Comparative Study of Machine Learning Methods for In-Vehicle Intrusion Detection

Ivo Berger[1], Roland Rieke[2,3](✉), Maxim Kolomeets[3,4], Andrey Chechulin[3,4], and Igor Kotenko[3]

[1] TU Darmstadt, Darmstadt, Germany
[2] Fraunhofer Institute SIT, Darmstadt, Germany
roland.rieke@sit.fraunhofer.de
[3] ITMO University, St. Petersburg, Russia
[4] SPIIRAS, St. Petersburg, Russia

Abstract. An increasing amount of cyber-physical systems within modern cars, such as sensors, actuators, and their electronic control units are connected by in-vehicle networks and these in turn are connected to the evolving Internet of vehicles in order to provide "smart" features such as automatic driving assistance. The controller area network bus is commonly used to exchange data between different components of the vehicle, including safety critical systems as well as infotainment. As every connected controller broadcasts its data on this bus it is very susceptible to intrusion attacks which are enabled by the high interconnectivity and can be executed remotely using the Internet connection. This paper aims to evaluate relatively simple machine learning methods as well as deep learning methods and develop adaptations to the automotive domain in order to determine the validity of the observed data stream and identify potential security threats.

Keywords: Machine learning · Automotive security · Internet of vehicles · Predictive security analysis · System behavior analysis · Security monitoring · Intrusion detection · Controller area network security

1 Introduction

Each modern vehicle can be regarded as a system of interconnected cyber-physical systems. When vehicles are to take over tasks which are up to now the responsibility of the driver, an increasing automation and networking of these vehicles with each other and with the infrastructure is necessary. In particular, autonomous driving requires both a strong interconnectedness of vehicles and an opening to external information sources and services, which increases the potential attack surface. An indispensable assumption, however, is that the vehicle can not be controlled unauthorized externally. Thus, IT security and data protection are enabling factors for the newly emerging Internet of Vehicles (IoV). Practical experiments [17] have already demonstrated that an attacker can gain remote access to an in-vehicle Electronic Control Unit (ECU) and recent advisories such as [10] also mention that public exploits are available.

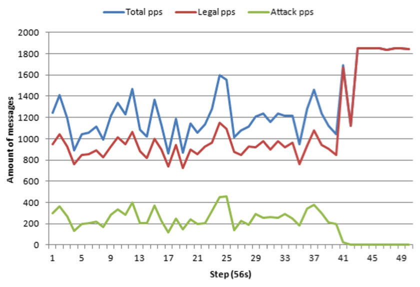
This work now examines various methods by which activities of an attacker already inside a vehicle could be detected. The Controller Area Network (CAN) bus is the standard solution for in-vehicle communication between the ECUs of the in-vehicle cyber-physical systems. Whilst offering high reliability the CAN bus lacks any kind of built-in security measures to prevent malicious interference by an outside party. This makes it easy for an attacker with access to one ECU to take over other critical cyber-physical systems within a vehicle. This could be done by broadcasting forged commands on the network, to gain the required knowledge, e.g. by running a fuzzing attack or to impair bus performance by performing a simple Denial of Service (DoS) attack.

In this work, we implemented and analyzed anomaly detection methods which can be applied to existing vehicle architectures as well as new designs by a software update or plug-in module as proposed in [19] without the adoption of new communication standards. The main contribution of this work is a comparative assessment of different Machine Learning (ML) methods with respect to usability for intrusion and anomaly detection within automotive CAN networks.

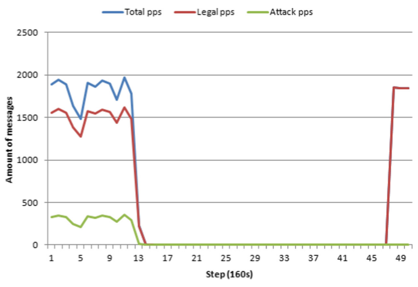
Section 2 introduces data sets from two different cars with and without attacks that have been used to evaluate the compared methods. Section 3 provides the background on four different ML technologies used in this work including training, validation and performance assessments. Section 4 presents the results of various experimental setups, while Sect. 5 discusses these outcomes and gives recommendations on feasible approaches for the domain of in-vehicle networks. Section 6 describes related work, and Sect. 7 concludes this paper.

2 Data Sets

On the CAN bus every attached device broadcasts messages. At the same time, devices listen for relevant information. We mapped the relevant data of CAN messages to the following tuple structure: $(time, ID, dlc, p_1, \ldots, p_8, type)$, where $time$ is the time when the message was received, ID comprises information about the type and the priority of the message, dlc (data length code) provides the number of bytes of valid payload data, $p1, \ldots, p8$ is the payload of 0–8 bytes, and $type$ marks the message (attack versus no attack). In cases where $dlc < 8$ we inserted dummy content in order to have a fixed tuple structure. For the experiments in this work we used five different data sets, namely, ZOE, $HCLR_{DoS}$, $HCLR_{Fuzzy}$, $HCLR_{RPM}$, and $HCLR_{Gear}$. The ZOE data set has been collected from a 10 min drive with a Renault Zoe electric car in an urban environment. It contains about 1 million messages and has been used before in [22] to perform behavior analysis by process mining. This data set contains no attack data. The other four data sets that we used have been published by the "Hacking and Countermeasures Research Labs" (HCRL) [7]. These data sets are fully labeled and demonstrate different attack types. The $HCLR_{DoS}$ data set contains DoS attacks. For this attack, every 0.3 ms a message with the ID "0000" is injected. Conversely, in the $HCLR_{Fuzzy}$ data set every 0.5 ms a completely random message is injected, whereas $HCLR_{RPM}$ and $HCLR_{Gear}$ contain spoofing attacks. In these data sets every millisecond a message with an ID related to gear respectively engine revolutions per minute is injected. The ID and message does not change. The linear charts of the ZOE, $HCLR_{DoS}$, and $HCLR_{Gear}$ data sets depicted in Fig. 1 show the composition of legal data and attacks. Figure 1a shows that in the $HCLR_{DoS}$ data set DoS messages decrease the number of legal packets, whereas Fig. 1b unveils a big gap in the traffic timeline of the $HCLR_{Gear}$ data set which could probably be a consequence of the spoofing attack. The linear charts of $HCLR_{Fuzzy}$ and $HCLR_{RPM}$ not shown here are similar to $HCLR_{Gear}$.



