Autoencoder Anomaly Detection on Large CAN Bus Data

Elena Novikova
elena.novikova@baesystems.com
BAE Systems, Inc.
Arlington, Virginia

Vu Le
vu.le@baesystems.com
BAE Systems, Inc.
Arlington, Virginia

Matvey Yutin
matvey.yutin@baesystems.com
myutin@berkeley.edu
BAE Systems, Inc.
Arlington, Virginia
University of California at Berkeley
Berkeley, California

Michael Weber
michael.weber@baesystems.com
BAE Systems, Inc.
Arlington, Virginia

Cory Anderson
cory.s.anderson@baesystems.com
BAE Systems, Inc.
Arlington, Virginia

ABSTRACT

It is well-known that the Controller Area Network (CAN) bus is insecure, and significant work has been done applying Intrusion Detection Systems (IDS) to it. However, many approaches to anomaly detection for CAN bus IDS have difficulty when attacked signals do not themselves appear anomalous in content or timing. In this paper we describe the successful development of an autoencoder neural network functioning as an anomaly detector operating over the complex relationships between multiple signals on the CAN bus to overcome this difficulty. Leveraging a large data set with ≈2.7 billion frames covering 22 days of operation, we encounter several new difficulties and present associated solutions for training and evaluating autoencoders processing large and sparse CAN bus data. We then apply the system to a public data set to provide reproducibility comparisons.

KEYWORDS

Neural networks, autoencoder, machine learning, intrusion detection, CAN Bus

ACM Reference Format:
Elena Novikova, Vu Le, Matvey Yutin, Michael Weber, and Cory Anderson. 2020. Autoencoder Anomaly Detection on Large CAN Bus Data. In *Proceedings of DLP-KDD 2020.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Large amounts of data flow through CAN buses in vehicles of all types. Many of these signals carry measurements of physical values; this information is instrumental for the proper operation of the vehicle. An attacker can manipulate these signal values to elicit undesirable or unexpected responses from control algorithms processing them, such as engine control units. Assessed individually, manipulation of values may be impossible to detect if the malicious values happen to be otherwise valid for the signal in question. However, given the nature of machinery, many dependencies and relationships exist between many of these measurements, some obvious, some subtle. In this paper, we describe the application of the autoencoder neural network as an anomaly detector able to extract these subtle relationships and detect otherwise invisible manipulations with reasonable runtime complexity. There are several statistical approaches that can be used to detect the relationships among signals, some simple such as K-nearest neighbors or complex such as cross-correlation and Pearson correlation analysis. These techniques require either preprocessing or simplifying assumptions: disregarding non-stationary behaviors, or remove some non-random components, such as trends due to seasons or other factors. The simplification or preprocessing steps may remove not only unessential nuances, but also important information. We demonstrate that an autoencoder can detect subtle change in the relationships without the need to modify or simplify data, representing big data with relatively small models. We also show how this approach can still be broadly deployed without requiring significant individualized training per vehicle [18].

A CAN bus is a peer-to-peer communications network that allows devices to communicate directly with each other using standard CAN protocols. CAN protocols are message-based protocols in which all messages are visible to all nodes in the network. In essence, each packet contains an arbitration field to determine the message's priority on the bus, a control field to indicate the length of data field and some control bits, a data field of eight bytes and a CRC field that carries the checksum of the message. There is no message authentication, tamper protection, or sender authentication in standard CAN protocols [4]. Previous attacks on the CAN bus have been documented in [5, 26].

2 RELATED WORKS

In this section, we review the literature on vehicular bus Intrusion Detection Systems (IDS). IDSs were originally designed to monitor networked computer systems used by humans [3]. As vehicle controls became more computerized, IDSs were deployed to protect

DLP-KDD 2020, August 24, 2020, San Diego, California, USA

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

in-vehicle networks. Hoppe et al. describe this history [12]. Vehicle bus IDS algorithms can be broadly categorized into four types: frequency-based, sequence-based, statistics-based, and machine-learning-based [16].

A frequency-based IDS tests frame arrival rates against predefined characteristics, and abnormal appearance frequencies trigger alerts. Groza et al. used Bloom filtering to test frame periodicity based on message identifiers [9]. The Bloom filtering algorithm has a time/space balance suited to embedded controllers, and effectively finds frame replay and modification attacks. Taylor et al. [24], Song et al. [21] and Gmiden et al. [8] also investigated frequency anomaly detection due to frame injection.

A sequence-based IDS detects attacks by comparing bus traffic to a set of known-to-be-valid message sequences. Stan et al. [22] and Ferling et al. [6] proposed methods to detect if frame arrivals complied with expected timing and sequence constraints. This type of attack is hard to detect with other algorithms since all packets are legitimate; only their order or timing is adjusted to serve malicious purposes. The other method of sequence-based IDS uses language to define the forbidden message sequences. The language serves as templates to detect the anomalous traffic [23]. Rieke et al. described a model-based method to detect unanticipated sequences of events in order to identify suspicious activities [20].

A statistics-based IDS performs various statistical tests on observed traffic and makes decisions based on the results. Genereux et al. extracted several timing-related features from messages on the bus and built histograms of each feature's distribution. These baseline histograms are then compared with observed traffic histograms to detect attacks [7]. Tomlinson et al. described in [25] a statistical technique to detect the timing changes in CAN traffic. The authors processed data into discrete, non-overlapping windows, and calculated some metric in each window, using the results for classification.

A machine-learning-based IDS involves techniques such as neural networks, support vector machines, and clustering to detect abnormal traffic. Avatefipour et al. reported using a modified one-class support vector machine on CAN traffic to quantify the correctly classified windows in test sets containing attacks in offline training [1]. Boumiza and Braham used a Hidden Markov Model to extract features from CAN packets and used them to build a proper model to detect the malicious packets [2]. Jin et al. used fuzzy rule interpolation techniques [13] to generate adjustable association rules during training, and used those during testing. Deep learning is also used in in-vehicle network security [14, 29].

The technique presented in this paper belongs to the machine-learning-based group. It uses an autoencoder to learn the relationships between physical signals. The learned relationships become the baseline for detecting malicious frames. The autoencoder has been used in vehicle IDS before: Weber et al. used an autoencoder to model each individual signal in vehicle IDS [27]. However, to the best of our knowledge, this is the first paper that describes using an autoencoder to detect anomalous signal correlations in a CAN bus IDS.

