A Data-Driven Minimal Approach for CAN Bus Reverse Engineering

Alessio Buscemi*, German Castignani†, Thomas Engel* and Ion Turcanu†
*Faculty of Science, Technology and Medicine (FSTM), University of Luxembourg
†Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg
{alessio.buscemi,thomas.engel,ion.turcanu}@uni.lu german.castignani@ext.uni.lu

***Abstract*—Current in-vehicle communication systems lack security features, such as encryption and secure authentication. The approach most commonly used by car manufacturers is to achieve security through obscurity – keep the proprietary format used to encode the information secret. However, it is still possible to decode this information via reverse engineering. Existing reverse engineering methods typically require physical access to the vehicle and are time consuming. In this paper, we present a Machine Learning-based method that performs automated Controller Area Network (CAN) bus reverse engineering while requiring minimal time, hardware equipment, and potentially no physical access to the vehicle. Our results demonstrate high accuracy in identifying critical vehicle functions just from analysing raw traces of CAN data.**

***Index Terms*—CAN Bus, Automated Reverse Engineering, In-Vehicle Networks, Signal Identification, Machine Learning**

I. Introduction

The evolution of automotive networks has mainly been shaped by the very stringent communication requirements of Electronic Control Units (ECUs) – embedded devices that control different parts of a vehicle's electronic system. This led to the development of dedicated network architectures and protocols able to meet these very specific requirements, such as Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST), and FlexRay [1]. Of these, CAN is the most popular and is considered the de facto standard for in-vehicle communication [2]. Communication on CAN bus inside today's automotive networks is typically not encrypted, and there is no protocol for secure ECU authentication. When an ECU sends a message or *frame* on the bus, it is received by all the ECUs connected to the same bus. An adversary capable of compromising any of the ECUs attached to the CAN bus can send fake messages to any other ECU in the network, hindering its correct operation.

The main reason for this lack of security mechanisms is that automotive networks used to be isolated from the outside world. This however has changed with the evolution of connected vehicles, which, on the one hand, enables new safety applications based on Vehicle-to-Everything (V2X) communications, while, on the other hand, exposing the in-vehicle network to external threats via remote access. The most prominent example is the attack performed by Miller and Valasek [3] in 2015, when the authors managed to drive a Jeep Cherokee off the road by remotely injecting messages from the Telematic Control Unit and In-Vehicle Infotainment systems.

Fortunately, an adversary gaining access to the in-vehicle network cannot directly perform a targeted attack. In fact, despite its lack of encryption and authentication protocols, the communication on the CAN bus is typically encoded according to a specific format designed by the car manufacturer. This format is proprietary to the manufacturer and the only way to disclose it is through *reverse engineering*. Manual reverse engineering [4] can only be performed with physical access to the vehicle, is typically time consuming, and does not scale. In recent years, several tools that achieve varying degrees of automation of the reverse engineering process, thus cutting the time and effort required, have been presented [5]–[9]. The reason the research community is interested in speeding up the reverse engineering process is both to demonstrate its feasibility from an attacker's perspective, and to allow researchers and car manufacturers discover potential vulnerabilities faster in order to design better defence mechanisms. However, these methods follow an "intrusive" approach in order to collect data from the CAN bus – they require physical access to the vehicle, installation of hardware, and request of diagnostic messages from the OBD-II port.

In this paper, we present a minimal approach towards fully-automated CAN reverse engineering and propose Critical Signals Identifier (CSI), a method that requires as input only the raw CAN data of the vehicle, collected by passively reading the CAN bus over a few seconds of driving. The proposed solution is validated on a dataset containing real logs of CAN data. The main novelties introduced by CSI are the division of signals into categories based on their length, the use of categorical features to represent CAN signals, and the employment of Integer Linear Programming (ILP) to refine the predictions made by the Machine Learning (ML) models. The proposed method represents a useful tool for researchers and companies that work on CAN bus security or exploit this data to provide manufacturer-independent aftermarket solutions, as it requires minimal time, equipment and manual effort compared to other state-of-the-art solutions.

II. Background and Related Work

A standard CAN data frame is composed of several fields, as illustrated in Figure 1. Among these, the two most relevant for this study are the CAN identifier and data fields. The former uniquely identifies a CAN frame and indicates its priority, while the latter contains one or more *signals*, i.e., the actual

1	11	1	2	4	0..64	16	2	7
SOF	CAN ID	RTR	Reserved	DLC	Data Field	CRC	ACK	EOF

Figure 1. Standard CAN frame structure. The numbers represent the bits dedicated to each field.

data to be carried. A signal corresponds to a chunk of data that encodes a vehicle function. The length of a signal is fixed, as well as its position within the payload.

