Detecting Attacks on the CAN Protocol With Machine Learning

Valliappa Chockalingam, Ian Larson, Daniel Lin, Spencer Nofzinger
University of Michigan
{*valli, anzallos, danielin, snofz*}@*umich.edu*

Abstract

Modern automobiles have seen a dramatic increase in the use of electronic systems over the last decade as they have gotten “smarter” and more technologically advanced. In doing so, a trust has been placed on increasingly complex embedded systems with the safety of both passengers and pedestrians at risk. In particular, one of the most common methods of connecting the different Electronic Control Units (ECUs) in a vehicle, the Controller Area Network (CAN) bus, has been repeatedly shown to be vulnerable to various intrusion based attacks. While securing communication protocols has received large amounts of attention within the security community, the unique constraints and complexities of embedded systems and the CAN protocol, which dates back to the 1980s, make many tried defenses impractical or impossible. Given the rise of the use of Machine Learning (ML) techniques in anomaly detection, in this paper we investigate the use of different ML algorithms in detecting anomalies in CAN packets or packet sequences, compare the algorithms with a discussion of some common attacks on the CAN protocol, and finally discuss some of the implementation details of such Anomaly Detectors.

1 Introduction

For most of its existence, the automobile has been a largely mechanical, combustion-powered system. While electrical systems have existed in cars for decades, it is only recently, i.e., over the last few decades, that automobiles have seen a rapid rise in the number and complexity of electronic components and can be viewed as large networks of complex computers. Additionally, this trend of the increasing electrification of vehicles is unlikely to be reversed any time soon as the automotive industry shifts towards autonomous behavior, electronic controls, and electric power.

A problem arises from the burgeoningly apparent issue that in any autonomous, wire-controlled, or electric-powered vehicle, numerous sensors and controllers must perform properly together to ensure the safety of both passengers and pedestrians. Indeed, as the automotive industry has changed, electronic control units, or ECUs, have been given partial or total control of more and more systems, a sizable number of which are safety-critical.

Historically, the auto industry has made safety a priority requirement of commercial vehicle designs, whether by choice or by government regulation. However, an area that has been neglected is the security of the ECUs that reside within a vehicle. This largely arises from the fact that the first version of the Controller Area Network (CAN) protocol, CAN V1.0, which provides a message-based communication channel for the ECUs to communicate, was designed in 1983. There have since been updates: CAN V2.0 in 1991, and CAN-FD (Flexible Datarate) in 2012. That said, the protocols, at least V1.0 and V2.0, were largely designed at a time in which security was not a major concern. To give an idea of the landscape of CAN V1.0, the Internet was not yet designed, Ethernet was just introduced (in 1980) and terms such as “virus” and “worm” were still rare. Furthermore, improving the security of a widely used protocol also poses problems for backwards compatibility. Thus even the updated versions pose problems for security.

An illustration of the problems in the CAN protocol is evident in recent work where security researchers have taken advantage of the lack of security measures in the CAN protocol and demonstrated that an attacker can easily gain access to particular ECUs, inject CAN packets (i.e., perform an intrusion based attack), and trivially cause harm [6, 3].

Our goal in this paper is to study typical attack patterns and develop a defense that leverages current strides in machine learning to detect intrusion using an analysis of CAN packets, and by extension, the general behavior of the CAN protocol.

We believe this is a particularly important problem to address as manufacturers continue to add more electronics and we enter into the age of self driving cars with vehicle-to-vehicle communication. Unlike an attack on a phone or a computer where the effects are not so severe, an attack on a vehicle is serious as there is a danger of loss of life. Given that there are already attacks that erase all evidence after a crash [6], mass surveillance is an issue being discussed lately, and the fact that vehicles are distributed systems with multiple manufacturers involved all of whom must be trusted, it is quite crucial to be able to detect attacks to design better security measures and build more secure vehicles.

