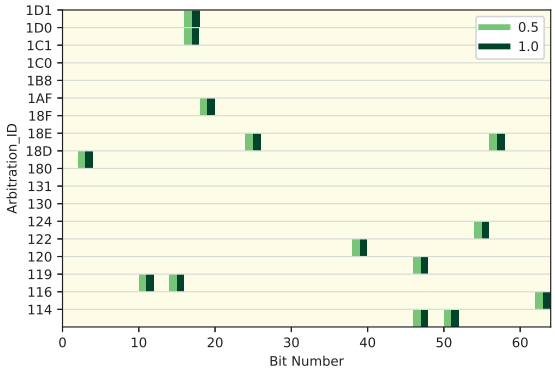
4.2 IDS through Analysis of CAN frame sequences

4.2.1 Input Data Preprocessing

The focus of this method is to identify an anomalous message in short time before attackers cause much damage on a targeted car. We implemented a way to identify anomalies in less than one second from the time the anomalies start to appear in the bus. We used 1 second so that we would be able to collect enough data for processing. For this reason, we have collected input and output values to our LSTM network in every 60ms seconds for car A and 75ms seconds for car B. We used different time ranges due to the behavior of each car's packets. The cars have a packet with two or more bits put together with the rest of the bits that are used as a counter.



(a) Car A



(b) Car B

Figure 4.2: Bits identified as a counter in the test cars

We used 75ms seconds for car B for the reason that the bit flip rate algorithm was not able to capture the transformation of the bits that are used as a

counter because car B has more number of bits, up to three, used as a counter. Figure 4.2 shows the counter bits available with each car. In the figure, Car A has up to three counter bits (0.25, 0.5, 1.0). Hence, three color codes. Car B has only two counters in some of the packets. The input to the neural network will be the values calculated in the specified time for a range of 1 second. Using the inputs collected in 1 second, we predict the next values. The values we used as an input to the network and as output are listed below.

- **Number of messages** n **-** This variable counts the number of messages that appear in the CAN bus in every 60ms for car A and 75ms for car B.

- **Average Time Difference** T_{diff} **-** In every range of milliseconds we collect messages associated with their respective arbitration IDs. Each of these payloads have timing information that shows the time the payloads appeared in the CAN bus. To calculate the average time difference we collected a bunch of messages with their particular timing information in the specified time range, then we took the average time difference between each packet.

- **Bit flip rate** b_0**,** b_1**,...,**b_{63} **-** A CAN payload can be 64 bits long or less depending on the arbitration ID. To calculate bit flip rate we employ an algorithm used by READ [14]. The bit flip rate is evaluated for each bit of the payload, independent of its neighbors. Bit flip rate algorithm counts the number of bit flips (from 0 to 1 and vice versa) occurrences among consecutive messages collected in the specified instant of time. Then the bit flip rate is obtained by dividing the number of bit flips to - n. The result from this is an array of k elements, each representing the bit flip rate for a single bit of the data field for a given ID, where k represents the number of bits included in the payload as defined by the value of the DLC field. The complete algorithm that calculates the bit flip rate is shown in Algorithm 1.

Algorithm 1 Pseudo-code of bit flip rate calculation

```
1:  function BitFlip-Rate(messageList, DLC)
2:      payloadLen ← len(messageList)
3:      bitFlip ← array(DLC)
4:      previous ← messgeList[0]
5:      while item in messageList do
6:          for ix in range(1..DLC) do
7:              if item[ix] ≠ previous[ix] then
8:                  bitFlip[ix] + +
9:      for ix = 0; ix < DLC; ix + + do
10:         bitFlip[ix] ← bitFlip[ix]/payloadLen
11:     return bitFlip
```

The advantage of using bit flip rate as an input feature to our neural network is it helps us to convert the counter bits to constant bit flip rate values saving the neural network from learning the flipping of these specific bits. For instance, for car A's arbitration ID 0CF00327, Figure 4.2a, it has 3 counter bits that count from 000 to 111. In the case of using bits as an input feature, the network has to learn how each of these bits are flipping from the previous counter to the next one but if we use bit flip rate these three bits will have a bit flip rate of 0.25, 0.5 and 1.0 for all the time windows in the training.

4.2.2 IDS Network Architecture

Our LSTM architecture is similar to the architectures used by the conventional LSTM except some changes in input features, network architecture and some parameter changes so as to fit with our training data. The Keras model [2] we implemented is depicted in Figure 4.3.

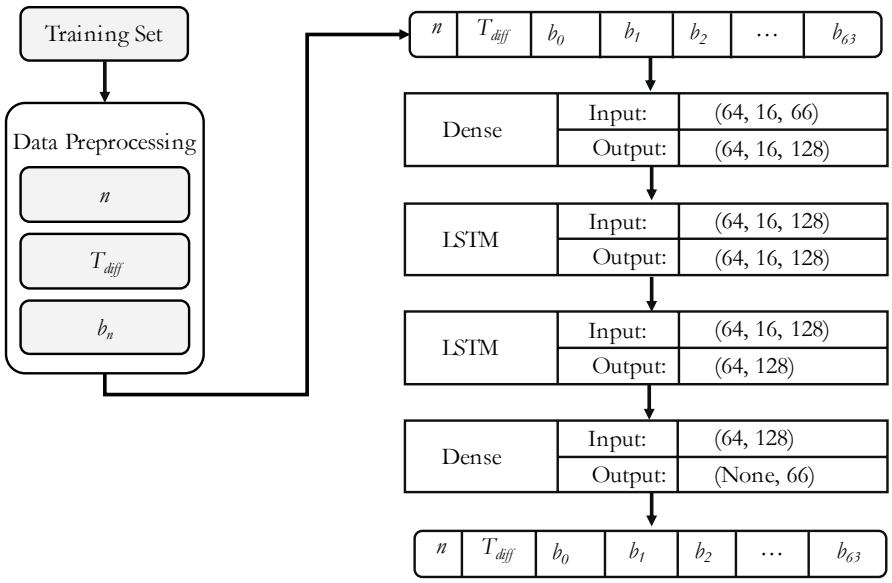


Figure 4.3: IDS Network Architecture

First we need to format our training data to a format that a neural network can understand. All our input data is numerical data, so we don't need any vectorization. But the input data is on a different scale. n is an integer that ranges from 0 to some unbound value and T_{diff} also varies accordingly. To bring all the input values to a comparable range, we used L_1 normalization that brings all the input values to a range between 1 and 0. Once we brought all the 66 features to a comparable range, a single tensor with a size of (t,66) is created, t is the total sequence of training data. Next, we converted this tensor to input and output sequence. This is created by continuously iterating over the training data to convert it to an overlapping input and output sequences. Each iteration goes through 16 (step) consecutive sequences and predicts the

subsequent data. A batch size of 64 is selected for the training, i.e during each batch the shape of the training data becomes (batch_size(64), look back(16), n_feature(66)). After we setup our input and out put sequence, we trained the preprocessed data using the architecture shown in Figure 4.3.