3 DATA DESCRIPTION

We performed our analysis and model construction using a large set of recorded CAN bus traffic that came from nine different vehicles of two types (we will call them types A and B). The data set from each vehicle comprises about 300M frames from several days of observation. We identified 81 signals common to all nine vehicles that represent measurements of continuous physical phenomena. Each signal has a fixed message rate between 1Hz and 50 Hz and has on the order of 1 million seconds' worth of data. To achieve a uniform sampling rate for all signals, we used their mean reported values in successive one-second windows. This allowed us to assign a single value for each "whole" second for each signal where at least one raw sample was present in that second.

4 SIGNAL GROUPS AND AUTOENCODER TRAINING ALGORITHM

We began by recognizing that particular groups of n signals, being physically or otherwise related to each other, self-organize into a (potentially, but not necessarily, complex) n-manifold, when considered in their n-dimensional space. An autoencoder should be able to capture that structure in each group during training and detect if values for a signal are out-of-character in the context of the state of the group as a whole. "Violations of character" can be attributed to, among other things, malicious manipulation of signal values.

When identifing signal groupings, we had to deal with the big data problem in the following atypical manner. During training, our resources may be considered to be significant, but the ultimate neural net models have to execute forward propagation on platforms with very limited computational and memory resources, such as a typical vehicle-embedded processing units. In a way, we had to "sparse out" the problem down to a very manageable size, and one of the ways was to limit the size of the neural net used in the autoencoder, which was directly related to the number of signals in the groups we were considering.

We experimented with different group sizes, looking for groups of physical signals that showed reasonably strong internal relationships. Among our 81 available signals, the number of promising groups dropped precipitously for group sizes greater than three, while the computational time for both training and forward propagation grew polynomially. Overall signal coverage in the final group set, coupled with the desire to enable real-time operation of a trained autoencoder-based IDS on an embedded platform, led us to define our groups using three signals only. Currently we manually defined groups. In the near future we will use clustering techniques to automate that process.

Our time-normalized data, as explained above, has at most one value per signal for each second; if any signal was not observed during a particular one-second window, that second was discarded for all signals. Since groups have three signals, mapping each signal to an axis in 3D space is a convenient way to visualize inter-signal relationships. Uniform sampling gave us regular time stamps and ability to reduce the combined time series of a given 3-signal group to a collection of (ordered) points in 3D space. Each point reflects the values of the three signals at one moment in time. In this initial investigation, the ordered nature of the points was discarded; i.e. we consider only the instantaneous values of the signals as represented

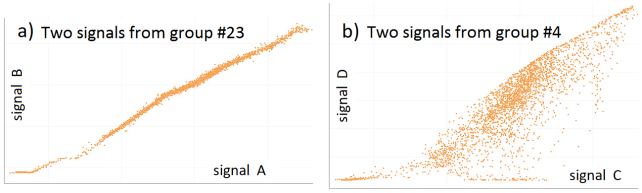

Figure 1: 2D projections of 3D patterns.

by the "point cloud," without regard to their evolution in time. Future work could consider how vehicle state moves from point to point in the resulting cloud, providing another opportunity to detect anomalous signal movement.

Two 2D plots of signal relationships are shown in Figure 1. The tight relationship between signals A and B in panel (a) of the figure is evident, even though the relationship is obviously non-linear. The relationship between signals C and D in panel (b) is more complex: an obvious upper bound on signal D can be seen, the value of that bound being a function of signal C. Also, some areas under the upper bound are more densely covered by the observed instances of (C, D) values than others.

Careful study of the data resulted in the identification of 32 groups of three signals exhibiting non-uniform point clouds. Fifteen signals could not be assigned to any group, and some signals belong to more than one group.

After identifying the groups, we proceeded to training of our autoencoders. An autoencoder is a specific type of feed-forward neural network whose task is to replicate its inputs [28]. The structure of an autoencoder is usually symmetric, with its shape resembling that of a butterfly, where the outermost layers (the input and the output layers) have the most nodes and the innermost layer has the fewest nodes. In our case, the configuration worked out slightly differently: the number of nodes in the input and output layers corresponds to the number of analyzed signals, which in our case is three. After a process described in section 5, we arrived at the non-canonical shape of 3-50-10-2-10-50-3. Autoencoder is an unsupervised learning model that is well-known in capturing the latent features of the datasets. A properly trained autoencoder can capture benign data distribution and become the base line for anomaly detection.

We used Pytorch [19] to train the autoencoder sensor. We experimented with different activation functions such as ReLU, Tanh,... and ended up using Sigmoid functions as activation functions [17], Mean Square Error is the loss function, and Adam is the first-order gradient-based optimization of the loss function, with learning rate of 0.001 [15]. The training ends when the loss stops decreasing.

5 CHOOSING NEURAL NET CONFIGURATION

Autoencoders are symmetric with respect to their center layer, flaring out from the center, towards the input and output layers. In our case, the selected input group size dictates that the input and output layers have three neurons. The center layer must have fewer neurons than the outermost layers to make the feature space smaller than the input space. This constraint forces the autoencoder to find the important features defining the input. The left half of the network, the encoder, encodes the input into a tight set of parameters representing the data features; the right half, the decoder, reconstructs the input using encoded features. The simplest non-trivial neural net that satisfies these conditions has a center layer of two neurons and outer layers of three neurons, i.e., a 3-2-3 configuration.

We started with this 3-2-3 configuration and estimated the quality of the training using several methods, pictured in Figures 2, 3, and 4. The same set of signals, "group #1," was used in all configurations presented in this section. The group #1 data set baseline has approximately 1.3 million data points, and all of these data were used as input to the first three network configurations.

Training was conducted until the average loss per epoch stopped decreasing. We use the words "loss" and "error" interchangeably here, because, in our setup, the loss function is the error of reconstruction, defined as the Euclidean distance between the original and the reconstructed points in the space of the normalized signals. The error is expressed in the units of standard deviation from the mean, since the signals are normalized. As seen in Figure 2 (yellow curve), the neural net 3-2-3 doesn't improve beyond error value 0.0048, which is too high for a reliable operation of an autoencoder-based anomaly detector. In reality, the situation is even worse, as the next two figures illustrate.