In this paper, we investigate the usefulness of machine learning in detecting attacks on a CAN bus. Classification machine learning algorithms can differentiate data into separate classes, sometimes better than a human can. Therefore, we decided to test the performance of applying machine learning to this problem. There are several different types of methods for classification, and so we want to test several and evaluate which have the best performance.

With the above noted, what follows is a brief roadmap of the paper and our contributions as a result:

- Construct several CAN packet/packet sequence anomaly detectors based on different Machine Learning algorithms.

- Consider common attacks on the CAN protocol and see how well the ML models perform under such scenarios.

- Provide recommendations for how the anomaly detectors can be used and implemented in practice.

The rest of this paper is structured as follows: In section 2, we discuss background information and related work. In section 3, we describe our approach. In section 4, we present our methods for detecting CAN fuzzing. In section 5, we give a summary of results along with some a discussion of the insights gained through the results. In section 6, we discuss future work. Finally, in section 6 we conclude.

2 Background and Related Work

Automobiles on the road today are controlled by a multitude of small microcontrollers, the ECUs, working in tandem. These ECUs are bridged together along CAN lines throughout the car. The ECUs control the majority of modern automobile functionality, from the media player to locking the breaks.

2.1 Attacks on the CAN Protocol

Recent research has shown that, if given access to a CAN bus, it is not very hard to forge packets and execute unwanted vehicle behavior. Koscher et al. used a technique called "Fuzzing", where an attacker sends random packets and hopes for some sort of system response. The goal in this attack scheme is to to discover packet combinations that control different ECUs [6]. They showed that, with a little reverse engineering and fuzzing, it is possible to gain complete control over an automobile's ECUs. They were able to lock the brakes, send arbitrary message to the media player and shut the car lights off, all through CAN packet injection. Even when some sort of cryptographic key was required by an ECU it was trivial to crack because of how small the CAN packets are. To reiterate, CAN was created many decades ago and was not built with the mindset that an attacker might be sharing the network.

There is still a question of how to get access to a CAN bus without direct physical proximity. In 2011, Checkoway et. al demonstrated that various entry attack vectors can be used in quite innovative manners [3]. For example, the authors showed that by using a malicious WMA audio file, access to the CAN bus can be gained. Another example is the use of malicious packets sent to the Telematics unit with a 3G IRC.

2.2 Machine Learning and Intrusion Detection

Recently, Machine Learning (ML) has been gaining increasing attention from the computer science community in general and the security community as well. One interesting use of ML has been in intrusion detection that use ML algorithms that are broadly known as anomaly detection, novelty detection or outlier detection algorithms. These algorithms have been used in applications ranging from detection of DoS attacks [4, 9, 8] to malware detection by Antiviruses [2]. One interesting example of these algorithms is "Comparing Anomaly-Detection Algorithms for Keystroke Dynamics" by Killourhy and Maxion [5]. In this paper, the authors compare different ML techniques such as Neural Networks, Support Vector Machines, k-Means and distance metrics such as the Mahalonobis distance, for anomaly based detection of impostors through keystroke dynamics data collected on typing passwords. As most of these ML techniques are quite general, i.e., "model-free," they have been applied to other areas as well.

2.3 Detection of Attacks in the CAN Protocol

In relation to the CAN protocol, there has not been much work in detection of attacks. Taylor et al. demonstrated in their paper "Frequency-based anomaly detection for the automotive CAN bus," that malicious packet injection of CAN messages can be detected with a reasonable false positive rate [10]. The paper used a one class support vector machine (OCSVM) that learns the distribution of a single class of data with features engineered from the frequency and delay of packets. Noting that frequency based methods seem to do well, the authors also mention that a simple hamming distance between successive packets is not sufficient and that the paper's focus is not on identifying anomalies based on packet content's themselves. Furthermore, the authors note that more research is needed to detect non-periodic malicious packets.