The architecture we selected for our IDS contains two non-recurrent hidden layers and two LSTM layers. The training first goes through a non-recurrent hidden layer with 128 units and later passes through two LSTM layers each with the 128 units. All of these three layers have a rectified linear unit (ReLU) activation function. The LSTM layers have also a 0.2 dropout that is used to overcome over-fitting during training. Later on, the output from the last LSTM layer is fed to a single dense layer with 66 units. The output from the last dense layer is a NumPy array with 66 units that has an output for each input feature.

After we generate our trained model, the IDS phase passes through the steps shown in Figure 4.4. First, we read the 1 second data collected from the available sequence of CAN data, let it be a dumped data as in offline detection or a live data being read from the CAN bus.

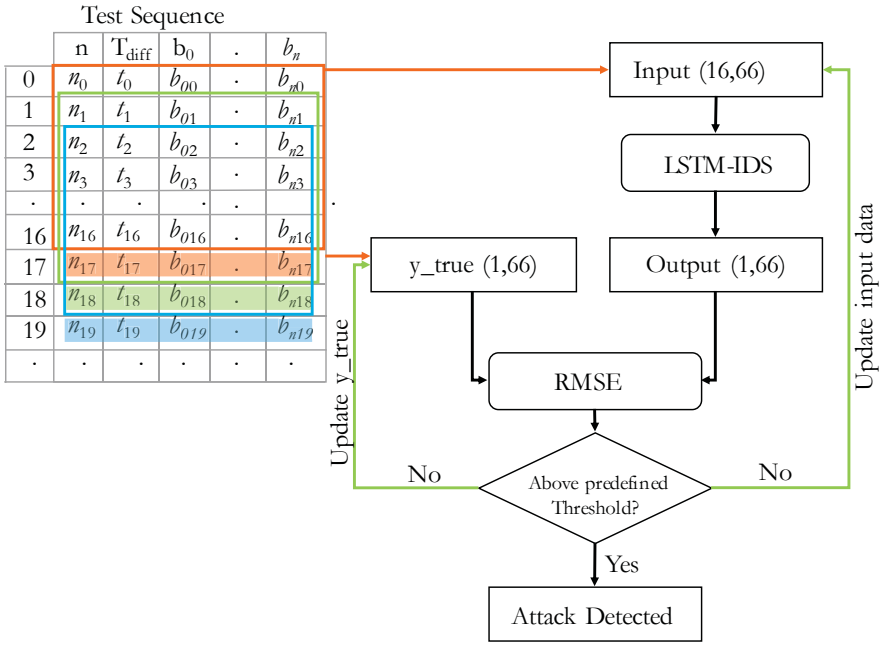


Figure 4.4: Anomaly Detection Process

The 1 second data is first preprocessed here again in the case of live intrusion detection to convert the raw packet data to suitable input variables for the neural network. But, for this detection algorithm we implemented most of the experiments in an offline mode. Hence, we have preprocessed the sequence beforehand. The input to the neural network has a shape of (16, 66), the first variable shows the number of steps we used to predict the successive values

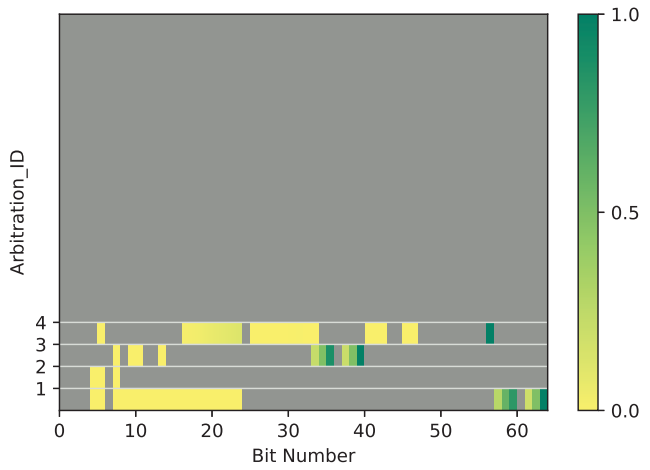
and the second value shows the number of features we created from the data. The trained model then outputs a numpy array with a shape of (1,66) which are the prediction results for all the input variables. Afterwards, we calculate the root mean square error (RMSE), Equation 4.7, of the predicted results and the true value, y_true, true value is the actual value picked from the test data. We selected Root Mean Square Error (RMSE) so as to penalize higher errors because mean square error gives a relatively higher weight to large errors than errors with smaller deviation from the actual value.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y\_true_i - Output_i)^2} \tag{4.7}$$

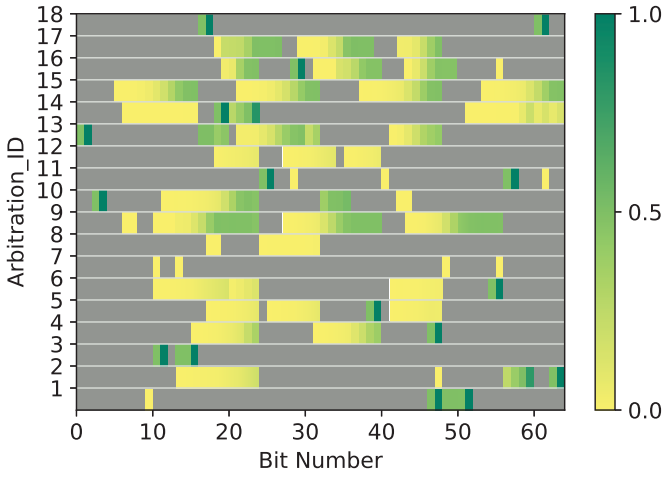
During testing, we have selected a predefined threshold value, depending on this value if the computed RMSE is above the predefined threshold value for each ID, an anomaly flag is generated implying the driver to take appropriate actions. However if the evaluated RMSE is with in a normal range, we increment the pointers of each input variable by one to read the next sequence of data. This process starts when the engine of the testing car is started and continues up until the engine of the vehicle is stopped.

4.2.3 Training and Test Set Description

To train our neural network we collected data from two vehicles, car A and car B. From car A, we collected packets for more than 130 hours with a total of 300 million packets. This car has 42 arbitration IDs with a total average of 680 messages in a second and out of these messages about 400 packets of them are associated with four arbitration IDs, however the rest of the messages are distributed among the other 38 arbitration IDs. For a practical intrusion detection mechanism, an anomaly should be detected way before it causes damage in the targeted host. So, we implemented our anomaly detector in a way that can look up data for a range of 1 second to predict a single future packet. But, we could not train the rest of the arbitration except the first four because we couldn't collect enough data that will be used as an input for prediction and due to the fact that LSTM performs better when the look back has a larger value [27] [8]. The neural network is trained to predict the bit flip rate in a range of 75ms seconds for car A. Figure 4.5a shows the bit flip rate value of all experimented arbitration IDs in car A.



