Examining the Use of Neural Networks for Intrusion Detection in Controller Area Networks

Camil Jichici, Bogdan Groza, and Pal-Stefan Murvay

Faculty of Automatics and Computers,
Politehnica University of Timisoara, Romania
Email: jichicicamil93@gmail.com, bogdan.groza@aut.upt.ro, pal-stefan.murvay@aut.upt.ro

Abstract. In the light of the recently reported attacks, in-vehicle security has become a major concern. Intrusion detection systems, common in computer networks, have been recently proposed for the in-vehicle buses as well. In this work we examine the performance of neural networks in detecting intrusions on the CAN bus. For the experiments we use a CAN trace that is extracted from a CANoe simulation for the commercial vehicle bus J1939 as well as a publicly available CAN dataset. Our results show good performance in detecting both replay and injection attacks, the former being harder to detect to their obvious similarity with the regular CAN frames. Nonetheless we discuss possibilities for integrating such detection mechanisms on automotive-grade embedded devices. The experimental results show that embedding the neural-network based intrusion detection mechanism on automotive-grade controllers is quite challenging due to large memory requirements and computational time. This suggests that dedicated hardware may be required for deploying such solutions in real-world vehicles.

Keywords: CAN bus · vehicle security · intrusion detection · neural networks

1 Introduction and Related Work

Contemporary vehicles are easy targets in front of well motivated adversaries, this was well proved by recent research. The main cause for this vulnerability is the inherent lack of security on Controller Area Network (CAN) which was designed decades ago without adversaries in mind.

Recently, a strong body of research has been focusing on the use of cryptographic authentication for the CAN bus. This includes the use of regular cryptographic message authentication codes [5], [6], [19], [4] but goes as far as using the physical layer to discard forged frames [10] or hide authentication bits in regular frames [24]. Attention is also payed to efficient allocation of signals in each frame [12]. Other works account for the physical layer in order to hide authentication bits within regular CAN bits or distinguish between nodes based on physical signal characteristics [15], [3], [2]. Other lines of work have been focusing in using characteristics of the physical layer to securely share a cryptographic key [7], [14].

The design of intrusion detection mechanisms for CAN is an even more recent preoccupation. The use of entropy characteristics of the frames was explored by [16] and

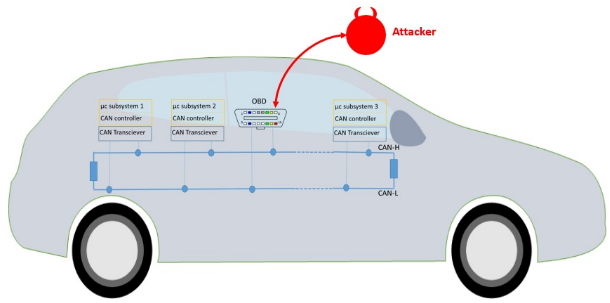

Fig. 1. Targeted scenario

[13]. In [11], the authors propose an Intrusion Detection System (IDS) based on remote frames. The idea proposed by the authors is to send a remote frame and then stores the offset and time intervals between the remote frame and the response data frame. The experimental results demonstrated that offsets exists between normal frames and attack frames [11]. In [20] the authors show that the timestamps of messages can be used to detect attacks. Specialized detection sensors are used in [17]. Hardware measurements such as clock-shews [1] voltage thresholds or signal characteristics [3], [15] may also set the stage for intrusion detection.

In [8] and [9] the authors propose an IDS based on deep neural networks. They use as input only the data-field of the CAN packet to detect the intrusion, which may not be sufficient to detect replay attacks (since replayed CAN frames are identical to genuine frames). In our proposal we use the timestamp of the CAN packet to circumvent this problem. A recurrent neural network is presented in [22]. The authors use two networks, one which is trained with the data packet and one which is trained with the CAN bus ID. A Markov Model is used in [18]. Finite-state automatons are used in [21] and multivariate time series in [23].

2 Background and Tools

In this section we discuss the adversary model then proceed to some background on neural networks and the tools that we use for evaluation.

2.1 Adversary Model

The setup that we address is depicted in Figure 1. Here an adversary connects over the On-Board Diagnostics (OBD), via some compromised device and injects frames on the bus. Our work considers two types of attacks which are modelled on the CANoe trace:

1. Injection of random data, an attack in which a the malicious CAN frame is injected with the same identifier as a genuine frame but has data field that is randomly generated. The timestamp of the injected frame is a random time value between the

timestamps of two regular CAN frames between which the injection takes place (we consider that this mimics a real-world attack scenario). The injection takes place at a random location in the regular trace. When analyzing traces for an attack on individual IDs, the injection occurs at random between two regular frames of the same ID. When analyzing the full trace, we consider that injection is again random, no later than few dozen frames from the genuine frame.