The goal of reverse engineering is to unequivocally locate the position of the signals inside the frames, identify their meaning, and interpret their values. In the manual approach, this is achieved by generating events that trigger ECU responses in order to spot differences with respect to the CAN normal traffic. This is often achieved by collecting diagnostic information when injecting messages with special IDs, called PIDs, in the OBD-II interface. OBD-II is a port present in all vehicles, whose main use is the check of emissions-related parameters.

The automated reverse engineering approach correlates data recorded on the CAN bus to ground truth provided by GPS, Inertial Measurement Unit (IMU) sensors installed in the vehicle, and/or the use of OBD-II PIDs. In addition, almost all algorithms, and especially those based on ML, make use of data from previously manually reverse engineered vehicles. The pipeline of these methods is typically divided into two main phases: (i) *tokenization*, which is the process of segmenting CAN frames payload into tokens, and (ii) *translation* – the process of decoding the content of the tokens, such as their meaning (i.e., in terms of telemetry or vehicle function). Note that a token, also described as a signal whose function and format have yet to be identified, is represented by its boundaries (start/end bit of the frame) and the order of the bytes in which it is located, called *endianness*. In this work, we focus mainly on the translation phase and assume the tokenization has already been performed (e.g., by using the algorithm proposed in [6]).

Jaynes et al. [5] propose a ML-based approach to identify the sender ECUs of CAN frames by analyzing the payload of the frames themselves. The authors train several classifiers to identify the content of messages concerned with five distinct car functions. The samples given as input to the classifiers are the data contained in chunks of one byte parsed to numerical values, obtained from the payload itself. The models have been tested on nine distinct car models, giving a top F-Score of 84.4 % with a Random Forests (RF) classifier.

Marchetti and Stabili [6] propose READ, a tokenization algorithm based on the comparison of the bit flip rates of consecutive bits within a CAN frame over a given period. This algorithm labels the signals according to three categories: physical, counters and cyclic redundancy check. READ was validated on a synthetic trace and 25 traces related to a total of 14 hours of driving sessions. On such a test set, the algorithm was able to correctly identify the type and boundaries of nearly 200 signals with precision greater than 90 %. However, this work focuses entirely on the tokenization phase and does not decode the signals themselves.

Verma et al. [7] introduce ACTT, which tokenizes and translates the signals matching the information obtained by injecting diagnostic messages through the OBD-II port. Like READ [6], ACTT identifies only the signals whose bits flip during the data collection. Testing ACTT on a single vehicle, the authors found that $\approx 70\,\%$ of its CAN traffic corresponds to constant bits. They could correctly identify the meaning of 16.8 % of the total bits in the trace.

Pesé et al. [9] propose LibreCAN, a tool that performs both tokenization and translation phases. The tokenization is achieved through a modified version of READ [6], able to identify a wider set of token labels according to their type. The translation is achieved by cross-correlating the CAN raw trace with data collected from OBD-II PIDs and IMU data collected with a phone aligned to the vehicle. LibreCAN managed to fully decode 24 signals for each of the four considered vehicles from the same car manufacturer with a top precision and recall of respectively 82.6 % and 44.1 %. However, LibreCAN makes use of injected diagnostic messages and IMU data, obtained with a tool expressly installed in the vehicle. Also, the tool was tested on car models from the same manufacturer, thus it is not clear whether this algorithm would perform as well on vehicle models from a different car manufacturer.

Unlike most of these works, this paper proposes an automated non-intrusive CAN reverse engineering approach that requires only raw CAN data as input. In particular, our approach does not require any additional data to be collected, such as from OBD-II PIDs or IMU data, which allows our solution to be potentially used on remotely collected CAN data.

III. Critical Signals Identifier

The goal of CSI is to identify a set of signals, referred to as *foreground set*, among a wider set of signals, the *background set*, from a trace of raw CAN data. The foreground set is composed of 10 signals, each representing a vehicle function, that we deem of primary relevance in a vehicle and critical for its safe operation. These functions are: engine speed, vehicle speed, wheel speeds, steering wheel angle, steering wheel side, throttle pedal position, battery voltage, engine coolant temperature, engine oil temperature, and odometer.