Instead of relying solely on average error, we assess training quality using the distribution of the error, as displayed in Figure 3. The histogram is shown in log-scale to address the sharp peaks, discussed below. The yellow curve shows the error distribution for net

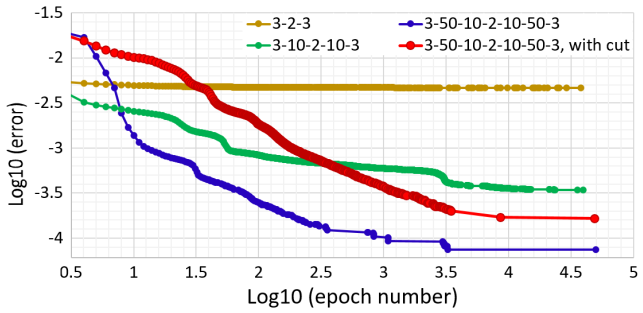

Figure 2: Average error for group #1 data set vs. epoch number for four neural net configurations.

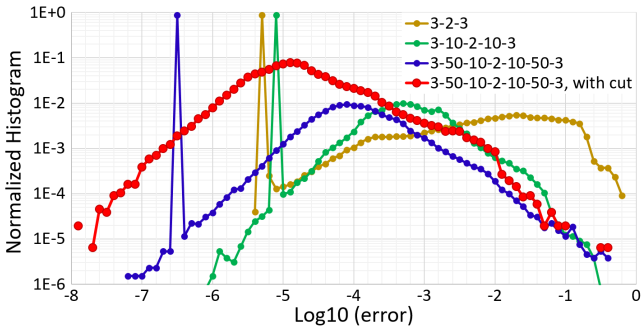

Figure 3: Reconstruction error in group #1.

configuration 3-2-3. It is clear that, even though the average error is 0.0048, many data points have poor reconstruction quality; the error value rises to 0.16 ($log_{10}(0.16) = -0.8$) quite often, and goes as high as 0.5 in a nontrivial number of cases. An error of 0.5 means that the position of the reconstructed point has no resemblance to the location of the original point.

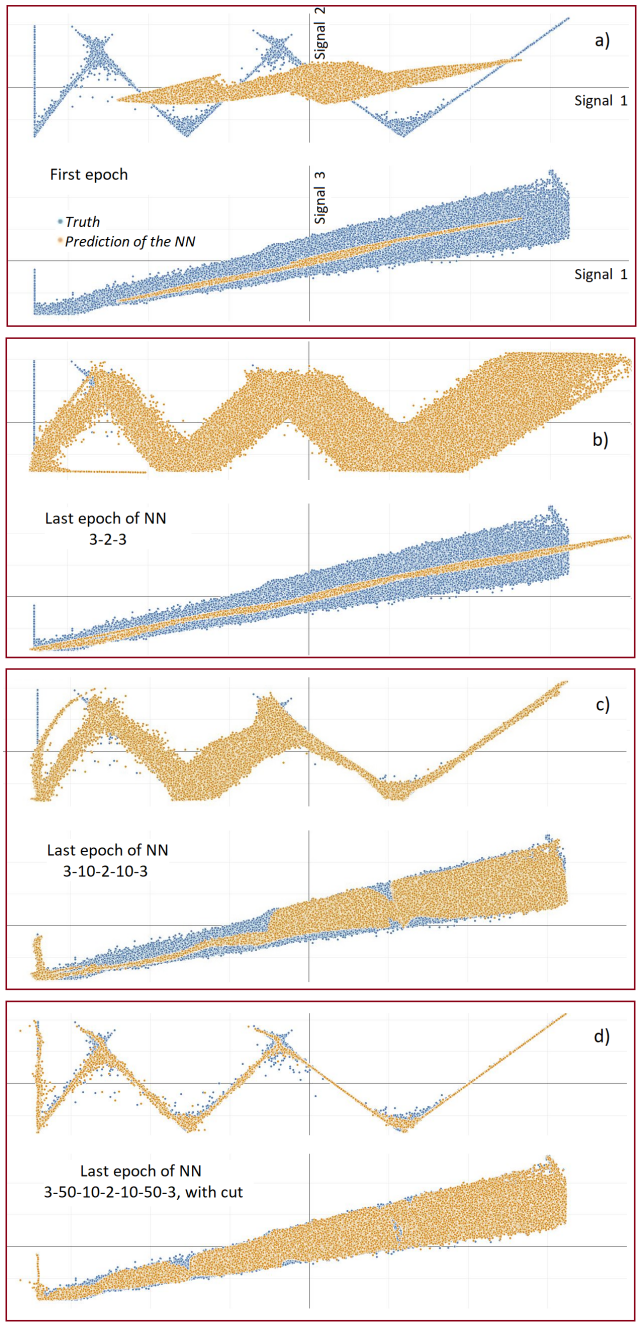

Figure 4: Training error visualization for group #1.

The second method for assessing training quality is a plot of input data vs. output data, as shown in Figure 4. The net's performance is presented as a match (or lack thereof) between the input "truth" (blue) values and the output "predicted" values (orange) for each 3D point in the data set. The 3D space of the normalized signal values is projected on to two 2D planes, one plane with the scatter plot of "signal 2" vs "signal 1," and the other plane with the scatter plot of "signal 3" vs "signal 1." Each panel in the figure reflects the performance of a neural net configuration after the first (top) or last (bottom three) training epochs. An ideal, perfectly trained autoencoder will produce exactly the same values at the output as were provided at the input. In an ideal situation, the output (orange) points would be exactly on top of the input (blue) points for all points in the training data set. All of the nets in Figure 4 were trained on the same group #1 data set.

The state of training after one epoch in Figure 4a is obviously poor, as it comes out of the first, random, approximation. Figure 4b shows the best the 3-2-3 net can achieve after training is complete (when the average error is no longer decreasing). The reconstructed result (orange) is much thinner in the signal-1-versus-signal-3 plane than the distribution of truth points (blue). Simultaneously, the zigzag pattern in the signal-1-versus-signal-2 plane is captured poorly, with lots of "overspray" around the truth pattern. Obviously, the neural net 3-2-3 cannot capture the richness of patterns present in the data — it simply does not have enough parameters.

The next neural net configuration we considered is more complex, with three hidden layers: 3-10-2-10-3. After training, this neural net yielded the green curves in Figures 2 and 3, and its performance is presented in Figure 4c. The average reconstruction error is reduced by more than an order of magnitude compared to the 3-2-3 net. The entire distribution of training error shifted to the left in Figure 3, and Figure 4c shows the greatly improved reconstruction precision. This neural net started to recognize the details of the patterns present in the data of group #1, such as: the span of the allowed values of signal 3 as a function of signal 1; the vertical "tail" at the left edge of both 2D projections; and the sharpness of the zigzag pattern. However, the recognition of these patterns is far from being complete.

