ECU-Secure: Characteristic Functions for In-Vehicle Intrusion Detection

Yannick Chevalier, Roland Rieke, Florian Fenzl, Andrey Chechulin, and Igor Kotenko

Abstract Growing connectivity of vehicles induces increasing attack surfaces and thus the demand for a sophisticated security strategy. One part of such a strategy is to accurately detect intrusive behavior in an in-vehicle network. Therefore, we built a log analyzer in C that focused on payload bytes having either a small set of different values or a small set of possible changes. While being an order of magnitude faster, the accuracy of the results obtained is at least comparable with results obtained using standard machine learning techniques. Thus, this approach is an interesting option for implementation within in-vehicle embedded systems. Another important aspect is that the explainability of the results is better compared to deep learning systems.

Key words: Controller area network security, Intrusion detection, Anomaly detection, Machine learning, Automotive security, Security monitoring

1 Introduction

Information Technology (IT) security and data protection are enabling factors for newly emerging intelligent distributed computing ecosystems such as the Internet of vehicles [15]. Increasing attack surfaces in intelligent autonomous vehicles are

Yannick Chevalier
Paul Sabatier University, Toulouse, France, e-mail: yannick.chevalier@irit.fr

Roland Rieke
Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany,
e-mail: roland.rieke@sit.fraunhofer.de

Florian Fenzl
University of Applied Sciences Mittelhessen, Giessen, Germany,
e-mail: florian.fenzl@mni.thm.de

Andrey Chechulin, Igor Kotenko
SPIIRAS, St-Petersburg, Russia, e-mail: chechulin|ivkote@comsec.spb.ru

inevitably caused by a strong interconnectedness of vehicles and the requirements for external information sources and services. Despite the complexity of modern vehicles with more than 100 interconnected Electronic Control Units (ECUs) and more than 100 million lines of code, an imperative need is that the vehicle cannot be controlled remotely by an attacker. Attacks based on remotely injecting messages to safety critical ECUs in order to influence physical actions such as steering and braking has already been demonstrated in [14]. It is thus very important to improve security of in-vehicle networks and as long as there are no effective means to prevent specific attacks, there should be methods in place to automatically detect them and react to the alerts. In principle, the detection of anomalies in the network traffic inside a vehicle that may be caused by intruders could be done remotely by sending all internal traffic to a security operations center. However, this would not only be a serious problem with respect to privacy concerns and regulations, it would also be inefficient, cause high costs and maybe not fulfill real-time reaction requirements.

Thus, in this work we consider a new in-vehicle anomaly detection method which aims at solving four requirements: It should (1) be as accurate as possible, because false alarms may lead to unnecessary degradation of vehicle usability. It should (2) be as resource efficient as possible. It should (3) not require hardware changes and not require additional third party software libraries because these may be not available for vehicle-specific embedded systems or have licenses which would require to make the in-vehicle software open source. Finally (4) the results of message evaluation with respect to anomalies should be explainable, in order to be able to judge about possible counteractions.

In order to address these requirements, we propose a logical analysis method which we compare with a simple neural network based methods which could probably be applied in embedded systems inside vehicles. We aim at better accuracy, faster and more resource efficient characterization of messages, portability to embedded systems without dependencies on libraries such as tensorflow, and rule-based reasoning so the results of message evaluation with respect to anomalies can be back-traced to the responsible rules. We evaluate the proposed method on data sets from the Controller Area Network (CAN) bus which is the standard solution for in-vehicle communication between ECUs.

Section 2 gives an overview on the background and related work. Section 3 introduces data sets from two different vehicles that have been used to evaluate the proposed method. Section 4 describes some results from tests with neural networks in order to provide a benchmark for our work. Section 5 presents the principles of the characteristic function approach while Sect. 6 describes its implementation and the results of various detection setups. Finally, Sect. 7 concludes this paper.