2. Replay of regular CAN frames, an attack in which the adversarial CAN frame has the same identifier and content. The frame has a random timestamp since it also mimics a real-world attack in which we assume that an adversary is replaying frames at random points in time. The injection is done at random locations, identical to the case of injection with random data.

Our adversary model is consistent with models from related work [11] where the authors define replay attacks and fuzzy attacks which are identical to our injections of random data. Additionally, in [11] the authors consider DoS attack by injecting the highest priority ID on the bus, i.e., 0x000h, but this attack is easy to detect since the ID does not occur in normal runs. Consequently we neglect this behaviours since detection would be trivial.

2.2 Neural Network Tools and Architecture

In our experiments we use both the Neural Network Toolbox made available by Matlab which is industry standard as well as an independent C++ implementation. The reason for using both these implementation is that the neural network toolset from Matlab is widely recognized for its performance and functionalities, however, for a microcontroller implementation an open-source C++ code is preferred. For this reason, the main results from the experimental section are done with Matlab but we also verified that similar results are obtain with the independent C++ code[1] which we also benchmark in the experimental section.

The Matlab toolbox provides plenty of algorithms and methods to solve classification problems. In particular, we used the *trainscg*, i.e., scaled conjugate gradient backpropagation, algorithm for training. The weights and bias values are updated by the algorithm using the scaled conjugate gradient method. The transfer function used between our layers is the hyperbolic tangent sigmoid transfer function *tansig* which returns a value in the range $[-1, 1]$. This function is recommended by the Matlab documentation as it offers good trade-offs where speed is important. The C++ implementation also uses the back-propagation algorithm and the a sigmoid function for activation.

For training and validation of the results, the dataset which we used is split in three parts:

1. training data (TD), which is the data used for training the network, i.e., updating the weights of the network,
2. validation data (VD), which is the data used to test how the neural network works with new data (this input is run at the end of each epoch),

[1] https://takinginitiative.wordpress.com/2008/04/23/basic-neural-network-tutorial-c-implementation-and-source-code/

3. test data (TsD) is used after the training phase (when the stop conditions are reached) and the correctness results are based on the output for this type of data.

We remark that in the C++ implementation the validation data is referred as generalization data and the test data is referred as validation data. This is simply a naming convention since the role of the sets is obvious from the implementation.

An epoch sums over the running time of the entire training data set plus the generalization set. The neural network runs continuously until the stop conditions are met. We now discuss this conditions in Matlab. Since the training was in the order of minutes or less and the training stage is an off-line process which does not require a real-time response, we leave the *maximum amount of time*, one of the stop conditions for the training, set to ∞ which is the default. This allows stopping on one of the following conditions: i) the *maximum epoch reached* (set to 1000), ii) the *performance goal* is reached (set to 0), iii) the performance gradient falls below the minimum gradient (set to 1e-06) or iv) the validation performance has consecutively increased more than 6 times.

The independent C++ implementation had similar stopping conditions. For example, if the training set accuracy and generalization set accuracy is less than the desired accuracy, the network will run until the maximum epochs is reached. Otherwise, the *training set accuracy* (TSA) can be used as stop condition and this represents the number of CAN packets that are correctly classified, i.e.,

$$TSA = 100\left(1 - \frac{NIC}{NT}\right)$$

where *NIC* is the number of incorrect results and *NT* the total number of CAN frames from the training set. Finally, another stop condition is the *generalization set accuracy* (GSA) which is identical to TSA, but computed on the generalization dataset.

The structure of the neural network consists in an input layer, a hidden layer and the output layer. These are shown in Figure 2. The neural network input accounts for the data field, the identifier (29 bits, for extended CAN frame) and also the delay between consecutive timestamps of the same ID (Δt). Mathematically the data input vector $I \in \{0,1\}^{105}$ is described as: $I = \left\{b_0, b_1, b_2, ...b_{104}\right\}$ where bits $b_0 ... b_{11}$ represent the delay Δ_t, bits $b_{12} ... b_{75}$ the 64-bit data field and bits $b_{76} ... b_{104}$ the 29-bit identifier. The output $O \in \{0,1\}$ is given as: $O = \left\{b_0\right\}$ where

$$b_0 = \begin{cases} 1, \text{represents an attack frame} \\ 0, \text{represents a regular frame} \end{cases}$$

3 Experimental Results

In this section we first discuss metrics for performance evaluation then we proceed to concrete results on detection accuracy. Finally, we present results on computational performance on automotive-grade microcontrollers.