3 Approach

3.1 Security Viewpoint and Threat Model

From a security standpoint, our approach can be described using a threat model. The threat model that we take can be seen through Figure 1 where we illustrate the attack tree, each of the nodes which we take to be "possible." In particular, note that we assume the attacker has already breached the CAN bus, like in [6]. Thus, we are assuming that an attacker already has access to the CAN bus and/or ECU hardware. The next line of defense is detecting when an attacker is performing some form of intrusion detection such as fuzzing the bus or trying reverse engineering. Both of these approaches are taken by an attacker with the aim of learning CAN packets that can execute the attacker's desired commands. The execution of such commands is ultimately what we want to prevent and our defense aims to stop the attacker from reaching this point (the root of the tree). The ML algorithms can be used to identify anomalous behavior that can be witnessed when the attacker is trying to learn CAN packets for executing malicious commands or indeed attempting to execute such malicious packets.

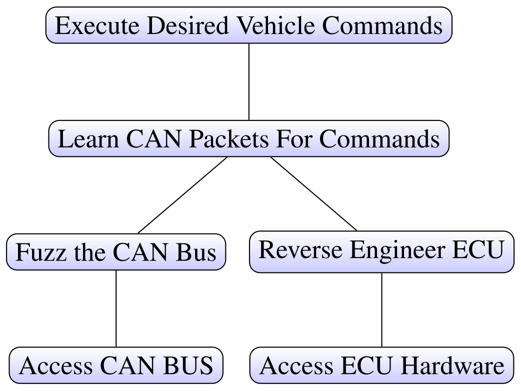

Figure 1: Attack Tree

3.2 Machine Learning Viewpoint

Our work looks at how reliable machine learning is at detecting anomalous behaviour on the CAN bus. We tested this by defining different potential algorithms and testing them against various forms of anomalous behavior. Thus, our approach from a Machine Learning point of view is similar to [5], in that we try various Machine Learning techniques and compare them. We decided to test many diverse kinds of algorithms in order to get an understanding of what kinds of methods would give us the most accurate predictions.

Notwithstanding the need for good accuracy, the simpler ML algorithms can also be helpful through the fact that simpler algorithms can likely more easily be implemented and distributed in the memory and computationally restrictive environments that vehicles posses. On the other hand, more complex approaches like Neural Networks can capture nonlinear interactions and memory based neural networks in particular can use internal memory that can act as a sort state representation and capture long term relations in the inputs that are particularly useful in the CAN protocol as different ECUs often communicate with different frequencies.

3.3 Industry and Operations Viewpoint

We believe that ML techniques are an interesting defense because CAN buses are typically complex, but relatively predictable, systems. While it is expected that designers would define acceptable bounds and conditions for data an individual ECU receives, universally detecting anomalies in any arbitrary message would require either detection and reporting measures for each ECU, or a high-level supervisor that understands the valid conditions and interactions of every message under every circumstance. While both options are theoretically possible, ECU-level action would require time to update ECUs across a supply chain, as well as trust that contractors would meet detection and reporting requirements. A high-level supervisor would in some sense remove the need to trust many different parties, but the amount of work and understanding to manually develop such an option is very high. We believe that a ML supervisor can solve the manpower issue because ML algorithms could be taught during the product development and testing of a vehicle, which should ideally include every realistic condition that could occur in a car.

4 Methodology

4.1 Training

For training the different ML algorithms, we require CAN packet data over a reasonable amount of time and data that is representative of the general nature of the CAN protocol.

With the above goals in mind, we obtained CAN packet data over fifteen minutes from the Michigan Solar Car Team for a total of approximately 150,000 packets.

The data obtained was in a human readable format that is not suited for Machine Learning techniques. This is primarily because the string type features needed to be encoded as integers, meaning data is lost, and, furthermore, the dimensionality of the human readable data was about 100. As discussed in prior work, the curse of dimensionality is an important issue to be wary of.

To reduce the dimensionality and to get the data in a better format, the human readable format was converted to the actual representation of the packets as a UNIX timestamp followed by eight hex digits representing the instruction.