Figure 2 illustrates the entire pipeline of our proposed solution. The CSI classification task is performed by ML models, trained on known data related to previously reverse engineered vehicles, which are applied to a trace of raw CAN data extracted from the vehicle we want to reverse engineer. It should be noted that, in this work, we assume the raw data is already divided into tokens (e.g., by applying the READ algorithm [6] or its modified version in [9]) before running the CSI algorithm.

A. Signal Categories

We assume signals representing the same vehicle functions have identical or similar length across vehicle models, due to the intrinsic amount of information they need to encapsulate. To validate this assumption, we conducted an analysis on a set of Database CAN (DBC) files related to 477 unique vehicle models provided by Xee [10]. A DBC file contains fundamental

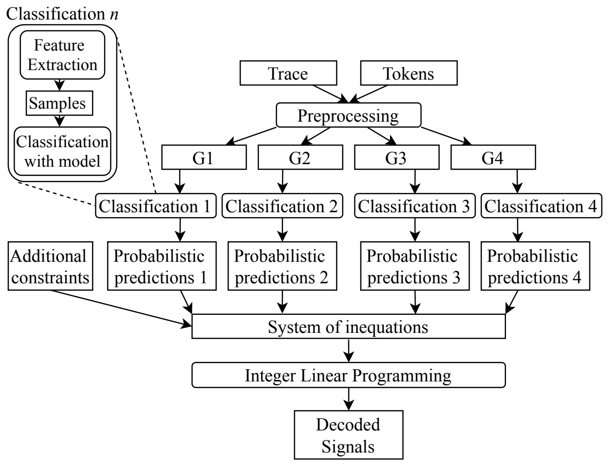


Figure 2. Pipeline of CSI.

information regarding the reverse-engineered CAN signals of a certain vehicle model. We discovered that about half of the signals related to the same telemetry have exactly the same length in all the vehicles. In addition, all considered vehicle functions but two have a Coefficient of Variation (CV) related to their length less than 0.5. CV is a normalized measure of the dispersion of frequency distribution, computed as standard deviation to mean ratio ($\frac{\sigma}{\mu}$). CV < 1 indicates a low variance, while CV ≥ 1 indicates a high variance.

As a result, we decided to divide the signals into four groups, according to their length: *G1* (1–4 bits), *G2* (5–10 bits), *G3* (11–17 bits), and *G4* ($\geq$ 18 bits). We tackle the reverse engineering process as four different classification tasks. The goal is to design features that better fit each group and build a more accurate model for the classification task.

B. Feature Extraction

The aim of a supervised ML classifier in the scope of CAN bus signal translation is to identify for each token the vehicle function that it represents. The classifier is trained to associate a sample representing each token with its vehicle function. Each sample is a combination of feature values calculated for a token. A feature describes a unique characteristic of a vehicle function. The features should make signals related to the same function recognizable across different vehicle models while differentiating them from other signals.

The main difficulty in finding good features for CAN signals is that most of them are directly or indirectly highly influenced by environmental factors, such as road/traffic conditions and the driving style. The risk is of identifying features that are not related to the intrinsic nature of the signal but, instead, are strongly influenced by the environmental factors. In related work, sensors such as GPS and IMU provide some useful ground truth about the driving context. For instance, by cross-correlating the speed pattern recorded by a GPS dongle with a set of tokens, it is possible to identify the signals related to the vehicle speed. In our use-case scenario, we do not have access to such ground data to match the CAN raw trace with. Therefore, we cannot make the signals comparable by scaling their values, given the absence of a reference truth.

To identify the set of features, we initially tried to exploit the correlation among signals across three domains: frame priority (inferred from the CAN ID), frame sending frequency, and payload. However, our preliminary analysis of the dataset revealed that the first two domains do not present high enough correlation values. For this reason, we focus only on the payload carried by each signal. The numerical values carried by the payload of each signal strongly depend on contextual factors and do not constitute a good base for fingerprinting the vehicle functions. For example, a model trained on traces related to cars going at most at 30 km/h is unlikely to recognize the vehicle speed in a trace from a car reaching 100 km/h.

To minimize the impact of these factors, we design features that are categorical. These features describe either (i) the presence or absence of a property (e.g., twinned or not), or (ii) a categorization or ranking of the signals inside the same trace with regard to a certain aspect (i.e., dynamicity rankings). These features fit the vehicle functions independent of the numerical values carried by the signals representing them.

