Prediction-Based Intrusion Detection System for In-Vehicle Networks Using Supervised Learning and Outlier-Detection

Moulay Abdelaziz Elaabid, Khaled Karray, Jean-Luc Danger, Sylvain Guilley

Prediction-based Intrusion Detection System for In-vehicle Networks using Supervised Learning and Outlier-detection

Khaled Karray[1,4], Jean-luc Danger[1,2],
Sylvain Guilley[2,1,3], and Moulay Abdelaziz Elaabid[4]

[1] Telecom Paristech, Paris, France
[2] Secure-IC, Paris, France
[3] École Normale Supérieure, Info dpt, Paris, France
[4] PSA-GROUPE, Paris, France

Abstract. Modern connected vehicles are composed of multiple electronic control units (ECUs) holding sensors, actuators but also wired and wireless connection interfaces, all communicating over shared internal communication buses. The cyber-physical architecture based on this ECU network has been proven vulnerable to multiple types of attacks leveraging remote, direct and indirect physical access. Attacks initiated from these access vectors go through the internal communication buses and spread over the whole network of ECUs. For this reason it is important to detect, and if possible to mitigate, attacks on the internal buses of the vehicle.
In this article, a novel intrusion detection system is developed to monitor vehicle state from information collected on internal buses. Based on supervised machine learning techniques, a normal behavior is learned and used as a reference to detect deviations. The principle is to learn how to predict the next state of the vehicle based on information and sensor values sent over communication buses. Experimental validation is conducted using data collected from different drivers. Results show that the approach is able to learn the nominal behavior with high accuracy for a single driver as well as for a set of different drivers. Results also demonstrate its ability to predict attacks with low false negative rate. This motivates the approach to be used for indirect and remote attacks intrusion detection as well as for safety purposes to detect sensor failures, lost connection with the sensor, etc.

Keywords: Automotive · Intrusion detection · Machine learning

1 Introduction

Two important requirements of today's cars are a high level of safety and connectivity with the outside world. This involves the use of advanced technologies based on a computing infrastructure composed of numerous electronic components –named Electronic Control Units (ECUs)– embedded inside the vehicle. These ECUs are in charge of processing sensed data through embedded sensors, and transforming them into commands for the actuators. For this purpose, ECUs share communication buses. These are used for periodic and event-based messages that allows the ECUs to monitor the vehicle state through the control and supervision of sensors and actuators states. The communication bus mostly used in the automotive domain is the Controller Area Network (CAN, ISO 11898), which connects together many ECUs.

Recently, the CAN protocol has become the center of multiple cyber-security issues [2,4]. In this context, Hoppe et al. [7] were the first researchers to point out the weaknesses of the CAN bus. These findings were further investigated and confirmed by Koscher et al. [11] and Checkoway et al. [2] who performed frame replay and frame injection attacks on a real vehicle. In these attacks, the attacker physically connects to the CAN network and replays or injects messages on the CAN bus. Miller and Valasek [15] showed that physical access to the communication bus was not necessary and showcased an attack granting remote control over a vehicle. In their experiments, the attacker remotely takes control of a legitimate ECU and use that ECU to send legitimate messages.
To protect against these attacks, multiple solutions have been proposed:

- Protecting the messages payload can be a good approach against an attacker that has physical access to the communication bus. Nilsson et al. [18] proposed to send message authentication codes over consecutive CAN frames to authenticate the messages. Hartkopp et al. [6] proposed to use Cipher based Message Authentication Code (CMAC) as a symmetric authentication measure between the sender and the receiver. These types of solution allow the receiving ECU to verify the integrity and/or the authenticity of the messages and to filter out forged information sent by the attacker (which is unauthentic).
- A second family of protection solutions is known as in-vehicle network Intrusion Detection and Prevention Systems. The role of these systems is to monitor the in-vehicle network for suspicious behavior like frame(s) injection and replay attacks and either physically kill suspicious frames by causing a frame error or by filtering them out. Examples of such detection mechanisms are presented for instance in the work of Taylor et al. [20], and the work of Marchetti et al. [14]. In general, state-of-the-art detection mechanisms can be categorized into two main classes: *rule-based* detection mechanisms and *statistical* detection mechanisms. We investigate more in details these types of solutions in section 2.1.
- Another type of protection solution, specific to the CAN bus focuses on protecting the identifier. These solutions are useful to protect against reverse engineering, replay and injection attacks for an attacker that has physical access to the CAN network. For instance Humayed et al. [8] presented a solution that can change a message identifier when an attack is detected, thereby stopping the targeted attack dead. Han et al. [5,12] proposed an identifier randomization function for the same purpose.

In the sequel we focus on in-vehicle intrusion detection techniques. State-of-the-art *rule-based* intrusion detection uses mechanisms known as identifier filtering, identifier timing and syntax check. Some of them also focus on payload content and implement what is known as *deep packet inspection* techniques.

Contributions. In this paper, we tackle the problem of deep packet inspection of in-vehicle networks from a practical viewpoint. For an attacker that gains control over an ECU, we consider that her capacity evolves from simply injecting an extra message on the communication bus, to being capable of modifying the content (payload) of a legitimate message. This evolution makes the classical detection mechanisms, based on identifier timing and syntax check, merely obsolete. In order to detect these kinds of attacks, a novel detection mechanism is developed. We formulate the problem in a way that allows to learn the normal behaviour of the system in terms of message payload content. Bad behaviour and bad payload content are flagged with outlier detection techniques. The method thus described can be adopted not only as an intrusion

detection mechanism, but also as an *online monitoring failure detection* and a *sensor rationality check* safety mechanisms as described by the "Road vehicles – Functional safety" standard ISO-26262 [9]. We validate in practice the model with real CAN traces collected from drive tests. We show that the approach is able to learn the nominal behavior with high accuracy and low false positives, for three different driving behaviors separately. Then we show that it is also able to learn a unified nominal behavior with high accuracy and low false positives, that can accommodate different driving behaviors. Finally we run an attack campaign in order to test the robustness of the detection rules, and demonstrate its ability to predict attacks with low false negative rate.