The final neural net configuration has five hidden layers. We gave the encoder a wide first hidden layer (50 neurons), which redundantly finds observable features of the input, and a tapering tail (50-10-2), which extracts the salient features from the high-dimensional space. We designed the decoder with the reverse of the encoder's architecture (2-10-50-3). The total number of parameters in this neural net is 1465, and, with the number of data points in the training set (1.3 million), we have about 890 data points per each parameter that needs to be trained.

After training, the 3-50-10-2-10-50-3 neural net configuration yielded the blue curves in Figures 2 and 3. Compared to the green curves of the 3-10-2-10-3 configuration, there is about three-quarters of one order of magnitude's worth of improvement in the average error, and about an order of magnitude shift to the left in the position of the main dome of the reconstruction error histogram.

However, the dominant feature of the error distribution in Figure 3 for all three configurations considered so far is the sharp peak at the left edge of the histogram. Investigation reveals that 88% of the data points forming this peak come from repetitive data points.

Specifically, we found the CAN bus recording contained stretches of time during which none of the signals in certain groups changed significantly, resulting in several very dense clusters of data points in the 3D space of normalized signal values. Training the neural net with all of these overlapping points results in over training for these clusters, wasting neural net parameters that overemphasize these, and under-representing the other, no less important parts of the manifolds and sub-volumes.

To solve this problem, we limit the number of duplicate data points present in the training set through a "bin and cut". The signals on the CAN bus used in this study are of two precisions, 8-bit and 16-bit. We bin the normalized axis of the 8-bit signals into 256 bins, 16-bit ones into 1000 bins. These bin sizes, we determined, capture and tame the sets of duplicate data that were causing the over-training. The binning results in a 3D grid, and we accept no more than 100 points from each cell, "cutting away" the extras. Applying this filtering to the group #1 data set results in a one-order-of-magnitude reduction in the number of data points to about 150,000 points available for training. Even after this drastic reduction, the data-point-to-net-coefficient ratio is still greater than 100.

The results of training the 3-50-10-2-10-50-3 neural net configuration with the bin-and-cut reduction of data are represented by red curves in Figures 2 and 3, and by the panel (d) in Figure 4. Comparison of the red and blue histograms in Figure 3 shows the red curve dome moving significantly to the left of the blue curve's dome, and the sharp peak of the blue curve is gone from the red curve. Filtering the data before training (making the data properly sparse) allows the autoencoder to capture more of the data set features that need to be learned instead of being overly focused on the repetitive data.

Interestingly, the final average training error is higher for the bin-and-cut data set (red curve) than for the unfiltered data set (blue curve) in Figure 2. The error among the large number of over-represented data points in the unfiltered data set is very small (confirmed with Figure 3), which reduced the average error. This case clearly demonstrates how dangerous it is to rely on a single metric when estimating the precision and quality of a specific neural net configuration. Close inspection of details may result in significant improvement of the final product.

The reconstruction precision for the neural net with five hidden layers trained on filtered data can be seen in Figure 4d. This configuration results in a close match of the reconstructed points with their original input points.

6 VERIFICATION OF RESULTS

The result of training a neural net on a certain data set is a model, where all 1465 coefficients of the neural net 3-50-10-2-10-50-3 are defined. This model needs to be verified, i.e., applied to a new data set that was not used in any form during training. Due to the large volume of data available, we were able to apply the model trained on our "base" vehicle to the data from the CAN buses of other vehicles. For the purposes of verification of our training and bin-and-cut algorithms, we performed these comparisons only among vehicles of the same type, for reasons discussed in the next section.

The verification results of the signal group #1 model are shown in Figure 5. Again, we use the distribution of reconstruction errors

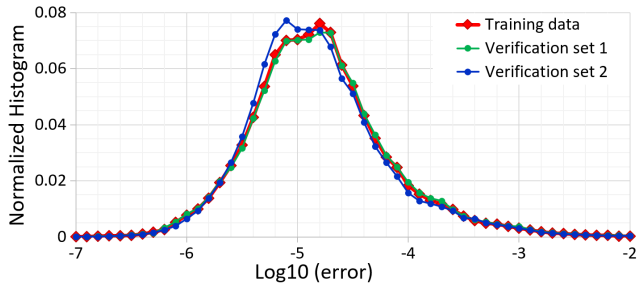

Figure 5: Verification of reconstruction, group #1.

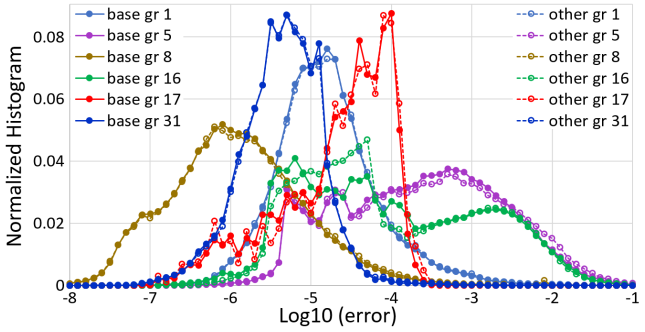

Figure 6: Application of the model trained on the base vehicle, to other vehicles, for several signal groups.

for a clearer picture of the model's performance. Data sets from two additional vehicles show an almost perfect match to the model's treatment of the training data set from the base vehicle.

It is important to note that this analysis allows us to conclude that models trained on one vehicle can be used on another vehicle, without re-training, as long as the characteristics of the vehicles are similar enough with respect to the signals assigned to a particular group.

Let us define "data X" as "the data from vehicle X." We further define "model X" as "the model trained on data X," and "model X's response to data X" (i.e., to the very data on which it was trained) as the "self-response." The self-response, as well as the response of model X to data Y, can be presented as a histogram (as in Figures 3 and 5), or as a scatter plot of 2D projections of the original and reconstructed points (as in Figure 4, and later in Figure 9). The self-response for six different models from vehicle #1 (the "base") are depicted in Figure 6, each with an associated plot of that model's response to the data of some other vehicle. The models of these six groups display different behavior, from group to group, but the response of the model to data from two different vehicles match surprisingly well. For anomaly detection purposes, it is especially important to check the high-error-value-tail part of the histogram for alignment, since this is the tail along which the cutoff between benign (acceptable) behavior and anomalous behavior is defined.

