Control System Level Intrusion Detection on J1939 Heavy-Duty Vehicle Buses

Camil Jichici, Adriana Berdich, Adrian Musuroi, Bogdan Groza

***Abstract*—As the security vulnerabilities of Controller Area Networks (CAN) become well known, heavy-duty vehicles implementing the SAE J1939 specification layer on this bus are immediate targets. Recently released standards provide clear cybersecurity requirements, but the exact methods to be implemented are not specified and remain up to the manufacturers. In this work we address adversary actions and countermeasures at the control system level for a heavy-duty vehicle J1939 CAN bus. This low level approach allows us to complement regular attacks with more knowledgeable attacks that may evade detection and discuss realistic countermeasures. Indeed, as we also show by experiments, traditional approaches based on machine learning algorithms will largely fail to detect such attacks. We present experiments based on a model that links between the Simulink environment, an extension of the MATLAB platform for the simulation of in-vehicle control systems, with the CANoe environment, which facilitates the simulation of in-vehicle networks.**

***Keywords*-CAN bus, vehicle security, intrusion detection, SAE J1939**

I. Introduction and motivation

The transportation infrastructure is continuously spreading with the over increasing demand for passengers to commute and for goods to be delivered. In this context, heavy-duty vehicles, sometimes also referred as commercial vehicles, like tractors, trailers or buses, play a key role. On highways, there are millions of heavy-duty vehicles traveling daily for hundreds of kilometers and thus it is essential to continuously improve their safety. The last decade brought several enhanced driver assisting technologies, like automatic emergency braking, lane departure warning, blind-spot detection, etc. An immediate target is to enable vehicles to travel autonomously, without any human intervention, which is of specific importance in case of heavy-duty vehicles due to the much longer distances they travel. In order to achieve all these capabilities, vehicles become complex cyber-physical systems equipped with dozens of ECUs (Electronic Control Units) running millions lines of code, using multiple sensors, actuators, cameras and radars.

Unfortunately, the very same scenario, turns vehicles into potential targets for well-motivated adversaries that can easily exploit existing security vulnerabilities. This has been proved by a strong body of research performed in the past decade. The first such reports from [1], [2] and [3] provide an extensive experimental analysis of attack surfaces and adversary capabilities inside vehicles. These attacks can be performed by an adversary that connects to the bus, e.g., via the OBD port [4], or compromises an existing unit, e.g., the telematics unit [5]. Some recent attacks were also done from remote. For example, a remote attack based on the exploitation of the web browser from a TESLA car was demonstrated in [6] and the authors from [7] managed to remotely compromise several safety critical ECUs from TESLA cars by exploiting the over-the-air (OTA) software update process. Regardless of the entry point, one of the main cause for the reported attacks is the insecurity of the CAN bus which was designed by BOSCH in the 80s without security specifications [8]. Security requirements were standardized only much more recently in the automotive industry, e.g., the ISO/SAE 21434 specifications [9].

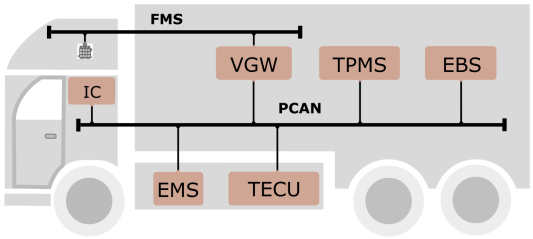
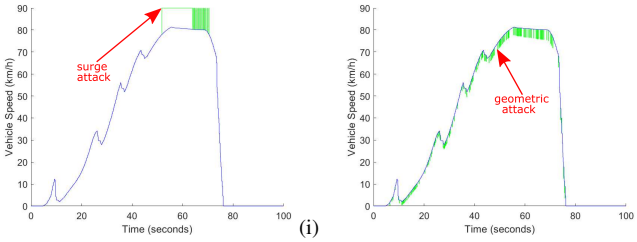


Fig. 1. Typical SAE J1939 CAN bus inside a heavy-duty vehicle

The SAE J1939 standard, developed since the mid 90s, complements the standard CAN bus protocol by adding specific features, to which we will refer later, and targets the CAN communication inside heavy-duty vehicles. Besides several J1939 protocol specific features (like address claims and multiframe transmissions), the standard also clarifies part of the frame content. While the standard is intended for the car and heavy-duty truck industry, this also makes it easier for the research community to design intrusion detection systems that are not content agnostic and account for the actual physical significance of the data carried by each frame. Indeed, most of the existing in-vehicle IDS proposals, which we discuss in the related work section, are content agnostic and one of the reasons is the fact that the content of the traffic in known only to the manufacturer (although it can be understood by proper reverse engineering as shown by some recent works [10], [11]). We depict the concrete J1939 network architecture that we address in Figure 1. The network includes two CAN buses, one of them, having six ECUs, is responsible for powertrain functions and the other is the Fleet Management System (FMS) which has one gateway ECU (more details

This paper was financially supported by the Project "Network of excellence in applied research and innovation for doctoral and postdoctoral programs / InoHubDoc", project co-funded by the European Social Fund financing agreement no. POCU/993/6/13/153437.

Camil Jichici, Adriana Berdich, Adrian Musuroi and Bogdan Groza are with the Faculty of Automatics and Computers, Politehnica University of Timisoara, Romania, Email: {camil.jichici, adriana.berdich, adrian.musuroi, bogdan.groza}@aut.upt.ro



(i) surge attack (ii) geometric attack

Fig. 2. Example of surge attack and geometric attacks on vehicle speed

about this network architecture will be discussed later). As expected, the J1939 specifications do not include security elements either. There are only some recent research proposal for securing J1939 buses which we discuss in the related work section. However, none of the existing works for securing the J1939 bus are addressing specific details at the control system level. Needless to say, these fine grain details can be exploited by adversaries and require distinct attention. Moreover, there are many recent proposals for deploying in-vehicle intrusion detection systems (IDS), which will be accounted in the related work section, but they come with three additional problems. Firstly, they require increased computation costs and memory, especially when relying on more demanding machine learning (ML) algorithms, as we show in the experimental section. Secondly, and perhaps a much more important aspect that is neglected by related works, it is nearly impossible to provide a dataset that covers all the states of a vehicle. Needless to say, a car is on road for years and travels several hundred thousand miles while the training datasets usually capture a few minutes, maybe hours or so, of runtime. Thirdly, small variations in the transmitted data may easily go undetected by the intrusion detection system. As a practical example, Figure 2 shows a surge attack in the left and a geometric attack on the right, such attacks were commonly accounted in control systems [12]. It is understandable that if the training was performed on data before the surge attack occurred, this attack will go unrecognized by a signature-based IDS. While for the geometric attack, the changes are initially very small and increase at a latter point which may evade detection by an anomaly-based IDS. Finally, by exploiting minute changes in the signals and the appropriate control system model, as discussed in this work, the detection mechanism requires only simple computations related to a change detection, e.g., a cumulative sum (CUSUM) which is a well-known sequential analysis technique. Because of the simplicity of such computations, the IDS can be easily integrated on existing microcontrollers (including low-end ones, such as the NXP S12 which we use in our experiments) and the IDS can actively block intrusion frames, if it is present on each microcontroller. This turns the proposed IDS into an intrusion prevention system since it can actively block the adversarial interventions. This approach complies with the recent specifications for IDS deployment from the AUTOSAR standard [13].

In the light of the above, the contributions of our work are threefold:

1) we develop an experimental setup which connects between CANoe, the industry leading software for simulating in-vehicle buses, and Simulink, the leading environment for the design of control systems,
2) we analyze attacks at the control system level, on various signals (vehicle speed, trip distance, engine speed, torque) that circulate on the bus of a J1939 heavy-duty vehicle and remain stealthy, being hard or impossible to detect by traditional intrusion detection systems and discuss the appropriate countermeasures,
3) we present concrete experimental results on both low-end (Freescale S12X) and high-end (Infineon TC series) microcontrollers, to prove that the solution can be deployed in real-world automotive scenarios.

Our work is organized as follows. In Section II we provide an overview of related works. Section III provides some background on CAN buses, relevant details of the SAE J1939 standard and some details on the CANoe-Simulink integration. In Section IV we present an overview of the J1939 CAN bus simulation, show the details for the control systems in Simulink and introduce the adversary model. Section V presents the experimental results with the attacks and countermeasures. Section VI holds the conclusions of our work.