Outline. The remainder of the paper is structured as follows. Section 2 gives some background on CAN intrusion detection mechanisms, machine learning techniques and the related work. Section 3 gives details about data collection and feature engineering. Section 4 presents practical validation results on real CAN traces. Section 5 concludes.

2 Background

2.1 Intrusion detection systems over CAN

Detecting intrusions on the in-vehicle communication buses is important as it can prevent attacks from spreading to other ECUs. It can be considered as the last line of defense after protecting ECUs interfaces from the outside world. Many mechanisms have been proposed to detect possible intrusion on the CAN bus. Figure 1 gives a high level overview of these mechanisms.

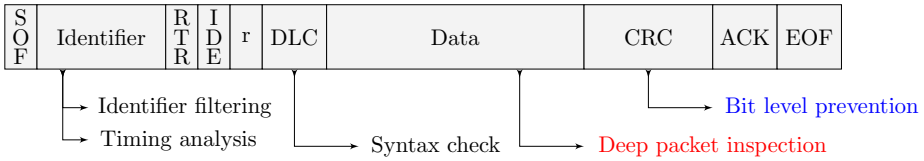
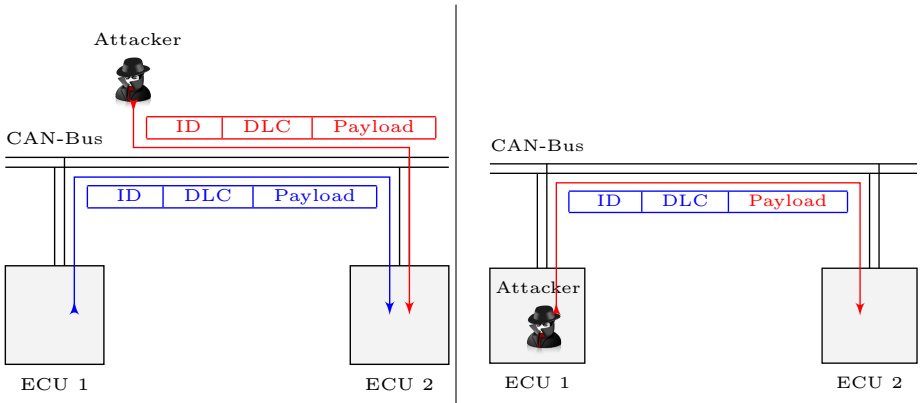


Fig. 1: High level synthesis of detection mechanisms applied to the CAN frame

Using the frame identifier, an intrusion detection system can establish a list of *allowed* and *forbidden* identifiers, based on which it can decide which frames to filter. This technique is best known as *identifier filtering* or *identifier white-listing* [16,15]. Such white list can also depend on the context of the vehicle: for instance the intrusion detection system may allow certain identifiers when the vehicle is on parking state, and reject them when the vehicle is moving. This technique is used in particular to enforce the diagnostic security policy by allowing diagnostic messages only in certain vehicle states. Another detection mechanisms that uses identifiers is *timing analysis* [7,16,3]. It is a very popular technique that works well with periodic messages. It consists in setting an acceptance time-window for each periodic message. If the same message is received outside of its acceptance time-window, the system shall consider it as an intrusion and shall filter it out.
Besides the identifier of the messages, the data length code (DLC) can also be exploited

to detect bad behaviour [16]. In fact, each manufacturer sets-up a proprietary protocol over the CAN standard. This protocol consists in creating a mapping between identifiers and payload information (sensor values for instance), also called signals, shared across all ECUs. This mapping defines a syntax that can be checked based on the payload length of each message. Messages that violate this syntax (i.e., messages sent with the wrong DLC) are then flagged as intrusions.

In this paper we distinguish between two attacker models (Figure 2). Figure 2a shows an attacker model that has direct physical access to the CAN bus. Since modification of a message on the fly is rather difficult (the message being protected with CRC mechanism), this attacker instead injects *extra* messages on the CAN bus. These messages will modify the proprietary communication protocol defined on top of CAN for instance by modifying the syntax of the message or its periodicity. These anomalies are caught by the classical detection mechanisms described previously. Therefore, an advanced attacker who has indirect and even remote access over a legitimate ECU (Figure 2b), might aim at modifying sensor information and commands directly on the payload without disrupting the defined protocol. Thus will not be detected by above-mentioned classical detection mechanisms. Consequently, we need to build mechanisms able to detect bad behaviour inside the payload. These mechanisms are referred to as *deep packet inspection*. The latter encompasses most safety checks. For instance, duplicated signals, process counters, checksum ... In this paper, we focus on *deep packet inspection* type of detection, as this



(a) Direct physical access to the CAN-Bus (b) Indirect/remote access to the CAN-Bus

Fig. 2: Attacker models. (a) State-of-the-art model. (b) Model investigated in this paper

detection mechanism is well adapted to sophisticated attacker model. Supervised machine learning techniques are used in order to build a nominal behaviour based on received signals; then outlier detection flags deviations from the previously built behavioral model.

2.2 Machine learning algorithms and their application

In practice there are multiple application domains where machine learning algorithms excel in prediction tasks. They are generally used to study correlation between different

inputs (also called features), to approximate an output function and/or to discover interesting data structures. For these reasons we decided to explore the use of machine learning techniques in the context of vehicle cyber-physical attacks and intrusion detection.

Machine learning algorithms can be divided into two main categories depending on the learning strategy:

1. *Supervised learning:* a machine learning algorithm is said to be using supervised learning strategy when the training set includes both the input data and the output data of the algorithm. In that sense the algorithm is training to learn a mapping function by minimizing a pre-defined cost function. The trained algorithm is then tested on some other examples that were not included in the training set. It is said to be generalizing well if the performance of the trained algorithm on the test set is comparable to its performance on the training set.
2. *Unsupervised learning:* a machine learning algorithm is said to be using unsupervised learning strategy if the training only includes the input data but not the expected output. In that sense, the machine learning algorithm is trying to discover interesting data structures.