We define a total of eight features, which require several processing steps to be generated from the CAN raw trace:

- **Number of Static Bits:** Unless the vehicle is pushed to its extreme performance with regard to a telemetry, some bits of the signal carrying this telemetry will never flip. The number of unchanging bits helps to identify signals carrying these telemetries.
- **Length:** The length of the signal itself is used as a feature.
- **Dynamicity Type:** This feature categorizes the behaviour of the signal in time. We define four signal types: status, counter, physical or other.
- **Short Signal Dynamicity Ranking:** It is related to short status signals. The feature ranks the dynamicity of the signals from “Low” to “High” based on the bit flip rate percentiles calculated taking into consideration signals of the same group and dynamicity type found in the trace.
- **Long Signal Dynamicity Ranking:** Same as Short Signal Dynamicity Ranking, but related to long physical signals.
- **Twinned:** Two or more signals within the same frame are twinned if they display a similar bit flip rate.
- **Stopping Dynamicity:** We assume the vehicle to reverse engineer is driven for a certain amount of time until it stops. We identify the parts of the trace related to the two phases of the driving session (driving and not driving) and find the speed-related signals whose values decrease down to the idle value. This feature encapsulates the temporal order in which these signals reach their idle status.
- **Idle Status:** This feature captures the idle status of the signals (i.e., their value), when the vehicle is not moving.

The samples of each group of signals are composed of a subset of these features, as shown in Table I.

C. Classification

Once the samples are generated for each token according to the features of each group, they are passed to an ML model

Table I
FEATURES FOR EACH GROUP OF SAMPLES.

Feature	**G1**	**G2**	**G3**	**G4**
Number of Static Bits		✓	✓	
Length	✓			✓
Short Signal Dynamicity Ranking	✓			
Long Signal Dynamicity Ranking		✓	✓	✓
Dynamicity Type	✓	✓	✓	✓
Stopping Dynamicity		✓	✓	
Idle Status		✓	✓	
Twinned			✓	

for the classification task, as shown in Figure 2. There are four ML models, each trained on samples of one of the four groups extracted from traces of previously reverse engineered vehicles. Our analysis of the dataset revealed that very rarely a CAN trace contains multiple instances of the same signal across frames with different IDs. It is, indeed, in the interest of manufacturers to avoid redundancy in a network which is already constrained by its low bandwidth. As a consequence, we can make some assumptions about the number of tokens that can be labelled as a certain signal in a trace. We define this number as the *cardinality* of the vehicle function. For example, we can assume that in every vehicle there can be at most four signals related to the wheel speeds. Therefore, the cardinality of wheel speed is four.

To the best of our knowledge, no classifier can be set to take into account the constraints during the prediction. Since the number of samples per class is not known a priori, in case of misclassification, a standard classifier would attribute the same label to an arbitrary number of tokens.

To overcome this limitation, instead of performing deterministic predictions, we set the classifier to output a list of probable labels/functions for each token. Subsequently, CSI converts each function predicted for each token into a variable, and the probability associated with it into a coefficient associated with the variable. All the variables are inserted into a system of inequalities, describing the constraints related to the cardinality of the vehicle functions. Then, the system is solved as a ILP problem, described as follows:

$$\text{maximize} \sum_{t\in T}\sum_{f\in F}\alpha_{t,f}p_{t,f} \tag{1}$$

$$\text{subject to:} \sum_{t\in T} p_{t,f} \le c_f, \forall f \in F \tag{2}$$

$$\sum_{f\in F} p_{t,f} \le 1, \forall t \in T \tag{3}$$

$$0 \le \alpha_{t,f} \le 1 \tag{4}$$

$$c_f \ge 1 \text{ integer} \tag{5}$$

$$p_{t,f} \in \{0,1\} \tag{6}$$

where T represents the set of tokens, F is the set of vehicle functions, $\alpha_{t,f}$ indicates the probability that token t contains the vehicle function f as predicted by the classifier, c_f represents the cardinality of the vehicle function f, and $p_{t,f}$ is a binary function defined as:

$$p_{t,f} = \begin{cases} 1, & \text{if the vehicle function } f \text{ is ultimately} \\ & \text{assigned to token } t \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

The constraints described by Equation (2) ensure that at most c_f tokens get associated with a certain vehicle function f, while Equation (3) ensures that each token is ultimately predicted to contain with at most one known vehicle function.

Table II
CSI PERFORMANCE FOR DIFFERENT CLASSIFIERS COMPARED TO BASELINE.