II. Related works

A comprehensive image of the recent technological challenges for automotive embedded systems, including standards, methodologies, hardware and software solutions, network architectures, functional safety and security design strategies is presented in [14]. Some vulnerabilities of in-vehicle networks, attack strategies and countermeasures are also summarized in [15]. Various practical attacks and defense mechanisms were proposed in the past decade but the attention was concentrated mostly on passenger cars, not on heavy-duty vehicles. Only a small number of recent works started to address the security of heavy-duty vehicles compliant with the SAE J1939 standard. The vulnerabilities of the SAE J1939 specific features were examined and an authentication protocol was proposed in [16]. The authors from [17] describe a DoS (Denial of Service) attack focused on the J1939 specific transport protocols used for multipacket transmissions. The authors in [18] showed that injection and replay attacks on J1939 CAN buses inside commercial vehicles can have serious consequences. Hariharan et al. [19] offer a verification and validation framework that relies on a security testing mechanism tailored for J1939 heavy-duty vehicle buses.

In what follows, we describe the recently proposed countermeasures as a response for the aforementioned attacks. The encryption of the J1939 diagnostic traffic was examined in [20]. Some steps were also done towards the deployment of intrusion detection systems on J1939. A machine learning based approach was proposed in [21] in order to detect DoS and fuzzing attacks and a precedence graph-based approach was evaluated in [22] for anomaly detection. Recently, a two-stage intrusion detection mechanism for J1939 was proposed in [23]. The first stage is responsible for checking the legitimacy of the encrypted addresses (source and destination) from the CAN ID while the second stage detects single bit changes of the

datafield by proper range checks. The detection of adversarial manipulations is facilitated by the avalanche effect of block ciphers as the datafields of CAN frames are encrypted. Another recent approach relies on both timing and data analysis in order to detect spoofing and masquerade attacks in J1939 and NMEA2000 networks [24]. The mechanism is able to detect manipulation attacks, i.e., single bit flips, by inspecting anomalous changes in the electric potential during a transition from a dominant to a passive state. However, none of the previous works on J1939 security accounts for the control system level, which is the main objective of our work.

Outside the J1939 context, there is a large number of works that are addressing the deployment of intrusion detection mechanisms on classical CAN. Issues related to attack surfaces, attack strategies, costs and real-time constraints, methods for implementing IDS and challenges are surveyed in [25]. A clock offset based algorithm for attacker identification in CAN networks is discussed in [26]. The use of clock skews has been originally proposed in [27]. The use of neural networks for intrusion detection was explored in [28] and [29]. Convolutional neural networks with long and short-term memory (LSTM) were proposed in [30] while LSTM autoencoders were investigated in [31]. Traditional machine learning algorithms were also used, e.g., k-Nearest Neighbor (k-NN) was analyzed in [32], support vector machine classifiers (SVM) in [32], [33] and [34] while decision trees were examined in [35]. A hybrid IDS based on discrete wavelet transform and SVM is suggested in [36]. Other lines of work use finite-state automatons [37], Hidden Markov Models [38] or entropy characteristics [39], [40]. The use of the time interval between remote frames and response data frames for detecting malicious activity is investigated in [4]. The work in [41] uses the deviation from the periodicity of the frames.

Most of the previous proposals for in-vehicle IDS are content-agnostic. However, as pointed out by many recent works on cyber-physical systems security, these methods may be quite ineffective. For example, it is clearly stated by the authors from [42] that, in case of cyber-physical systems (which include the automotive domain), intrusion detection systems that do not use domain-specific knowledge will perform poorly. And, as acknowledged by the authors, the work in [43] has already proved that data with high variability makes it impossible to deliver high detection rates and low positive rates at the same time. This makes it clear that content-agnostic approaches have limitations and should be used only when specific information is not available. While this is the case in many applications, fortunately, in-vehicle control systems do have a well-understood physical behavior that can be used for a finer-grain detection of adversarial manipulations.

As we later show in the experimental results, traditional machine learning approaches, like k-nearest neighbors (k-NN), decision trees classifier (DTC) and the random forest classifier (RFC), fail to detect the attacks that we analyze (the results are summarized in Table III). Also, simple metrics like the Hamming distance would be ineffective. For example, in case of a surge attack, a Hamming distance equal to just 1 will cause exponential variation depending on the bit which is affected, e.g., the most significant bit. Historically speaking, the Hamming distance was always effectively used for error detection and correction, but in case of intrusion detection it is generally used only if a more effective domain-specific metric is unknown [44]. Keeping in mind that physical processes have monotonic variations, using state predictors and checking cumulative sums of the recorded errors, i.e., CUSUM, is likely more effective. This kind of approach is commonly used for detecting intrusions in cyber-physical systems [45], [46].

III. Background

In this section we discuss the CAN background and the SAE J1939 standard specific features. Then we proceed to a presentation of the CANoe environment and its integration with the Simulink environment.

A. CAN bus description

The effectiveness and simplicity of the classical Controller Area Network (CAN) bus has been proven by more than three decades of use, during which it remained the most employed solution for internal communication inside vehicles. Newer extensions of the CAN protocol, i.e., the CAN with Flexible Data Rate (CAN-FD) and more recently, the CAN Extra Long (CAN-XL), set the room for future uses of the CAN protocol, at least for the decade that follows. The CAN protocol supports transmission rates of up to 1 Mbit/s when high-speed CAN is employed. Enhanced speed performance and payload can be accomplished by using the CAN-FD or the CAN-XL protocol. CAN-FD supports speeds from 2 to 5 Mbit/s and up to 64 data bytes, while CAN-XL enables transmissions at 10 Mbit/s and payloads up to 2048 bytes.

As we depict in Figure 3, each microntroller or interface is connected to the bus via two wires (CAN-High and CAN-Low) terminated by 120 Ohm resistors. Bus communication is mediated by the CAN data frames which are broadcast by each CAN node. The extended CAN data frame format is presented in Figure 4. The start of a CAN frame is indicated by a dominant start-of-frame bit (SOF). Each CAN frame has an 11 bit ID (identifier) in case of the standard format or a 29 bit ID in case of the extended format. In addition to the role of uniquely identifying a CAN message, the CAN ID also ensures the bus arbitration mechanism, i.e., if multiple nodes try to occupy the bus at the same time, the node which transmits the lowest valued ID wins the arbitration. Only nodes interested for certain CAN IDs take into account the signals that are packed inside the CAN frame. The control field specifies the amount of data (0-8 bytes) that will be packed inside the data field. An error detecting code, i.e, a 15 bit Cyclic Redundancy Check (CRC), is computed over the frame content in order to verify data correctness and protect communication in case of accidental alterations, e.g., errors inflicted by noise, of the frame content. Each node that correctly receives a frame confirms this by writing a 0 in the ACK slot, which remains otherwise set to 1. The end of the CAN message is marked by a recessive EOF bit.

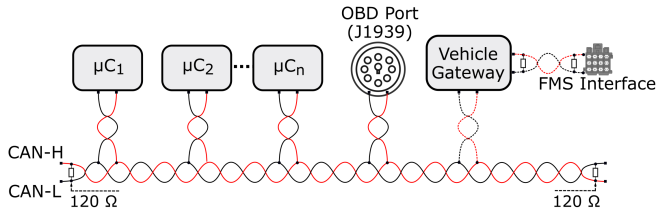


Fig. 3. SAE J1939 typical CAN bus topology

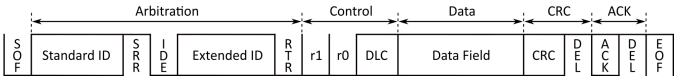


Fig. 4. Extended CAN data frame

B. SAE J1939 features

The J1939 standard collection was released by the Society of Automotive Engineers (SAE) and is dedicated to a specific sector from the automotive industry, i.e., commercial vehicles which includes various industry groups, varying from agricultural (tractors) and construction equipment (mobile cranes, trucks) or on-highway equipment (highway cabs) to marine (yacht, boats) or industrial processes. Concretely, SAE J1939 defines a full communication stack on top of the CAN protocol by adding the data link and network upper layers. The specifications inside the SAE J1939-71 [47] document are relevant for the proposed IDS since they outline specifics related to in-vehicle frames which can be also used for a better design of an IDS. This is not the case with regular passenger vehicles where there are no such specifications.

A specific choice in the J1939 specification is the use of extended CAN IDs that have 29 bits in length (regular identifiers have only 11 bits). Another one accounts for the employment of specific transport protocols that enables payloads of up to 1785 bytes through multi-frame transmissions. The J1939-71 document describes the signals that are enclosed by the standardized J1939 frames. Please note that sometimes the terms parameter and signal are used interchangeably in the J1939-71 standard (as well as in many other automotive specifications) referring to the data packed inside the CAN frame that corresponds to a specific variable, e.g., vehicles speed, brake switch, cruise control status, etc. Another aspect of J1939 networks is the integration of source and destination addresses into CAN IDs, each address having the role of identifying a node from the network.

A supplemental material for the J1939-71 document is the J1939 Digital Annex [48] which presents the signals packed inside J1939 messages in an easier form. An example of signal layout inside a J1939 frame is presented in Table I for ID 0x18FEF100 which is further used in our experiments. The second column specifies the start position of the parameter inside the payload at byte level and for each byte it mentions the bit position. The third column indicates the parameter length in bits and the last column describes the meaning of the parameter. The parameter marked with bold text, corresponding to the wheel-based vehicle speed, will be used in our experimental evaluation.