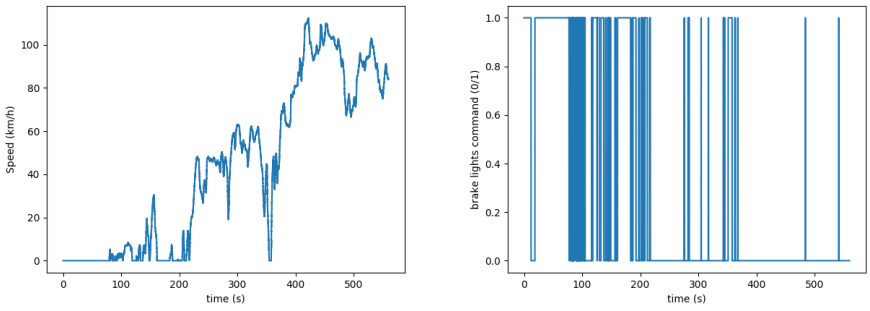
Machine learning techniques have been used previously in the context of *deep packet inspection* for intrusion detection. Kang et al. [10] train a deep neural network structure to classify normal *versus* attack packets using probability-based feature vectors of packet payload bits. Training data were generated by the *Open Car Test-bed and Network Experiments (OCTANE)* packet generator [1]. Normal and attacked packets were necessary in order to train the algorithm. Loukas et al. [13] use sensor input features along with recurrent neural network (RNN) to detect attacks on vehicles. The detection mechanism also consists in learning to classify whether the vehicle is under attack or not with a training data that included both attacked and normal packets. An important limitation of the work of Kang et al. [10] and Loukas et al. [13] is that the intrusion detection system is trained to recognize specific attacks. An important effort is devoted to generate attacked packets in order for the detection module to learn the attack profile. Taylor et al. [21] use long short-term memory networks to detect attacks on the CAN bus. The approach was applied to the identifier, and learns to predict the next packet identifier on the CAN bus. Highly surprising bits are then flagged as anomalous. This method draws its strength from repetitive periodic sequences. This is why it is applied to the identifier field. Nevertheless, this is hardly the case for payload information that holds sensor information. Narayanan et al. [17] propose to build Hidden Markov Model of the normal behaviour of the car based on sensor values (or signals). Their work shows that it is possible to detect data manipulation attacks like speed discontinuity. In their work, Narayanan et al. focus on signal changes rather than signal values, i.e., gradients of signals. As a result, the built model can serve to detect *signal jumps* types of anomalies and cannot be used for prevention. Besides, their work does not evaluate the True Positive Rate and False Positive Rate of the detection principle.

An important limitation of the previous approaches is that during training, data representing both attacked and non-attacked states is needed to learn to recognize attacks. In order to produce this kind of data we need to select and perform multiple attacks on the vehicle. Thus it is challenging to generate the data for a large range of attacks. Besides, the intrusion detection system learns only to recognize performed attacks included in the training set. Another downside is that the approach allows only to predict whether the vehicle is under attack or not but does not deliver more detailed information useful to investigate on the cause of the attack.

In order to overcome these limitations, we propose a different formulation of the problem. In fact, instead of predicting whether the vehicle is under attack (or not) based on payload inputs, we break down the payload information into signals according to the manufacturer proprietary protocol and we train a machine learning algorithm to predict the next signal value based on other signals. The idea is then to compare the predicted signal and the received signal. Under the assumption that the predictor is *accurate enough*, we assume the following as a security metric: if the difference between the prediction and the received value is *large enough*, then, with a high probability, the vehicle is being attacked and that the predicted signal is the potential cause of the attack.

Input signals are sensor values sent from one ECU to the other ones. They can either be real-valued or categorical signals:

- An example of real-valued signal is the speed of the vehicle (Figure 3a). It is sent over 2 bytes of payload information. The received value is then an integer between 0 and 65535. A multiplication by 0.01 is necessary to recover the actual measurement of the sensor to make speed range in $[0,655.35]\,km/h$.
- An example of categorical signal is the brake lights command signal (Figure 3b). It is sent over 1 bit of payload data. The received value is a binary information (0/1) indicating whether to activate the brake lights (1) or not (0).



(a) Speed signal (b) Brake lights command signal

Fig. 3: Example of real-valued and categorical signals

2.3 Problem formulation

In what follows we formulate our problem as a supervised machine learning problem. Let $\mathcal{D}=\{(x_i,y_i)\}_{i\in[1,N]}$ be the set of input-output pairs. Here $\mathcal{D}$ is the collected Data set, and N is the number of observed examples. Each training input $(x_i)_{i\in[1,N]}$ is a d-dimensional vector of components representing signal values/states $(s_i^{(1)},s_i^{(2)},...,s_i^{(d)})$. These are called features and are stored in an $(N\times d)$ matrix X (Figure 4). The output $(y_i)_{i\in[1,N]}$ is stored in a 1-dimensional vector y and represents the target signal that we want to predict. It can be either real-valued (in this case we will talk about *regression*) or a categorical value (in which case we will talk about *classification*), depending on the signal type.

The object of supervised machine learning is to assume the existence of some unknown function $<f>$ that maps the inputs to the outputs, as in (1):

$$f(x)=y, \quad \forall(x,y)\in\mathcal{D}. \tag{1}$$

The goal of the learning process is to estimate the function $< f >$ given a labeled training set and then to make predictions on unseen data x_u using the estimated function $\hat{y}=\hat{f}(x_u)$. We denote the probability distribution over possible labels, given the input vector x_u and the training data set $\mathcal{D}_{train}$ by $p(y|x_u,\mathcal{D}_{train})$. This probability is conditional on the input vector x_u and the training set $\mathcal{D}_{train}$. When approximating the function $< f >$, we will use a machine learning model M_θ, where M is the model, and θ denotes the parameters of the model. The probability distribution over possible labels becomes also conditioned by the chosen model, $p(y=\hat{y}|x_u,\mathcal{D}_{train},M_\theta)$.
When using regression parametric models, we assume that the estimated function used for the prediction introduces a residual error ϵ between the predictions and the ground truth:

$$y=\hat{y}+\epsilon. \tag{2}$$

We make the assumption that the residual error term ϵ has a Gaussian normal distribution, $\epsilon\sim\mathcal{N}(\mu,\sigma^2)$. More explicitly we will assume that the probability distribution over possible labels is as follows:

$$p(y|x_u,\mathcal{D}_{train},M_\theta)=\mathcal{N}(\mu_\theta(x_u),\sigma^2). \tag{3}$$