2 Background and Related Work

The most important topic in vehicle security should be the design of secure architecture, protocols, hardware and software. Hardening the system by reducing the

attack surface is a common incremental approach. For example, [21, 25] list possible intrusion points together with proposals for countermeasures such as cryptography, anomaly detection and ensuring software integrity by separating critical systems. However, most of the currently discussed intrusion prevention measures require hardware changes, thus conflicting with backwards-compatibility. Therefore, it has been proposed by some actors that, in the CAN context, intrusion detection should be used in a defense-in-depth approach [6]. CAN intrusion detection methods can be grouped into four categories, namely, detecting ECU impersonating attacks, detecting specification violations, detecting message insertions, and detecting sequence context anomalies. The work on detection of ECU impersonating attacks such as [5, 3] in most cases uses some kind of physical fingerprinting by voltage or timing analysis with specific hardware. This work seeks to mitigate the general problems of missing authenticity measures in CAN bus design and thus is complementary to the work presented in this paper. The detection of specification violations assumes that the specification of normal behavior is available and thus there is an advantage that no alerts based on false positives will be generated. Specification based intrusion detection methods can use specific checks, e.g. for formality, protocol and data range [11], a specific frequency sensor [8], a set of network based detection sensors [16], or specifications of the state machines [20]. The detection of message insertions can be based on different technologies such as analysis of time intervals of messages [19] or LSTM [24]. The methods for detection of sequence context anomalies comprise process mining [18], hidden Markov models [12, 17], OCSVM [23], neural networks [9], and detection of anomalous patterns in a transition matrix [13]. Comparisons of different Machine Learning (ML) algorithms are given in [4], [22], and [2]. OCSVM, Self Organizing Maps, and LSTM are used in [4] while LSTM, Gated Recurrent Units (GRU) and Markov models are used in [22]. OCSVM, SVM, sequential neural networks and LSTM are used in [2]. A detailed review on intrusion detection systems for in-vehicle networks is given in [1].

Our training sets and ML benchmark described in Sect. 3 are based on [2, 7]. We consider the method proposed in this paper in particular with respect to payload analysis to be more accurate than the methods based on SVM, OCSVM and similar approaches and to be much faster and more resource efficient than the methods based on different kinds of neural networks.

3 Simulated In-Vehicle Attacks

For the evaluation of our work we used five different data sets from two different vehicles, namely, $HCLR_{DoS}$, $HCLR_{fuzzy}$, $HCLR_{gear}$, ZOE_{fuzzy}, and $ZOE_{payload}$. We mapped the relevant data of the CAN messages to the following tuple structure: $(time, ID, dlc, p_1, \ldots, p_8, type)$, where *time* is the time when the message was received, *ID* comprises information about the type and the priority of the message, *dlc* (data length code) provides the number of bytes of valid payload data, and $p1, \ldots, p8$

is the payload. The messages are labeled by *type* (attack versus no attack). Three of these data sets that we used have been published by the "Hacking and Countermeasures Research Labs" (HCRL) [7]. The $HCLR_{DoS}$ data set contains DoS attacks. For this attack type, every 0.3 milliseconds a message with the ID "0000" is injected (cf. m2 in Table 1). Conversely, in the $HCLR_{fuzzy}$ data set every 0.5 milliseconds a random message is injected (cf. m3 and m4 in Table 1). The $HCLR_{gear}$ contains spoofing attacks, where every millisecond a message with an ID related to gear is injected, whereby payload does not change (cf. m5 and m6 in Table 1).

The *ZOE* data set with about 1 million messages has been collected from a 9 minutes drive with a Renault Zoe electric car in an urban environment. It has been used before to evaluate process mining [18] as well as ML methods [2]. The *ZOE* data set contains 110 different IDs which makes it much more difficult to reach a good detection accuracy compared to the relatively simple *HCRL* data sets with maximal 38 different IDs. The *ZOE* data set originally contains no attack data but for this work we have created a merged data set ZOE_{fuzzy} by injecting 32000 generated messages with random ID and payload, similar to the $HCLR_{fuzzy}$ data set. We have also created another data set $ZOE_{payload}$ where we similarly injected 10000 generated messages with randomly generated payload but only with message IDs which have been used in the original data set. Thus, these messages are more difficult to detect.

Table 1 Exemplary messages in data sets HCRL and ZOE

Message	Time	ID	Length	p1	p2	p3	p4	p5	p6	p7	p8	Type	Comment
m1	0.851863	1264	8	0	0	0	128	0	105	209	19	1	HCRL: normal msg
m2	0.852103	0	8	0	0	0	0	0	0	0	0	-1	HCRL: DoS attack
m3	0.972222	1869	8	68	51	82	16	80	85	48	212	-1	HCRL: fuzzy attack
m4	0.982961	1139	8	148	217	62	32	201	26	23	44	-1	HCRL: fuzzy attack
m5	1.348859	1087	8	1	69	96	255	107	0	0	0	-1	HCRL: gear attack
m6	1.349963	1087	8	1	69	96	255	107	0	0	0	-1	HCRL: gear attack
m7	0.010919	504	8	248	4	255	239	254	0	10	13	1	ZOE: normal msg
m8	0.015625	1656	8	243	99	108	24	188	209	74	171	-1	ZOE: fuzzy attack