TABLE I
SPECIFIC J1939 SIGNALS FOR ID 0X18FEF100 ACCORDING TO [48]

No.	Start position	Length	J1939 signal
1.	Byte 1 - bit 1	2 bits	Two Speed Axle Switch
2.	Byte 1 - bit 3	2 bits	Parking Brake Switch
3.	Byte 1 - bit 5	2 bits	Cruise Control Pause Switch
4.	Byte 1 - bit 7	2 bits	Park Brake Release Inhibit Request
5.	**Byte 2 - bit 1**	**16 bits**	**Wheel-Based Vehicle Speed**
6.	Byte 4 - bit 1	2 bits	Cruise Control Active
7.	Byte 4 - bit 3	2 bits	Cruise Control Enable Switch
8.	Byte 4 - bit 5	2 bits	Brake Switch
9.	Byte 4 - bit 7	2 bits	Clutch Switch
10.	Byte 5 - bit 1	2 bits	Cruise Control Set Switch
11.	Byte 5 - bit 3	2 bits	Cruise Control Coast (Decelerate) Switch
12.	Byte 5 - bit 5	2 bits	Cruise Control Resume Switch
13.	Byte 5 - bit 7	2 bits	Cruise Control Accelerate Switch
14.	Byte 6 - bit 1	8 bits	Cruise Control Set Speed
15.	Byte 7 - bit 1	5 bits	PTO Governor State
16.	Byte 7 - bit 6	3 bits	Cruise Control States
17.	Byte 8 - bit 1	2 bits	Engine Idle Increment Switch
18.	Byte 8 - bit 3	2 bits	Engine Idle Decrement Switch
19.	Byte 8 - bit 5	2 bits	Engine Diagnostic Test Mode Switch
20.	Byte 8 - bit 7	2 bits	Engine Shutdown Override Switch

C. CANoe-Simulink integration

CANoe is a market-leading development environment for the design, real-time simulation and testing of in-vehicle buses. It also provides integration with other industry standard tools like Simulink which is a MATLAB-based environment that can be used for simulating dynamical systems (such as in-vehicle control systems).

A common CANoe simulation setup is suggested in Figure 5. A simulated bus in the CANoe environment is depicted on the right side. This bus is compatible with hardware devices developed by Vector which are used for transmitting CAN frames on the actual physical bus from the left side. In this example the simulation has three CAN nodes. Different software layers are employed in order to describe the behavior of a CAN node. On the upper side is the application layer that interacts with the bus signals or with the values provided by several sensors or actuators, which can be retrieved as inputs from the graphical interface. For the application layer, the CANoe environment uses code written in CAPL (Communication Access Programming Language). CAPL is an event-controlled programming language having a slightly similar syntax with the C language and complemented by various specific functions that react to the real-time CAN communication events, e.g., reception of a CAN frame. In our setup, we use the CAPL code to mimic a real-world adversary that injects the malicious frames on the CAN bus. The interaction with the CAN parameters and frames or with sensor measurements is facilitated by the use of a CAN database. Furthermore, the CANoe environment supports the extension or even the complete replacement of the CAPL application layer with the Simulink models (a feature which we actually used in this work).

The connection between the application layer and the physical layer is handled by the following three layers: the interaction layer (IL), the network management layer (NM) and the transport protocol layer (TP). The IL assures the connection between the application layer and the low level

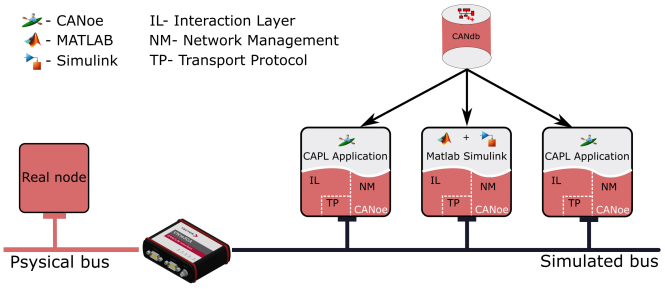


Fig. 5. CANoe simulation setup

drivers. The proper operation of a CAN network is enabled by the NM layer which performs several actions for bus management. The TP manages the organization of the data link layer. There are some circumstances when data larger than 8 bytes (which is the maximum limit for standard CAN) need to be transmitted over CAN and cannot be packed into a single CAN frame, e.g., in case of diagnostic buffers, multi-frame messages, etc. The TP is responsible for splitting the data into several frames and re-packing the data on the receiver side.

The interaction between Simulink and CANoe at run-time can be achieved in three different ways as depicted in Figure 6. The first one, shown in Figure 6 (i), is the offline mode in which the simulation is executed inside the Simulink environment which acts as the lead system while the CANoe environment is managed as a secondary system. The offline mode uses the simulation time-base controlled by Simulink and is mostly used for debugging purposes. In this mode, a real-time simulation cannot be run and the interaction with hardware devices is not available. In synchronized mode, which is shown in Figure 6 (ii), similar to the offline mode, the Simulink environment is the host for the simulation. The difference from the previous mode is that the time-base for the simulation is now taken from the CANoe environment. Consequently, the simulation happens in real-time and the opportunity for communication with Vector hardware devices is also available. Last but not least, the hardware-in-the-loop mode, depicted in Figure 6 (iii), operates differently from the two aforementioned modes. In this setting, the CANoe environment is the host for the simulation while the Simulink models are integrated inside CANoe simulation through DLLs (Dynamic Link Libraries) built from the C Code generated by the Simulink Coder. In this work we use the synchronized mode which copes with real-time constraints and allows us to quickly change and test the Simulink models.

IV. Control Systems and Adversary Model

This section presents the control system level from the J1939 simulation and the adversary model that we use.

A. J1939 simulation and Simulink models

We address the J1939 simulation setup for a heavy-duty vehicle that has already been depicted in Figure 1 and is made available by the CANoe tool[1]. The J1939 network

[1]https://www.vector.com/int/en/products/products-a-z/software/canoe

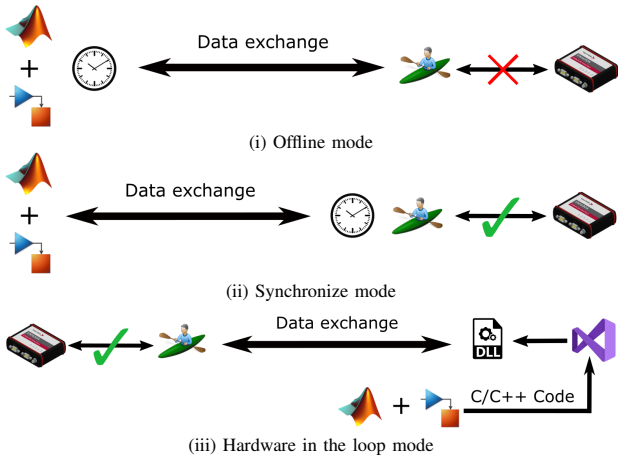


Fig. 6. Interfacing CANoe with Simulink

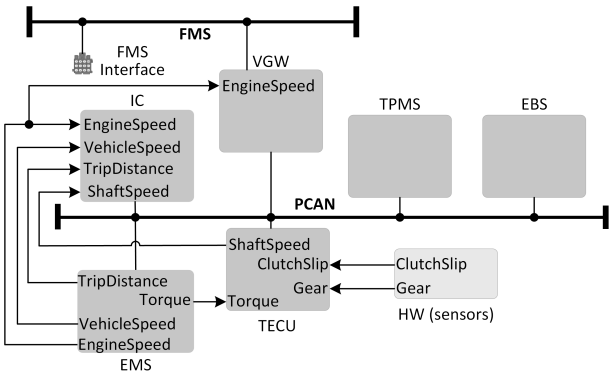


Fig. 7. Signal mapping between ECUs on the PCAN bus

configuration contains two CAN buses, one for the Powertrain and the other for the Fleet Management System (FMS). The Powertrain bus has six ECUs. One of them plays the role of a gateway, being also connected to the second CAN bus. The main function of the employed ECUs, their assigned addresses and signals are presented in Table II. The Tx column contains the signals that are transmitted by the ECUs while the received signals are presented in the Rx signals column. The flow of the signals between ECUs is shown in Figure 7. The FMS bus has just one ECU, i.e., the same gateway ECU (VGW) that is also present in the first network. The role of this ECU is to select the Parameter Groups according to the FMS standard [49] and forward them from one CAN bus to the other.

We extend this J1939 CAN bus simulation by linking it to Simulink models for the control systems, which allows us to add adversarial behavior. To protect the CAN bus against adversarial interventions, we need to make predictions for the signals that are sent through the network. We depict how these signals are computed, also showing the Simulink blocks, in what follows.