In order to estimate the model parameters $<\theta>$, we use the maximum likelihood estimator that maximizes $p(\mathcal{D}_{train}|\theta)=\prod_{i=1}^{N}p(y_i|x_i,\theta)$. It is equivalent to finding the model parameters $\hat{\theta}$ that minimizes the negative log-likelihood which is the sum of residual errors $\sum_{i=1}^{N}(y_i-\hat{y}_i)^2=\sum_{i=1}^{N}\epsilon_i^2$:

$$\hat{\theta}=\underset{\theta}{\operatorname{argmin}}\sum_{i=1}^{N}(y_i-\hat{f}_\theta(x_i))^2. \tag{4}$$

Once optimal parameters $\hat{\theta}$ are estimated, the prediction model outputs a predicted signal estimation $\hat{y}_u=\hat{f}_{\hat{\theta}}(x_u)$ for an unseen input vector x_u. The received signal value y is then compared to the estimated signal value. An alert is raised if the two signals are *not similar*

$$Alert=1 \iff |\hat{y}-y|\geq t_p. \tag{5}$$

When using classification parametric models, where the output is one out of C classes, we model the probability over possible labels with a categorical distribution. Let $y_{ij}=I(y_i=j)$ be the one-hot encoding of y_i:

$$p(y|x_u,\mathcal{D}_{train},M_\theta)=\prod_{j=1}^{C}\mu_{\theta,j}(x_u)^{I(y=j)}. \tag{6}$$

In order to estimate the model parameters $<\theta>$, we use the maximum likelihood estimator that maximizes $p(\mathcal{D}_{train}|\theta)=\prod_{i=1}^{N}p(y_i|x_i,\theta)=\prod_{i=1}^{N}\prod_{j=1}^{C}\mu_{\theta,j}(x_i)^{I(y_i=j)}$. This is equivalent to minimizing the negative log-likelihood which is the cross entropy function:

$$\hat{\theta}=\underset{\theta}{\operatorname{argmin}}\sum_{i=1}^{N}\sum_{j=1}^{C}y_{ij}\log(\mu_{\theta,j}(x_i)). \tag{7}$$

Once we have the optimal model parameters $\hat{\theta}$, for each unseen input vector x_u, we make a prediction in favor of the class where the probability distribution is the highest: $\hat{y}_u=\underset{j\in[1,C]}{\operatorname{argmax}}(\mu_{\hat{\theta},j}(x_u))$.

Once optimal parameters $\hat{\theta}$ are estimated, the prediction model outputs a predicted signal estimation $\hat{y}_u=\hat{f}_{\hat{\theta}}(x_u)$ for an unseen input vector x_u. The received signal value

y is then compared to the estimated signal value. An alert is raised if the two signals are *not similar*:

$$Alert = 1 \iff \hat{y} \neq y. \tag{8}$$

3 Data collection and feature engineering

3.1 Data collection

In order to provide training vectors, the best way is to collect data directly from a real vehicle. For this purpose we prepared a CAN acquisition device. The device is composed of a Raspberry Pi with additional CAN-Bus hardware module running a Linux kernel with *SocketCAN* drivers. We equipped a vehicle with the acquisition device connected directly to different CAN buses in order to have direct access to *all* sensor information, although not all of them will be used during training. We collected CAN traces from one vehicle for three different drivers, driving in different circuits for about 90 minutes each. Circuits consisted of multiple driving conditions including city driving, vehicle parking, highway driving, etc. During those data collections, drivers were asked to drive normally but also to perform rare but legitimate scenarios like activating cruise control, activating lane keep assist, activating emergency breaking, etc. For safety reasons no attacks were performed during data collections step.

3.2 Feature engineering

After raw data acquisition, the second step consists in preparing the data for processing. In this step, the goal is to select and arrange the features in a form that would be useful during training step. Each CAN identifier sent over the CAN bus has a payload that is composed of one or multiple signals. A signal is an information (sensor value, ECU state, counter, checksum, ...) that can occupy one or multiple bits or bytes depending on the nature of the information. Extraction of signals requires the knowledge of the proprietary protocol of the car manufacturer. Signals included in the payload for safety reasons, like checksums, process counters, duplicated signals, are checked by safety functions and problems with those signals, if any, would be handled by appropriate safety mechanisms. Thus, they are not relevant for this task and therefore are not selected. Typically we are interested in physical sensor values like speed, acceleration, RPM, etc. The set of those signals defines the state of the vehicle and constitutes the input features that are relevant for learning the normal behaviour and evolution of the car states. The second selection criteria is the relevance with respect to the target signal. In fact, the dimensionality of the training vectors equals the number of selected signals. However, in general, machine learning algorithms do not work well with high dimensional inputs. Indeed, as input vectors dimensions grow, the performance deteriorates, due to the curse of dimensionality. As a result, we choose to select only signal with high correlation with the target signal. For instance, the engine oil temperature has no influence on the vehicle speed, thus would not be selected when building a predictor for the speed signal. On the other hand, the acceleration of the vehicle is highly correlated to the speed of the vehicle, thus will be selected as an input to predict the speed. Using this selection criteria we can guarantee that signals that can *explain the most* the target signal are used for prediction. Signals are featured in the form of a matrix where columns represent signals and lines

represent signal values evolution over time. For each received CAN message that holds selected signal, a new line is added to the matrix where all signals keep their previous values/states except the one that has just been received. Figure 4 gives more details about how to construct the features matrix.