4 Baseline Benchmark: Neural Network

As a benchmark for the evaluation of our approach we used a neural network with a structure that is based on a `Sequential` model from the `keras.models` package. Neural networks are the standard for deep learning and can model very complex nonlinear relationships. A fully connected neural network utilizes a number of layers with each layer supporting an arbitrary number of neurons. Data is propagated from

the input to the output layer using weighted connections between the neurons of these layers. In order to get a very high accuracy, we used two dense layers with 50 neurons each for the training data sets (trainable parameters: 3,201) and processed the learning phase in 20 epochs. Tensorflow 11 with Binary Crossentropy, Adam and Accuracy was used as loss, optimizer and performance metric. The validation of the neural network was done with a standard train/test split of the original data. In case of the ZOE_{fuzzy} and $ZOE_{payload}$ data sets we used random attacks initialized with different seed for training and detection phases.

Table 2 Neural network results (Data set: log-file with simulated attacks; TP/FP/TN/FN: true/false positives/negatives; Precision (Positive Predictive Value) $PPV = TP/(TP+FP)$; Recall (True Positive Rate) $TPR = TP/(TP+FN)$; Accuracy $ACC = (TP+TN)/(TP+TN+FP+FN)$; Train: time for training the model on i7; i7: time for running detection on i7; ARM: time for running detection on ARM; Real: elapsed time in log-file).

Data set	TP	FP	TN	FN	ACC	PPV	TPR	Train	i7	ARM	Real
$HCRL_{DoS}$	587521	0	3078250	0	100%	100%	100%	5m6s	14m37s	144m46s	47m
$HCRL_{fuzzy}$	491829	118	3346895	18	99.99%	99.97%	99.99%	5m22s	15m8s	153m14s	91m
$HCRL_{gear}$	597252	136	3845754	0	99.99%	99.97%	100%	6m10s	16m57s	175m7s	133m
ZOE_{fuzzy}	31050	135	1012856	950	99.89%	99.56%	97.03%	1m23s	4m4s	41m31s	9m
$ZOE_{payload}$	9284	964	1012027	716	99.83%	90.59%	92.84%	1m26s	4m2s	40m21s	9m

The results in Table 2 show an accuracy above 99.83% for all used data sets. Neural networks have the drawback of needing anomalous training data to be useful. Training for all known attacks is very difficult because the training set has to cover many variations of attack types. Furthermore, usually a data set covering all possible 'normal' behaviors of vehicles is not available.

The training of the models has been done on a laptop with Intel i7-8550U CPU. The detection tests have been executed on the same laptop and also on a Raspberry Pi with ARMv7 CPU. In order to run the detection in real-time within an in-vehicle network it is a necessary precondition that the detection is faster than the real elapsed time as listed in the last column of Table 2. By comparison with the detection times on a Raspberry Pi (cf. column *ARM*), it is evident that an IDS using these neural network models would not be fast enough on an embedded system with such processing power.

5 Principles of the Characteristic Function Approach

We aim at synthesizing a formal model for the network from a log of events. In contrast with other tentatives we focus on the fact that every good message sent in the network and thus occurring in the log is to be accepted by a device in this network. Thus and in contrast with natural systems, we know there exists, implemented in the

devices connected to the CAN bus, tests that are employed to accept only legitimate messages. Conversely, legitimate messages sent on the bus are constructed so as to pass these tests.

Under this assumption our approach consists in first choosing a set of tests that is likely relevant to the communications on the CAN bus, second in analyzing the log to keep only tests that are likely to be implemented by the devices or that are consequences on the messages' payload of the information conveyed. Third we apply the tests computed during the training phase on messages from another log.

Considerations on the tests space. The first step consists in choosing a set of simple tests that are likely to be relevant. The space of all possible tests will then be all the possible conjunctions and disjunctions of these simple tests. We model packets by an ID and a sequence of bytes, *i.e.* 256-valued integers. This ID determines a class to which each packet belongs. We assume that all packets in a given class are similar enough so that some tests exist that are valid on all messages on the class and are not vacuous.