1) Vehicle speed prediction: In Figure 8 we describe the Simulink model which computes the vehicle speed in km/h based on shaft vehicle speed in rpm which comes from the TECU controller as follows:

$$v = \frac{3600}{1000} \times r \times \frac{2 \times \pi}{60} \times v_{shaft},$$

TABLE II
BRIEF DESCRIPTION OF NODES EMPLOYED IN POWERTRAIN NETWORK

Node	Addr.	Function	Tx signal	Rx signal
EMS	0x00	Engine management system	Vehicle Speed, Trip Distance, Engine Speed, Torque	–
TECU	0x03	Transmission ECU	Shaft Speed	Clutch Slip, Gear, Torque
IC	0x17	Instrument cluster	–	Vehicle Speed, Shaft Speed, Trip Distance, Engine Speed,
VGW	0xE6	Vehicle gateway	–	Engine Speed

where v is the vehicle speed in km/h, r is the diameter of the shaft axle in meters (in our case 0.151 meters) and v_{shaft} is the shaft vehicle speed measured in rpm. In general, inside cars, the shaft vehicle speed is computed by the TCU controller based on a PWM signal which measures the rotation of the main axle.

2) Trip distance prediction: In Figure 9 we depict the calculation of the trip distance based on the vehicle speed, as follows:

$$dist = \int \frac{v \times 0.1}{3600}\, dx,$$

where $dist$ is the trip distance, v is the vehicle speed and the coefficient $\frac{0.1}{3600}$ is used to convert the vehicle speed from km/h into m/s.

3) Acceleration estimation: In Figure 10 we compute the acceleration based on vehicle speed as follows:

$$acc = \frac{\mathrm{d}v}{\mathrm{d}t},$$

where acc is the acceleration measured in m/s^2, v is the vehicle speed and t represents time. In order to use the computed acceleration to predict other vehicle signals, i.e., the engine speed and torque, we used a low-pass filter on the acceleration to remove undesired spikes.

4) Engine speed prediction: Based on the filtered acceleration and gear, in Figure 11, we depict the prediction of the engine speed. To achieve this, we used lookup tables (a commonly used object in automotive and control systems projects) with 3 to 100 breakpoints, calibrated from a short run-time during which various gear-shifts took place. These allow us to determine the engine speed based on the filtered acceleration and gear. The lookup tables were calibrated based on the signals recorded inside the CANoe simulation, i.e., for each gear we chose 3 to 100 values of the filtered acceleration and set the value of the engine speed that corresponds to it (intermediary values can be determined by interpolation). To improve the accuracy of the prediction, when necessary, we switch between two lookup tables based on the clutch slip value. This is needed because when the clutch is engaged, the engine speed ramps faster, while when the clutch is disengaged, the engine speed is more stable. In case of idle speed, if the vehicle is not moving and the acceleration or brake pedals are not pressed by the driver, the predicted engine speed is set to the idle speed, i.e., $250rpm$.

5) Torque prediction: Similar with the prediction of engine speed, in Figure 12, we predict the engine torque using one-dimensional lookup tables based on the filtered values for the acceleration and gear. When necessary, we switch between three lookup tables. This is needed because the torque behavior is different when the accelerator pedal value is pressed more than 70% or when the power take-off module is engaged. When the clutch is engaged, we do not have a driver torque request, the torque losses are higher and the value of the actual torque is set to the minimum value, i.e., -1650Nm, according to the CANoe model.

We note that for the last two signals, engine speed and torque, we don't have a concrete physical model and the prediction is done based on look-up tables constructed from the behavior of the parameters inferred from previous runs of the CANoe model. For this reason, in the experimental section, the accuracy of the intrusion detection on these parameters will be lower and should serve only as an example. Better predictions can be achieved with improved models that were out of reach for us at the moment.

B. Adversary model

Existing works focusing on attacks and intrusion detection for CAN buses consider three types of attacks: replay, DoS and fuzzing attacks. These attacks can be easily circumvented by checking that the IDs are part of the legitimate set (in case of fuzzing when IDs are random or DoS which is caused by high priority IDs that are not part of the network) and that the rate at which the IDs arrive is the expected one (which is usually not the case for replays). Moreover, changes in the datafield can be detected if, having the predicted value of the signal $y'_{\blacklozenge}(k)$ and a bias b, a change detection mechanism is introduced by checking the following recurrent sum [12]: $S_{\blacklozenge}(k) = \max\{0, S_{\blacklozenge}(k-1) + |y_{\blacklozenge}(k) - y'_{\blacklozenge}(k)| - b\}, S(0) = 0$. By comparing this cumulative sum with an empirically defined threshold τ, an intrusion can be detected. This approach for change detection, introduced in [50], abbreviated as CUSUM is commonly used in intrusion detection systems. A very recent work dedicated to intrusion detection on J1939 also uses it [24], but here we account for the control system level and stealthy attacks, which were not addressed before in the context of in-vehicle networks.

There are three flavors of stealthy attacks that can evade the detection mechanism based on cumulative sums (CUSUM) as pointed out in [12]: surge attacks, bias attacks and geometric attacks. Previous works on CAN bus intrusion detection have not considered these alternatives in the adversary model. Therefore, we specifically focus on these attacks in our work. Since replay and DoS attacks can be detected by simply inspecting the arrival rate of frames (both these attacks require multiple frames that would be unusual as CAN IDs are sent at fixed periods), our adversary model incorporates the following types of modification attacks:

1) *fuzzing attacks* - are the modification attacks in which random values are injected in the datafield of CAN frames,
2) *surge attacks* - are the modification attacks in which the value of the signal is set to the maximum value (or

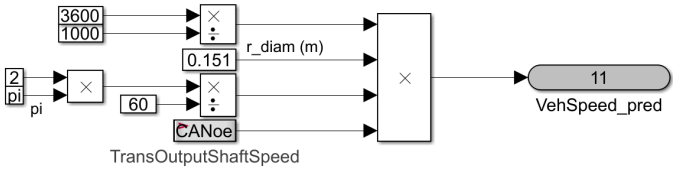


Fig. 8. Prediction of vehicle speed based on shaft speed

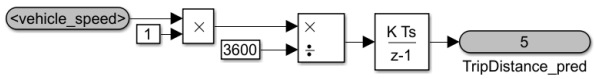


Fig. 9. Prediction of trip distance based on vehicle speed

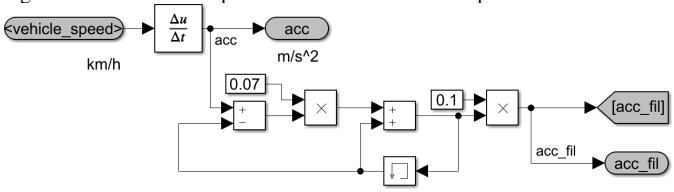


Fig. 10. Acceleration estimation

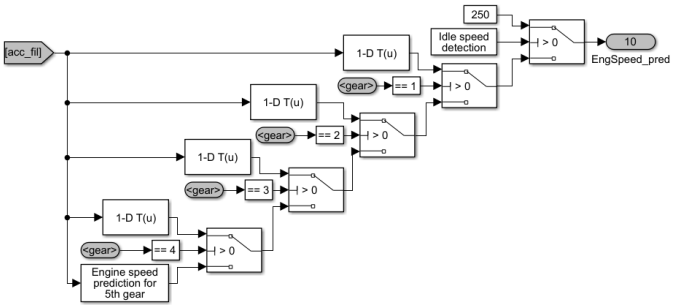


Fig. 11. Prediction of engine speed based on acceleration and gear

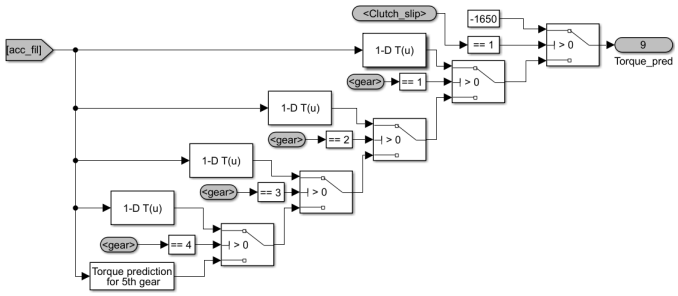


Fig. 12. Prediction of torque based on acceleration and gear and clutch slip

minimum value) such that it will inflict the maximum damage on the system and yet remain undetected, i.e., the attack value at step $k+1$ will be $y_{\blacklozenge,\max}$ only if the corresponding sum at the next step $S_{\blacklozenge}(k+1) \leq \tau$ while otherwise the attack signal will stay at $y'_{\blacklozenge}(k) + |\tau + b - S_{\blacklozenge}(k)|$,

3) *bias attacks* - are the modification attacks in which a small constant $c = \tau/n + b$ is added at each step to the attacked signal, i.e., $\widetilde{y}_{\blacklozenge}(k) \leftarrow y_{\blacklozenge}(k) + \tau/n + b$, ensuring that the attack remains undetected for n steps,
4) *geometric attacks* - are the modification attacks in which a small drift is added to the attacked signal in the beginning and the drift becomes increasingly larger in the next steps using a geometric expansion, i.e., $\widetilde{y}_{\blacklozenge}(k) \leftarrow y_{\blacklozenge}(k) + \beta\alpha^{n-k}$ where α is fixed and $\beta = \frac{(\tau+nb)(\alpha^{-1}-1)}{1-\alpha^n}$.