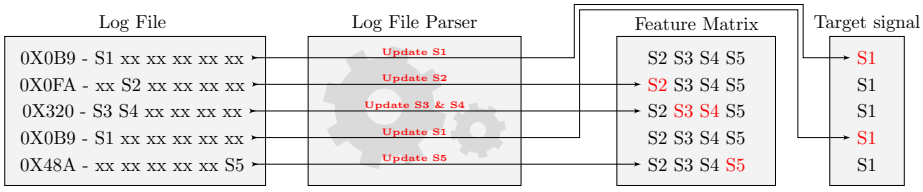


Fig. 4: Parsing the log file and building the training data.

4 Experimental validation and discussion

In order to validate the approach, we conduct some experiments to predict two target signals, one of each type (categorical and real-valued), using five selected input signals. To this end, a total of six signals are extracted. For each target signal, the remaining five are used as input features.

- Speed, is a real-valued signal sent from the Electronic Stability Program (ESP) and that is generated by an embedded speed sensor.
- Acceleration, is real-valued signal that is sent from the Electronic Stability Program (ESP) and generated by an acceleration sensor.
- Engine rotational speed expressed in revolutions per minute (RPM), is a real-valued signal sent by the Engine Control Module (ECM).
- Torque, is a real-valued signal sent by the Engine Control Module (ECM) that contains the engine torque.
- Gearbox position, is a categorical signal sent by the Electronic Shifter Module (ESM), that indicates the gear lever position.
- Brake lights command is a categorical signal that is sent from the Electronic Stability Program (ESP) module to control brake lights.

Experimental validation is conducted in two steps. First we train and evaluate the detection rules using collected data and without performing any attacks. This step gives us the True Negative rate, that we define hereafter as the accuracy (Acc) of the supervised learning algorithm, which will be formally introduced in the section 4.1. The False Positive rate is then derived from the accuracy and equals $(1-Acc)$. Then we conduct an attack campaign and measure how many of the performed attacks are detected. This step gives us the True Positive rate and the False Negative rate. Table 1 defines the metrics that will be used in the sequel.

4.1 Validation metrics:

Regression metrics for real-valued signals: The *accuracy* (denoted as Acc) of a machine learning prediction algorithm is generally measured using the coefficient of determination R^2. The R^2 coefficient of determination is a statistical measure of how well the

Table 1: Detection metrics

	Detected	Not-detected
No-attack	$FP=1-Acc$	$TN=Acc$
Attack	TP	FN

regression predictions approximate the observed target values. The closer it is to 1, the more accurate the prediction is. An R^2 of 1 indicates that the regression predictions perfectly fit the data. We can express the prediction accuracy with the following:

$$Acc=R^2=1-\frac{\sum(\hat{y}_i-y_i)^2}{\sum(y_i-E(y_i))^2}=1-\frac{\sigma_\epsilon^2+\mu_\epsilon^2}{\sigma_y^2}, \tag{9}$$

where $\sum(\hat{y}_i-y_i)^2$ is the residual sum of squares, $\sum(y_i-E(y_i))^2$ is the total sum of squares, σ_ϵ^2 and μ_ϵ^2 are respectively, the standard deviation and mean of the error term, and σ_y^2 is the standard deviation of the target signal y.

Intuitively, comparing the quality of the predictors can be based on the mean and variance of the prediction error ϵ. Ideally the error has to be centered around zero (unbiased predictor) with the smallest possible variance.

To define an intrusion detection system based on the predictor we need to define an *acceptable deviation* of the prediction that can be tolerated. Beyond this *acceptable deviation*, the received signal can be considered way off compared to the prediction and an alarm should be raised. This *acceptable deviation* or detection threshold t_p for the predictor defines the false positives statistically generated by the predictor (red bars in Figure 6). More formally we can define the *false prediction*, as follows:

$$FP_{t_p}(y,\hat{y})=\begin{cases}1 & \text{if } |y-\hat{y}|\geq t_p,\\ 0 & \text{if } |y-\hat{y}|<t_p.\end{cases} \tag{10}$$

Tweaking this parameter t_p helps increase/decrease the false positives probability of the intrusion detection rule that will be defined based on this predictor. The new accuracy measure with respect to t_p becomes $Acc_{t_p}=P(|\epsilon|<t_p)$.

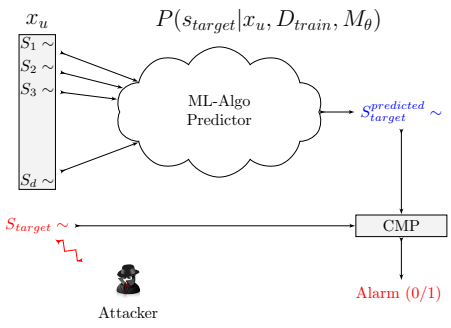


Fig. 5: Prediction principle

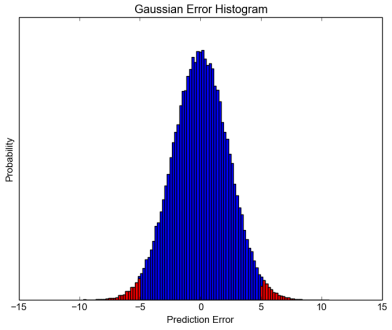


Fig. 6: Gaussian shaped prediction error

Classification metrics for categorical signals: The default accuracy metrics used in machine learning classification tasks is the correct classification ratio:

$$Acc = \frac{\#\text{ correct predictions}}{\#\text{ use-cases}} \tag{11}$$

Unlike regression, for classification it is straightforward to define a false prediction which in this case is simply a mis-classification. More formally we can define the mis-classification function as the following:

$$MC(s,p) = \begin{cases} 1 & \text{if } class(s) \neq class(p), \\ 0 & \text{if } class(s) = class(p). \end{cases} \tag{12}$$

4.2 Predicting a real-valued signal: speed

For regression problems, we chose to validate the approach we described in previous sections on a signal that is important from a safety standpoint. The speed information is sent by the Electronic Stability Program over the CAN bus for the other ECUs to be used in other functions. Besides being displayed for user-information, it is used to compute the effort to be applied on brakes when emergency brakes are activated, to decide when to activate airbags in case of an accident, also to decide if the car doors should be open or closed, and whether or not to accept diagnostics commands and a lot of other functions. In the performed experiments, the goal is to compare between different machine learning algorithms, as each algorithm has a different way of capturing dependencies between input features and the target signal. We used a data set of 10^6 input vectors from each drive test. The data set was split into a training set and a test set of 0.7 and 0.3 size ratio respectively. All experiments are done with the `scikit-learn` library [19].