(a) $HCLR_{DoS}$: DoS attacks



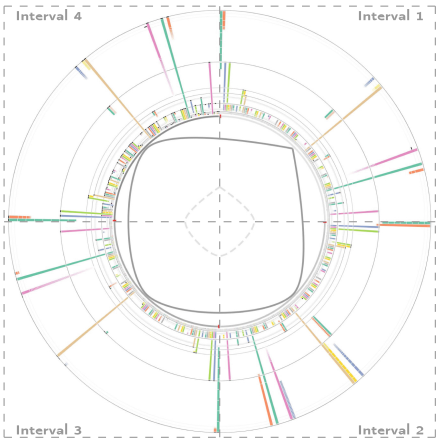
(b) $HCLR_{Gear}$: spoofing

Fig. 1. Attack influence analysis by linear graphs

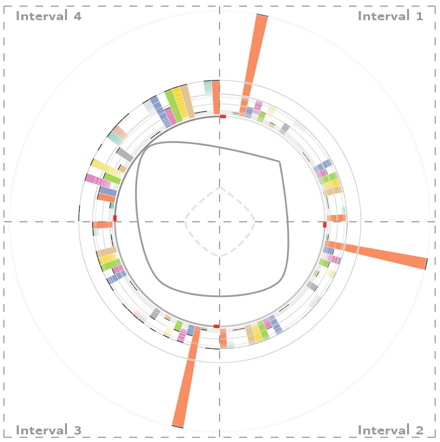
In order to get some more insight into the contents of the CAN data sets, we have visualized them by radial time-intervals using the method described in [12]. Figure 2 shows differences between ZOE and $HCLR_{DoS}$ traffic. The significantly higher number of bars in Fig. 2a is due to a higher number of different IDs in the ZOE traffic. In Fig. 2a the traffic without attacks has no outstanding bars whereas in Fig. 2b solid orange bars are outstanding in comparison to other bars. These bars represent DoS attacks which are decreasing the number of legal messages in the first three intervals during the attack. The radial visualizations of the other HCRL data sets with fuzzing and spoofing injections not shown here did not provide more insights.

3 Machine Learning Methods

This section introduces the compared algorithms including training, validation and performance assessments. The problem of detecting anomalies and attacks in CAN data differs from most ML applications as it does not present a clear



(a) *ZOE* data without attacks



(b) $HCLR_{DoS}$ attacks (orange bars)

Fig. 2. Attack influence visualization by radial time intervals, each summarizing one quarter of the period represented by the data set; traffic is separated in four radial time intervals that consist of bars; each ID is represented as a bar whose height equals the number of messages; bars consists of arcs which represent payload – the more messages with same payload the higher is the arc; so solid (or almost solid) arcs depict messages with few different payloads; transparent bars depict messages with big payload variety. (Color figure online)

classification problem. Anomalies and attacks are by nature unpredictable and thus it is not possible to obtain representative data to train a classifier. Thus, the approach taken in this work is to fit a model for the regular system behavior which can detect deviations from the norm. One aspect of system behavior is the range of values for the IDs and the payload of CAN messages, another aspect is the temporal behavior, resulting in a novelty detection problem and a time series analysis task. Some ECUs send messages periodically and thus are relatively easy to validate. Even though the collection of a representative set of anomalies and attacks is not possible it is beneficial to be able to detect and prevent known types of attacks such as DoS or fuzzing attacks. This presents a standard classification task where a model is trained to differentiate between regular and anomalous CAN communications. Another aspect is the practicality of possible solutions in real-world scenarios. For the deployment in vehicles the trained models need to be able to validate the incoming data steam in real-time, requiring efficient models and thus restricting their complexity.

The training of all models was done using the data sets introduced in Sect. 2 or subsets thereof in order to achieve reasonable training times. The data was split into training and validation sets to get a realistic performance estimate for each model. The validation of the Support Vector Machine (SVM) and standard neural network models use accuracy and confusion matrices. The Long Short Term Memory (LSTM) network was validated by predicting the next message ID based on a window of preceding messages and comparing it to the actual message. All experiments in this paper are written in `Python 3.6`. They utilize the `pandas` [16] library to read and transform the data and `scikit-learn` [21] and `keras` [6] with the `tensorflow` [1] back-end for the ML itself. For visualization `matplotlib` [9] and `seaborn` [28] were used. We now give a short introduction on how each of the methods operate.