These adversarial actions and the countermeasures that we address are independent from the entry point that is exploited by the adversary. As stated in the introduction, most of the attacks ask for an adversary that connects to the bus or compromises a device from remote. It is worth noting that a recent work [51] proposes a clever kind of attack in which regular CAN frames are hidden inside CAN-FD frames. Indeed, this possibility has not been previously explored. However, regardless of how the attack is performed, since the IDS that we design is supposed to run locally on each node, the attack should be detected when the frame content is inspected. Thus, our proposal should provide resilience to this kind of attack too.

Our adversary model considers that the intruder has a fixed success probability in modifying each frame. The rationale behind this model goes as follows. The way in which attacks are commonly performed on the CAN bus by injecting additional (adversarial) messages on the network, is correct from a networking perspective, but it offers a slightly misleading image regarding the way in which in-vehicle controllers operate. Concretely, in-vehicle controllers run tasks that are scheduled at precise time intervals, for example 10-100ms, and consume one CAN message during each subsequent execution. In general, messages are buffered on the controller and for every CAN ID, each new message will overwrite the past one. Therefore, even if an adversary injects multiple messages between two legitimate messages, only one message is going to be consumed by the controller, i.e., the most recent one. Consequently, given that a number of adversarial messages compete with legitimate messages on the bus, attacks are probabilistic. This explains why rather than injecting messages, we set a probability, e.g., 25%, that a message is or not corrupted, which is consistent with the reality at the controller level addressed in this paper.

Also, besides the obvious possibility that one task running on the ECU is corrupted and simply replaces legitimate messages from another task with corrupted ones, there are two more considerations at the networking layer that make this adversary model (which replaces legitimate frames with corrupted ones rather than injecting new messages) more realistic. One is that ECUs may be subject to *bus off* attacks which will keep the ECU dormant for a while, during which the adversary may inject malicious messages. These attacks were first demonstrated in [52]. The other is that messages may pass through a corrupted gateway that may alter them. Nonetheless, vehicles are now moving from domain oriented architectures, in which ECUs that implement similar functionalities are

grouped on the same bus, toward zone oriented architectures, where ECUs are grouped under a zonal controller according to the location where they operate. In this newer paradigm, even messages that are responsible for the same functionality may need to be routed by gateways and thus have significantly higher chances for being manipulated.

V. Experimental results

In this section, we first discuss some J1939 specific parameters then we introduce the metrics employed in our performance assessment of the IDS. Afterwards, we present the detection results and some computational requirements on automotive-grade controllers.

A. Specific attacks on J1939 signals

For our work we target J1939 specific signals that can be easily identified based on J1939 specifications [47]. Concretely, our evaluation considers 4 parameters: vehicle speed (km/h), trip distance (km), engine speed (rpm) and torque (Nm). To provide a better understanding about the behaviour of the attacks and the impact they produce, we depict the variations of the J1939 signals under several attack scenarios.

Figure 13 shows the variation of the vehicle speed. The legitimate signal is marked with blue while the modifications inflicted by the adversary are marked with green. Figure 13 (i) shows the fuzzing attacks which appear as spikes around the legitimate signal. When a surge attack occurs, i.e., Figure 13 (ii), the falsified signal reaches $90km/h$ but this happens only as the legitimate signal approaches the targeted value. Only small deviations of the legitimate signals occur in case of the bias attacks from Figure 13 (iii) and the geometric attacks from Figure 13 (iv). The difference between these last two types of attack is that the distance from the genuine signal to the attack signal remains constant in case of the bias attacks while the distance grows slowly for the latter. As can be seen in Figure 14 the trip distance signal has a different waveform compared to other parameters, i.e, the signal is incremented for every $125m$ traveled. Under the surge attack, presented in Figure 14 (ii), the altered signal exhibits a top level of $1.2km$. The behavior for the bias and geometric attacks, depicted in Figures 14 (iii) and (iv), is comparable with the behavior for the previous signal. The range in which the legitimate engine speed signal varies is greater, i.e., 0–$2050rpm$, as can be seen in Figure 15. In this case, the manipulated signal reaches $2250rpm$ when a surge attack is mounted, i.e., Figure 15 (ii). The deviations from the legitimate signals are greater than those from the vehicle speed signal when the bias and geometric attacks take place, i.e., Figures 15 (iii) and (iv). We omit the plots for the torque signal since the attacks are similar to the case of the attacks on the engine speed.

B. Metrics for evaluating detection efficiency

Intrusion detection is a binary classification issue between legitimate and intruder frames. Thus, a performance assessment has to rely on the four commonly used parameters: true positives (TP), i.e., the number of intrusion messages that are correctly labelled as attacks, true negatives (TN), i.e., the number of legitimate messages that are correctly labelled as legitimate, false positives (FP), i.e., the number of legitimate messages that are incorrectly labelled as attacks, and false negatives (FN), i.e., the number of intrusion messages that are incorrectly labelled as legitimate.

We will also use as a metric the false positive rate, i.e., the ratio between the number of frames that are wrongly reported as attacks and the total number of legitimate frames, i.e., $FPR = FP/(FP + TN)$. And also, the false negative rate (FNR) which is the ratio between the number of frames that are incorrectly reported as legitimate and the total number of intrusions, i.e., $FNR = FN/(FN + TP)$. It is also easy to derive two more metrics that we are going to use as they are commonly employed in related works too. The *accuracy* combines all four parameters in order to get the ratio between the number of correctly classified frames (intrusions or legitimate frames) and the total number of frames: $Accuracy = (TP + TN)/(TP + TN + FP + FN)$. The *precision* is the ratio between the correctly classified intrusion frames and all the frames classified as intrusions: $Precision = TP/(TP + FP)$.

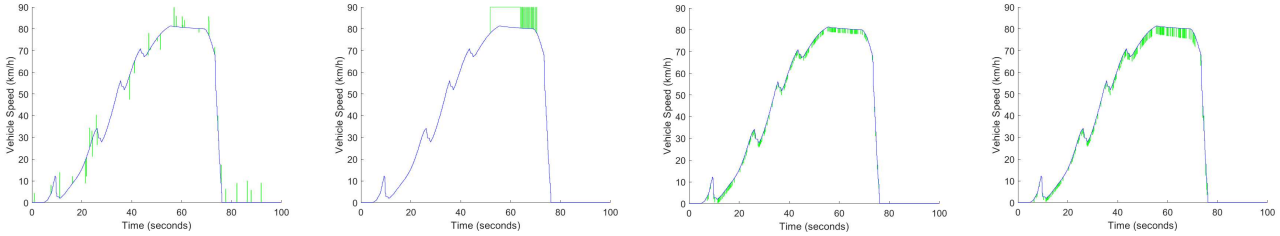
C. Results on detecting intrusions

On each CAN signal, we programmed the adversary to act with a predefined attack probability p_{att} (this complies with the usual attack model in which legitimate traffic is injected with adversarial actions). This attack probability is employed for the fuzzing, bias and geometric attacks while the surge attack takes place once the conditions described in the adversary model from the previous section are reached in order to avoid detection. Attacks resulting in values that are out range, i.e., negative values, will not be carried since they can be immediately detected by range checks. The change detection mechanism runs in the Simulink environment and retrieves data from the CANoe simulation with which it communicates in synchronized mode.