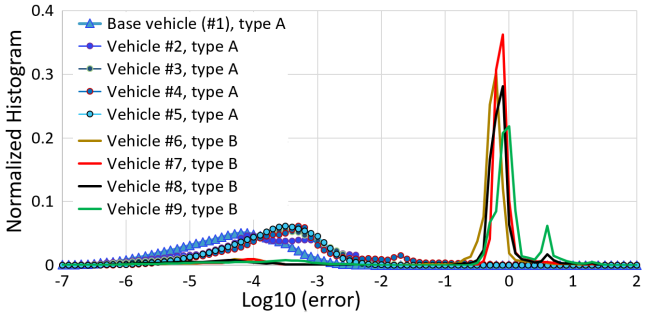

Figure 7: Model for vehicle #1, group #6: self-response and response to data from other vehicles.

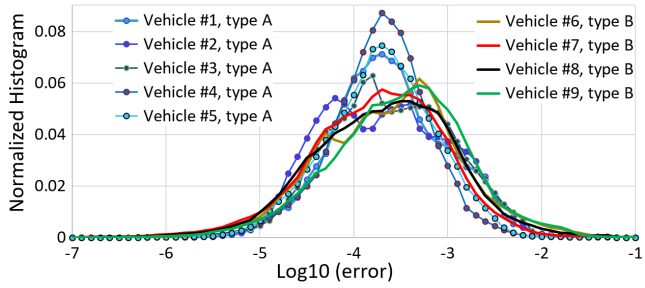

Figure 8: Response of the combined model to the data of every vehicle, one at a time (group #6).

7 THE MODEL FOR ALL VEHICLES

Not all signal groups behave as well as those shown in Figure 6; for some groups, model X's response to data Y is very different than its self-response. One such example is group #6, as shown in Figure 7.

Not only do the base model's responses to the data of various other vehicles not match, but we also see a clear separation of response histograms into two groups, one having responses resembling the self-response of vehicle #1, and the other group that is very different from that self-response. This separation corresponds to physical characteristic differences between the two vehicle types. When the systems are so different, can we still come up with a model that will work on all vehicles of both types, with the goal of reducing logistics and possible mistakes "in the field?" The answer is "yes, we can."

Consider a combined data set from all vehicles, normalized, binned, and cut as described above. We trained the 3-50-10-2-10-50-3 neural net using this combined data set. The patterns present in the data of each separate vehicle will be preserved, and will only be amended by the data from the rest of the vehicles. Overall, the combined patterns can be expected to be more complex, and the neural net may lose precision that the crisper patterns represent in each vehicle separately. However, the neural net still recognizes the combined pattern.

We define the model trained on the data coming from all vehicles as a "combined model." The combined model's response to each vehicle's data individually is shown in Figure 8. Comparing with Figure 7 clearly shows how the model's response to the two types of vehicles (type A and type B) has changed, becoming more cohesive and uniform. However, the response is not ideal, probably indicating that a more complex neural net should be used to learn a richer feature set of the combined data. Another indicator of the need to increase the complexity of the neural net is the worsening (shift to the right) of the position of the peak in responses of the combined model (Figure 8), when compared to the self-response of the vehicle #1 model (base vehicle in Figure 7). The loss of precision is approximately half an order of magnitude.

More details of the combined model's performance can be seen in Figure 9, where the response from two models (model of vehicle #1, and the combined model) is shown as a scatter plot projection onto one of the 2D planes.

The self-response of the vehicle #1 model is shown in panel (a). The model is well-trained; the positions of the reconstructed (orange) points are very close to the positions of the original (blue) points. Applying vehicle #9 data to the vehicle #1 model results in a poor match, seen in panel (b). Vehicle #9's input data points are shown in green in panel (b); these points have a different structure than vehicle #1's input data points. It is not surprising that the neural net trained to see the patterns of blue dots from panel (a) cannot recognize the green dots from panel (b), and in fact, is treating them as anomalous, giving very large errors of reconstruction.

When the data sets from many vehicles are combined, the patterns of type A vehicles (e.g., vehicle #1) are combined with the

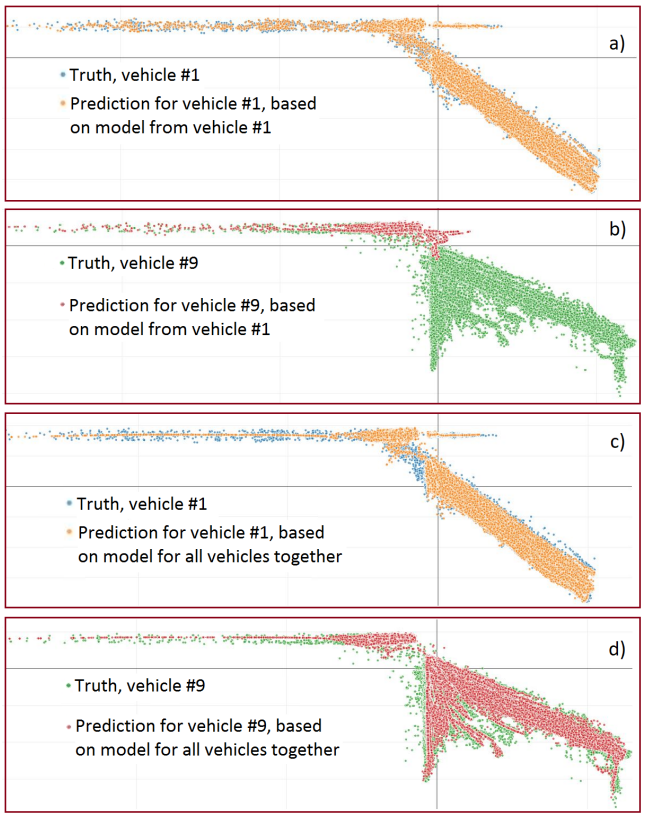

Figure 9: Training error visualization, group #6.

patterns of type B vehicles (e.g., vehicle #9). The combined model can then recognize patterns from both types of vehicles and can deal with variations of the patterns inside one vehicle type. The combined model does have a slightly poorer fit to the data of a specific vehicle when compared to the self-response of that vehicle's model. Compare panels (a) and (c) in Figure 9: the points reconstructed in panel (c) have larger error than in panel (a). This is the trade off required for a combined model to respond well a broader range of data from any specific vehicle within a fixed net configuration. The response of the combined model to the vehicle #9 data is now reasonable (Figure 9d), while the response of the vehicle #1 model to the same data was very poor (Figure 9b).