3.1 One-Class Support Vector Machines

For anomaly detection One-Class Support Vector Machines (OCSVM) were used, which are an adaption of classic SVMs to be trained with one class with the goal of learning a border which encompasses the training data. OCVSMs are linear classifiers but can make use of nonlinear kernels to represent more complex data structures. They were used with success in [4,27] and this work used the hyper-parameters suggested in [4]. For OCSVMs the `sklearn.svm.OneClassSVM` and `numpy` [20] packages were used to filter out anomalous data from the training set. The `scikit-learn` [21] metrics `accuracy_score` and `confusion_matrix` were used to calculate scores from the predictions on the test set. To visualize the results the metrics for all data sets were saved and displayed using a `seaborn` [28] heat-map for the confusion matrices (Fig. 4) and a simple line graph for the accuracy per subset size (Fig. 3).

3.2 Support Vector Machines

SVMs are linear "max-margin" classifiers as they try to find a hyper-plane separating the data with the greatest possible margin to the closest entry of each class. They are linear but can use kernels to model nonlinear data structures whilst maintaining low hardware requirements when classifying. As they are very similar to OCSVMs the hyper-parameters from [4] were used here as well. Our implementation of the regular SVMs is almost identical to the OCSVM, except that `sklearn.svm.NuSVC` was used instead of `sklearn.svm.OneClassSVM`.

3.3 Neural Networks

Neural networks are the standard for deep-learning and can model very complex nonlinear relationships. The most basic version is the fully connected neural network. It utilizes an arbitrary number of layers with each layer supporting an arbitrary number of neurons. Data is propagated from the input to the output layer using weighted connections between the neurons of these layers, resulting in very complex structures and thus a large amount of trainable parameters and thus flexibility even for relatively small networks. They are usually trained using some form of gradient descent and are prone to overfitting due to their great flexibility. In consequence, the goal was to find the smallest possible network to achieve a good accuracy. Therefore, the anomalous class was set to 0 to work properly for binary classification and all features of the complete set were scaled using the `MinMaxScaler` from `scikit-learn` [21] before training.

The neural networks were implemented using `keras` [6] `Sequential` model from the `keras.models` package and `keras.layers.Dense` as its fully connected layers. From initial test it was found that one hidden layer and one epoch is sufficient for these data sets. For easier testing both layers used the same number of neurons. Binary Crossentropy, Adam and Accuracy was used as loss, optimizer and performance metric (see Listing 1.1).

Listing 1.1. Neural Network: Model and Training

```
def train(x: np.ndarray, y: np.ndarray, split, batch_size,
    neurons):
    # define and compile the model
    model = Sequential()
    model.add(Dense(neurons, activation='relu', input_shape=(x
        .shape[1],)))
    model.add(Dense(neurons))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam'
        , metrics=['accuracy'])
    # train model
    model.fit(x, y, epochs=1, batch_size=batch_size, shuffle=
        True, verbose=0)
    return model
```

Due to the good optimization of the `tensorflow` [1] back-end the model wasn't tested with different subset sizes but different neuron counts.

3.4 Long Short Term Memory Neural Networks

LSTMs are a derivation of recurrent neural networks for time sequence classification and prediction. They differ from standard neural networks by keeping previous states and thus are able to capture temporal relationships. LSTMs in particular keep a very recent as well as a long-standing state and are able to detect relationships between relatively distant events as well as directly consecutive ones opposed to simpler recurrent neural networks which only remember recent states. LSTMs are trained with time sequences, requiring to pre-process the data sets into message sequences with the window size as a configurable parameter. Furthermore, they are not trained to classify a message as anomalous or non-anomalous but to predict future messages or validate if new messages concur with the learned behavior.

Training a LSTM to predict or validate new messages requires the time series that is the training data to be transformed into a supervised learning problem. This is achieved by using message sequences of a certain window size with the message ID that followed it instead of single messages with a binary label. This enables the LSTM to learn the behavior and temporal relationships between data points. The original version of the code used was taken from [2] and has been adapted and simplified for this work. The next pre-processing step is the transformation of the time stamps to time deltas per ID, i.e. that the time column gives the seconds since the last occurrence of that ID instead of a comparatively arbitrary time stamp. This is done using `pandas` [16] split-apply-combine `pandas.DataFrame.grouby` and `apply` functions. To make the problem easier to solve and thus the training times shorter the predictions were limited to the message IDs instead of predicting/validating whole messages. To achieve good results the IDs had to be transformed from simple numbers to categories `keras` [6] and `tensorflow` [1] can properly handle. This process utilizes the `scikit-learn` [21] `LabelEncoder` and `keras` [6] `to_categorical` functions which first encode the IDs as labels and then transform them into a one-hot encoded `numpy` [20] array. The last step before applying the time series transformation is a `MinMaxScaler`. For usage with `keras` [6] the result of the time series transformation has to be reshaped into a three-dimensional array containing the original data point, the following ten steps and the corresponding label.

Listing 1.2. LSTM: Training

```
def train(x, y, batch_size, neurons=10):
    # define and compile the model
    model = Sequential()
    model.add(LSTM(neurons, input_shape=(x.shape[1], x.shape
        [2])))
    model.add(Dense(y.shape[1], activation='softmax'))
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam', metrics=['accuracy'])
    # train the model
    model.fit(x, y, epochs=5, batch_size=batch_size, shuffle=
        False, verbose=0)
    return model
```

Listing 1.2 shows the actual training process which is quite similar to that of a regular neural network. Differences are in the used loss function (categorical vs. binary crossentropy) and that the data isn't shuffled for LSTMs as that would destroy any temporal relationships in the data.