Finally, to avoid overfitting and for observing the performance of models that learn using gradient descent (the Neural Network based models) as they were being trained, a validation set was used with early stopping. More specifically, we used 70% of the training data for training and 30% of the data for validation. The validation error was measured after every gradient update, i.e. after every minibatch consisting of 20 packets or packet sequences in the case of LSTMs. If the error did not reduce in the last 50 iterations, the algorithm was stopped to avoid overfitting.

Now, we discuss the algorithms themselves. The first algorithm we used was a simple linear One Class Support Vector Machine (SVM). OCSVMs have been used in prior work relating to CAN intrusion detection [3] and thus provide a good baseline. Additionaly, they are more interpretable in the sense that it is easier to understand how such models make decisions as opposed to a neural networks which are harder to inspect. The second method we tested was the Self Organizing Map (SOM). Looking at an SOM is useful because it provides the user with a clear visual comparison of two different inputs and we can observe clustering as more input is read. The final model, in some ways the most complex, we looked at was a Long Short-Term neural network (LSTM). LSTMs are very powerful in the sense that they theoretically are Turing Complete and can thus perform very complex classifications. However, more practically, using LSTM allowed us to look at the validity of a packet in context with the sequence of packets that came before it. Also, they allow for the prediction of entire future packet sequences. While this was not done in this work, we leave it as an open question of whether prediction multiple steps ahead can help.

What follows below is a brief introduction to the various methods along with a few notes about each, in particular what hyperparameters were used, if any.

4.1.1 One Class Support Vector Machine

Support Vector Machines (SVMs) are a classification method whereby a decision function is created with the best separating margin. In other words, the best function for separating two classes of data is looked for. One Class Support Vector Machines (OSCVMs) are a particular variant of SMVs where training is done using only one class of data. This is especially suited to anomaly detection as we generally have only data from the non-anomalous class. This is also the case with the CAN packets we collected; they all belong to the non-anomalous class.

SVMs can be used with kernel functions that allow for more complex decision functions. We tried both a linear kernel and a non-linear (Radial Basis Function) kernel. We also varied the hyperparameter ν which controls for what fraction of the training set can be misclassified or, equivalently, the number of support vectors (the number of datapoints on or within the margin generated.) The closer ν is to 1, the longer the training takes and more of the non-anomalous packets are allowed to be misclassified as anomalous packets.

4.1.2 Self Organizing Map

Self-Organizing Maps or Kohonen Maps is a type of artificial neural network designed to learn a mapping of the input space of training examples onto a low-dimensional, discretized map, generally a 2 dimensional square grid, in an unsupervised manner. The main hyperparameter we changed was the size of this grid. In particular, we used 3x3, 5x5 and 9x9 grids.

4.1.3 Long Short Term Memory Neural Network

Long Short Term Memory Neural Networks (commonly abbreviated as LSTMs) are a recurrent type of neural network particularly well suited to time series data where a sequence of data is passed as input and predictions about the future are made. We made use of an LSTM network with one hidden layer. The hyperparameters we were interested in were the number of packets to take as input (the fixed size window used during training) and the number of neurons in the hidden layer. For the former, we considered 25 and 50 packets and for the latter we considered 10, 25 and 50 neurons. Finally, it should be noted that since LSTMs take sequences as input rather than single packets, packet sections were be taken from the training set and the model was trained to reduce the error in its prediction of the next packet.

4.2 Evaluating Anomaly Detection (Testing)

There is an interesting Chicken-and-Egg question that arises with anomaly detection. In particular, anomaly detection is quite a peculiar type of problem as we generally do not know how to judge the performance of an anomaly detector. This is simply because we don't really know what is anomalous; this is the reason we look to constructing models that can somehow infer the similarity within the non-anomalous class and discriminate anomalous from non-anomalous data.