TABLE III
Engine speed - results with machine learning algorithms

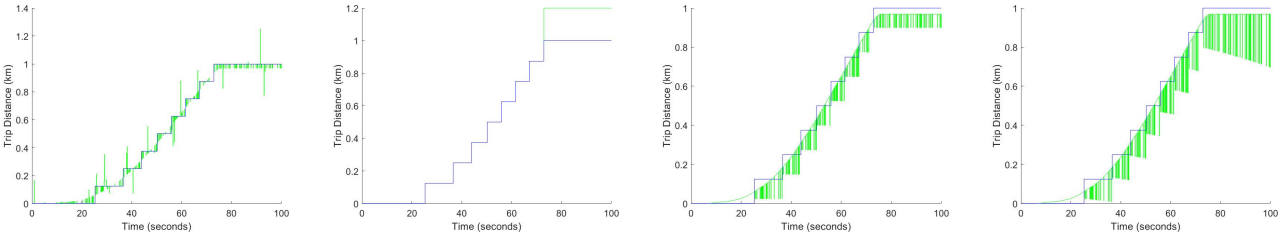
Algorithm	Attack type	FPR (%)	FNR (%)	Accuracy (%)	Precision (%)
k-NN	fuzzing	16.20	74.87	69.50	33.33
k-NN	surge	1.18	0.00	99.38	98.68
k-NN	bias	1.06	98.54	82.25	22.22
k-NN	geom	0.15	98.26	83.00	50.00
DTC	fuzzing	28.26	37.95	69.38	41.44
DTC	surge	0.00	0.00	100.00	100.00
DTC	bias	0.45	100.00	82.50	0.00
DTC	geometric	9.19	99.26	75.50	1.61
RFC	fuzzing	12.73	55.38	76.88	53.05
RFC	surge	0.00	0.00	100.00	100.00
RFC	bias	0.60	100.00	82.38	0.00
RFC	geometric	0.15	99.26	83.00	50.00

First, we note that the machine learning algorithms cannot cope with the modifications attacks. To implement these algorithms in our system, we use the Statistics and Machine Learning Toolbox provided by MATLAB as well as the Python scikit-learn library https://scikit-learn.org/stable/ and



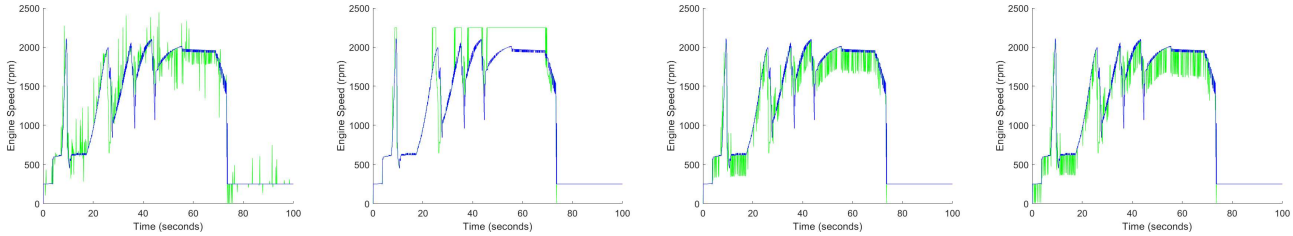
(i) fuzzing attack (ii) surge attack (iii) bias attack (iv) geometric attack

Fig. 13. The behavior of wheel based vehicle speed signal for different types of attacks



(i) fuzzing attack (ii) surge attack (iii) bias attack (iv) geometric attack

Fig. 14. The behavior of trip distance signal for different types of attacks



(i) fuzzing attack (ii) surge attack (iii) bias attack (iv) geometric attack

Fig. 15. The behavior of engine speed signal for different types of attacks

the sklearn-porter library https://github.com/nok/sklearn-porter to generate C code in order to measure the runtime of the detection algorithms on the embedded devices. We build our classification models based on portions of the full CAN traces. We split the CAN traffic in two parts: 20% for training and 80% for testing. The results from Table III rely on classical machine learning algorithms: k-nearest neighbors (k-NN), decision trees classifier (DTC) and the random forest classifier (RFC). The attack was carried only on frames that contain the targeted signal. As can be seen, the machine learning based IDS has considerable failures in detecting fuzzing, bias and geometric attacks (the surge attacks are the only attacks on which the detection works). The false negative rate is over 98% in case of the bias and geometric attacks, so these attacks remain largely undetected. For the fuzzing attacks the FNR is smaller, around 37%-74%, but still unsatisfactory. Cells that contain a lower precision are marked with grey.

On the contrary, the change detection mechanism is highly successful. Now we discuss the detection results that are shown in Table IV. The bias and the threshold values for the IDS and the adversary were empirically determined. If we employ the same threshold values for the adversary and the IDS, only small deviations from the legitimate signals occur, and the impact is very small while the attack goes undetected by the IDS. For example, in case of the vehicle speed, the IDS threshold is at $2km/h$, and, for the engine speed, the threshold is at $150rpm$ - such variations, even if produced by the adversary to mislead the driver, will have almost no effect. Thus, the adversary will be interested in causing higher variations that can have more relevance in misleading the driver (like changing the reported speed with $10km/h$). This explains the selection of the thresholds from Table IV.

When using larger thresholds for the adversary than for the IDS, the attack has a significant impact but the IDS will be able to detect it. The results for the first two parameters (vehicle speed and trip distance) are excellent, i.e., the accuracy and precision are close to 100%. This is due to the fact that both these signals are accurately predicted. On the other hand, for the other two signals, Engine Speed and Torque, the detection exhibits a lower accuracy and precision, i.e., 94%-99% and 95%-98%. There is an increase of the FPR and FNR, but this still seems acceptable at 0%-1% and 0%-20% respectively. This is caused by a poorer estimation of the legitimate signals with the lookup tables described previously. Better estimations can be achieved based on the mechanical parameters of the vehicle which are out of scope for our work here.

The previous experiments were based on a 100 seconds simulation, having a 0.1 second simulation step, which leads to a total of 1000 CAN frames. True indeed, other works based on in-vehicle collected CAN bus data use traces of one or more hours. This is not necessarily a better approach for testing the validity of an IDS since it is the variability of the data, not the duration of the trace, which determines the complexity of detecting attacks. The rather short trace of 1000

CAN frames that we used accounts for significant variations of the recorded parameters, while a several hours long trace is not necessarily more convincing if the vehicle parameters do not exhibit significant variations.

To prove this, we also run the detection mechanism in a scenario that lasts more than an hour. For this we created a more intricate traffic trace that contains several specific actions, e.g., gear shifts, sudden braking, speed variations, etc. The results are presented in Table V. With a few exceptions, the false positives and negatives dropped significantly compared to the short trace. That is, the false positives dropped below 1%, while the false negatives, with the exception of fuzzing attacks, are between 0%-5%. On a closer inspection, the cause for the drop in false positives became immediately visible for us: almost all of the false positives are caused by gear shifts due to the imperfect look-up tables that we used for deriving the engine speed and torque. The long trace that we used had 30 gear shifts for more than one hour, while the short trace had 10 gear shifts during 100 seconds. This also suggests that to avoid false alarms, abrupt changes in engine speed or torque that happen during a gear shift may be neglected.

It is difficult to delve further into this matter since unexpected gear shifts may occur due to various innocuous reasons. For example, a driver may downshift gears instead of pressing the brake pedal to slow down the car or to increase torque when approaching a slope (in case of a manual gearbox, otherwise, the same actions will be done by a dedicated ECU). None of these actions is an attack, yet they may lead to abrupt changes in engine speed and torque.

By consulting the related works, the reported FPR is 1.60% in [28], 4.68% in [36] or even 6.45% in [33]. The work in [23] reports a FPR of 0% with the use of cryptographic security. Regarding the FNR, this is 2.80% in [28], 2.40% in [36] and 3.90% in [33]. Again, the work in [23] reports a 0.01% FNR but by using cryptographic security. Therefore, if better FPR and FNR are needed, the only solution would be to rely on cryptography and add security elements to each CAN frame. According to the AUTOSAR SecOC [53], 32 bits of security elements are recommended for each CAN frame, consisting of an authentication tag and a timestamp. In this case, the authentication tag is 24–28 bits (according to security profiles 1–3) and the chances of forgery 2^{-24}–2^{-28} [53]. We note however that the freshness parameter is just 0–8 bits (depending on the security profile) and thus the chance of successful replay attacks is still at least 2^{-8} which is quite high, the freshness parameter repeats after only 256 frames.

The impact of the aforementioned attacks on safety is immediate since all these signals, e.g., vehicle speed, torque and engine speed, are used by various ADAS (Advanced Driver Assistance Systems) systems and they can also produce unintended acceleration/deceleration of the vehicle which may lead to collisions. Note that a mere $5km/h$ deviation from the actual vehicle speed translates into $1m/s$ which may have serious consequences when estimating the time to a collision.

D. Runtime performance

We evaluate the performance of the detection algorithms on four automotive-graded platforms from well-known manufacturers: S12XF (NXP), SAM V71 (Microchip), TC277 and TC297 (Infineon). One of them is from the low-end sector, the 16-bit S12XF clocked at maximum 50MHz that is equipped with 512KB of Flash and 32KB of RAM. The other controllers are high-end operating on 32 bits. The Microchip SAM V71 Xplained Ultra board is powered by the ATSAMV71Q21 microcontroller, which follows the ARM Cortex M7 specification and is capable of a clock frequency of 300MHz.

The Infineon devices are powered by the TC277 and TC297 microcontrollers, both of them based on the Aurix TriCore architecture but with different quantities of available resources, e.g., the TC277 board clocks at 200MHz and the TC297 at 300MHz. The experimental setup employed in our evaluation is depicted in Figure 16. All processors were configured to run at the maximum supported speed.