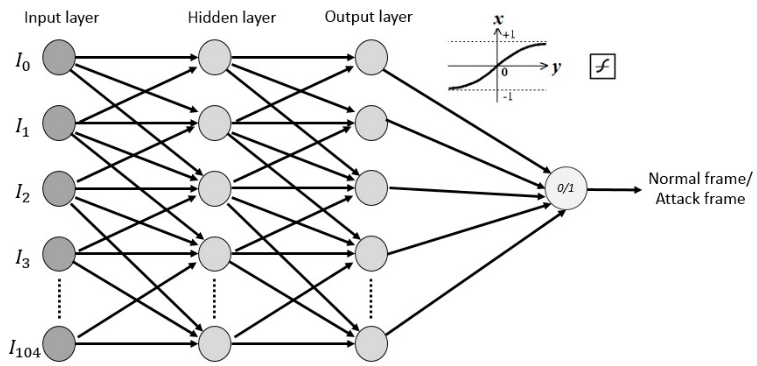

Fig. 2. Structure of the neural network

3.1 Metrics for Evaluating the Detection Rate

We evaluate success rate of the detection mechanism based on the usual four parameters: true positives (TP), i.e., the number of frames that are correctly reported as intrusions, true negatives (TN), i.e., the number of frames that are correctly reported as genuine, false positives (FP), i.e., the number of frames that are incorrectly reported as intrusions, and false negative (FN), i.e., the number of frames that are incorrectly reported as genuine. Based on these, we calculate the following: the sensitivity or the true positive rate $TPR = TP/(TP+FN)$, the false negative rate $FNR = FN/(TP+FN)$, the specificity or the true negative rate $TNR = TN/(TN + FP)$ and the fall-out or the false positive rate $FPR = FP/(FP + TN)$.

In addition to these, the validation set MSE (Mean Squared Error) is also used. This represents the average of the sum of the squared errors for each pattern in the validation set:

$$MSE = \frac{\sum_{i=0}^{n}(DesiredValue - ActualValue)^2}{n},$$

where n is the number of CAN frames from validation set.

3.2 Results on Detection Accuracy

We use a CANoe trace from a J1939 simulation. We compute the detection rates on portions of traces containing all packets with a particular CAN identifier. We consider three rates of injection: 5%, 10% and 20% of attack frames in the trace. For each injection rate we have five cases of splitting the data for training (TD), validation (VD) and testing (TsD). These are shown in Table 1.

To begin with, we have conducted experiments on the real-world CAN bus data made public by the authors in [11]. The dataset includes traces for both regular network traffic and for the case of adversarial interventions. Unfortunately, for the later

Table 1. The five cases in which we split the dataset

I	60% TD	20% VD	20% TsD
II	40% TD	20% VD	40% TsD
III	20% TD	20% VD	60% TsD
IV	10% TD	10% VD	80% TsD
V	5% TD	5% VD	90% TsD

case, the traces do not have a mark to separate between injected frames and genuine frames. Consequently, we have only considered traces for fuzzy attacks and assumed that randomized frames are injections while the rest of the frames are genuine. The results, presented in Table 2, were excellent at almost 100% detection accuracy, but this is mostly due to the simplicity of the attack trace, e.g., the attack is carried on a single ID with low entropy. In what follows we complicate these experiments to test the limits of the neural network based detection.

Table 2. Experimental results - efficiency rates regarding fuzzy attacks dataset

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
N/A	I	1	7.07e-07	9.6381e-07	24	2454 99.92%	5586 100%	2 0.08%	0 0%
N/A	II	1	5.21e-07	7.5294e-04	31	5277 99.96%	10806 100%	2 0.04%	0 0%
N/A	III	1	8.22e-07	7.4193e-07	28	7860 99.97%	16266 100%	2 0.03%	0 0%
N/A	IV	1	6.08e-07	1.4023e-06	27	10462 99.98%	21707 100%	2 0.02%	0 0%
N/A	V	1	6.49e-07	1.4732e-06	26	11827 99.98%	24363 100%	2 0.02%	0 0%

Results on single ID with low vs. high entropy. We now discuss detection rate on monitoring a single ID from our CANoe J1939 trace. Monitoring for a single ID is relevant as a baseline since it is expected that extending detection to all the IDs from the trace will require a larger neural network which in turn requires more computational power and storage space. Also, we note that in the trace that we use some of the IDs carry entropy that is close to 0, i.e., almost constant data-fields, while other frames have 12-13 bits of entropy. We present results for both these situations in Tables 3 and 4 for injections with random data. For replay attacks the results are available in Tables 5 and 6. Extended results on this dataset are deferred for the Appendix of this work in Tables 15, 16, 17 and 18. In case of injections with random data, for the higher entropy ID there is a slight increase in the false-negative rate, but detection rate is still close to 100% percents. For replay attacks, somewhat poorer results were obtained for the low-entropy ID. But the true negative rate stays at 100% while the true positive rate may occasionally drop to around 70%.