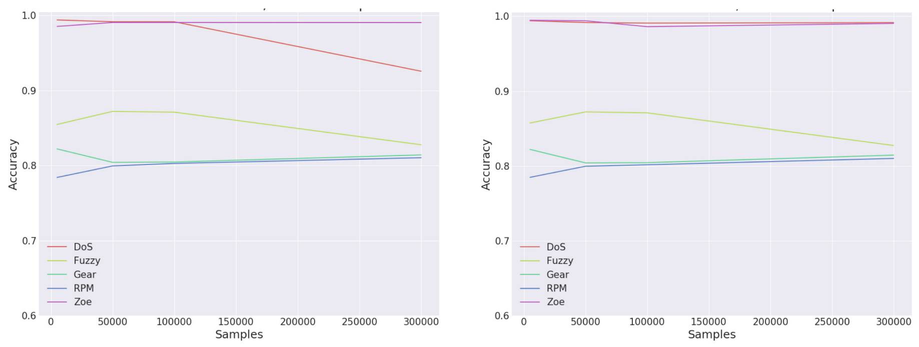
The scoring is essentially the same as for the neural networks with the exception that the predictions are given as probabilities per category which have to be transformed to the one-hot encoding in the test set by setting the category with the highest probability to one and all others to zero.

4 Results

This section presents the results of all methods mentioned in Sect. 3. It will introduce the metrics used and discuss the performance of each method with regard to the nature of the data sets and validation methods.

4.1 One-Class Support Vector Machines

The OCSVMs were validated with subsets of the data sets described in Sect. 2 of different sizes between 5,000 and 300,000 messages using two different approaches: The ZOE data set consists of non-anomalous data only, resulting in a validation error that is equivalent to the false negative rate, i.e. it was tested which percentage of the validation data was misclassified as anomalous. The training portion of the HCRL data set was cleaned of anomalous data and the trained model tested with both classes using the accuracy for performance assessment as well as confusion matrices where appropriate with the true label on the y-axis and the predicted label on the x-axis.



(a) OCSVM with time-stamps (b) OCSVM without time-stamps

Fig. 3. OCSVM results

The high accuracy on the $HCLR_{DoS}$ data set is expected as the anomalous entries are easily detectable for any subset of the data. Figure 3a, however, shows a slight worsening with increasing subset sizes. As the model was trained with all fields, including the timestamps, the result suggest that the OCSVM is detecting messages with a timestamp outside of the learned boundaries as anomalous. Excluding the timestamps from training and testing confirms this as it results in almost perfect accuracy for all subset sizes (see Fig. 3b). As seen in Fig. 3, the OCSVMs accuracy on the ZOE data is almost perfect. Considering that this data set only consists of regular data the good result comes from a too great similarity of the training and test data sets. The result thus lacks informative value about the effectiveness of OCSVMs in anomaly detection. The performance on the $HCLR_{Fuzzy}$ data set is pretty high on a subset of 50,000 messages and declines with increasing message count. This can be explained with the randomized generation of the anomalous data in this set. With increasing subsets the amount of completely random data increases as well, which in turn increases the amount of anomalous data that looks like regular data by chance, resulting in deterioration of the results. This is confirmed by the confusion matrix in Fig. 4. The model predicts the regular class almost exclusively and the performance changes are a result of changes in the test set rather than changes in the model.

The accuracy for the spoofing data sets first shows a slight dip for 50,000 samples and then recovers with larger subsets. The confusion matrices in Fig. 4 show that this is purely due to changes in the test data set as the model only predicts the regular class. The anomalous data in these sets only differ in the timestamp. There is nothing in the ID or payload of these entries that makes them distinct from other non-anomalous messages, making it impossible for a SVM to separate between classes. The confusion matrices (Fig. 4) show heavy bias towards the regular class for all data sets except $HCLR_{DoS}$ and thus that all changes in the performance of the models are due to changes in the composition of the test data and not an improvements of the models itself. We also observed that the removal of the timestamps only has a noticeable effect on the $HCLR_{DoS}$ results. This can be explained with the mentioned bias as well as any potential changes are shadowed by the almost exclusive prediction of the regular class for all other data sets.

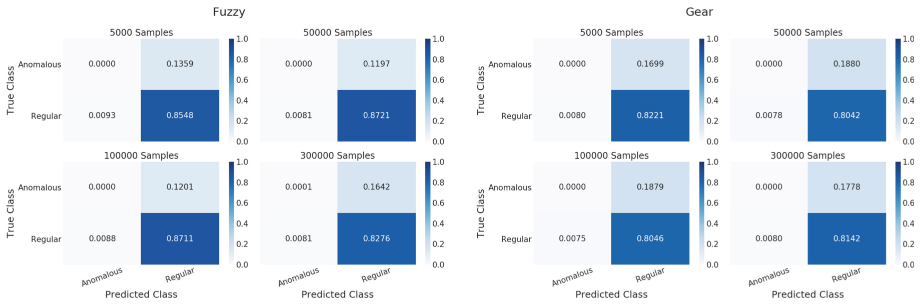
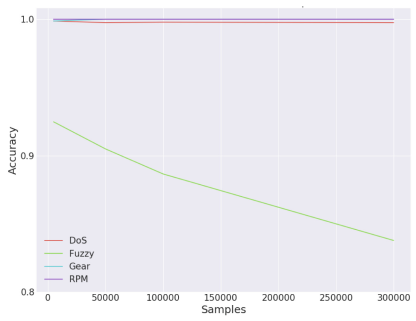