8 USE CASE

We selected a 100-second segment of input data from our proprietary data set, and created a "disturbance" by halving the values of purple signal. We then feed the modified data set to our anomaly detector built around the autoencoder and observe the resulting raw and filtered detections. The results are presented in Figure 10.

When comparing the original data (Figure 10a) to the modified data (Figure 10b), it is clearly difficult to detect the disturbance without relating the purple signal to the other signals from the groups that it belongs to. The modified purple signal retains its original range and statistical characteristics; however, the autoencoder easily detects this anomaly because it produces a large reconstruction error in excess of a reasonable threshold.

The warnings raised for various groups are shown in Figure 10c. The location, height, and color of each vertical bar indicates the anomaly time, severity (degree of deviation from normal), and source group respectively. The modified purple signal belongs to several groups, and the warnings for these groups inside the disturbance time window are the legitimate ones. However, a total of 12 groups generated warnings, many of them outside of the disturbance time window. To prevent these warnings from being reported as false positives, we employ a filter algorithm which requires the warnings to be continuous, to have significant average severity, and to have a large absolute severity value at least briefly. Warning sequences meeting these criteria are eligible to be promoted to alarms.

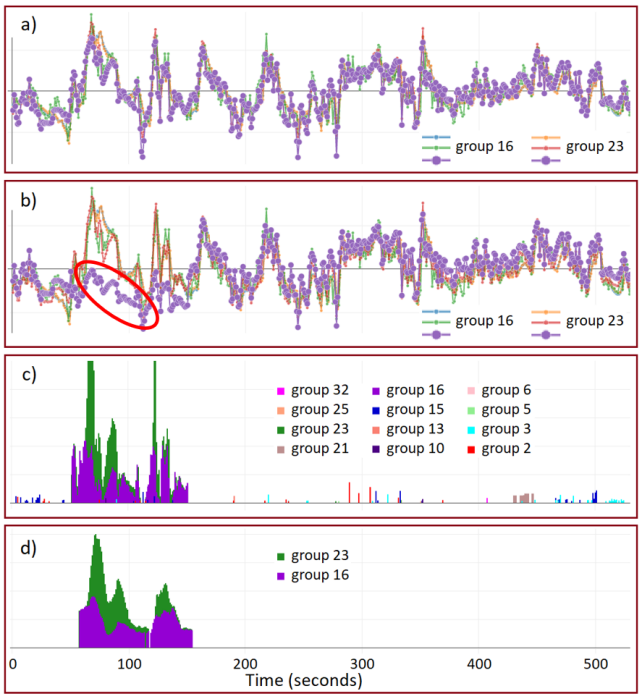

Figure 10: Performance of autoencoder during a case study.

The result of this filtering is shown in Figure 10d. Only warnings from groups #16 and #23 pass and are promoted to alarms; the other weak, brief warnings are filtered out.

9 REPRODUCIBILITY WITH SYNCAN DATA

As mentioned in [10], publishing the real CAN traffic used in this paper for reproducibility is impossible due to intellectual property restrictions. To adapt to this situation, we resort to the published data set SynCAN that Hanselmann et al. [11] gracefully produced and published in github.

The data consists of a training data set and six test data sets. The data is represented in CSV format, with the following columns: Label, Timestamp (in milliseconds), ID, Signal-1, and three optional columns (Signal-2, Signal-3, Signal-4). Labels 0 and 1 are for benign and anomalous data entries, respectively, ID is the CAN packet ID, each packet has at least one signal, with the maximum of four. There are a total of twenty different signals (label types) in the records.

The training data set has four CSV files, each contains roughly seven million benign packets, accounting for about 13 hours of operation. The test data sets have the same format, with some packets labeled "1," meaning "anomalous." We used three test sets for verification of the performance of our autoencoder as an anomaly detector: "test_plateau," "test_playback" and "test_continuous." Each of these sets contains multiple (≈100) attacks that last for several seconds.

The SynCAN data set has sampling frequencies of 22.2Hz, 33.3Hz or 66.6Hz. We scaled the time axis to have 2, 3, or 6 samples per one "time tick," by first converting time from milliseconds to seconds, and then dividing by 0.09. The sampling rate change does not affect the outcome of training and/or testing, but allows for easier handling of data. All figures below that include a time axis use these "time ticks" as the unit. We divided the set of 20 signals into 11 groups, as seen in Table 1.

Each group contains three signals that are related to each other in some way; we used our visual tools to find related signals. We trained our SynCAN autoencoders and prepared the models to be used in attack detection. Our preliminary testing using self-response histograms (similar to the blue curve in Figure 5) showed that our SynCAN-models for 11 groups behave as expected, and we proceeded to work with the three testing sets that contain attacks. For the convenience of representation, we shifted our (already stretched) time scale in each of the "attack" files so that the beginning of each file corresponds to time tick "zero."

We first considered "plateau" attacks. During "plateau" attack, one of the signals becomes constant for the duration of the attack,

Table 1: List of groups and their signal components.

Group #	Signal 1	Signal 2	Signal 3
1	ID1-Sig1	ID2-Sig2	ID5-Sig1
2	ID1-Sig1	ID2-Sig2	IDA-Sig4
3	ID2-Sig1	ID3-Sig2	ID7-Sig1
4	ID2-Sig3	ID8-Sig1	ID7-Sig1
5	ID2-Sig2	ID5-Sig1	IDA-Sig4
6	ID5-Sig2	ID6-Sig1	IDA-Sig3
7	ID9-Sig1	IDA-Sig1	IDA-Sig4
8	ID1-Sig1	ID4-Sig1	ID5-Sig1
9	ID1-Sig2	ID3-Sig1	ID7-Sig2
10	IDA-Sig2	ID6-Sig2	ID7-Sig2
11	ID1-Sig2	ID6-Sig2	ID7-Sig2

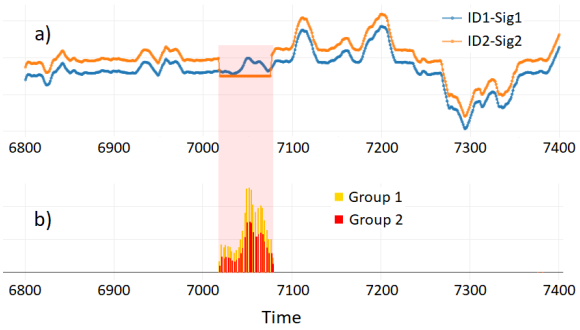

Figure 11: A detection of a SynCAN "plateau" attack.