While the statements above may paint a bleak picture, all is not lost. Generally, for testing anomaly detectors we can construct anomalous data based on what kinds of anomalies generally might arise or on how the anomalies might be generated.

With respect to the CAN protocol, we look to two different types of anomalies. The first anomalous behaviour we want to test against is "fuzzing". Fuzzing is a method in which random data is sent into a system or network in order to elicit some sort of response. This a common and easily-identifiable attack against a CAN bus as an attacker can reverse engineer the response garnered from the automobile in question to a packet they sent. Once they've reverse engineered enough valid packets they have a library of functionality that they can use on the bus. It is assumed that fuzzing occurs away from the victim, in a lab where the attacker has access to a car of the same model. In our defense scenario, we propose that, if the car detects fuzzing, an alert will be sent to the manufacturer for further investigation. This defense is a first time detection system operating under the assumption that someone else has not already fuzzed the target car and posted the commands to a database on-line. We think this is a fair assumption given that to our knowledge no wide spread attacker CAN databases exist.

Now, to test for fuzzing, we first need to generate a dataset consisting of fuzzed data.

One way we did this is simply by using training data and validation data, i.e., non-anomalous data, along with a probability p that denotes how many of the hex digits in an instruction are changed randomly. In other words, the method of fuzzing we use is to keep the UNIX timestamp the same and simply change the hex-digits in prportion to the probability p. Note that we chose $p \in \{0.1, 0.2, 0.4, 0.6, 0.8\}$. Also, this hex-swapping method was tested on the algorithms that work on single CAN packets.

Another way we introduced fuzzing was to add Gaussian Noise to the UNIX timestamps of some packets in a packet sequence. Specifically, we used an exponential distribution with scale parameter 100 to generate a large dataset consisting of packets which consists of a fuzzed packet on average every 100 packets. The reason for this choice is that we found that commercial software in fact uses exponentially distributed packet frequency bursts to test for the reliability of CAN buses [1]. The fuzzing of the packets selected by the exponential distribution was done using Gaussian Noise with $\mu = 0$ and $\sigma \in \{0.01, 0.05, 0.1, 0.2, 0.5, 1\}$. The reason for using zero-mean is that it is generally quite a good assumption and furthermore, for timestamps there seems to be no reason to bias the addition of noise so that packets seem to arrive before or after when the non-anomalous packet arrived. The use of different standard deviations was done mainly to measure the sensitivity of the anomaly detection modesl. Now, having this large dataset, we then constructed packet sequences from this dataset just as in training. And finally, measuring the error in the prediction of the next packet is used as the metric for deciding whether a packet sequence is anomalous (the reasoning for this is that the higher the error, the more likely the packet sequence probably hasn't been seen before and is hence anomalous).

The second form of anomalous behaviour we tested against was to introduce small groups of misplaced non-anomalous packets in an otherwise non-anomalous packet sequence, i.e., we essentially created a dataset with jumbled non-anomalous data. In this attack scenario we assume that an attacker has already gained access to valid CAN commands and is able to send packets to manipulate the car behavior. This method as well as the Gaussian Noise addition were only applied on LSTMs as packet sequences are used as inputs.

5 Results and Discussion

Below we present the results for each of the machine learning algorithms we tested along with discussions for what the results suggest in terms of possible implementations and from a CAN protocol behavior perspective.

5.1 One Class Support Vector Machines

Upon training the OCSVMs, we performed fuzzing with the hex-changing method. Since we used three different values for ν, two different kernels and five different probability of hex-changing, there are a lot of results we obtained. However, in summary, we found that Linear Kernels work quite well, while Nonlinear Kernels generally take a long time to optimize. We visualized the results with confusion matrices that represent the true positive/false positive/true negative/false negative data.