(a) Car A



(b) Car B

Figure 4.5: bit-flip rates of both cars. The grey color shows constant bits that don't flip throughout the whole training. The color bar shows the bit flip rate values

To further check the feasibility of our system, we also collected small data from a different car, car B. Car B has comparatively higher number of messages, 2575 packets per second, than car A. For this car, we trained 18 LSTM networks out of 68 arbitration IDs for the same reason we explained above. We drove this particular car for 30 minutes and collected 3.8 million packets. Figure 4.5b shows the bit flip rate value of all experimented arbitration IDs in car B. We trained both of the vehicles with the 70%, 15%, 15% principle of training data, validation data and testing data sizes. And the testing data is further split into two, the first half for simulating the listed attacks and the

other half is left intact.

4.3 Intrusion Detection Through Analysis of ID Sequences

This section describes a different approach of detecting intrusions through ID sequence prediction. In this method the LSTM network is trained to predict a subsequent arbitration ID by looking at the last 20 arbitration IDs. Even though this method can only identify specific types of anomalies, it leaves no arbitration IDs behind.

4.3.1 Input Data Preprocessing

We used the same data like in the first approach. This method is only implemented for car A as we have collected more data set from this car. Car A has 42 arbitration IDs that contain data packets along with them.

The input to the LSTM network is only a sequence of IDs. Like all types of neural networks LSTM also accepts only numeric tensors. To convert the sequence of the IDs to a numeric tensor we vectorized each input ID. First each arbitration ID is tokenized to a numeric value. Next this sequence of numbers are later one hot encoded before we fed it to the network. From here what the network has to do is give us a sigmoid probability for each of these values in the tensor. Figure 4.6 shows the processes of converting the string sequences to number sequences.

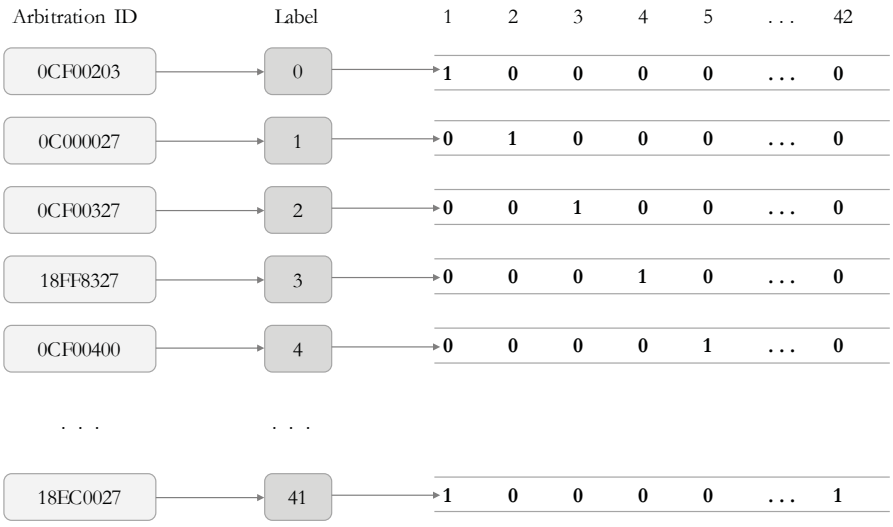


Figure 4.6: Data Encoding process

4.3.2 IDS Network Architecture

The input to the neural network is the sequence of IDs extracted from CAN packets. The network architecture of this IDS is also similar with the architecture used by the first approach other than the input tensors and similar attributes that we had to change to fit to this method's training data. The architecture consists of three dense layers and two LSTM layers (Figure 4.7).

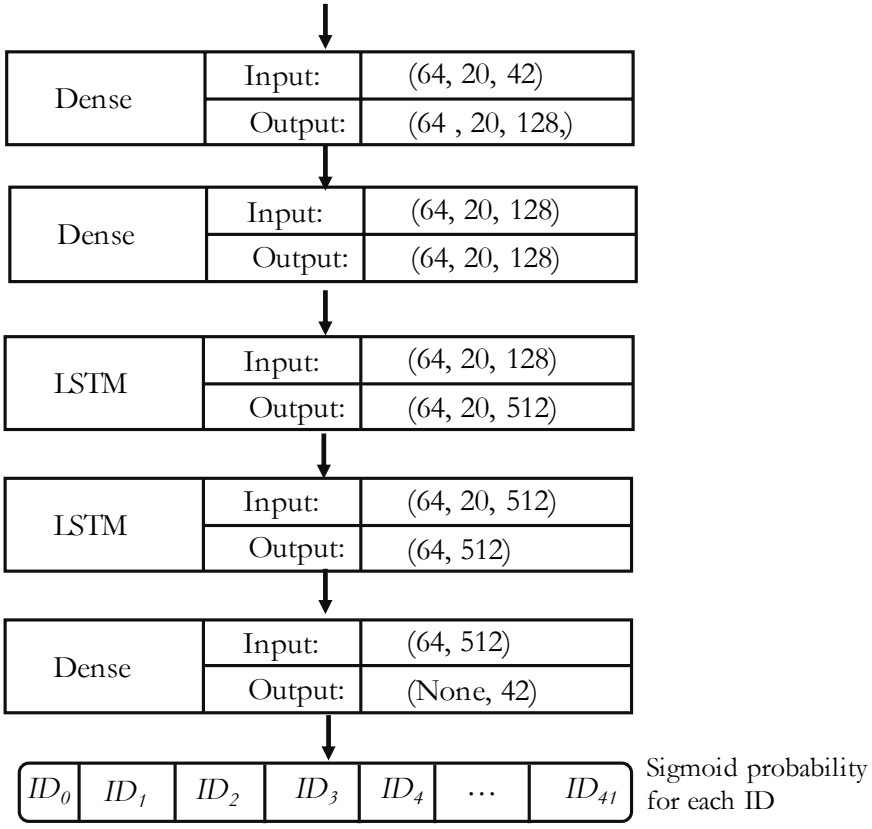


Figure 4.7: ID prediction network architecture

First the input to the network is changed to a suitable format as explained in the last section. Then a tensor of high dimensional vector is fed to the two dense layers each with 128 units and *tanh* activation function. Later on there is a dropout of 0.2 which we used to fight over-fitting during training. The output from this layer then passes to two LSTM layers with 512 units each. This two LSTM layers have *tanh* activation function and 0.2 drop out value. The final layer in this architecture is a dense layer with 42 units that provides us with sigmoid probability values of each arbitration ID. Unlike the first method which we needed to train a different network architecture for each arbitration ID, this method only needs a single network training.