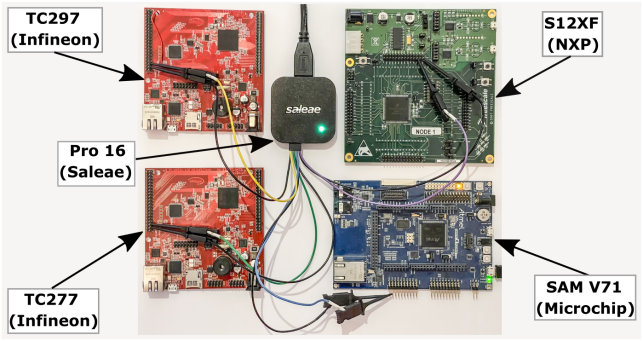


Fig. 16. Experimental setup employed in our evaluation

Regarding the detection algorithms, both DTC and RFC were configured to operate with the default parameters and the number of estimators for RFC was set to 15. k-NN was not ported on the controllers since it requires to much memory for storing the neighbors information regarding legitimate or intrusion frames. Table VI summarizes our results. We only show performance for the engine speed as the results are similar for the rest. Columns two and three specify the classifier that was employed and the attack type. Finally, the last four columns present the execution time of the classifier. Not surprisingly, the RFC executes substantially slower than the DTC as it relies on multiple estimators. On the low-end platform, for both machine learning classifiers, the runtime varies from 60.74μs to 715μs while the simpler change detection mechanism takes only 39μs. On the high-end platforms, the runtime of the machine learning classifiers is between 0.74μs and 18.89μs while for the change detection mechanism is between 0.13μs and 1.32μs. This allows us to conclude that, besides the much better detection rate, which was proved in the previous section, the change detection classifier is 2-65 times faster. Also, the change detection mechanism has extremely low memory requirements since it needs only a few lines of code while the machine learning algorithms have a footprint of 10-30Kb.

VI. Conclusion

Our work shows that despite increasing efforts in addressing CAN bus intrusions by the mean of machine learning algorithms, such algorithms cannot cope with modifications

TABLE IV
WHEEL BASED VEHICLE SPEED, TRIP DISTANCE, ENGINE SPEED AND TORQUE - DETECTION RESULTS FOR 1000 CAN FRAMES (SHORT 100S SIMULATION)

Adversary / IDS parameters					Detection results							
J1939 signal	**Att. type**	**Att. prob.**	**Bias Adv. / IDS**	**Threshold Adv. / IDS**	**TN (frames)**	**TP (frames)**	**FP (frames)**	**FN (frames)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**Precision (%)**
Vehicle Speed (km/h)	fuzzing	25%	- / 0.5	- / 2	759	233	0	8	0.00	3.32	99.20	100.00
	surge	-	5 / 0.5	10 / 2	734	266	0	0	0.00	0.00	100.00	100.00
	bias	25%	5 / 0.5	10 / 2	842	158	0	0	0.00	0.00	100.00	100.00
	geometric	25%	5 / 0.5	10 / 2	838	162	0	0	0.00	0.00	100.00	100.00
Trip Distance (km)	fuzzing	25%	- / 0.05	- / 0.075	759	229	0	12	0.00	4.98	98.80	100.00
	surge	-	0.2 / 0.05	0.25 / 0.075	614	386	0	0	0.00	0.00	100.00	100.00
	bias	25%	0.2 / 0.05	0.25 / 0.075	842	158	0	0	0.00	0.00	100.00	100.00
	geometric	25%	0.2 / 0.05	0.25 / 0.075	847	153	0	0	0.00	0.00	100.00	100.00
Engine Speed (rpm)	fuzzing	25%	- / 100	- / 150	753	196	6	45	0.79	18.67	94.90	97.03
	surge	-	400 / 100	600 / 150	636	357	7	0	1.09	0.00	99.30	98.08
	bias	25%	400 / 100	600 / 150	820	172	6	2	0.73	1.15	99.20	96.63
	geometric	25%	400 / 100	600 / 150	820	174	6	0	0.73	0.00	99.40	96.67
Torque (Nm)	fuzzing	25%	- / 200	- / 250	752	192	7	49	0.92	20.33	94.40	96.48
	surge	-	500 / 200	1000 / 250	698	277	12	13	1.69	4.48	97.50	95.85
	bias	25%	500 / 200	1000 / 250	752	230	7	11	0.92	4.56	98.20	97.05
	geometric	25%	500 / 200	1000 / 250	752	239	7	2	0.92	0.83	99.10	97.15

TABLE V
WHEEL BASED VEHICLE SPEED, TRIP DISTANCE, ENGINE SPEED AND TORQUE - DETECTION RESULTS FOR 36000 CAN FRAMES (LONG 1H SIMULATION)

Adversary / IDS parameters					Detection results							
J1939 signal	**Att. type**	**Att. prob.**	**Bias Adv. / IDS**	**Threshold Adv. / IDS**	**TN (frames)**	**TP (frames)**	**FP (frames)**	**FN (frames)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**Precision (%)**
Vehicle Speed (km/h)	fuzzing	25%	- / 0.5	- / 2	26988	8676	0	336	0.00	3.73	99.07	100.00
	surge	-	5 / 0.5	10 / 2	2036	33964	0	0	0.00	0.00	100.00	100.00
	bias	25%	5 / 0.5	10 / 2	27202	8798	0	0	0.00	0.00	100.00	100.00
	geometric	25%	5 / 0.5	10 / 2	27192	8808	0	0	0.00	0.00	100.00	100.00
Trip Distance (km)	fuzzing	25%	- / 0.05	- / 0.075	26988	8974	0	38	0.00	0.42	99.89	100.00
	surge	-	0.2 / 0.05	0.25 / 0.075	35221	779	0	0	0.00	0.00	100.00	100.00
	bias	25%	0.2 / 0.05	0.25 / 0.075	27071	8929	0	0	0.00	0.00	100.00	100.00
	geometric	25%	0.2 / 0.05	0.25 / 0.075	27076	8422	0	502	0.00	5.63	98.61	100.00
Engine Speed (rpm)	fuzzing	25%	- / 100	- / 150	26851	7481	137	1531	0.51	16.99	95.37	98.20
	surge	-	400 / 100	600 / 150	25145	10758	97	0	0.38	0.00	99.73	99.11
	bias	25%	400 / 100	600 / 150	27021	8803	137	39	0.50	0.44	99.51	98.47
	geometric	25%	400 / 100	600 / 150	27021	8798	137	44	0.50	0.50	99.50	98.47
Torque (Nm)	fuzzing	25%	- / 200	- / 250	26805	7023	183	1989	0.68	22.07	93.97	97.46
	surge	-	500 / 200	1000 / 250	34352	1450	179	19	0.52	1.29	99.45	89.01
	bias	25%	500 / 200	1000 / 250	26805	8828	183	184	0.68	2.04	98.98	97.97
	geometric	25%	500 / 200	1000 / 250	26805	8964	183	48	0.68	0.53	99.36	98.00

TABLE VI
PERFORMANCE OF IDS ALGORITHMS ON AUTOMOTIVE PLATFORMS

J1939 sig.	**Algorithm**	**Attack**	**Execution time on target (μs)**			
			S12XF	**SAM V71**	**TC277**	**TC297**
Engine Speed	DTC	fuzzing	199.12	18.89	2.34	1.86
		surge	60.74	5.49	0.91	0.74
		bias	130.23	12.48	1.64	1.28
		geometric	153.83	15.07	1.94	1.51
	RFC	fuzzing	575.20	16.92	13.12	8.55
		surge	355.65	7.69	7.41	5.35
		bias	620.38	15.44	8.91	6.41
		geometric	715.00	16.98	9.15	6.56
	Change detection	**all**	**39.10**	**1.32**	**0.34**	**0.13**

inside the data carried by CAN frames. The shortcomings are due to two obvious facts: the limited time available for training which implies that the training dataset cannot cover all possible states of the bus, and the small changes that can be induced by adversaries making the attack stealthy. By using specific change detection mechanisms, the computational costs are lower and attack detection has much higher accuracy and precision. The implementation of such mechanisms does require redundancy, but we have shown that this is possible in the J1939 model that served our experiments. Consequently, as there are little or no papers to address such fine grain details, we hope that our work paves way in this direction.

REFERENCES

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive Experimental Analyses of Automotive Attack Surfaces." in *USENIX Security Symposium*. San Francisco, 2011.

[3] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *Black Hat USA*, 2014.

[4] H. Lee, S. H. Jeong, and H. K. Kim, "OTIDS: A Novel Intrusion Detection System for In-vehicle Network by using Remote Frame," in *Procedings of PST (Privacy, Security and Trust)*, 2017.