In the first experiment, we train and evaluate detection rules for each driver separately. We used four types of machine learning algorithms: k-nearest neighbors (KNN), Decision Tree, neural network with logistic perceptron and neural network with rectified linear unit (Relu) perceptron. For each type of machine learning algorithms, we used different tuning parameters to progressively give them the ability to capture more complex dependencies, but also that increase the complexity of the learning algorithm. For instance, this consists in increasing the depth of a decision tree or in increasing the number of neurons and layers for neural networks. Table 2 reports evaluation metrics of the tested algorithms.

First, we note that the results of KNN is merely provided as a baseline. In fact using KNN is advantageous as it gives a very precise local approximation for dense and uniformly distributed training set. It is nevertheless not useful in the context of embedded systems as it needs all the training data in memory in order to make a prediction. Second, each algorithm performs approximately similarly on the three drivers. Third, for a given algorithm, we note that as we increase the complexity (tuning parameters) of the learning algorithm, the accuracy improves. The rule becomes progressively able to capture more dependencies. As a result, it becomes necessary to take into consideration the added complexity compared to the gain in accuracy. For the decision tree algorithm, changing the tree depth from 20 to 40 does not improve significantly the accuracy. Similarly increasing the number of neurons in the Logistic-Neural-Network up to 80 neurons, and increasing the number of layers in the Relu-Neural-Network up to 10 layers does not have a significant effect on the accuracy for all three drivers. We conclude that as the complexity of the algorithm increases, its ability of capturing more dependencies also increases, but reaches a a certain limit beyond which it is no longer advantageous to increase

Table 2: Prediction accuracy of detection rules for $tp = \pm 5\, km/h$ trained and tested with data captures from three different drive tests

ML-Algorithm	Tuning	Driver 1		Driver 2		Driver 3	
		$Acc(\%)$	Acc_{t_p}	$Acc(\%)$	Acc_{t_p}	$Acc(\%)$	Acc_{t_p}
KNN regression	$k=1$	99.97	99.66	99.97	99.77	99.66	99.22
KNN regression	$k=2$	99.97	99.78	99.97	99.82	99.71	99.40
KNN regression	$k=3$	99.97	99.76	99.97	99.78	99.71	99.40
Linear Regression	Null	79.88	23.28	83.47	22.61	74.42	59.89
Decision Tree	depth = 10	99.71	98.19	99.67	98.39	98.58	96.17
Decision Tree	depth = 20	99.97	99.89	99.97	99.93	99.67	99.29
Decision Tree	depth = 40	99.97	99.92	99.97	99.96	99.77	99.59
Neural Net (Logistic)	1 Layer, 30 neurons	98.97	94.74	98.22	88.52	75.67	84.94
Neural Net (Logistic)	1 Layer, 35 neurons	98.96	94.90	98.35	88.95	80.38	83.02
Neural Net (Logistic)	1 Layer, 40 neurons	99.01	94.76	98.62	88.66	82.10	84.97
Neural Net (Logistic)	1 Layer, 80 neurons	99.15	94.74	99.07	92.58	97.54	94.20
Neural Net (Relu)	1 Layer, 10 neurons	99.31	92.82	99.11	87.91	97.26	92.52
Neural Net (Relu)	1 Layer, 20 neurons	99.25	92.44	99.35	93.58	97.32	92.55
Neural Net (Relu)	1 Layer, 40 neurons	99.36	93.75	99.29	92.42	97.61	93.65
Neural Net (Relu)	5 Layer, 10 neurons	99.53	95.19	99.46	94.52	97.67	94.11
Neural Net (Relu)	10 Layers, 10 neurons	99.55	95.36	99.55	95.37	98.37	95.90

the complexity. Overall, and for all three drivers, we can establish that the best results were reported for the decision tree algorithm tuned with depth parameter equals to 40.

4.3 Predicting a categorical signal: brake lights command

For classification problem, we choose to validation the approach on the *brake-lights-command* categorical signals. In order for the accuracy metric to make sense, test data should be balanced, i.e., the number of test vectors should be roughly the same for each class. Results are reported in Table 3.

A similar test procedure was also used for the *brake-lights-command* signal. We notice that there are small differences in the accuracy for the same rule when comparing between different drivers. In fact, practically all the tested rules perform better on the first and second driver than on the third driver. An explanation of this result might be that the third drive test contained singular use-cases that did not appear frequently enough, thus the rules did not train well enough in order to recognize them. An easy solution to overcome this limitation is to collect more data for these specific use-cases. We also notice that the decision tree algorithm tuned with depth parameter equals to 40, reported the best performance for all three drivers.

4.4 Unification of detection rule

In the previous section, we reported results on the accuracy of the predictors trained and evaluated for each driver separately. The resulting detection rules could be influenced by the driving behaviour of the driver. In this section we investigate the possibility of building one single detection rule that can accommodate all three drivers. According to the previous results, the Decision Tree algorithm outperforms the rest of the algorithms for

Table 3: Prediction Accuracy of detection rules for the *brake-lights-command* signal