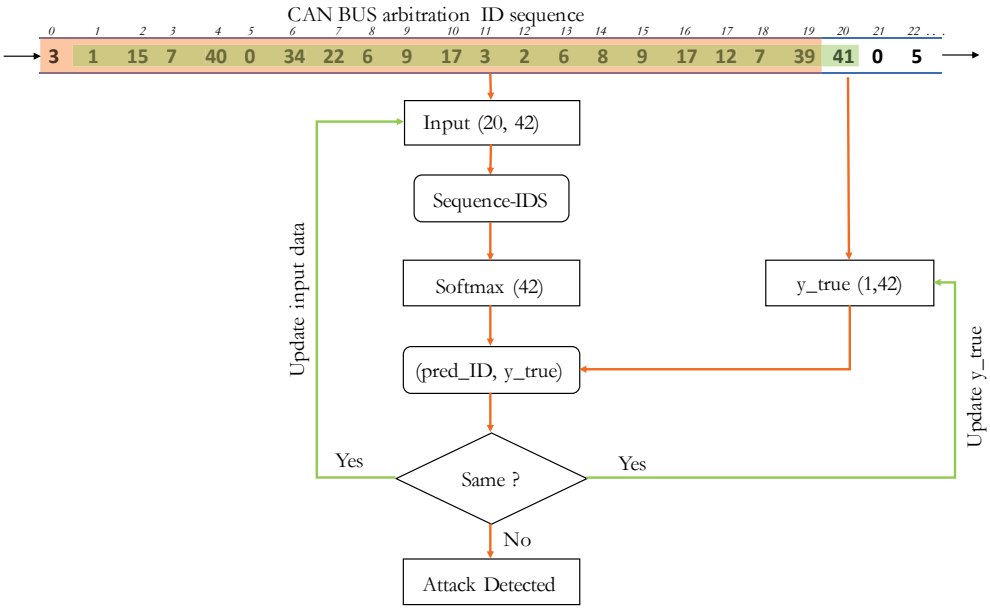


Figure 4.8: ID Sequence Prediction IDS process

The ID sequence intrusion detection method process is shown in Figure 4.8. After we trained the network, each time a message appears in the CAN bus, we collect the first 20 IDs of these messages. Using these 20 messages as input to the trained network, we get a softmax probability to the prediction of the next arbitration ID. The one given with the highest probability will be the predicted arbitration ID. Next, we compare the predicted arbitration ID with the one that has appeared after 20^{th} arbitration ID. If the predicted and true arbitration are ID are not the same an anomaly signal is flagged. But, if both of these values are the same we update the input and the true values in the next step. The input value will be a tensor that grabs 20 arbitration IDs again but this time the start pointer is updated by one to point to the ID next to the first one. And the last pointer will also be incremented by one to incorporate the last predicted arbitration ID. Using these 20 arbitration IDs we again go through the same process to monitor for intrusions. This process also starts from when engine of the car is started and continues till the car's engine is stopped.

4.3.3 Training and Test Set Description

The training data is extracted from the sequence of data we collected from car A. We used the 36 million packets that are collected in the first 100 hours. Out of this data we used 70% for training, 15% for validation and the last 15% is used as a testing data. We used 2-fold cross validation to further validate our proposed network architecture. Figure 4.9 shows the scattered sequence of the first 1000 IDs, each color represents a single arbitration ID of the 42

IDs. As we can see it in the figure some of the IDs appear periodically and some appear randomly but there seems to be some pattern in the sequence of the IDs. Hence, the target of this method would be to check if the LSTM would be able to learn this pattern of sequences. When the engine of the car starts some of the the arbitration IDs (e.g. 1, 2, 3, and 4) start to appear a bit late by around 0.2 seconds than the others. This is not considered in the training because this will not have a significant effect in a data size of 36 million sequences.

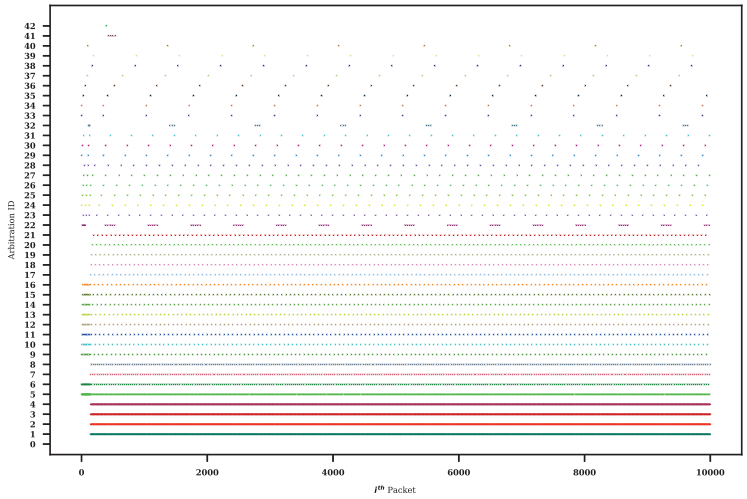


Figure 4.9: Scattered plot of the first 10000 packet sequences of all the 42 arbitration IDs.

In this method, insertion and drop attacks are simulated. When intruders attack a vehicle there will be some deflection from the normal sequence of the arbitration IDs as most attacks either remove a packet or add a new packet to the bus. Hence, what this method does is, it checks if any ID has appeared in the CAN bus out of a sequence. When a new attack packet appears in the CAN bus, this IDS method will first grab the last 20 IDs sequences to predict the already appeared attack packet's ID. Since this was an attack packet the IDS will predict a different arbitration ID value than this one. If the predicted one is different from the already appeared packet's ID, this IDS creates an alert message about this attack or further checks the log loss value evaluated for four consecutive predictions.

Predicting for ID can give us more false positive values due to the randomness of the ID sequence. To solve this issue, we used log loss as our anomaly signal [20]. We selected log loss because log loss penalizes higher errors than low errors. After we get the softmax probability for each arbitration ID we calculated the log loss of the predicted ID and the true ID. Let the true labels

for our predicted arbitration ID be encoded as 1-of-K binary indicator matrix Y, i.e.,$Y_{i,k} = 1$ if sample i has label k taken from the set of K arbitration ID labels. Let P be a matrix probability estimates, with $p_{i,k} = Pr(t_{i,k} = 1)$. Then the log loss of the whole set is calculated by using Equation, 4.8.

$$L_{\log}(Y, P) = -\log Pr(Y|P) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} (y_{i,k} \log P_{i,k}) \tag{4.8}$$

Using this value as a signal we further improved the detection accuracy. And unlike the first method which provides us an anomaly signal value in every 20 ID sequences, this one gives an anomaly signal in every 100 ID sequences. The collective anomaly signal is generated by taking the average of 5 consecutive log loss values. This value is then compared with a predefined threshold to see if its above it or not.

4.4 Evaluation Metrics

We used recall, precision, and F_1 score to show the performance results of our proposed method. These metrics are commonly used in binary classification problems. We also used these results to compare our proposed method with the conventional LSTM-IDS method. Recall shows how good the trained network is at identifying anomalies from a collection of our test data or the ability to find all the relevant cases, anomalies in our case. We used Equation 4.9 to calculate the recall where TP stands true positive to indicate the anomalies identified as relevant and FN is for false negative to show the anomalies that were identified as benign but anomaly packets.

$$Recall = \frac{TP}{TP + FN} \tag{4.9}$$

And precision shows how much of the identified anomalies were actually anomalies or how much of the identified anomalies where correctly labeled. we used Equation 4.10 to calculate the precision. In the equation, false positive (FP) denotes the number of normal records that are identified as an anomaly.

$$Precision = \frac{TP}{TP + FP} \tag{4.10}$$

Additionally, we used F_1 score to compare the proposed and conventional classification methods. F_1 score shows a measure of a test's accuracy. It considers both precision and recall of the test to compute the score. Equation 4.11 shows the formula used to evaluate the F_1 score of our experiment.

$$F_1 score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.11}$$

5 Experiments

5.1 Experiment Environments

The data we collected from both cars are captured by manually connecting a Raspberry Pi with PiCAN 3 device to each car as shown in Figure 5.1. PiCAN 3 board provides CAN-Bus capability for the Raspberry Pi. It uses the Microchip MCP2515 CAN controller with MCP2551 CAN transceiver.