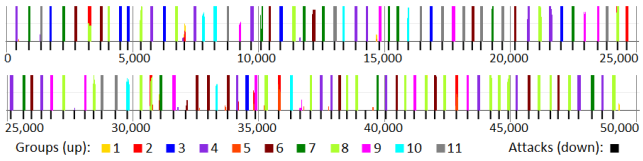

Figure 12: Statistics of detection of "plateau" attacks.

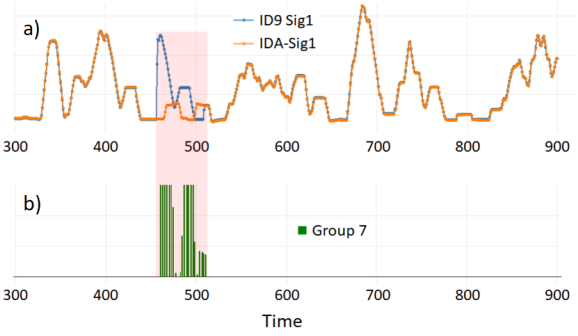

Figure 13: A detection of a SynCAN "replay" attack.

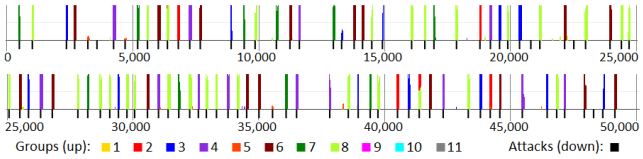

Figure 14: Statistics of detection of "playback" attacks.

and the other two remain "live." When it happens, chances are good that the combined values of the three signals will go outside of the relationship the autoencoder is trained to accept as normal for this specific CAN bus.

A good example of such a detection is shown in Figure 11. The value of the signal under attack, ID1-Sig1, "jumps" down between time ticks ≈7020 and ≈7070 (Figure 11a). This type of attack cannot be detected based on range, since the "misleading" value is well within the usual range of the signal, as can be seen in Figure 11a. As shown in Figure 11b, the autoencoders for groups #1 and #2 show detections all the way through the time frame of the attack: the height of the colored vertical bars in Figure 11b depicts the strength of the detection.

The overall statistics of our detection for all "plateau" attacks can be seen in Figure 12, where all attacks and detections are shown against the time axis in "ticks." The attacks are represented as black bars descending from the time axis, and the color of the detections shows the group(s) that caught them. Our system detected 115 out of 116 attacks.

Our next case is the "playback" attacks. A signal is "hijacked" and its value is changed over the time frame of the attack. The value is changed by replacing a portion of the time series by another portion of the same time series, taken from a bit earlier. An example of such an attack, and of its detection, is presented in Figure 13.

It can be seen that the values of signal ID9-Sig1 are overridden between time ticks ≈ 452 and ≈ 512. The values for this override were taken from the time span immediately preceding the attacked time span, as can be seen in Figure 13a. The signal under attack belongs to group #7, and the autoencoder easily detected this replay. As described above, our application of the autoencoder as an anomaly detector is based on the fact that it can recognize the points in time when the signals do not relate to each other in the same way as they do in the training data. Close inspection of the detection strength (the height of the green bars in Figure 13b) shows that it is proportional to the magnitude of the substituted value's departure from the original value.

The signal IDA-Sig1, which is also a member of group #7, is following the signal ID9-Sig1 - in benign data. Comparison of these two signals provide an excellent measure of the discrepancy between the original values of ID9-Sig1 and the "replay" values. Notice that the autoencoder doesn't have difficulties typical for the methods that work with one signal at a time: a replay attack is very hard to detect using methods based on properties of only one signal, because the substituted values come with the properties that look benign to a detector that doesn't relate those values to the values of the other signals on the CAN bus. The quality of the detection of all "playback" attacks is shown in Figure 14: our system detected 89 out of 93 attacks.

The "continuous" attack is shown in Figure 15a). Two closely related signals from this group are shown, and it is clear that IDA-Sig4 is attacked around time tick 17,695, where its value is replaced

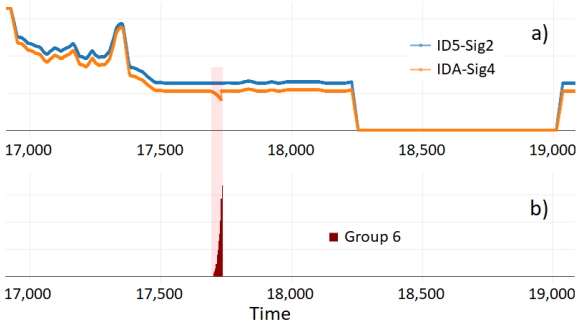

Figure 15: A detection of a SynCAN "continuous" attack.

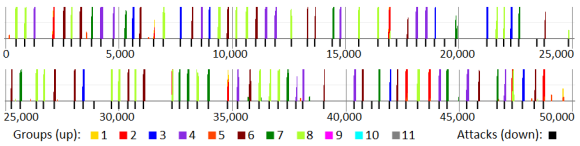

Figure 16: Statistics of detection of "continuous" attacks.

by a progressively different value (a continuous downward ramp is evident). The height of the brown bars in Figure 15b is proportional to the significance of the detection raised by autoencoder of the group #6. The stronger the discrepancy between the original value and the substituted value, the easier it is to detect the attack.

The statistics of our detection of "continuous" attacks is presented in Figure 16: our system detected 91 out of 96 attacks.

10 CONCLUSIONS

In this paper we described the process and results of developing an autoencoder-based anomaly detection capability backed by a uniquely large (≈2.7 billion frames) CAN bus data set, along with some insights enabled by such data. We further presented detailed results of anomaly detection using our autoencoder. We illustrated how to keep data unbiased when handling sparse data set. Importantly, the constructed model can be reused for similar types of vehicles without losing precision. However, if we can accept a bit lower precision then we can build a model that can fit all available types of vehicles. That capability is valuable in a sense that we can project our model to even unseen vehicles.