The confusion matrices for the best classifier (in terms of the overall false positive rate being the lowest) we obtained, OCSVM with $\nu = 0.01$ and a linear kernel, is shown in Figure 2 where a Spectrum color map is used meaning Purple represent a high concentration or fraction and red represents a low concentration or fraction. Clearly the concentration of the packets are in the boxes where the True Label is identical to the Predicted Label (i.e., the classification is correct.)

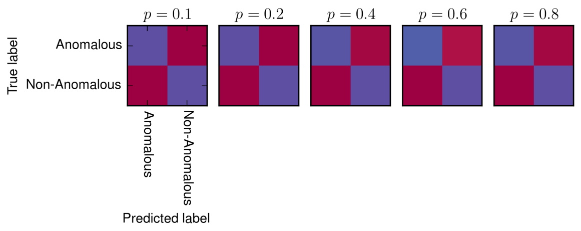

Figure 2: Confusion Matrices for best OCSVM model found

Overall, the above classifier obtained a 7% false positive rate. Thus, OCSVMs seem to be good classifiers for detecting anomalous CAN packets. However, we would like a classifier that can ideally keep being trained and refined. Additionally, a classifier which provides a probabilistic measure can be useful in determining a threshold for when to notify humans about possible intrusions. SVMs generally do not allow for these kind of properties. Also, as more data is used, SVMs generally become prohibitive in time complexity and require large amounts of memory which may certainly be a luxury in embedded systems used in automobiles today.

5.2 Self-Organizing Map

For analyzing the performance of the Self-Organizing Map, we used the Average Euclidean Distance metric which gives an idea of how the weights are distributed along the grid. The results for the tests on non-anomalous packets are given in Figure 3.

Kohonen Map Dimensions	Avg. L2-Distance for Non-Anomalous Training Set	Avg. L2-Distance for Non-Anomalous Validation Set
3 x 3	22.583	131.400
5 x 5	28.344	128.593
9 x 9	31.149	127.411

Figure 3: Avg. L2-Distance Results

As seen in the table, the Average Euclidean Distance for the Validation set is in general much higher than that of the training set even though the Validation Set (which is a fraction of the training set) consists of only valid packets. This is undesirable in an intrusion detection system as this would lead to a very high false positive rate. It is for this reason that we did not test how the SOM model handles fuzzing or intrusion detection. That said, we were also able to get nice visualizations of the clustering effects as shown in Figure 4.

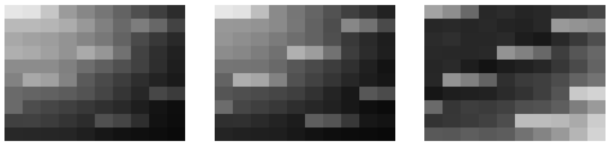

Figure 4: Cue Heatmaps for non-anomalous packets

Shown above are "Cue Heatmaps" where a cue, i.e., a CAN packet, is passed through the SOM and the activations of the neurons are observed. As can be seen above, the activations of the first two packets (from the left) appear the same while the third looks quite different. In fact, the first two packets are from the training set while the rightmost one is from the validation set. Should the SOM method improve and perform better in the future, we suggest the use of such visual representations as they are in some sense easy to interpret and can provide the passengers in automobiles and OEMs a high level overview of what the CAN traffic looks like when intrusion is detected. That said, like with most neural networks, grasping the semantic meaning of these activations is an open question.

5.3 Long Short-Term Memory

Firstly, it should be noted that the model we discuss henceforth is the one with a 25 packet window and 10 hidden neurons. The primary reason for this is training time. The models with more hidden neurons or a larger packet window took a long time to train and this can be prohibitive in embedded systems.

Now, for observing the false-positive and true-positive rate as a function of the threshold, a common method is to plot the Receiver Operating Characteristic (ROC), as done in [10]. The ROC curve essentially arises from a parametric function that takes in a threshold, observes the prediction probabilities and calculates the false positive and true positive rates given the true outputs. Using this method, we obtain the ROC curves with different values for the standard deviation in the Guassian noised timestamps (Figure 5).