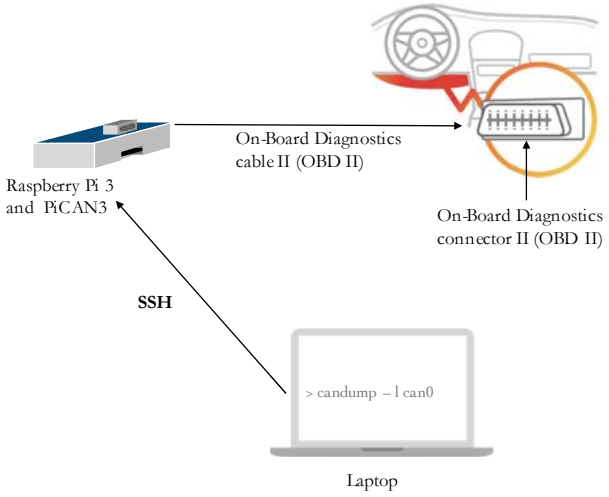


Figure 5.1: Vehicles CAN data Collection

For Car A total of 130 hours of driving data has been gathered to a sum of more than 300 million packets. Out of all these packets, 180 million of it falls in the category of the four arbitration IDs which we trained for our model. In addition, we used the same device to further prove our model in a second car, car B. For car B, we drove it around for only 30 minutes which counts to a total of 3.8 million packets.

5.2 Experimental Results and Discussion

5.2.1 Intrusion Detection through analysis of data packet sequences

For each arbitration ID, we used different threshold values so as to get the best performance and depending on the property of the corresponding arbitration

IDs it gets easier for the method to identify the anomalies for the IDs with a larger number of counter bits (0C000027) than the once which have no counter bits like in arbitration ID_0CF00203, Table 5.1, which got no counter bit at all, Figure 4.2a.

Table 5.1 and Table 5.2 show the performance results of our method for each arbitration ID. In the tables, each row contains performance values in an order of precision and recall for all the tested arbitration IDs.

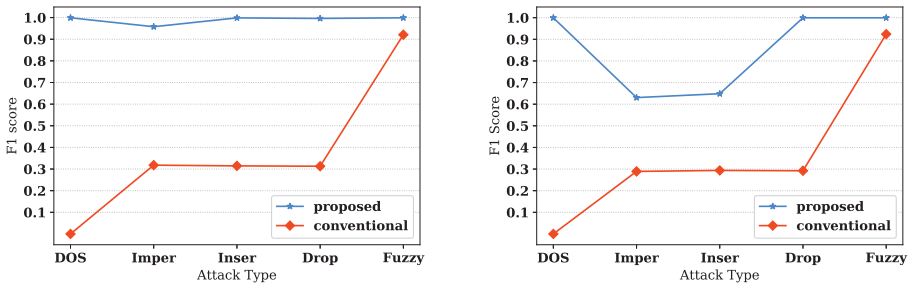
Table 5.1: car A performance results for all attack types with an order of precision and recall in each row

Arbitration ID	Metrics	Impersonation	Drop	Insertion	DoS	Fuzzy
0C000027	Precision	0.997	0.997	0.997	0.998	0.997
	Recall	0.921	0.995	1.0	1.0	1.0
0CF00203	Precision	0.997	0.999	0.999	0.9992	0.999
	Recall	0.461	1.0	0.480	1.0	1.0
0CF00327	Precision	0.997	0.997	0.997	0.998	0.998
	Recall	0.991	1.0	1.0	1.0	1.0
18FF8327	Precision	0.995	0.990	0.992	0.991	0.996
	Recall	0.760	0.406	0.4834	0.439	1.0

Table 5.2: car B performance results for all attack types with an order of precision and recall in each row

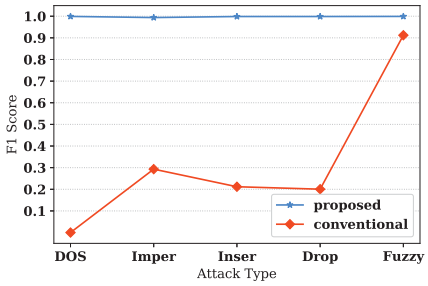
Arb ID.	Metrics	Impersonation	Drop	Insertion	DoS	Fuzzy
114	Precision	0.993	0.993	0.993	0.994	0.993
	Recall	0.767	1.0	1.0	1.0	1.0
116	Precision	0.931	0.926	0.922	0.939	0.993
	Recall	0.910	0.842	0.805	0.786	1.0
119	Precision	1.0	1.0	1.0	1.0	1.0
	Recall	0.817	1.0	1.0	1.0	1.0
120	Precision	0.963	0.944	0.879	0.977	0.970
	Recall	0.792	0.515	0.223	0.446	1.0
122	Precision	0.931	0.800	0.8975	0.818	0.943
	Recall	0.818	0.242	0.530	0.198	1.0
124	Precision	0.952	0.913	0.928	0.897	0.957
	Recall	0.901	0.477	0.591	0.283	1.0
130	Precision	0.956	0.429	0.945	0.333	0.971
	Recall	0.667	0.023	0.523	0.015	1.0
131	Precision	0.918	0.545	0.913	0.583	0.929
	Recall	0.847	0.092	0.806	0.107	1.0
180	Precision	0.905	0.906	0.806	0.571	0.917
	Recall	0.857	0.865	0.376	0.120	1.0
18D	Precision	0.934	0.929	0.918	0.941	0.937
	Recall	0.948	0.873	0.754	0.777	1.0
18E	Precision	1.0	1.0	1.0	1.0	1.0
	Recall	0.777	0.738	1.0	1.0	1.0
18F	Precision	0.922	0.654	0.932	0.820	0.935
	Recall	0.8233	0.131	0.9539	0.209	1.0
1AF	Precision	0.897	0.895	0.870	0.923	0.910
	Recall	0.856	0.841	0.659	0.822	1.0
1B8	Precision	0.895	0.870	0.788	0.907	0.904
	Recall	0.901	0.712	0.394	0.712	1.0
1C0	Precision	0.636	0.867	0.429	0.930	0.890
	Recall	0.216	0.804	0.093	0.848	1.0
1C1	Precision	0.869	0.889	0.611	0.920	0.905
	Recall	0.699	0.842	0.165	0.830	1.0
1D0	Precision	1.0	1.0	1.0	1.0	1.0
	Recall	0.712	1.0	1.0	1.0	1.0
1D1	Precision	0.949	0.900	0.861	0.940	0.964
	Recall	0.705	0.341	0.235	0.406	1.0

Car A's comparison graphs in the Figures 5.2a - 5.2d shows how much using our method outperforms the conventional method, which only considers the raw packet bits. Using bit flip rate as an input helps us to smooth out the various transitions the neural network has to learn if we used a raw bit as an input. Moreover, Our LSTM's input includes average time difference and number of packets to help us get improved detection accuracy.