REFERENCES

[1] Omid Avatefipour, Ameena Saad Al-Sumaiti, Ahmed El-Sherbeeny, Emad Mahrous Awwad, Mohammed A. Elmeligy, Mohammed A. Mohamed, and Hafiz Malik. 2019. An Intelligent Secured Framework for Cyberattack Detection in Electric Vehicles' CAN Bus Using Machine Learning. *IEEE Access* 7 (2019), 127580–127592.

[2] Safa Boumiza and Rafix Braham. 2019. An Efficient Hidden Markov Model For Anomaly Detection In CAN Bus Networks. *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (September 2019). https://doi.org/10.23919/softcom.2019.8903789

[3] Guy Bruneau. 2019. The History and Evolution of Intrusion Detection. (2019). https://www.sans.org/reading-room/whitepapers/detection/paper/344

[4] Steve Corrigan. 2016. *Introduction to the Controller Area Network (CAN).* Texas Instruments.

[5] Roderick Currie. 2016. Developments in Car Hacking. https://www.sans.org/reading-room/whitepapers/ICS/developments-car-hacking-36607

[6] Benedikt Ferling, Justyna Chromik, Marco Caselli, and Anne Remke. 2018. Intrusion Detection for sequence-based attacks with reduced traffic models. *19th Internation GI/ITG Conference on "Measurement, Modelling and Evaluation of Computing Systems* (February 2018). http://www.mmb2018.de

[7] Sebastien Genereux, Alvin Lai, Craig Fowles, Vincent Roberge, Guillaume Vigeant, and Jeremy Paquet. 2019. MAIDENS: MIL-STD-1553 Anomaly-Based Intrusion Detection System Using Time-Based Histogram Comparison. *IEEE Trans. Aerospace Electron. Systems* (April 2019). https://doi.org/10.1019/TAES.2019.2914519

[8] Mabrouka Gmiden, Mohamed Gmiden, and Hafedh Trabelsi. 2016. An Intrusion Detection Method for Securing In-Vehicle CAN bus. *17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA2016)* (December 2016), 19–21.

[9] Bogdan Groza and Pal-Stefan Murvay. 2019. Efficient Intrusion Detection With Bloom Filtering in Controller Area Networks. *IEEE Transactions on Information Forensics and Security* 14, 4 (April 2019), 1037–1051.

[10] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. 2019. CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data. *arXiv preprint arXiv:1906.02492* (2019).

[11] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. 2019. SynCAN Dataset. (2019). https://github.com/etas/SynCAN

[12] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. 2009. Applying Intrusion Detection to Automotive IT - Early Insights and Remaining Challenges. *Journal of Information Assurance and Security* 4 (2009), 226–235.

[13] Shangzhu Jin, Yanling Jiang, and Jun Peng. 2018. Intrusion Detection System Enhanced by Hierarchical Bidirectional Fuzzy Rule Interpolation. *2018 IEEE International Conference on Systems, Man, and Cybernetics* (2018). https://doi.org/10.1109/SMC.2018.00010

[14] Min-Joo Kang and Je-Won Kang. 2016. Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. *PLoS One 11(6)* (June 2016). https://doi.10.1371/journal.pone.0155781

[15] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *Proceedings of 3rd International Conference on Learning Representations (ICLR 2015)* (2015).

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

[17] Peter Norvig and Stuart J. Russell. 1994. *Artificial Intelligence: A Modern Approach.* Prentice Hall.

[18] Elena I. Novikova, Vu Le, Michael Weber, Cory Anderson, and Samuel N. Hamilton. 2020. Best Practices For Ground Vehicle Intrusion Detection Systems. *Ground Vehicle Systems Engineering and Technology Symposium* (2020). To appear.

[19] Adam Paszke and Sam Gross et al. 2019. Pytorch: An Imperative Style, High-Performance Deep Learning Library. *Proceedings of 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)* (2019).

[20] Roland Rieke, Marc Seidemann, Elise K. Talla, Daniel Zelle, and Bernhard Seeger. 2017. Behavior Analysis for Safety and Security in Automotive Systems. *Proceedings of 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP 2017* (March 2017). https://doi.10.1109/pdp.2017.67

[21] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. 2016. Intrusion Detection System Based on the Analysis of Time Intervals of CAN Messages for In-Vehicle Network. *Proc. 2016 International Conference on Information Networking (ICOIN)* (January 2016), 63–68.

[22] Orly Stan, Yuval Elovici, Asaf Shabtai, Gaby Shugol, Raz Tikochinski, and Shachar Kur. 2017. Protecting Military Avionics Platforms from Attacks on MIL-STD-1553 Communication Bus. (2017). https://arxiv.org/abs/1707.05032

[23] Ivan Studnia, Eric Alata, Vincent Nicomette, Mohamed Kaaniche, and Youssef Laarouchi. 2015. A language-based intrusion detection approach for automotive embedded network. *Proceedings of 21st IEEE Pacific Rim International Symposium on Dependable Computing, IEEE 2015* (2015).

[24] Adrian Taylor, Nathalie Japkowics, and Sylvain Leblanc. 2015. Frequency-Based Anomaly Detection for the Automotive CAN bus. *Proc. World Congress on Industrial Control Systems Security* (2015), 45–49.

[25] Andrew Tomlinson, Jeremy Bryans, Siraj A. Shaikh, and Harsha K. Kalutarage. 2018. Detection of Automotive CAN Cyber-Attacks by Identifying Packet Timing Anomalies in Time Windows. *Proceedings of 48th Annual International Conference on Dependable Systems and Networks Workshops* (2018).

[26] Chris Valasek and Charlie Miller. 2014. Adventures in Automotive Networks and Control Units. *Technical Paper, IOActive, Inc.* (2014).

[27] Marc Weber, George Wolf, Eric Sax, and Bastian Zimmer. 2018. Online Detection of Anomalies in Vehicle Signals using Replicator Neural Networks. *6th escar USA 2018* (June 2018).

[28] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. 2017. Autoencoder-based Feature Learning for Cyber Security Applications. *2017 International Joint Conference on Neural Network (IJCNN)* (2017). https://doi.org/10.1109/IJCNN.2017.7966342

[29] Jiayan Zhang, Fei Li, Haoxi Zhang, Ruxiang Li, and Yalin Li. 2019. Intrusion detection system using deep learning for in-vehicle security. *Ad Hoc Networks* 95 (December 2019). https://doi.org/10.1016/j.adhoc.2019.101974