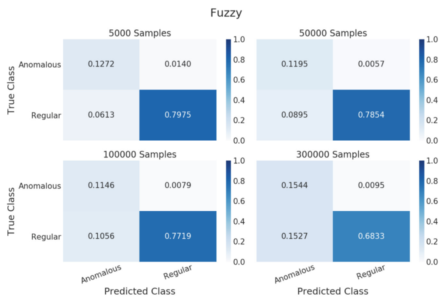
Fig. 4. OCSVM confusion matrices

4.2 Support Vector Machines

SVMs are very similar to OCSVMs as described in Sect. 3.2, hence the method of validation is as described for OCSVMs in Sect. 4.1 with the important difference that SVMs are classifiers and thus were trained with regular and anomalous entries. As SVMs don't support training on data sets with only one class the ZOE data set is excluded from these results. Furthermore, the SVM implementation in `scikit-learn` [21] is not multithreaded and had very long training times when training with timestamps. For these reasons only results without timestamps are presented. The results in Fig. 5a show that knowledge about the anomalous entries significantly improves accuracy on the impersonation data sets, achieving perfect classification on all but the $HCLR_{Fuzzy}$ data set even with the smallest subset. Looking at the very obvious distinction between regular and anomalous data points in these sets (see Sect. 2) the good performance is as expected as the continuously worsening performance on the $HCLR_{Fuzzy}$ data set.



(a) SVM without timestamps



(b) SVM confusion matrix

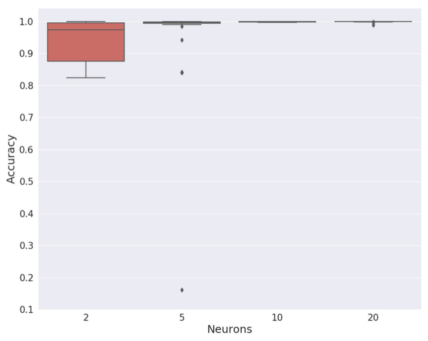
Fig. 5. Results from support vector machines

Considering that all anomalous entries for this set are random and the resulting possibility of entries falling within the value range of regular entries there are no support vectors that can describe the difference comprehensively. Thus, the accuracy declines with increasing subset size as more and more false negatives are introduced which is shown in the corresponding confusion matrix in Fig. 5b. Despite the clear decline the classification is still surprisingly good considering the simple linear kernel and the random nature of the anomalous entries.

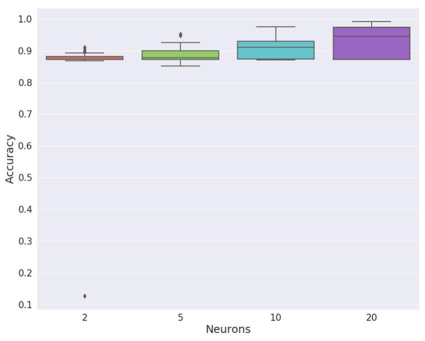
4.3 Neural Networks

The validation of the neural network was done with a standard train/test split of the original data and the performance of networks with different amounts of neurons compared in order to find the simplest possible network to solve the problem. As neural networks are very flexible and even small ones already have a good amount of variables this paper examines a network with only one hidden layer for neurons counts of 2, 5, 10 and 20, going from extremely simple to fairly complex models. The deep learning results are presented as confidence intervals which are obtained using the bootstrap method with 50 iterations and a sample size of 800.000 per iteration.

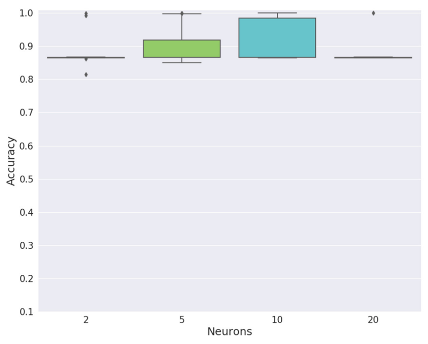
The very good to perfect results in Fig. 6 for all used data sets show the great flexibility of neural networks. For all data sets the intervals reach 99% even for the simplest network. The explanation for the good performance can be found in the very simple structure of the anomalous entries for the $HCLR_{DoS}$, $HCLR_{Gear}$ and $HCLR_{RPM}$ data sets: in each case there is one exact value combination that has to be detected. Whilst the OCSVMs had problems (cf. Sect. 4.1) with the $HCLR_{Gear}$ and $HCLR_{RPM}$ data sets as their anomalous entries values are within range of regular ones a neural network can learn to single out this exact combination as being anomalous and thus achieve the seen results. The intervals and the outliers in particular show that the networks performance depends greatly on the samples used.