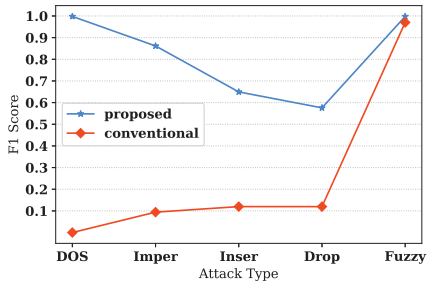


(a) ID_0C000027

(b) ID_0CF00203



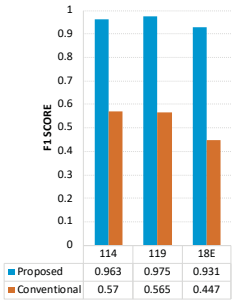
(c) ID_0CF00327



(d) ID_18FF8327

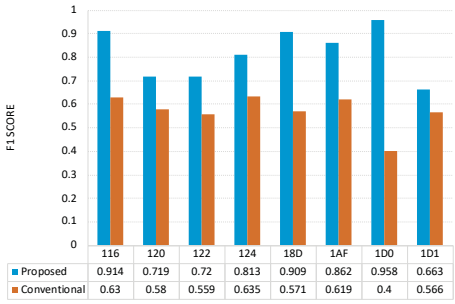
Figure 5.2: Car A, comparison graphs between proposed and conventional method for all attack types

In the comparison figures, the detection performance of the arbitration IDs with a higher number of counter bits is more accurate than the one with no counter bit. Figure 5.3 shows car B's average F_1 score of all attack types except DoS attack for both conventional and proposed methods. The conventional method didn't detect any of the DoS attacks, Figure 5.2, because their method doesn't consider if messages are arriving at an expected time range. DoS attack is simulated by delaying packets to a range of 0.05ms. Delaying these messages doesn't make any difference in the conventional method but the number of messages and average arrival time variables of the proposed method are affected when messages are delayed. Proposed method can detect with a precision and recall values as shown in Table 5.2. The conventional method has no way of telling if a message is delayed due to different attacks, DoS attack in our case, because the conventional method only considers the packet bits.



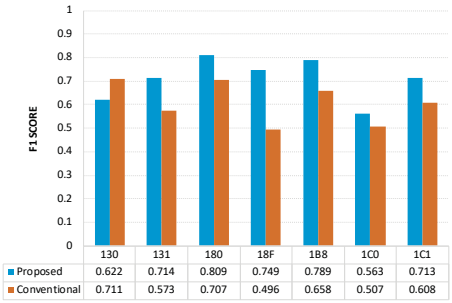
	114	119	18E
Proposed	0.963	0.975	0.931
Conventional	0.57	0.565	0.447

(a) 4bit(2+2)counter IDs



	116	120	122	124	18D	1AF	1D0	1D1
Proposed	0.914	0.719	0.72	0.813	0.909	0.862	0.958	0.663
Conventional	0.63	0.58	0.559	0.635	0.571	0.619	0.4	0.566

(b) 2 bit counter IDs



	130	131	180	18F	1B8	1C0	1C1
Proposed	0.622	0.714	0.809	0.749	0.789	0.563	0.713
Conventional	0.711	0.573	0.707	0.496	0.658	0.507	0.608

(c) No counter bit IDs

Figure 5.3: Car B, F1 score Comparison results between conventional method and proposed method

Limitation

This method highly depends on the frequency of the arbitration IDs. Since, the drivers should be notified about an attack in as short time as possible, it becomes unrealistic to make predictions for non-frequent arbitration IDs. This method considers data of about 1 second to identify anomalies. For those arbitration IDs which only appear in an average arrival time of more than 1 second, we can't get any data from the CAN bus to make our intrusion detection. Hence, this method only identifies targeted attacks of limited arbitration IDs. The other limitation with this implementation also falls on the execution time of the trained network. The trained network takes about 0.02 ms to execute. During this time some packets will appear in the CAN bus that won't be considered due to the delay of the system. If the attackers can learn this delay, it will be possible to schedule their attacks during this time.

5.2.2 Intrusion Detection through analysis of ID sequences

For this method we proposed two ways of identifying anomalies. The first one is to hard code an ID prediction neural network. The purpose of this network would be to give us a softmax probability of all the classes, all the arbitration IDs. And the class with highest softmax probability value would

be the subsequent ID prediction of the network. Unlike the first method which takes data of the whole 1 second to make a prediction, this approach tests for anomalies by only considering 20 sequence of arbitration IDs for a CAN bus with 680 messages in a second.

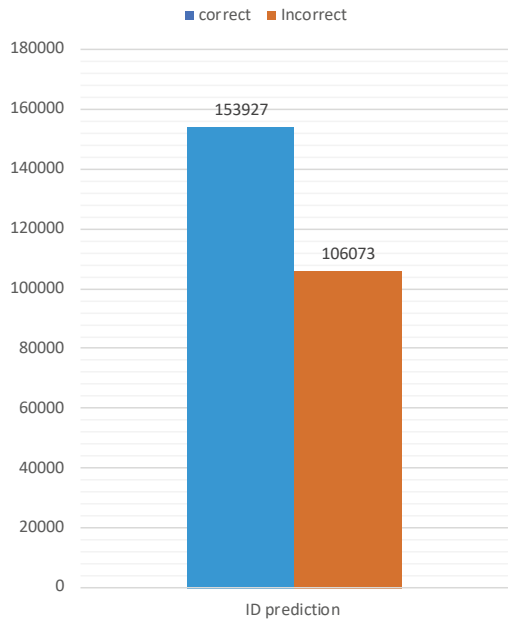
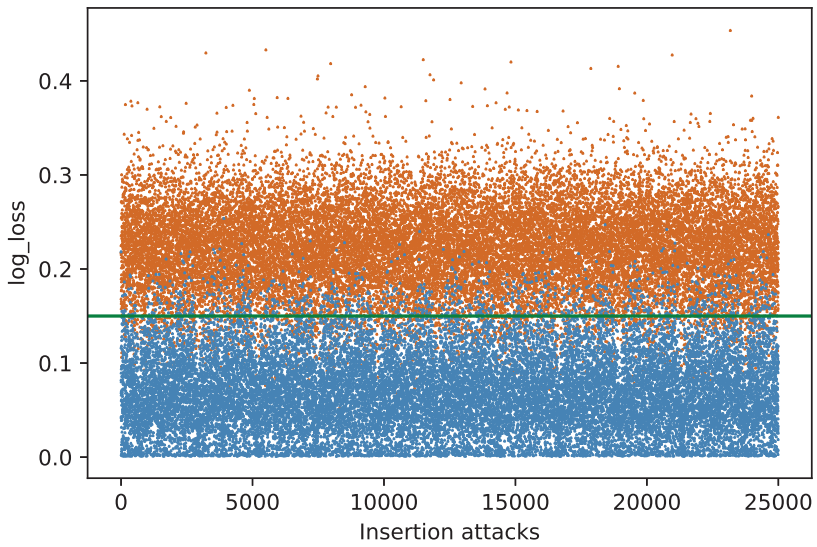
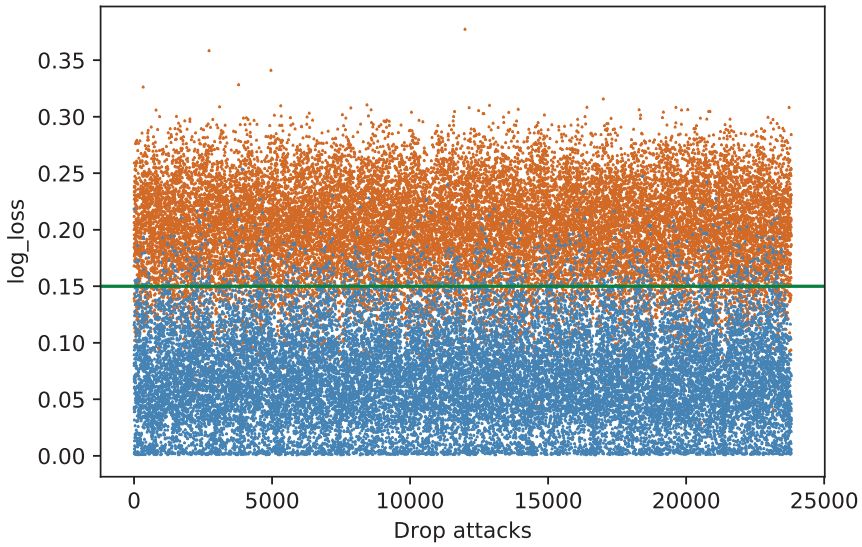