Table 3. Result on injections with random data over a low-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	0	8.79e-07	1.0677e-07	29	7519 100%	1518 100%	0 0%	0 0%
20%	II	0	9.23e-07	5.7362e-07	27	15079 100%	2995 100%	0 0%	0 0%
20%	III	1	7.21e-07	1.7136e-06	26	22645 100%	4466 100%	0 0%	0 0%
20%	IV	0	7.34e-07	1.1782e-06	31	30152 100%	5996 100%	0 0%	0 0%
20%	V	1	8.12e-07	7.9371e-05	25	33871 100%	6794 99.99%	0 0%	1 0.01%

Table 4. Result on injections with random data over a high-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	1	9.63e-07	3.7269e-07	28	7562 100%	1475 100%	0 0%	0 0%
20%	II	1	9.57e-07	7.5004e-06	28	15093 100%	2981 100%	0 0%	0 0%
20%	III	1	6.32e-07	3.281e-05	28	22645 100%	4457 100%	0 0%	0 0%
20%	IV	0	8.27e-07	3.8664e-05	28	30161 100%	5987 100%	0 0%	0 0%
20%	V	0	7.31e-07	5.3963e-06	29	33890 100%	6774 99.97%	0 0%	2 0.03%

Results on full trace. The results are now extended for attacks over the full trace as presented in Tables 7 and 8. In this case the attacks are carried by selecting a message at random from the trace and re-injecting it at some random point no later than a few dozen frames afterward. The data-field of the injected frame is either identical to the genuine one (replay attacks) or replaced by random data. The results start to degrade a bit as false-positives start to appear. Still, these stay at several percents and only in case of 20% replays they increase to 13.83%. A bigger concern are the false negatives which also increase at almost 43.58% in case of 5% replays. This may be due to the low density of the training set since the rate drops at 17.64% as soon as the injection rate is increased at 20%.

Table 5. Result on replay attacks over a low-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	6	1.10e-03	2.9128e-04	26	7530 100%	1507 100%	0 0%	0 0%
20%	II	6	2.32e-03	6.1636e-06	37	15086 100%	2909 97.36%	0 0%	79 2.64%
20%	III	6	6.64e-04	6.0327e-03	33	22601 100%	4297 95.28%	0 0%	213 4.72%
20%	IV	6	2.74e-03	4.3834e-03	16	30081 100%	5639 92.95%	0 0%	428 7.05%
20%	V	6	4.23e-03	2.5314e-02	13	33886 100%	5855 86.36%	0 0%	925 13.64%

Table 6. Result on replay attacks over a high-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	6	9.61e-07	2.0248e-06	43	7557 100%	1449 97.91%	0 0%	31 2.09%
20%	II	1	6.70e-07	3.7639e-03	45	15051 100%	2992 98.97%	0 0%	31 1.03%
20%	III	6	6.64e-04	6.0327e-03	33	22592 100%	4431 98.05%	0 0%	88 1.95%
20%	IV	1	8.85e-07	4.2954e-05	43	30147 100%	5936 98.92%	0 0%	65 1.08%
20%	V	0	5.92e-07	1.8005e-05	42	33886 100%	6585 97.12%	0 0%	195 2.88%

Table 7. Results on injections with random over the full trace

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	1	9.15e-07	9.4955e-04	47	79842 99.68%	3833 98.26%	257 0.32%	68 1.74%
5%	V	1	7.51e-07	8.2342e-04	50	89782 99.70%	4368 98.18%	269 0.3%	81 1.82%
20%	IV	0	6.07e-07	1.6087e-03	48	80137 99.64%	15552 99.85%	288 0.36%	23 0.15%
20%	V	1	8.34e-07	3.6169e-04	41	89850 99.62%	17700 99.42%	347 0.38%	103 0.58%

Table 8. Results on replay attacks over the full trace

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	1.41e-02	1.8857e-02	213	78749 98.37%	2445 61.96%	1305 1.63%	1501 38.04%
5%	V	6	3.47e-03	2.1721e-02	167	88401 98.20%	2526 56.42%	1622 1.80%	1951 43.58%
20%	IV	6	2.01e-02	3.8511e-02	150	69220 86.17%	12909 83.26%	11106 13.83%	2765 17.64%
20%	V	6	9.12e-03	3.7179e-02	158	80157 88.91%	14430 80.88%	10001 11.09%	3412 19.12%