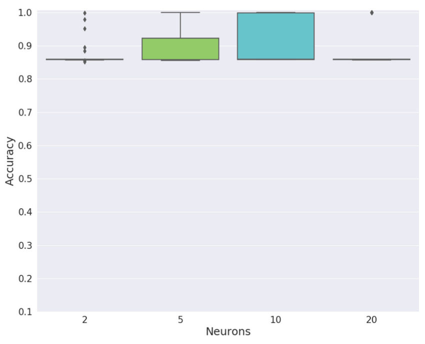
(a) $HCLR_{DoS}$ dataset



(b) $HCLR_{Fuzzy}$ dataset



(c) $HCLR_{Gear}$ dataset



(d) $HCLR_{RPM}$ dataset

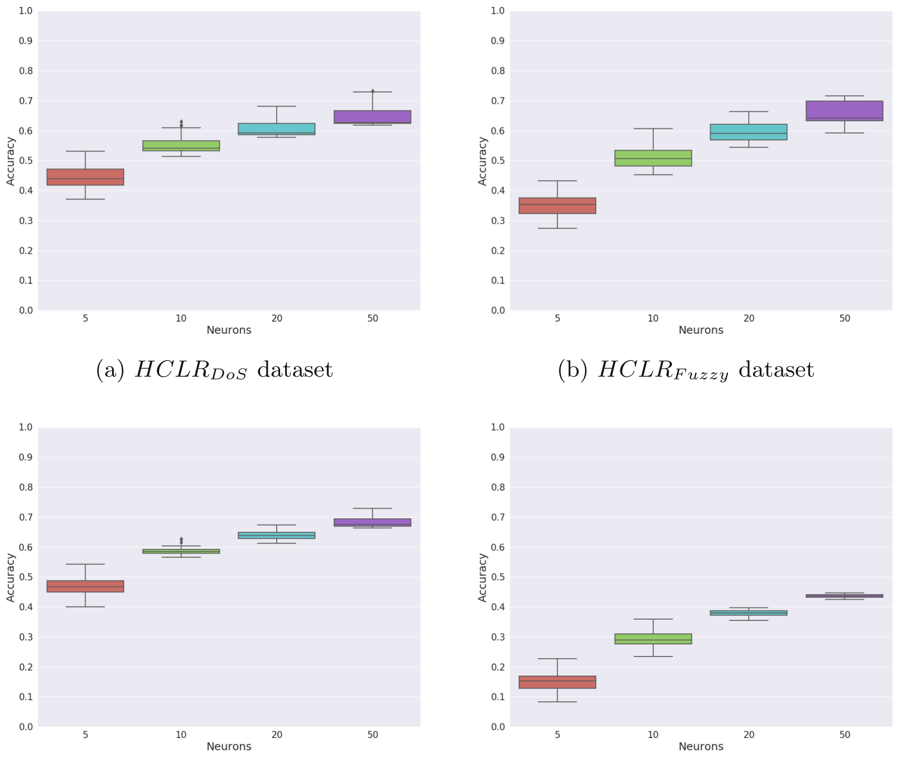
Fig. 6. Neural network confidence intervals

The case where the very good results are not as obvious is the $HCLR_{Fuzzy}$ data set, as it has randomly generated anomalous entries which can not be as easily differentiated from the regular ones, which is supported by the need of at least 5 neurons to surpass the 99% accuracy threshold. In this case the great flexibility of the network enables it to learn which value combinations, for example in which range an ECU's payloads are, are valid and thus to distinguish them from the random entries very well. Another observation is that the intervals are generally larger then for the other data sets. As we didn't use stratified splits this suggests that a certain minimum of both regular and random data is required for the network to learn a good model.

The outliers seen in most results illustrate the general importance of having the right data. Out results show that with a proper amount of data to train the model neural networks are capable of detecting complex anomalies reliably.

4.4 Long Short Term Memory Neural Networks

To validate the LSTMs the whole data set was transformed to message sequences as explained in Sect. 3.4 from which the first 80% were used to train bootstrap networks while the remaining 20% validated the message ID predictions the LSTM made. The results graphs (Fig. 7) show the networks performance for a window size of 10 and each individual graph plots confidence intervals on accuracy against neuron count of the LSTM layer. The LSTMs were trained and tested without anomalous entries in order to measure their capability to model the regular data stream.



(a) $HCLR_{DoS}$ dataset

(b) $HCLR_{Fuzzy}$ dataset

(c) $HCLR_{Gear}$ dataset

(d) ZOE dataset

Fig. 7. LSTM confidence intervals

For the $HCLR_{DoS}$ and the spoofing data sets the accuracy is quite good considering the relatively small samples and networks used as well as the complexity of the problem. On all of these data sets the network achieves an accuracy up to 50% with only 5 neurons and up to 60% on the DoS data set with 20 neurons. The very similar and good performance is due to equal and relatively low number of regular message IDs in the sets at only 26. Performance on the $HCLR_{Fuzzy}$

data set is noticeably but not significantly worse for the simplest LSTM and very similar to the other HCRL sets for 10 or more neurons. The lower accuracy on the LSTM with 5 neurons is due to the higher number of IDs at 38. The ZOE data set with its considerably higher number of message IDs at 110 performs by far the worst at a maximum accuracy just above 40%. Considering that random performance for 110 IDs is at only 0.9% an accuracy over 40% is more than 42 times better than random and thus still quite good. As the data set is from a different source than the others (see Sect. 2) it suggest that the Renault Zoe has more complex internals than the vehicles used to acquire the HCRL data sets.

For all tested data sets there is a clear correlation between message ID frequency and the LSTMs categorical accuracy on that ID. The periodic occurrence of these highly frequent IDs and the possible triggering of reactions to certain periodic IDs explains the good predictions. Consequently, the worst performance can be observed on infrequently occurring IDs, especially as they contain IDs triggered by outside events and thus are simply unpredictable.