Figure 5.4: Results of ID prediction, predicting a single subsequent arbitration ID for every 20 ID sequences

From here we take the true value from the CAN bus and compare both of these values, results are shown in Figure 5.4. If both of this values fall in the same category of arbitration IDs then we take it as the CAN bus is sane but if the both of these values are not equal, an anomaly signal is generated. As we can see it from the results, out of 160 thousand sequence of size 21 it only detected around 60% of the total. This result might help in identifying some attacks but deploying it in a real vehicle would give us a lot of false positives.



(a) Insertion Attack



(b) Drop Attack

Figure 5.5: scattered results of Insertion and Drop attacks for the first 8000 mix of attack and benign sequences

Therefore we devised a second approach using log loss anomaly signal. Instead of us using ID prediction, we used log loss. We calculate log loss value of the softmax output and the true value of five consecutive predictions for both the anomalous sequence and sane sequence. By making the softmax a bit flexible and identifying a single anomaly in a every 5 predictions it gives

us better results. Figure 5.5 shows the log loss scattered plot for both the insertion and benign results respectively. Figure 5.6 also shows how relaxing the detection methodology can help us in improving the detection capability. In the previous approach a single prediction was being made in every 20 arbitration IDs. The previous approach mainly focuses on taking the maximum softmax probability of a class to make a prediction. But, this method calculates the average log loss value of the softmax values and true values to test if the evaluated value is above a predefined value. We make a single decision threshold for every 100 frames, unlike the first one which only considers the first 20 frames.

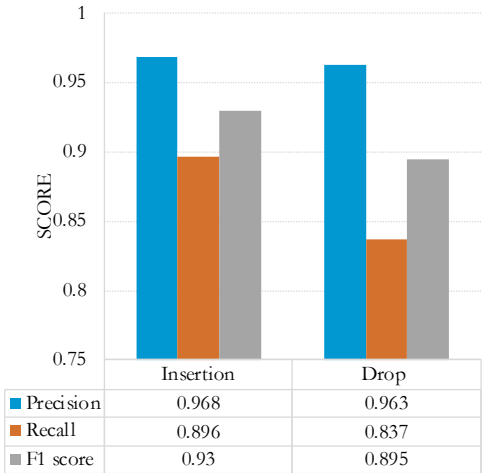


Figure 5.6: Detection scores for both insertion and drop attacks

Limitations

Unlike the first method which can't be implemented for all the arbitration IDs, this method can be used to identify any intrusions in the whole network by not depending on weather arbitration IDs are frequent or not. Even though this has some advantages over the first one, it can only detect specific types of attacks. Attacks that can be identified by this method are those that can deviate the flow of messages disturbing the normal sequence of arbitration IDs. If attackers can compromise a single ECU, they might over pass this method by sending normal sequence of arbitration IDs with spoiled data portion of the packets. And similar to the first one, it also leaves some messages behind when enough messages are collected for processing. During processing for intrusion detection this method also has some delays. During this delay, 1 or 2 messages can appear in the CAN bus that the method will skip.

5.3 Execution Time

Implementation of LSTM-IDS in the in-vehicle network would require a fast processor that is capable of collecting bit flip rates in n time window and make a prediction in as short time as possible. But, there is always a significant delay from the time when the packets appear in the CAN bus to the time when these packets are collected for analysis. Figure 5.7 shows the execution time of our detection process. The values in the box plot show the execution time of the models simulated in Ubuntu 18.04 OS, Intel Xeon E5-1620 CPU and GM200 (GeForce GTX TITAN X) CUDA 6144 cores GPU that has a clock speed 1000MHz and a RAM size of 16GB.The first box plot in the table We implemented the LSTM-IDS in this computer by using a SocketCAN [21] that contains CAN drivers and a networking stack for Linux. We played the dumped file in the terminal with vcan0 interface and created a program that collects the packets through SocketCAN API and does the prediction. The box plot shows the execution time it takes for the trained model and make predictions out of collected data. The three boxes show the execution time for the conventional, Proposed CAN frame sequence analysis and ID sequence analysis models in the ordered they appeared from left to right. As we can see it in the figure, the conventional method takes longer time than the proposed models. The conventional method looks back 98 messages to predict a single packet, however the proposed method only looks back 15 or 17 messages to predict an anomaly. Since we need one to test for intrusions every second, the conventional method takes the raw packet bits that sums to a total of 98 messages in a second. But the proposed method extracts only 1 feature raw for every 6 or 7 rows in the proposed method and this makes the proposed method a little bit faster than the proposed method. There seems to be some relation ship between the look back value and the execution time of the trained network. The trained network is fast for the proposed frame sequence predictor than the ID sequence predictor and conventional frame sequence predictor because the proposed method has a look back of 17 or less while the other methods have a look back of 20 and 98.