Performance metric	**CSI**				**Baseline**
	RF	MLP	SVM	NB	RF
Accuracy [%]	91.6	93.0	91.7	89.3	12.7
Balanced Accuracy [%]	73.0	80.0	72.4	63.4	12.8
F1-Score [%]	91.7	92.8	92.1	89.2	11.5
Balanced F1-Score [%]	74.1	79.9	74.4	59.9	11.4

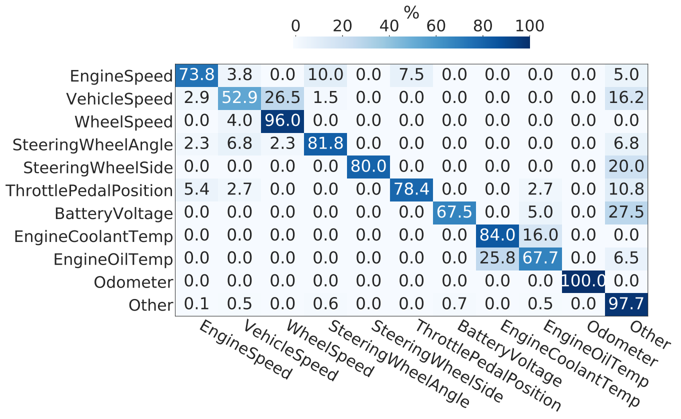
IV. PERFORMANCE EVALUATION

To evaluate the performance of our proposed solution, we used a set of 60 s-long CAN traces collected with a `PCAN-USB FD` [11] dongle connected to the CAN bus of the following vehicle models: Audi A3 2012, BMW X1 2015, Kia Sportage 2016, Mercedes A-Class 2018, Peugeot 307, 2008, Megane 4 2016, Volkswagen Golf 5 2009 and Volvo XC40 2018. The ground truth for the content of these CAN traces was obtained through manual reverse engineering. To obtain more samples to train the classifiers better, each trace is divided in 10 smaller traces of about 6 s each through windowing. A total of 2,302 signals were found in these 80 subtraces. The background set is composed of signals related to 59 different vehicle functions.

For the classification task, we tested four of the most used ML classifiers in the related work: Support Vector Machine (SVM), Naive Bayes classifier (NB), Random Forests (RF), and Multilayer Perceptron (MLP). To cross-validate CSI, we trained iteratively each classifier on samples belonging to all but one vehicles and used the samples of the remaining vehicle as a test set. To optimize the performance of each classifier, we balanced the training set with SMOTE [12], a well-known technique for over-sampling the minority classes, and we extensively tuned its hyperparameters.

The evaluations presented in the following refer to the combined predictions achieved on the different test sets. We compare the performance of CSI with the method proposed by Jaynes et al. [5], which we refer to as *Baseline*. To the best of our knowledge, this is the only related work that performs automated reverse engineering based only on raw CAN data. Table II shows the *accuracy* and *F1-score* achieved by CSI with all the considered classifiers, as well as the results obtained with the Baseline method. For the latter we show only the results achieved using RF, the best performing classifier in [5].

The accuracy is defined as the number of correct predictions divided by the number of total predictions made. The F1-Score is defined as the harmonic mean between precision and



	EngineSpeed	VehicleSpeed	WheelSpeed	SteeringWheelAngle	SteeringWheelSide	ThrottlePedalPosition	BatteryVoltage	EngineCoolantTemp	EngineOilTemp	Odometer	Other
EngineSpeed	73.8	3.8	0.0	10.0	0.0	7.5	0.0	0.0	0.0	0.0	5.0
VehicleSpeed	2.9	52.9	26.5	1.5	0.0	0.0	0.0	0.0	0.0	0.0	16.2
WheelSpeed	0.0	4.0	96.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SteeringWheelAngle	2.3	6.8	2.3	81.8	0.0	0.0	0.0	0.0	0.0	0.0	6.8
SteeringWheelSide	0.0	0.0	0.0	0.0	80.0	0.0	0.0	0.0	0.0	0.0	20.0
ThrottlePedalPosition	5.4	2.7	0.0	0.0	0.0	78.4	0.0	0.0	2.7	0.0	10.8
BatteryVoltage	0.0	0.0	0.0	0.0	0.0	0.0	67.5	0.0	5.0	0.0	27.5
EngineCoolantTemp	0.0	0.0	0.0	0.0	0.0	0.0	0.0	84.0	16.0	0.0	0.0
EngineOilTemp	0.0	0.0	0.0	0.0	0.0	0.0	0.0	25.8	67.7	0.0	6.5
Odometer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0
Other	0.1	0.5	0.0	0.6	0.0	0.0	0.7	0.0	0.5	0.0	97.7

Figure 3. Confusion matrix of the predictions performed by CSI with MLP.

recall. Due to the presence of the background set, consisting of numerous signals, and to the fact that the foreground signals are present in different number in different vehicles, the test sets are imbalanced. As a consequence, the accuracy and F1-Score are influenced by the number of elements in each class given as input to the model. In this case, the results are heavily conditioned by the performance on the background set, as its number of samples is consistently higher than any other class. For this reason, we also report the balanced (or weighted) accuracy and F1-Score.