One important metric in comparing Binary Classifiers is the Area under the ROC curve (the AUC). As can be

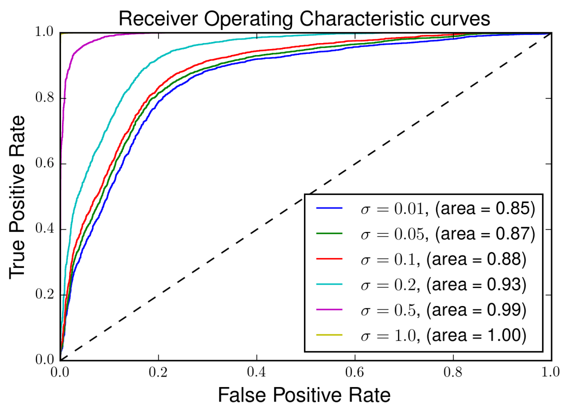

Figure 5: ROC Curves for LSTM Models with Gaussian Noise Timestamps

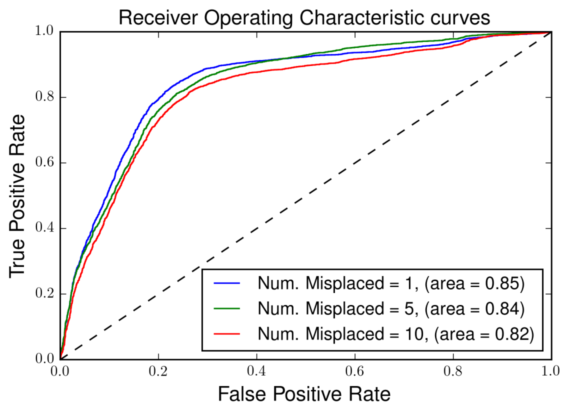

Figure 6: ROC Curves for LSTM Models with Misplaced Packets

seen in Figure 5, the AUC definitely increases as the variance in the noise is increased (as one would expect). On the flipside though, the AUC remains quite high even when the variance is reduced to near 0. This shows that the LSTM models are generally quite robust and can perform quite well with this kind of fuzzing whereby timestamps are changed. What this means is that LSTMs can be good tools for detecting attacks that rely on timestamps or in some sense look to exhausting resources over a short period of time, such as the Denial of Service attacks. In particular, LSTMs can work quite well to ensure a sort of rate-limiting since CAN buses generally operate with very low bandwidth links ($\approx$ 500 Kbps).

Finally, Figure 6 presents the ROC curves for the misplaced packets based anomalies where correct or "valid" packets were inserted into a sequence of otherwise non-anomalous packets.

As can be seen in the Figure, the AUCs for the curves is lower than that of the Gaussian Noised timestamps dataset. However, in this case, it seems that including multiple misplaced packets doesn't really affect the anomaly detection much (the AUCs are approximately the same). This again shows that the LSTM model is quite robust. That said, a crossover point is apparent between the green and blue curves. This suggests that with a more thorough analysis there might be a point until which observing multiple anomalous packets can give better predictions, but after that the performance reduces.

6 Future Work

Our tests in this paper were done in a simulated environment, so the next step for our work is to evaluate our methods in a live environment. This is an important step to take because the unique limitations of embedded systems restrict the computation power available for a machine learning program. We have access to the University of Michigan's solar car, which we believe would be useful for live testing the performance of the different algorithms because we have access to the true meaning of every CAN message. We would load our different trained machine learning algorithms onto a small board, like a Raspberry Pi, and attach it to the CAN bus of the solar car. Then, we would simulate fuzzing while the car was executing regular CAN activity, and deduce whether our trained algorithm was properly alerting us to the active fuzzing. Ultimately, we would investigate ways to optimize the algorithm for the embedded environment.