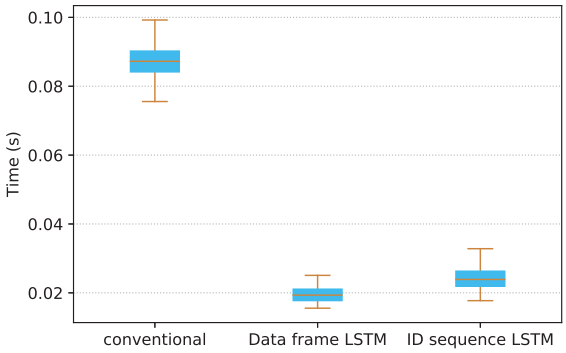


Figure 5.7: Execution Time for both conventional and proposed methods

When our system enters the detection phase, one or two messages appear in the CAN bus that will be skipped unless we use a buffering mechanism. For these messages to not be skipped while we are testing a time window we implemented a buffering mechanism that can keep these messages in a buffer until the prediction is done. Even though this method leaves no packets behind but it also destroys our goal of identifying intrusions in a short time because if packets start to pile up in the buffer it would start to take longer time to identify the anomalies. Therefore, the best option would be to implement an IDS that collects messages as soon as the decision for the state of the messages is checked for benignness. Hence, Skipping one or two messages until the collected sequence passe through the LSTM-IDS is the best option we have so far.

6 Future Work

Through the over all evaluation, we proved that further including data from timing and count of packet in a time window has a great advantage in detecting anomalies. Further more, we also discussed on how LSTM can be used to learn the sequence of arbitration IDs. Although the proposed methods can identify anomalies at an acceptable accuracy, there seems to be a poor performance for those arbitration IDs which contain no counter bits in the first method. In this paper, we had to implement LSTM network for each arbitration ID due to the different DLC lengths associated with each arbitration ID. But for the case of our first car, CAR A, it uses CAN 2.0B and all the messages have the same DLC size. Therefore for our future work, we would implement a single LSTM network as in the ID sequence analyzer or any other related neural network that takes all the arbitration ID's packets as an input to predict a forthcoming packet and its respective ID. This would help us to implement an overall IDS that does not leave out any packets behind. Moreover, the neural network was trained in a principle of 70% training data, 15% validation data and 15% testing data, but for the feature we would also like to use a different principle, k-fold cross validation, to validate the approach in a different angel.

Deployment of this research in production environment can be a bit expensive due to the high processing GPUs we need to help us test through all the packets that appear in the CAN bus. Even with the high speed GPU we have in our PC, the the IDS systems skip one or two packets when it is implemented in a virtual online environment. Thus, as a future work we also need to improve the execution speed of our system so as no packet would be skipped with out testing. Additionally, by taking the advantage of both the ID sequence prediction and packet sequence anomaly detection a single IDS system can be created that passes through both the approaches in order to identify for anomalies.

7 Conclusion

In this paper, we presented on how anomalies can be detected using LSTM for in-vehicle networks. The first detection method works for targeted attacks meant for a specific ECU. LSTM networks are trained to predict forthcoming payloads and related attributes by looking at information that has already appeared in the CAN bus at some instant in time. The predicted values are compared with actual values, that are either sent by an intruder or the benign ones and depending on how close our prediction is with the actual payload, we managed to effectively identify anomalies in an improved accuracy than the conventional LSTM-IDS. To train the network we reverse engineered the messages to identify the counter bits associated with each packet which improved the overall detection method up to 98%. The second strategy focuses on the analysis of ID sequences to identify anomalies. It takes as input the sequence of IDs in one 1 second to identify if there has been anomalies in this time window.

Bibliography

[1] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[2] P.W.D. Charles. Project title. https://github.com/charlespwd/project-title, 2013.

[3] S. Corrigan. *Introduction to the Controller Area Network (CAN).* SLOA101B–August. Texas Intruments Incorporated, 2002–Revised May 2016.

[4] S. Corrigan. Introduction to the controller area network (can), 2002–Revised May 2016.

[5] Felix A Gers, Jurgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Not provided*, 1999.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[7] Gang Han, Haibo Zeng, Yaping Li, and Wenhua Dou. Safe: Security-aware flexray scheduling engine. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 8. European Design and Automation Association, 2014.

[8] Weicong Kong, Zhao Yang Dong, Youwei Jia, David J Hill, Yan Xu, and Yuan Zhang. Short-term residential load forecasting based on lstm recurrent neural network. *IEEE Transactions on Smart Grid*, 10(1):841–851, 2017.

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

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

[11] Takuya Kuwahara, Yukino Baba, Hisashi Kashima, Takeshi Kishikawa, Junichi Tsurumi, Tomoyuki Haga, Yoshihiro Ujiie, Takamitsu Sasaki, and Hideki Matsushima. Supervised and unsupervised intrusion detection based on can message frequencies for in-vehicle network. *Journal of Information Processing*, 26:306–313, 2018.

[12] Keen Security Lab. Car Hacking Research. https://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/, 2016. [Online; accessed 26-may-2019].

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

[14] Mirco Marchetti and Dario Stabili. Read: Reverse engineering of automotive data frames. *IEEE Transactions on Information Forensics and Security*, 14(4):1083–1097, 2018.

[15] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015:91, 2015.

[16] Michael R Moore, Robert A Bridges, Frank L Combs, Michael S Starr, and Stacy J Prowell. Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: a data-driven approach to in-vehicle intrusion detection. In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, page 11. ACM, 2017.

[17] Philipp Mundhenk, Sebastian Steinhorst, Martin Lukasiewycz, Suhaib A Fahmy, and Samarjit Chakraborty. Lightweight authentication for secure automotive networks. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 285–288. EDA Consortium, 2015.

[18] Anvardh Nanduri and Lance Sherry. Anomaly detection in aircraft data using recurrent neural networks (rnn). In *2016 Integrated Communications Navigation and Surveillance (ICNS)*, pages 5C2–1. IEEE, 2016.

[19] Christopher Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs, 2015. [Online; accessed 26-may-2019].

[20] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[21] Volkswagen Research. Controller area network protocol family (aka socketcan). https://www.kernel.org/doc/Documentation/networking/can.txt.

[22] D-7000 Stugart 1 Robert Bosch GmbH, Postach 50. *Bosch CAN Specification*. Chunk Powers, 1991.

[23] Florian Sagstetter, Martin Lukasiewycz, Sebastian Steinhorst, Marko Wolf, Alexandre Bouard, William R Harris, Somesh Jha, Thomas Peyrin, Axel Poschmann, and Samarjit Chakraborty. Security challenges in automotive hardware/software architecture design. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 458–463. EDA Consortium, 2013.

[24] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*, pages 63–68. IEEE, 2016.

[25] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. Frequency-based anomaly detection for the automotive can bus. In *2015 World Congress on Industrial Control Systems Security (WCICSS)*, pages 45–49. IEEE, 2015.

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

[27] Qianlong Wang, Yifan Guo, Lixing Yu, and Pan Li. Earthquake prediction based on spatio-temporal data mining: an lstm network approach. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[28] Yujing Wu, Yeon-Jin Kim, Zheyan Piao, Jin-Gyun Chung, and Yong-En Kim. Security protocol for controller area network using ecandc compression algorithm. In *2016 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–4. IEEE, 2016.