In principle the test space encompasses all boolean functions on messages or sequences of messages. However a succinct analysis already delineates a few types of tests that may be useful for the analysis of logs:

- some tests are concerned about the syntactic content of the packet, such as the presence of a padding constant or the presence of a specific value, denoting *e.g.* a more precise type for the packet;
- some tests are computed on the whole packet, such as an error-correcting code;
- some tests are domain specific and relate to the possible evolutions of physical data between consecutive packets or the set of possible values of some data;
- some tests depend on the internal state of the devices, a packet being acceptable at some point of their execution but not at another point.

For the sake of simplicity we consider in this paper only tests performed independently on the different fields of messages, as well as on their ID and their time, translated into first a 32-bits integer, then treated as four different 1-byte fields. That is, we consider only the first and third cases of the preceding list. We detail below how the interesting fields are discovered and how we construct rules from their values.

Classes of messages. Messages are placed in classes depending on the value of their IDs. A value test or a difference test has to be valid on all messages of a given class during training to be incorporated in the monitor.

Automatic fields. A field is *automatic* if the device receiving and accepting this packet tests whether the value of the field is equal to a constant in its program. It is expected that, if different packets can be sent from one device to another, at least one automatic field exists so that the receiver can derive the type of the received packet. The statistical characteristic of such fields are that they should have only *a few* legitimate values, and that these values should have no other detectable relations.

There is obviously some arbitrariness in deciding what *a few* means. Since the tests performed are not based on any hints from the protocol, we have arbitrarily decided to define a small set of different values to be the square root of the total

number of possible different values, that is less than 16 values among the 256 possible ones. In future work we plan to adapt this choice *wrt* the number of messages in the class. Tests relevant to automatic fields are *value tests* in which we record all the different values occurring in a field during training. If the number of different values is more than 16, we perform no value test on that field during monitoring. Otherwise we verify during monitoring that the value in that field for a message is among the ones seen during training. To sum up, value tests are a conjunction, on all fields f, of a disjunction $f = v_1 \vee \ldots \vee f = v_k$ with $k \leq 16$, or of the *true* constant $\top$ if more than 16 different values have been encountered.

Physical values. These are values that are assumed to evolve slowly. For these values we assume a bound on the difference between the value present in the current packet *wrt* the value occurring in the last preceding similar packet. For these fields the analyzer keeps track of the value in the last accepted message and compares that value with the one in the current message. As in the case of value tests, these difference tests are performed during monitoring only if a small (less than 16) number of changes have been observed during the training phase. Re-using the same notation as above, but now denoting f the value of a field in the last accepted packet, and f' its value in the packet under analysis, difference tests are a conjunction, on all fields f, of a disjunction $(f' - f) = v_1 \vee \ldots \vee (f' - f) = v_k$ with $k \leq 16$, or of the *true* constant $\top$ if more than 16 different values have been encountered for the difference between the values for that field between a message and its predecessor.

Random values. There are fields for which no relation was found in the data set among the ones that were searched for. In the data sets considered, a post-analysis of the rules has shown that in several cases these fields are often related with the physical value fields, and that the data conveyed were actually 2-bytes values. The analyzer does not perform any test on these fields, as per the construction described both the value and the difference tests are reduced to the $\top$ constant for these.

6 Implementation and Evaluation of Characteristic Function Approach

Our approach consists in using tests to first classify messages into classes, and second to characterize messages in a given class by the set of tests they pass. The monitor only implements tests that are satisfied by all messages in a given class. In the experiments of Table 3 the only classification performed is on the ID field. The tool outputs, for each class, the tests that are to be performed on packets of that class. We believe this information to be very valuable for future work.

First, it permits to compute the probability that a random message satisfies all the tests in the class, and thus allows us to evaluate the robustness of the monitor against the injection of random messages. Assuming that in a given class there are n fields classified as automatic and m fields classified as physical, and that tests on fields all accept the maximum of 16 values, a random message in that class has

a probability $(\frac{16}{256})^{n+m} = 2^{-4\cdot(n+m)}$ to be accepted. This small but non-negligible probability explains the occurrences of false negatives in Table 3.

Second, given that the rules generated implement simple tests, it is also in theory possible for a human to better understand the system by looking at the rules produces, and eventually produce new (and less generic) tests beyond those described in this paper. A side result of this is that it is also quite easy to build a fake traffic that will be accepted by a monitor once we know its rules.

Third, it permits to focus further classification work on classes for which only a few fields are tested. Though this is outside the scope of this paper, a manual analysis of the rules produced and of the messages in these classes strongly suggests new test functions tailored to handle these cases.