5 Discussion

The OCSVM results show very clearly that this comparatively simple method of novelty detection only works for very basic anomaly detection. As OCSVMs try to find a boundary which contains all or most of the seen data it can only detect anomalies which differ significantly from the normal data in terms of raw field values. Considering the observed heavy bias towards the regular class it can still be useful: if it does classify a message as being anomalous there is a high chance that it's correct. Theissler [27] has also conducted a more sophisticated approach. He used Support Vector Data Descriptors, a derivation of OCSVMs, trained with message sequences instead of individual messages with better results and very low to no false anomalies. The low false negative rate and the ability to train them without anomalous data is a quite important aspect. Combined with their relatively simple and thus fast classification makes them a good practical choice for real-time classification in a vehicle.

Regular SVMs share the good results and speed of OCSVMs but require anomalous data for training. Practical use would only come from the classification of attacks which cannot be easily specified such as the fuzzing attack. For any simple specification violating these specifications could be used directly to verify the data stream without the need to train a model. Because of that and the significant decline on performance of SVMs on the $HCLR_{Fuzzy}$ data set the real-world applicability of SVMs for the here evaluated use case is very limited.

Neural networks share the major drawback of needing anomalous training data to be of any use but show impressive performances on all tested data sets. The results suggest that they are able to learn the ECU behavior very well and thus detect diverging data points. Even on the randomly generated fuzzing attack their accuracy was close to perfect. Considering that only a very simple network with one hidden layer and two neurons per layer is needed to achieve this performance and thus classifying very fast, even without a graphics processing unit, they could be a powerful tool to simplify automated specification checking. In practice, they can be trained with regular and randomly generated data and automatically derive specifications for non-anomalous data. This would require additional testing on more diverse data sets in order to generalize this approach.

LSTMs are by far the most complex and thus computationally intensive of the here presented learning algorithms. Our results show their ability to learn the behavior at least partially and they have been applied to the more difficult problem of prediction complete messages with success in [4]. The performance shows diminishing returns when using more than 20 neurons and further simplification might be possible by excluding messages triggered by external events. This opens the possibility of improving the network's performance while reducing its complexity as in the present experiments the accuracy is clearly linked to the number of message IDs. Considering all of the above points LSTMs present a practical and potentially the most powerful approach of anomaly detection out of the methods analyzed in this work.

It is interesting to note that none of the ML methods indicated the big gaps in some of the data sets found by the visualization technique (cf. Fig. 1b).

6 Related Work

A collection of possible intrusion points together with proposals for countermeasures such as cryptography, anomaly detection and ensuring software integrity by separating critical systems are presented in [25,30]. Whilst the proposed measures should prove effective most of them require hardware changes, conflicting with backwards-compatibility. CAN intrusion detection methods can be grouped into four categories: (1) Work on detection of ECU impersonating attacks such as [3,5] in most cases uses some kind of physical fingerprinting by voltage or timing analysis with specific hardware. This work seeks to mitigate the general problems of missing authenticity measures in CAN bus design and thus is complementary to the work presented in this paper. (2) Work on detection of specification violations assumes that the specification of normal behavior is available and thus there is an advantage that no alerts based on false positives will be generated. The specification based intrusion detection can use specific checks, e.g. for formality, protocol and data range [13], a specific frequency sensor [8], a set of network based detection sensors [18], or specifications of the state machines [24]. (3) Work on detection of message insertions can be based on various technologies like analysis of time intervals of messages [23] or LSTM [29]. (4) Work on detection of sequence context anomalies comprises process mining [22], hidden Markov models [14,19], OCSVM [27], neural networks [11], and detection of anomalous patterns in a transition matrix [15]. In most cases the authors of the above mentioned papers described experiments with one specific method. However, because the authors use different data sets for their experiments the results of their work are not directly comparable.

Therefore, we compared different ML algorithms with the same data sets. The closest work to our paper is [4] and [26] which also provides results on method comparison. OCSVM, Self Organizing Maps, and LSTM are used in [4] and LSTM, Gated Recurrent Units (GRU), as well as Markov models are used in [26]. However, in [4] only one small training set of 150,000 packets from the Michigan Solar Car Team was used, and [26] is more focused on attack generation.

7 Conclusion

In conclusion this study has shown the potential of ML for anomaly detection in CAN bus data. Even simple linear methods like OCSVMs can yield good results when detecting simple attacks while more complex neural networks are capable to learn "normal" message content from CAN data. The most sophisticated models, namely LSTMs, are able to learn ECU behavior adequately. Even the deep learning approaches can be kept relatively simple meaning all analyzed methods should be able to detect anomalies in real-time even on low-end hardware. Combined with the existing research ML promises to be an effective way to increase vehicle security. The injected attacks are relatively trivial in nature requiring additional research with more diverse and complex intrusions as well as the comparison of methods used in other research to the here present ones. Focused tests, potentially in cooperation with vehicle manufacturers, have to provide further insights in the prediction capabilities of LSTMs. Furthermore, real-world tests on practical hardware are needed to confirm that real-time detection is indeed possible. Based on reliable anomaly detection, appropriate reactions such as simple notifications or automated prevention measures need to be investigated.

Acknowledgements. This research is partially supported by the German Federal Ministry of Education and Research in the context of the project secUnity (ID 16KIS0398) and by the Government of Russian Federation (Grant 08-08), by the budget (project No. AAAA-A16-116033110102-5).