Our approach significantly outperforms the state-of-the-art solution in all the considered metrics. The main reason for the poor performance achieved by [5] is that the samples are generated from all the bytes contained in the payload of a frame. The authors do not consider the possibility that a CAN frame can contain multiple signals. Therefore, the sample generation is negatively affected by the presence of unrelated chunks of payload that are associated to the signals.

Figure 3 illustrates the normalized confusion matrix, offering a complete view of the predictions made for each class by CSI with MLP, which is the best performing algorithm. The x-axis represents the predicted labels, while the y-axis the actual labels. The label "Other" corresponds to the background set. The main diagonal shows the rate of correctly classified samples for each class. Since the confusion matrix is normalized, the overall accuracy is calculated as the mean of these rates. The other slots report the rate of misclassification for each class (the rows being the false negatives and the columns the false positives).

The difference in the performance achieved by CSI for different signals can be explained by (i) the different quality of fitting provided by the features for each vehicle function and (ii) the presence of signals related to more or less similar vehicle functions. For instance, the Engine Oil Temperature is mostly misclassified with Engine Coolant Temperature and vice versa. This indicates a high correlation between these two vehicle functions, which leads to a higher difficulty in correctly distinguishing them.

Our results show that CSI is able to identify with good accuracy safety-relevant signals on a CAN bus among a wide set of signals. The diverse set of tested vehicles argues in favour of the universality of this method. Moreover, since CSI does not require any manual intervention in the reverse engineering process, the execution time of the algorithm is much faster compared to other state-of-the-art approaches, such as [9].

V. Conclusion

In this paper we have presented CSI, an automated CAN bus reverse engineering algorithm that only requires raw CAN data as input. We validated the proposed solution on a dataset of real CAN traces collected from eight driving vehicles. Our solution is characterized by a minimal approach in terms of employed hardware and execution time. Using a combination of ML and ILP, CSI can automatically identify safety-relevant vehicle functions with an accuracy up to 93 %.

Future work includes the design of an accurate tokenization algorithm, which in this work was solely simulated. In addition, our method identifies the signals without extracting the actual value included in these signals. One potential improvement could include a method to identify the parameters (e.g., scaling factor) needed to extract the actual physical values. Finally, new features can be designed to improve the classification accuracy and extend the number of identified signals.

Acknowledgement

We acknowledge support from the National Research Fund (FNR) under grant number PRIDE15/10621687. We thank Xee for the provided datasets we used to validate our solution.

References

[1] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.

[2] International Organization for Standardization, "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," ISO 11898-1, Dec. 2015.

[3] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," p. 91, 2015.

[4] C. Quigley, D. Charles, and R. McLaughlin, "CAN Bus Message Electrical Signatures for Automotive Reverse Engineering, Bench Marking and Rogue ECU Detection," in *SAE Technical Paper*, SAE International, Apr. 2019.

[5] M. Jaynes, R. Dantu, R. Varriale, and N. Evans, "Automating ECU identification for vehicle security," in *15th International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2016, pp. 632–635.

[6] M. Marchetti and D. Stabili, "READ: Reverse engineering of automotive data frames," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1083–1097, 2018.

[7] M. Verma, R. Bridges, and S. Hollifield, "ACTT: Automotive CAN tokenization and translation," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2018, pp. 278–283.

[8] M. R. Moore, R. A. Bridges, F. L. Combs, and A. L. Anderson, "Data-Driven Extraction of Vehicle States From CAN Bus Traffic for Cyberprotection and Safety," *IEEE Consumer Electronics Magazine*, vol. 8, no. 6, pp. 104–110, 2019.

[9] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin, "LibreCAN: Automated CAN Message Translator," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2019, pp. 2283–2300.

[10] Xee. (2020), [Online]. Available: https://www.xee.com/en/ (visited on 05/27/2020).

[11] PEAK System. (2020), [Online]. Available: https://www.peak-system.com/PCAN-USB-FD.365.0.html (visited on 06/03/2020).

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.