Table 3 Results of characteristic functions approach (Data set: log-file with simulated attacks; TP/FP/TN/FN: true/false positives/negatives; Precision (Positive Predictive Value) $PPV = TP/(TP+FP)$; Recall (True Positive Rate) $TPR = TP/(TP+FN)$; Accuracy $ACC = (TP+TN)/(TP+TN+FP+FN)$; Train: time for training the model on i7; i7: time for running detection on i7; ARM: time for running detection on ARM; Real: elapsed time in log-file).

Data set	TP	FP	TN	FN	ACC	PPV	TPR	Train	i7	ARM	Real
$HCRL_{DoS}$	587521	0	3078250	0	100%	100%	100%	0.9s	1m1s	17m2s	47m
$HCRL_{fuzzy}$	491847	0	3347013	0	100%	100%	100%	1.2s	1m2s	17m48s	91m
$HCRL_{gear}$	597252	0	3845890	0	100%	100%	100%	1.2s	1m22s	20m27s	133m
ZOE_{fuzzy}	31985	0	1012991	15	99.99%	100%	99.95%	0.3s	0m19s	5m1s	9m
$ZOE_{payload}$	9910	0	1012991	90	99.99%	100%	99.10%	0.3s	0m19s	4m50s	9m

A final comment on the experiments shown in Table 3 is that we have encountered no False Positive, which shows that though it is arbitrary, the heuristic threshold of 16 is not too high as it does not classify a field that contains random values into an automatic field, *i.e.* no overfitting has been observed. This however should not be interpreted as an impossibility for our approach to suffer from over-fitting. Especially a training dataset which is too short would tend to produce illegitimate value tests, *e.g.* for the fields recording the timestamp of the packet.

Implementation. The algorithm has been implemented in C. The log is first translated if necessary into a binary file which is then mmap'd to an array of structures, each structure representing a packet. This array is then analyzed independently by different modules. Each analysis module constructs a balanced binary tree mapping an ID to the result of the analysis on this ID. The *monitor* uses this structure to parse a log file and test whether a packet shall be accepted.

Resources. During training all the log is virtually available in memory, though we rely on the operating system to optimize speed and memory consumption. By construction the memory needed by the monitor is linear in both the number of different IDs and in the number of fields.

As can be seen in Table 3, the results are very encouraging against the different attacks considered. It shall be noted that knowing the results of the analysis modules,

it is also quite easy to construct attacks (*i.e.*, add additional messages) that follow a pattern that will be accepted by the analyzer.

7 Conclusion

We have seen in previous work [2] that neural network approaches to anomaly detection deliver good results but that it is hard to implement this kind of detection in-vehicle because of restictions with respect to on-board resources of typical ECUs used in vehicular systems. Thus, we have started to analyze logs using a bind and branch approach that was very accurate but lacked robustness. From this experience we built a log analyzer in C that focused on payload bytes having either a small set of different values or a small set of possible changes. The results obtained are at least comparable with results obtained using standard ML techniques.

We will work in the near future on refining the analysis to *guess* the functions employed by the devices to test whether the packet shall be accepted. In order to evaluate our approach in a realistic context, we will further test our approach in a setup of several Raspberry Pis equipped with CAN-bus boards which use components such as the Microchip MCP2515 CAN controller with MCP2551 CAN transceiver that are typical for automotive ECUs. We furthermore plan to extend our approach to CAN with flexible data-rate (CAN-FD) which is an extension of the original CAN bus protocol with higher bandwidth. Another possible evaluation could improve the baseline neural network performance on micro-controllers utilizing optimized neural network functions such as CMSIS-NN [10].

Acknowledgements This research is partially supported by the German Federal Ministry of Education and Research in the context of the project VITAF (ID 16KIS0835).

References

1. Al-Jarrah, O.Y., Maple, C., Dianati, M., Oxtoby, D., Mouzakitis, A.: Intrusion detection systems for intra-vehicle networks: A review. IEEE Access **7**, 21,266–21,289 (2019). DOI 10.1109/ACCESS.2019.2894183
2. Berger, I., Rieke, R., Kolomeets, M., Chechulin, A., Kotenko, I.: Comparative study of machine learning methods for in-vehicle intrusion detection. In: Computer Security. ESORICS 2018 International Workshops, CyberICPS 2018 and SECPRE 2018, Barcelona, Spain, September 6-7, 2018, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 11387, pp. 85–101. Springer, Cham (2019). DOI 10.1007/978-3-030-12786-2_6
3. Cho, K., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection. In: T. Holz, S. Savage (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., pp. 911–927. USENIX Association (2016)
4. Chockalingam, V., Larson, I., Lin, D., Nofzinger, S.: Detecting attacks on the CAN protocol with machine learning (2016)