ML-Algorithm	Tuning	Driver 1 *Acc*(%)	Driver 2 *Acc*(%)	Driver 3 *Acc*(%)
KNN classification	$k=1$	98.96	98.45	97.27
KNN classification	$k=2$	98.70	98.11	96.14
KNN classification	$k=3$	98.89	98.34	97.22
Logistic Regression	Null	93.68	93.01	90.62
Decision Tree	depth = 10	96.72	95.80	94.65
Decision Tree	depth = 20	99.10	98.63	97.12
Decision Tree	depth = 40	99.36	99.00	97.77
Neural Net (Logistic)	1 Layer, 30 neurons	95.86	94.23	94.56
Neural Net (Logistic)	1 Layer, 35 neurons	95.82	94.11	94.48
Neural Net (Logistic)	1 Layer, 40 neurons	96.01	93.88	94.57
Neural Net (Logistic)	1 Layer, 80 neurons	95.97	94.15	94.55
Neural Net (Logistic)	5 Layer, 30 neurons	95.22	93.43	94.80
Neural Net (Relu)	1 Layer, 10 neurons	96.25	94.59	94.23
Neural Net (Relu)	1 Layer, 20 neurons	96.56	95.26	94.33
Neural Net (Relu)	1 Layer, 40 neurons	96.70	95.38	94.48
Neural Net (Relu)	5 Layer, 10 neurons	96.70	95.49	94.49
Neural Net (Relu)	10 Layers, 10 neurons	96.72	95.67	94.70

both predicted signals. Thus, we use Decision Tree algorithm to build the detection rules in this section. In order to train the algorithm we combine the data sets collected during the three drive tests and we split the resulting data set into 0.7 and 0.3 ratio training set and test sets. We report results of the accuracy on the test set as well as on the three data sets separately for the speed signal in Table 4 and for *brake-lights-command* in Table 5. Results show that, for both signals, the resulting detection rules have a high accuracy level on the combined data set as well as on data from each individual driver. This shows that it is possible to build a single detection rule that can accommodate the three drivers.

Table 4: Prediction Accuracy of the unified detection rules for the speed

ML-Algorithm	Tuning	All		Driver 1		Driver 2		Driver 3	
		Acc(%)	Acc_{t_p}	*Acc*(%)	Acc_{t_p}	*Acc*(%)	Acc_{t_p}	*Acc*(%)	Acc_{t_p}
Decision Tree	depth = 40	99.95	99.66	99.97	99.77	99.98	99.77	99.76	99.43

Table 5: Prediction Accuracy of the unified detection rules for the *brake-lights-command*

ML-Algorithm	Tuning	All *Acc*(%)	Driver 1 *Acc*(%)	Driver 2 *Acc*(%)	Driver 3 *Acc*(%)
Decision Tree	depth = 40	98.16	99.37	98.16	97.97

4.5 Evaluation against attacks

In order to evaluate the effectiveness of the detection rule, we conduct a test campaign against simulated attacks. Since we claim that our model can detect attacker that has full control over one of the ECUs (Figure 2b), the simulated attacks consist in replacing the data content of the messages with an attacked content. Thus the attacker is showcasing a *Man-in-the-middle attack* between the signal generator (sensor) and the receiver ECU on which we install the intrusion detection system.

Attacks against real-valued signal: For the speed signal monitoring we perform three types of attacks:

- Random speed injection: in this attack, the attacker substitutes the real sensor value with a random value.
- Speed offset injection: in this attack, the attacker adds to the real speed sensor value an offset value.
- Speed Denial of service (signal drop): in this attack, the attacker interrupts the sending of the frame causing the speed signal to freeze at the last sent value.

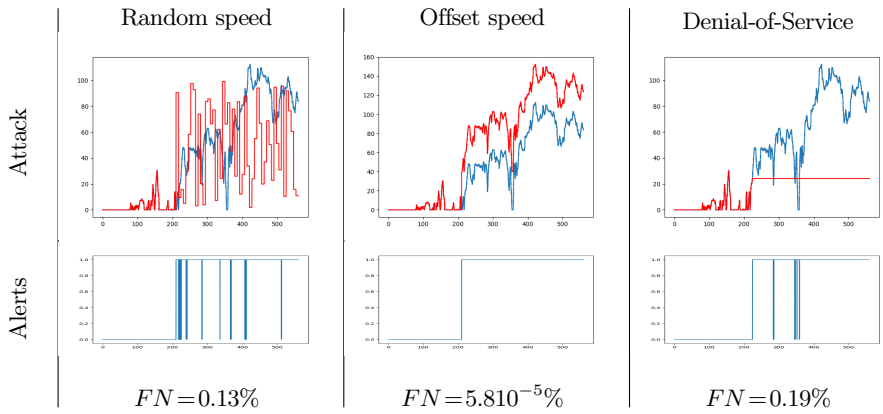


Fig. 7: Alerts raised by the decision tree (depth=40) detection rule tested on three different attacks on the *speed* signal. On top is the ground truth and attacked signals: the blue signal represents the ground truth sensor value, the red signal is the attack signal. On the bottom is the Alerts raised by the detection rule when receiving the attack signal.

Figure 7 shows the attack use-cases on the speed signal. Note that the detection rule is set to raise an Alert as long as the received speed value (injected by the attacker) is outside the acceptance interval of $\pm 5\,km/h$ of the predicted speed value. Thus we consider that an attack is happening if the injected speed signal is outside of this acceptance interval. We can see from the Alerts raised by the detection rule that:

- For the random speed injection attack: as long as the injected speed value is outside the acceptance window, alerts are raised. The alert is not raised when the injected speed value is close to the ground truth value. We obtained 0.13% of false negatives when performing this attack.
- For the speed offset attack, we can see that, the alert is raised as soon as the attack started. In fact, since the speed offset of the attack is set to $+40\,km/h$, the received signal is always outside the acceptance window. The detection in this case is perfect and we obtained $5.810^{-5}\%$ of false negatives.
- For the Denial of service attack, the same reasoning applies. The injected speed is frozen at around $20\,km/h$, which means that most of the time the alarm is raised as the received speed is outside the acceptance window. But as soon as the ground truth speed value approaches the injected value, the alarm turns off. We obtained 0.19% of false negatives on this attack.

Attacks against categorical signal: For the *brake-lights-command* signal monitoring we perform three types of test:

- Random command injection: in this the attack, the attacker substitutes the real command with a (0/1) random command.
- Inverse command injection: in this attack, the attacker inverts to the real command.
- Denial of service (force to 0): in this attack, the attacker always sends the 0 command value.