Results on full trace with reduced network size. Reducing the network size is mandatory since the computational and memory load may be too high for automotive-grade controllers as we discuss in the next section. Tables 9 and 10 hold results for injections with random data and replays over the full trace with a network that has a hidden layer reduced to 1/4. Tables 11 and 12 hold the same results for a network with a hidden layer reduced to 1/16. As expected, the performance does degrade with a reduce network size. Still, the results are satisfactory at both 1/4 and 1/16 size of the hidden layer. Again the most significant degradation is for the false-negatives in case of replay attacks. This improves as soon as replays are increased to 20% suggesting the need for a bigger training set.

Table 9. Results on injections with random data over the full trace at 1/4 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	0	9.35e-07	4.9171e-04	45	79842 99.68%	3846 98.28%	257 0.32%	67 1.72%
5%	V	1	4.60e-07	2.0366e-03	46	89665 99.57%	4275 96.09%	386 0.43%	174 3.91%
20%	IV	6	5.42e-07	7.3191e-04	52	80153 99.66%	15462 99.27%	272 0.34%	113 0.73%
20%	V	1	9.90e-07	1.0166e-03	50	89471 99.20%	17695 99.39%	726 0.80%	108 0.61%

Results on a longer trace. Finally, we experiment with a longer trace of 500,000 frames. The objective was to determine if the longer trace will increase the false-positives rate. This however appears to remain stable and correlate only with the injection rate and learning time which was already obvious from the previous experiments. For brevity, the results are moved to Appendix A in Tables 19, 20, 21 for injections with random data and Table 22 for replays.

Table 10. Results on replay attacks over the full trace at 1/4 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	8.93e-03	1.8363e-02	281	78719 98.33%	2476 62.75%	1335 1.67%	1470 37.25%
5%	V	6	1.07e-02	1.8056e-02	214	89050 98.92%	2711 60.55%	973 1.08%	1766 39.45%
20%	IV	6	1.82e-02	3.4832e-02	306	72010 89.65%	12742 81.29%	8316 10.35%	2932 18.71%
20%	V	6	1.09e-02	3.4798e-02	154	84019 93.19%	14046 78.72%	6139 6.81%	3796 21.28%

Table 11. Results on injections with random data over the full trace at 1/16 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	1.79e-04	1.1984e-03	53	79818 99.65%	3757 96.31%	281 0.35%	144 3.69%
5%	V	6	3.15e-05	1.6704e-03	39	89444 99.33%	4229 95.06%	607 0.67%	220 4.94%
20%	IV	6	4.0e-05	1.1148e-03	42	80007 99.48%	15401 98.88%	418 0.52%	174 1.12%
20%	V	6	7.50e-05	2.3672e-03	54	88763 98.41%	17439 97.96%	1434 1.59%	364 2.04%

Table 12. Results on replay attacks over the full trace at 1/16 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	7.86e-03	2.8314e-02	125	79528 99.34%	1791 45.39%	526 0.66%	2155 54.61%
5%	V	6	4.45e-03	2.6876e-02	113	88184 97.96%	1871 41.79%	1839 2.04%	2606 58.21%
20%	IV	6	1.35e-02	4.1603e-02	189	63747 79.36%	12704 81.05%	16579 20.64%	2970 18.95%
20%	V	6	1.69e-02	4.5909e-02	202	74453 82.58%	13068 73.24%	15705 17.42%	4774 26.76%

3.3 Computational Results

The successful deployment of an IDS in real life vehicular applications depends on the computational constraints of the embedded platforms employed in its implementation. We examine the computational performance of our proposal by measuring the runtime of the detection algorithm on three automotive grade platforms. The first, representing the low-performance device group, is a NXP S12XF512 microcontroller. From the high performance sector we employed an Infineon AURIX TC297 microcontroller and a Renesas RH850/E1x. The S12XF chip comes with 32KB of RAM, 512KB of Flash and a 16 bit main core (a coprocessor is also available for reducing peripheral interrupt load) that can provide a top operating frequency of 50MHz. On the other hand, the AURIX platform is equipped with 728KB of RAM, 8MB of Flash and three 32-bit cores running at up to 300MHz. The Renesas platform which we evaluated using a dedicated simulator offers 352KB of RAM and 4MB of Flash and two 32 bit cores clocked at up to 320MHz.