5. Choi, W., Joo, K., Jo, H.J., Park, M.C., Lee, D.H.: Voltageids: Low-level communication characteristics for automotive intrusion detection system. IEEE Transactions on Information Forensics and Security **13**(8), 2114–2129 (2018)
6. ENISA: Cyber security and resilience of smart cars. Tech. rep., ENISA (2016). DOI 10.2824/87614
7. Hacking and Countermeasure Research Lab (HCRL): Car-Hacking Dataset for the intrusion detection. http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset (2018). [Online; accessed 28-Jun-2018]
8. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks – practical examples and selected short-term countermeasures. Reliability Engineering & System Safety **96** (2011)
9. Kang, M.J., Kang, J.W.: A novel intrusion detection method using deep neural network for in-vehicle network security. In: 2016 IEEE 83rd Vehicular Technology Conference (VTC Spring) (2016)
10. Lai, L., Suda, N., Chandra, V.: CMSIS-NN: efficient neural network kernels for arm cortex-m cpus. CoRR **abs/1801.06601** (2018). URL `http://arxiv.org/abs/1801.06601`
11. Larson, U.E., Nilsson, D.K., Jonsson, E.: An approach to specification-based attack detection for in-vehicle networks. In: Intelligent Vehicles Symposium, 2008 IEEE, pp. 220–225 (2008)
12. Levi, M., Allouche, Y., Kontorovich, A.: Advanced analytics for connected cars cyber security. CoRR **abs/1711.01939** (2017)
13. Marchetti, M., Stabili, D.: Anomaly detection of CAN bus messages through analysis of id sequences. In: 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 1577–1583 (2017)
14. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle. Tech. rep., IOActive Labs (2015)
15. Müller-Quade, J., et al.: Cybersecurity research: Challenges and course of action. Tech. rep., Karlsruher Institut für Technologie (KIT) (2019). DOI 10.5445/IR/1000090060
16. Müter, M., Asaj, N.: Entropy-based anomaly detection for in-vehicle networks. In: 2011 IEEE Intelligent Vehicles Symposium (IV), pp. 1110–1115 (2011)
17. Narayanan, S.N., Mittal, S., Joshi, A.: Obd securealert: An anomaly detection system for vehicles. In: IEEE Workshop on Smart Service Systems (SmartSys 2016) (2016)
18. Rieke, R., Seidemann, M., Talla, E.K., Zelle, D., Seeger, B.: Behavior analysis for safety and security in automotive systems. In: Parallel, Distributed and Network-Based Processing (PDP), 2017 25nd Euromicro International Conference on, pp. 381–385. IEEE Computer Society (2017)
19. Song, H., Kim, H., Kim, H.: Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network, vol. 2016-March, pp. 63–68. IEEE Computer Society (2016)
20. Studnia, I., Alata, E., Nicomette, V., Kaâniche, M., Laarouchi, Y.: A language-based intrusion detection approach for automotive embedded networks. In: The 21st IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2015) (2014)
21. Studnia, I., Nicomette, V., Alata, E., Deswarte, Y., Kaâniche, M., Laarouchi, Y.: Security of embedded automotive networks: state of the art and a research proposal. In: M. ROY (ed.) SAFECOMP 2013 - Workshop CARS of the 32nd International Conference on Computer Safety, Reliability and Security (2013)
22. Taylor, A., Leblanc, S.P., Japkowicz, N.: Probing the limits of anomaly detectors for automobiles with a cyber attack framework. IEEE Intelligent Systems **PP**(99), 1–1 (2018)
23. Theissler, A.: Anomaly detection in recordings from in-vehicle networks. In: Proceedings of Big Data Applications and Principles First International Workshop, BIGDAP 2014 Madrid, Spain, September 11-12 2014 (2014)
24. Wei, Z., Yang, Y., Rehana, Y., Wu, Y., Weng, J., Deng, R.H.: IoVShield: An Efficient Vehicular Intrusion Detection System for Self-driving (Short Paper), pp. 638–647. Springer International Publishing, Cham (2017)
25. Wolf, M., Weimerskirch, A., Paar, C.: Security in Automotive Bus Systems. Proceedings of the Workshop on Embedded Security in Cars (July), 1–13 (2004)