A notable variation of our work would be to apply the machine learning algorithms to detecting anomalies on a single CAN address, rather than in the stream of all messages. We believe that this would be a more realistic application of machine learning because it would reduce the computational demand of running the algorithm; instead of listening to all messages, including ones that are unimportant to safety, the algorithm could instead only monitor safety-critical messages. In addition, a message-level algorithm would not need to be retaught every time a message in the car is changed, unlike a bus-level algorithm.

An area for improvement of a bus-level algorithm would be detecting relationships between valid messages. While we have investigated the ability to detect malicious valid commands, our work largely looked at individual changes in a stream of data, rather than correlations between pieces of the data stream. [7] showed that a real car is likely to have the same information car mirrored in different messages. Detecting relationships between messages would be extremely useful because an attacker would need to spoof multiple messages. This defense would be especially useful if the relationships between messages are not readily apparent to an attacker, or even the system designers. However, this type of detection is more difficult because it relies on relationships between messages that may only hold under certain scenarios, and a machine learning technique will likely have difficulty detecting relationships with acceptable confidence under the constraints of an embedded system.

With regards to the trained models we have as of now, we haven't thoroughly tested them to see if there is an arbitrary way to avoid detection. Further testing on these models is needed to determine their adequacy for real world application. There is also other areas to explore in terms of how we apply our machine learning. For example, we can train models for each specific type of message, to better detect anomalies as the domain for valid messages may be more specific.

An important area of work, while not directly related to our investigations, is determining how to ensure our defense would work as intended. We briefly proposed that the automaker could be notified of a suspected attack, which could allow preemptive action to be taken. However, there are many challenges to ensuring that any such alerts would not be blocked by an attacker. In addition, just ensuring that an attacker is not able to remove any ML defense is another important challenge that we have not yet investigated.

7 Conclusion

We have shown efficient methods for detecting certain classes of anomalies on the CAN bus. It is our hope that car manufacturers could implement a detection scheme like ours as a fundamental part of the CAN bus. As newer cars have Internet capability, if fuzzing is detected, the car manufacturer, and potentially the owner of the car, could be notified that suspicious activity has occurred on the vehicle. Coupled with identifying information of the vehicle, these alerts could assist automakers and law enforcement in locating possible attackers and preempt any attacks.

We stress that, even if the accuracy of fuzzing detection methods like ours approaches perfect accuracy, the true security effectiveness of fuzzing defense will become weaker as criminal interest grows in hacking cars and CAN databases are developed. We believe that our work can be an adequate defense against fuzzing attacks, and thus slow the progress of attacks, but our work does not replace the need for car manufacturers to improve their security at a more fundamental level.

Attacks on the CAN bus are a dangerous reality, and we predict their frequency will increase in the coming years. It is our hope that significant corporate and academic interest is generated in this field to create prevention and detection methods to stop these potentially deadly attacks, before a large-scale compromise occurs.

References

[1] RTaW simulator. `http://www.realtimeatwork.com/software/rtaw-sim/`. Accessed: 2016-04-03.

[2] Bazzi, A., and Onozato, Y. Ids for detecting malicious non-executable files using dynamic analysis. In *APNOMS* (2013), pp. 1–3.

[3] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium* (2011), San Francisco.

[4] Cheung, S., and Levitt, K. N. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *Proceedings of the 1997 workshop on New security paradigms* (1998), ACM, pp. 94–106.

[5] Killourhy, K. S., and Maxion, R. A. Comparing anomaly-detection algorithms for keystroke dynamics. In *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on* (June 2009), pp. 125–134.

[6] Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on* (2010), IEEE, pp. 447–462.

[7] Miller, C., and Valasek, C. Adventures in automotive networks and control units. *DEF CON 21* (2013), 260–264.

[8] Moore, D., Shannon, C., Brown, D. J., Voelker, G. M., and Savage, S. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS) 24*, 2 (2006), 115–139.

[9] Roesch, M., et al. Snort: Lightweight intrusion detection for networks. In *LISA* (1999), vol. 99, pp. 229–238.

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