Our detection algorithm was implemented to run on a single-core using training data that is stored in the Flash memory. We assume that the weights are computed off-line and already available as a result of an initial training step. Three sets of weights were used in our tests corresponding to the network with a full size hidden layer, a hidden layer reduced to 1/4 and 1/16 respectively. Tests were made using set of weights stored both as single and double precision floating point values. Tables 13 and 14 holds the run-times measured for the detection algorithm in the analysed scenarios. We were unable to obtain results for running the detection algorithm on the S12XF platform with the full size hidden layer due to limitation in the employed compiler.

Table 13. Computational results on single precision floats

Platform	Full hidden layer	1/4 hidden layer	1/16 hidden layer
S12XF512	n/a	$52.3ms$	$13.22ms$
TC297	$3.904ms$	$899\mu s$	$237.5\mu s$
RH850/E1x-FCC1	$2.697ms$	$667.1\mu s$	$157.6\mu s$

Table 14. Computational results on double precision floats

Platform	Full hidden layer	1/4 hidden layer	1/16 hidden layer
S12XF512	n/a	$110.5ms$	$25.76ms$
TC297	$15.26ms$	$3.744ms$	$822\mu s$
RH850/E1x-FCC1	$2.680ms$	$671.3\mu s$	$162.73\mu s$

As expected, the low-end platform bring on a considerable performance bottleneck making the deployment of the detection algorithm only feasible for a reduced neural network size and a CAN network with few nodes sending messages with high cycle times (i.e. greater than 100ms). High performance platforms prove to be more suitable for implementing the detection algorithm. However, using the full version of the proposed neural network may still be problematic for handling CAN traffic with cycle times in the order of 10s of milliseconds.

4 Conclusion

Neural networks prove to be effective in detecting intrusions on CAN but limitations exists. As expected, the results are split between the two types of attacks replay vs. modification attacks. For replay attacks detection rate is lower because injected frames are identical to genuine frames. In this case the time-stamp is the only indicator. In case of injections with random data, the attack is easily detected by the network. From the detection point of view, the results are satisfactory and neural networks looks like a promising mechanism for detecting intrusions on the CAN bus. The more significant problem comes from computational and storage requirements as neural networks do not appear suitable for low-end automotive-grade controllers. High-end controllers may cope with neural networks of reduced size, but computational demands are still high. Thus detection may not be always carried locally on each node, unless dedicated hardware is added. The solution may be to rely on gateways equipped with stronger cores that can filter traffic in real time. Such a solution may be subject of future investigations for us. Nonetheless, we plan as future work extending this evaluation over more complex real-world in-vehicle traces from CAN, CAN-FD and FlexRay.

Acknowledgement. This work was supported by a grant of Ministry of Research and Inovation, CNCS-UEFISCDI, project number PN-III-P1-1.1-TE-2016-1317, within PNCDI III (2018-2020).

References

1. K.-T. Cho and K. G. Shin. Fingerprinting Electronic Control Units for Vehicle Intrusion Detection. In *25th USENIX Security Symposium*, 2016.
2. W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee. Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks. *IEEE Trans. Vehicular Technology*, 67(6):4757–4770, 2018.
3. W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee. VoltageIDS: Low-Level Communication Characteristics for Automotive Intrusion Detection System. *IEEE Transactions on Information Forensics and Security*, 2018.
4. B. Groza, P.-S. Murvay, A. Van Herrewege, and I. Verbauwhede. LiBrA-CAN: a Lightweight Broadcast Authentication protocol for Controller Area Networks. In *11th International Conference on Cryptology and Network Security, CANS 2012, Springer-Verlag, LNCS*, 2012.
5. B. Groza and S. Murvay. Efficient protocols for secure broadcast in controller area networks. *IEEE Transactions on Industrial Informatics*, 9(4):2034–2042, 2013.