References

1. Abadi, M., Agarwal, A., Barham, P., et al.: TensorFlow: large-scale machine learning on heterogeneous distributed systems. In: 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, pp. 265–284 (2016)
2. Brownlee, J.: How to convert a time series to a supervised learning problem in Python. https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/ (2018). Accessed 28 June 2018
3. Cho, K., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, 10–12 August 2016, Austin, TX, USA, pp. 911–927. USENIX Association (2016)
4. Chockalingam, V., Larson, I., Lin, D., Nofzinger, S.: Detecting attacks on the CAN protocol with machine learning (2016)
5. Choi, W., Joo, K., Jo, H.J., Park, M.C., Lee, D.H.: Voltageids: low-level communication characteristics for automotive intrusion detection system. IEEE Trans. Inf. Forensics Secur. **13**(8), 2114–2129 (2018)

6. Chollet, F., et al.: Keras. https://github.com/keras-team/keras (2015)
7. Hacking and Countermeasure Research Lab (HCRL): Car-hacking dataset for the intrusion detection. http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset (2018). Accessed 28 June 2018
8. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks - practical examples and selected short-term countermeasures. Reliab. Eng. Syst. Saf. **96**, 11–25 (2011)
9. Hunter, J.D.: Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**(3), 99–104 (2007)
10. ICS-CERT: Advisory (ICSA-17-208-01). https://ics-cert.us-cert.gov/advisories/ICSA-17-208-01, July 2017. Accessed 17 Sept 2018
11. Kang, M.J., Kang, J.W.: A novel intrusion detection method using deep neural network for in-vehicle network security. In: 2016 IEEE 83rd Vehicular Technology Conference (VTC Spring) (2016)
12. Kolomeets, M., Chechulin, A., Kotenko, I.: Visual analysis of CAN bus traffic injection using radial bar charts. In: Proceedings of the 1st IEEE International Conference on Industrial Cyber-Physical Systems, ICPS-2018, Saint-Petersburg, Russia, pp. 841–846. IEEE (2018)
13. Larson, U.E., Nilsson, D.K., Jonsson, E.: An approach to specification-based attack detection for in-vehicle networks. In: 2008 IEEE on Intelligent Vehicles Symposium, pp. 220–225, June 2008
14. Levi, M., Allouche, Y., Kontorovich, A.: Advanced analytics for connected cars cyber security. CoRR abs/1711.01939 (2017)
15. Marchetti, M., Stabili, D.: Anomaly detection of CAN bus messages through analysis of ID sequences. In: 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 1577–1583, June 2017
16. McKinney, W.: Data structures for statistical computing in Python. In: Proceedings of the 9th Python in Science Conference 1697900(Scipy), pp. 51–56 (2010)
17. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle. Technical report, IOActive Labs, August 2015
18. Müter, M., Asaj, N.: Entropy-based anomaly detection for in-vehicle networks. In: 2011 IEEE Intelligent Vehicles Symposium (IV), pp. 1110–1115, June 2011
19. Narayanan, S.N., Mittal, S., Joshi, A.: OBD SecureAlert: an anomaly detection system for vehicles. In: IEEE Workshop on Smart Service Systems, SmartSys 2016, May 2016
20. Oliphant, T.E.: Guide to NumPy. Methods **1**, 378 (2010)
21. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**(Oct), 2825–2830 (2012)
22. Rieke, R., Seidemann, M., Talla, E.K., Zelle, D., Seeger, B.: Behavior analysis for safety and security in automotive systems. In: 2017 25nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 381–385. IEEE Computer Society, March 2017
23. Song, H., Kim, H., Kim, H.: Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network, pp. 63–68. IEEE Computer Society, March 2016
24. Studnia, I., Alata, E., Nicomette, V., Kaâniche, M., Laarouchi, Y.: A language-based intrusion detection approach for automotive embedded networks. In: The 21st IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2015, November 2014

25. Studnia, I., Nicomette, V., Alata, E., Deswarte, Y., Kaâniche, M., Laarouchi, Y.: Security of embedded automotive networks: state of the art and a research proposal. In: ROY, M. (ed.) SAFECOMP 2013 - Workshop CARS of the 32nd International Conference on Computer Safety, Reliability and Security, September 2013
26. Taylor, A., Leblanc, S.P., Japkowicz, N.: Probing the limits of anomaly detectors for automobiles with a cyber attack framework. IEEE Intell. Syst. **PP**(99), 1 (2018)
27. Theissler, A.: Anomaly detection in recordings from in-vehicle networks. In: Proceedings of Big Data Applications and Principles First International Workshop, BIGDAP 2014, 11–12 September 2014, Madrid, Spain (2014)
28. Waskom, M., Meyer, K., Hobson, P., Halchenko, Y., et al.: Seaborn: v0.5.0, November 2014
29. Wei, Z., Yang, Y., Rehana, Y., Wu, Y., Weng, J., Deng, R.H.: IoVShield: an efficient vehicular intrusion detection system for self-driving (short paper). In: Liu, J.K., Samarati, P. (eds.) ISPEC 2017. LNCS, vol. 10701, pp. 638–647. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72359-4_39
30. Wolf, M., Weimerskirch, A., Paar, C.: Security in automotive bus systems. In: Proceedings of the Workshop on Embedded Security in Cars (July), pp. 1–13 (2004)