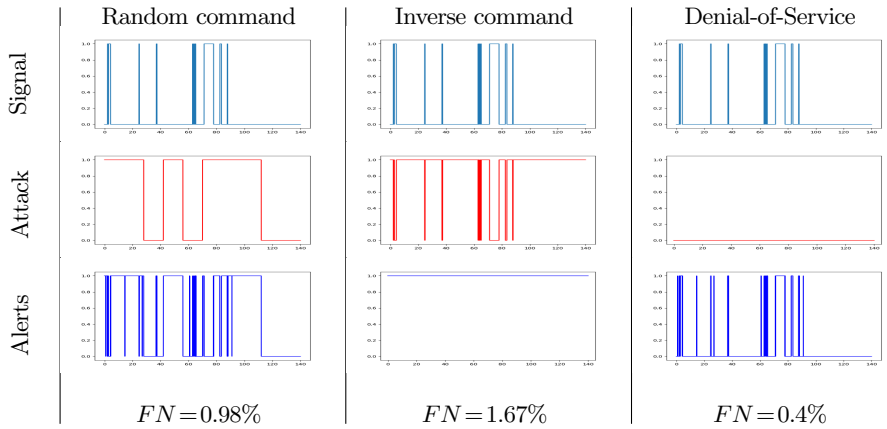


Fig. 8: Alerts raised by the decision tree (depth=40) detection rule tested on three different attacks on the *brake-lights-command* signal. On top is the ground truth command, in the middle is the attack command and on the bottom is the Alerts raised by the detection rule when receiving the attack signal.

Figure 8 shows the attack use-cases on the *brake-lights-command* signal. Note that the detection rule is set to raise an Alert as long as the received command value (injected by the attacker) differs from the predicted command. Thus we consider that an attack is happening if the injected command signal is different from the real *brake-lights-command* signal. We can see from the Alerts raised by the detection rule that:

- For the random command injection: as long as the injected command differs from the ground truth command, alerts are raised. The alert is not raised when the injected and ground truth commands are the same. We obtained a false negative rate of 0.98%
- For the Inverse command attack, we can see that, the alert is raised as soon as the attack started. In fact, since the injected command is always the opposite of the ground truth command, the predicted signal is always different from the received signal. Thus an attack is detected from the start, and we obtained a false negative rate of 1.67%
- For the Denial of service attack, the injected command is set to 0. The ground truth *brake-lights-command* have occurrences of about 70% and 30% for 0 and 1 respectively. Thus, we consider that there is an attack only 30% of the time. Similarly, the alerts were raised when the injected command differs from the ground truth command. We obtained a false negative rate of 0.4%

5 Conclusion

In this article we introduced a novel in-vehicle intrusion detection system capable of detecting an attacker with full control over an ECU. This intrusion detection system is based on detection rules built with supervised machine learning techniques. The rules learn nominal behavior of the system and make predictions for individual signal value. Alarms are raised when the predicted signal value is not similar to the received value. We showed first the effectiveness of the detection rules for separate drivers, then for a small set of drivers. We also showed the effectiveness of the detection rules against examples of attacks. The advantage of the proposed method relatively to previous work is that it only needs collected data to learn nominal behavior, and does not need examples of attacks in order to recognize them. Plus, it gives the ability to target individual signals (for instance most safety critical). Since the detection rules are actually signal predictors, theoretically the approach could be used for prevention as well. One may consider the false positive rate of 1% not low enough given the high number of frames used within the communication buses. For this purpose we can account for successive alerts as a remedy. In fact, in order to effectively influence the behavior of the car, the attacker needs to send successive attack frames. Thus, we can consider that an isolated detection alert could be ignored, and focus on successive alerts. This technique can tremendously reduce the number of false positives.

References

1. P Borazjani, C Everett, and Damon McCoy. Octane: An extensible open source car security testbed. In *Proceedings of the Embedded Security in Cars Conference*, 2014.
2. Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.
3. Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927. USENIX Association, 2016.
4. Ian D Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. Fast and vulnerable: A story of telematic failures. In *WOOT*, 2015.
5. Kyusuk Han, André Weimerskirch, and Kang G Shin. Automotive cybersecurity for in-vehicle communication. In *IQT QUARTERLY*, volume 6, pages 22–25, 2014.
6. Oliver Hartkopp, Cornel Reuber, and Roland Schilling. MaCAN - Message Authenticated CAN. In *Escar Conference, Berlin, Germany*, 2012.
7. Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive CAN networks–practical examples and selected short-term countermeasures. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2008.
8. Abdulmalik Humayed and Bo Luo. Using ID-Hopping to Defend Against Targeted DoS on CAN. In *Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles*, pages 19–26. ACM, 2017.
9. ISO. ISO 26262-5:Road vehicles – Functional safety – Part 5: Product development at the hardware level. *International Organization for Standardization*, 2011.
10. Min-Joo Kang and Je-Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016.
11. Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

12. Han Kyusuk, Weimerskirch Andr, and G. Shin Kang. A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier. In *Embedded Security in Cars, Escar Europe*, 2015.
13. George Loukas, Tuan Vuong, Ryan Heartfield, Georgia Sakellari, Yongpil Yoon, and Diane Gan. Cloud-based cyber-physical intrusion detection for vehicles using deep learning. *IEEE Access*, 6:3491–3508, 2018.
14. Mirco Marchetti and Dario Stabili. Anomaly detection of CAN bus messages through analysis of ID sequences. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 1577–1583. IEEE, 2017.
15. Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015, 2015.
16. Michael Müter, André Groll, and Felix 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.
17. Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. OBD_SecureAlert: An Anomaly Detection System for Vehicles. In *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.
18. Dennis K Nilsson, Ulf E. Larson, and Erland Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5. IEEE, 2008.
19. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
20. Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. Frequency-based anomaly detection for the automotive CAN bus. In *Industrial Control Systems Security (WCICSS), 2015 World Congress on*, pages 45–49. IEEE, 2015.
21. Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 130–139. IEEE, Oct 2016.