6. O. Hartkopp, C. Reuber, and R. Schilling. MaCAN-message authenticated CAN. In *10th Int. Conf. on Embedded Security in Cars (ESCAR 2012)*, 2012.
7. S. Jain and J. Guajardo. Physical Layer Group Key Agreement for Automotive Controller Area Networks. In *Conference on Cryptographic Hardware and Embedded Systems*, 2016.
8. M.-J. Kang and J.-W. Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016.
9. M.-J. Kang and J.-W. Kang. A novel intrusion detection method using deep neural network for in-vehicle network security. In *Vehicular Technology Conference (VTC Spring), 2016 IEEE 83rd*, pages 1–5. IEEE, 2016.
10. R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata. CaCAN - Centralized Authentication System in CAN (Controller Area Network). In *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.
11. H. Lee, S. H. Jeong, and H. K. Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. *Privacy, Security and Trust (PST) 2017*, 2017.
12. C.-W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli. Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems. *IEEE Embedded Systems Letters*, 7(1):11–14, 2015.
13. M. Marchetti, D. Stabili, A. Guido, and M. Colajanni. Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms. In *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–6. IEEE, 2016.
14. A. Mueller and T. Lothspeich. Plug-and-secure communication for CAN. *CAN Newsletter*, pages 10–14, 2015.
15. P.-S. Murvay and B. Groza. Source identification using signal characteristics in controller area networks. *IEEE Signal Processing Letters*, 21(4):395–399, 2014.
16. M. Müter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1110–1115. IEEE, 2011.
17. M. Müter, A. Groll, and F. C. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 92–98. IEEE, 2010.
18. S. N. Narayanan, S. Mittal, and A. Joshi. OBD_SecureAlert: An Anomaly Detection System for Vehicles. In *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.
19. A.-I. Radu and F. D. Garcia. LeiA: a lightweight authentication protocol for CAN. In *European Symposium on Research in Computer Security*, pages 283–300. Springer, 2016.
20. H. M. Song, H. R. Kim, and H. K. Kim. Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*, pages 63–68. IEEE, 2016.
21. I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi. A language-based intrusion detection approach for automotive embedded networks. *International Journal of Embedded Systems*, 10(1):1–12, 2018.
22. A. Taylor, S. Leblanc, and N. Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pages 130–139. IEEE, 2016.
23. A. Theissler. Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection. *Knowledge-Based Systems*, 123:163–173, 2017.
24. A. Van Herrewege, D. Singelee, and I. Verbauwhede. CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus. In *ECRYPT Workshop on Lightweight Cryptography*, volume 2011, 2011.

Appendix - Results on Various Injection Rates over a Single ID and a Longer Trace of 500,000 Packets

Table 15. Result on injections with random data over a low-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	I	0	7.88e-07	9.4077e-07	28	7537 100%	370 100%	0 0%	0 0%
5%	II	0	9.36e-07	7.0086e-06	26	15078 100%	736 100%	0 0%	0 0%
5%	III	1	7.64e-07	1.6502e-05	26	22603 100%	1119 100%	0 0%	0 0%
5%	IV	1	9.78e-07	6.7137e-05	25	30113 100%	1511 99.67%	0 0%	5 0.33%
5%	V	1	6.33e-07	8.5875e-05	25	33892 100%	1683 99.53%	0 0%	8 0.47%
10%	I	6	7.95e-07	1.2784e-07	20	7562 100%	721 100%	0 0%	0 0%
10%	II	0	7.17e-07	1.3031e-07	23	15073 100%	1494 100%	0 0%	0 0%
10%	III	1	8.39e-07	1.0475e-06	27	22603 100%	2248 100%	0 0%	0 0%
10%	IV	1	6.92e-07	2.5506e-05	25	30133 100%	3002 100%	0 0%	0 0%
10%	V	1	7.53e-07	2.7009e-05	27	33892 100%	3385 100%	0 0%	0 0%

Table 16. Result on injections with random data over a high-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	I	0	9.73e-07	1.7407e-06	29	7535 100%	372 100%	0 0%	0 0%
5%	II	0	8.36e-07	1.4209e-05	26	15068 100%	746 100%	0 0%	0 0%
5%	III	1	6.65e-07	2.1e-05	29	22584 100%	1136 99.82%	0 0%	2 0.18%
5%	IV	0	8.58e-07	1.6136e-04	27	30126 100%	1498 99.67%	0 0%	5 0.33%
5%	V	0	6.57e-07	1.0028e-03	25	33898 100%	1665 98.81%	0 0%	20 1.19%
10%	I	1	7.10e-07	1.2579e-06	26	7518 100%	765 100%	0 0%	0 0%
10%	II	0	7.63e-07	2.8341e-06	25	15057 100%	1510 100%	0 0%	0 0%
10%	III	1	8.85e-07	1.556e-05	20	22615 100%	2234 99.91%	0 0%	2 0.09%
10%	IV	1	6.04e-07	7.2312e-07	17	30151 100%	2983 99.97%	0 0%	1 0.03%
10%	V	0	7.23e-07	9.4239e-05	27	33884 100%	3390 99.91%	0 0%	3 0.09%

Table 17. Result on replay attacks over a low-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	I	6	1.59e-03	6.2598e-04	13	7561 100%	346 100%	0 0%	0 0%
5%	II	6	5.11e-04	3.2658e-04	27	15082 100%	693 94.67%	0 0%	39 5.33%
5%	III	6	7.18e-04	1.1147e-03	19	22618 100%	1031 93.39%	0 0%	73 6.61%
5%	IV	6	5.87e-03	4.7389e-03	11	30132 100%	1375 91.85%	0 0%	122 8.15%
5%	V	6	1.18e-03	1.9846e-02	14	33894 100%	1308 77.44%	0 0%	381 22.26%
10%	I	6	4.98e-04	4.4403e-05	25	7549 100%	734 100%	0 0%	0 0%
10%	II	6	6.35e-03	1.0116e-03	24	15081 100%	1455 97.91%	0 0%	31 2.09%
10%	III	6	1.86e-03	7.7252e-04	16	22608 100%	2126 94.78%	0 0%	117 5.22%
10%	IV	6	4.99e-03	8.5069e-03	13	30120 100%	2803 92.97%	0 0%	212 7.03%
10%	V	6	1.69e-02	1.9815e-02	12	33882 100%	2638 77.70%	0 0%	757 22.30%

Table 18. Result on replay attacks over a high-entropy ID

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	I	0	9.62e-07	4.106e-08	42	7519 100%	388 100%	0 0%	0 0%
5%	II	0	8.82e-07	6.7873e-08	47	15072 100%	742 100%	0 0%	0 0%
5%	III	6	5.35e-05	4.8317e-05	43	22568 100%	1154 100%	0 0%	0 0%
5%	IV	0	8.44e-07	8.9661e-08	53	30116 100%	1458 96.36%	0 0%	55 3.64%
5%	V	6	1.35e-03	1.1116e-02	27	33875 100%	1486 87%	0 0%	222 13%
10%	I	0	7.19e-07	4.7954e-08	51	7536 100%	747 100%	0 0%	0 0%
10%	II	6	5.74e-04	1.8338-04	27	15068 100%	1469 98%	0 0%	30 2%
10%	III	0	8.57e-07	4.0524e-08	48	22608 100%	2126 94.96%	0 0%	117 5.04%
10%	IV	0	9.36e-07	1.5564e-08	68	30116 100%	2831 93.77%	0 0%	188 6.23%
10%	V	6	3.33e-03	7.7698e-03	13	33877 100%	3124 91.88%	0 0%	276 8.12%

Table 19. Results on injections with random data over a longer trace of 500,000 frames

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	1	9.04e-07	3.4937e-05	95	400193 100%	19757 99.75%	0 0%	50 0.25%
5%	V	6	2.04e-05	6.4378e-03	53	450013 99.98%	22302 99.59%	93 0.02%	92 0.41%
20%	IV	1	8.20e-07	6.1195e-05	58	401825 100%	78159 99.98%	0 0%	16 0.02%
20%	V	0	7.50e-07	5.5172e-03	52	450822 99.98%	89024 99.93%	95 0.02%	59 0.07%

Table 20. Results on injections with random data over a longer trace of 500,000 frames and 1/4 network size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	1	9.04e-07	1.0093e-04	51	400193 100%	19708 99.5%	0 0%	99 0.5%
5%	V	6	2.19e-04	6.357e-03	38	450014 99.98%	22023 98.34%	92 0.02%	371 1.66%
20%	IV	1	8.35e-07	1.1475e-04	66	401825 100%	78106 99.91%	0 0%	69 0.09%
20%	V	1	6.39e-07	5.5222e-03	50	450822 99.98%	89007 99.91%	95 0.02%	76 0.09%

Table 21. Results on injections with random data over a longer trace of 500,000 frames and 1/16 network size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	6.24e-05	1.3187e-04	62	400193 100%	19677 99.34%	0 0%	130 0.66%
5%	V	6	1.41e-06	6.7372e-03	74	450013 99.98%	22082 98.61%	93 0.02%	312 1.39%
20%	IV	6	1.28e-04	5.2868e-04	96	401825 100%	77864 99.60%	0 0%	311 0.4%
20%	V	6	7.85e-05	5.0288e-03	63	450836 99.98%	88707 99.58%	81 0.02%	376 0.42%

Table 22. Results on replay attacks over a longer trace of 500,000 frames

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	2.21e-02	2.6788e-02	79	399912 99.97%	9267 46.41%	122 0.03%	10699 53.59%
5%	V	6	1.23e-02	3.3719e-02	62	449910 99.96%	8981 40.04%	160 0.04%	13449 59.96%
20%	IV	6	6.93e-02	1.2133e-01	41	392552 97.71%	35752 45.70%	9215 2.29%	42481 54.30%
20%	V	6	4.06e-02	6.7429e-02	78	329834 73.15%	59553 66.83%	121051 26.85%	29562 33.17%