[5] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: A story of telematic failures," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[6] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking Tesla from Wireless to CAN bus," *Briefing, Black Hat USA*, vol. 25, pp. 1–16, 2017.

[7] S. Nie, L. Liu, Y. Du, and W. Zhang, "Over-the-air: How we remotely compromised the gateway, BCM, and autopilot ECUs of Tesla cars," *Briefing, Black Hat USA*, 2018.

[8] *CAN Specification Version 2.0.*, Robert BOSCH GmbH, 1991.

[9] "ISO/SAE 21434:2021 Road vehicles — Cybersecurity engineering," ISO, Standard, 1st edition, Aug 2021.

[10] W. Choi, S. Lee, K. Joo, H. J. Jo, and D. H. Lee, "An Enhanced Method for Reverse Engineering CAN Data Payload," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3371–3381, 2021.

[11] 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, 2019.

[12] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: risk assessment, detection, and response," in *ACM symposium on information, computer and communications security*, 2011, pp. 355–366.

[13] *Requirements on Intrusion Detection System*, AUTOSAR, 2020.

[14] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems," *IEEE Trans. on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2019.

[15] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-Vehicle Network Attacks and Countermeasures: Challenges and Future Directions," *IEEE Network*, vol. 31, no. 5, pp. 50–58, 2017.

[16] P. Murvay and B. Groza, "Security Shortcomings and Countermeasures for the SAE J1939 Commercial Vehicle Bus Protocol," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4325–4339, 2018.

[17] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical DoS Attacks on Embedded Networks in Commercial Vehicles," in *Information Systems Security*. Springer, 2016, pp. 23–42.

[18] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, "Truck Hacking: An Experimental Analysis of the SAE J1939 Standard," in *10th USENIX Workshop on Offensive Technologies*, 2016.

[19] S. Hariharan, A. V. Papadopoulos, and T. Nolte, "On in-vehicle network security testing methodologies in construction machinery," in *27th International Conference on Factory Automation*, 2022, pp. 1–4.

[20] M. Zachos, ""securing j1939 communications using strong encryption with fips 140-2"," in *SAE Technical Paper*. SAE International, 03 2017.

[21] H. Shirazi, I. Ray, and C. Anderson, "Using Machine Learning to Detect Anomalies in Embedded Networks in Heavy Vehicles," in *Foundations and Practice of Security*, 2020, pp. 39–55.

[22] S. Mukherjee, J. Walkery, I. Rayz, and J. Daily, "A Precedence Graph-Based Approach to Detect Message Injection Attacks in J1939 Based Networks," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017, pp. 67–6709.

[23] C. Jichici, B. Groza, R. Ragobete, P.-S. Murvay, and T. Andreica, "Effective Intrusion Detection and Prevention for the Commercial Vehicle SAE J1939 CAN Bus," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–15, 2022.

[24] M. Rogers, P. Weigand, J. Happa, and K. Rasmussen, "Detecting CAN Attacks on J1939 and NMEA 2000 Networks," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–15, 2022.

[25] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, and K. Li, "A Survey of Intrusion Detection for In-Vehicle Networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2020.

[26] Y. Chen, W. Hu, M. Alam, and T. Wu, "Fiden: Intelligent Fingerprint Learning for Attacker Identification in the Industrial Internet of Things," *IEEE Trans. on Industrial Informatics*, vol. 17, no. 2, pp. 882–890, 2021.

[27] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th {USENIX} Security Symposium*, 2016, pp. 911–927.

[28] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS one*, vol. 11, no. 6, p. e0155781, 2016.

[29] S. Boumiza and R. Braham, "An Anomaly Detector for CAN Bus Networks in Autonomous Cars based on Neural Networks," in *Intl. Conf. on Wireless and Mobile Comp., Networking and Comm.* IEEE, 2019, pp. 1–6.

[30] H. Sun, M. Chen, J. Weng, Z. Liu, and G. Geng, "Anomaly detection for in-vehicle network using CNN-LSTM with attention mechanism," *IEEE Trans. on Veh. Tech.*, vol. 70, no. 10, pp. 10 880–10 893, 2021.

[31] S. Longari, D. H. Nova Valcarcel, M. Zago, M. Carminati, and S. Zanero, "CANnolo: An Anomaly Detection System Based on LSTM Autoencoders for Controller Area Network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1913–1924, 2021.

[32] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, "Classification Approach for Intrusion Detection in Vehicle Systems," 2018.

[33] M. Al-Saud, A. M. Eltamaly, M. A. Mohamed, and A. Kavousi-Fard, "An intelligent data-driven model to secure intravehicle communications based on machine learning," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 6, pp. 5112–5119, 2019.

[34] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, 2015, pp. 45–49.

[35] D. Tian, Y. Li, Y. Wang, X. Duan, C. Wang, W. Wang, R. Hui, and P. Guo, "An Intrusion Detection System Based on Machine Learning for CAN-Bus," in *International Conference on Industrial Networks and Intelligent Systems*. Springer, 2017, pp. 285–294.

[36] A. Kavousi-Fard, T. Jin, W. Su, and N. Parsa, "An Effective Anomaly Detection Model for Securing Communications in Electric Vehicles," *IEEE Transactions on Industry Applications*, pp. 1–1, 2020.

[37] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks," *Intl. Journal of Embed. Sys.*, vol. 10, no. 1, pp. 1–12, 2018.

[38] S. N. Narayanan, S. Mittal, and A. Joshi, "OBD_SecureAlert: An Anomaly Detection System for Vehicles," in *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[39] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. IEEE, 2016, pp. 1–6.

[40] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Intell. Vehicles Symposium*. IEEE, 2011, pp. 1110–1115.

[41] M. Gmiden, M. H. Gmiden, and H. Trabelsi, "An intrusion detection method for securing in-vehicle CAN bus," in *17th Intl. Conf. on Sciences and Techn. of Automatic Ctrl. & Comp. Eng.* IEEE, 2016, pp. 176–180.

[42] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[43] D. Hadžiosmanović, L. Simionato, D. Bolzoni, E. Zambon, and S. Etalle, "N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2012, pp. 354–373.

[44] C. Sinclair, L. Pierce, and S. Matzner, "An application of machine learning to network intrusion detection," in *Proceedings 15th annual computer security applications conference*. IEEE, 1999, pp. 371–377.

[45] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on industrial control systems," in *ACM Conf. on Computer and Communications Security*, 2016, pp. 1092–1105.

[46] F. Akowuah and F. Kong, "Real-Time Adaptive Sensor Attack Detection in Autonomous Cyber-Physical Systems," in *2021 IEEE 27th Real-Time and Embedded Tech. and Applications Symposium*, 2021, pp. 237–250.

[47] "J1939-71 – Vehicle Application Layer," SAE International, Standard, May 2006.

[48] "Digital Annex of Serial Control and Communication," SAE International, Standard, January. 2020.

[49] "FMS-Standard description," ACEA Task Force HDEI/BCEI, Standard, September. 2021.

[50] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[51] M. Ziv, "CANCAN: Encapsulation of CAN-FD Messages for Circumvention of Security Measures," CYMOTIVE Technologies LTD, Report, June. 2022.

[52] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1044–1055.

[53] *Specification of Secure Onboard Communication*, R20-11 ed., AUTOSAR, November 2020, no. 654.

Camil Jichici is a PhD student at Politehnica University of Timisoara (UPT) since 2018 and worked as a young researcher in the PRESENCE project. He received the Dipl.Ing. degree in 2016 and MsC. degree in 2018, both from UPT. His research interests are on the security of in-vehicle components and networks. He is also working as a software integrator in the automotive industry for Continental Corporation in Timisoara since 2014.

Adriana Berdich is a PhD student at Politehnica University of Timisoara (UPT). She received the Engineer title in 2017 and MsC. degree in 2019, both from UPT. She has a 7-year background in the automotive industry as a Function Software Developer with main focus on torque structure and vehicle motion functions, through all phases of the V-model. Between 2015-2018 she has been working as a software developer in the automotive industry for Continental Corporation in Timisoara. Currently, she continues as a function developer in the automotive industry for Vitesco Technologies focusing on vehicle power-train applications. She was also a research student in the PRESENCE project (2019-2020) focusing on environment-based device association inside cars.

Adrian Musuroi is pursuing a Ph.D. at Politehnica University of Timisoara (UPT) focusing on the security of automotive networks. He graduated his B.Sc. in 2018 and his M.Sc. studies in 2020 at the same university. In 2019 Adrian joined the automotive industry as an embedded software developer at HELLA Romania. Since 2021 he is a cybersecurity engineer at ZF Group.

Bogdan Groza is Professor at Politehnica University of Timisoara (UPT). He received his Dipl.Ing. and Ph.D. degree from UPT in 2004 and 2008 respectively. In 2016 he successfully defended his habilitation thesis having as core subject the design of cryptographic security for automotive embedded devices and networks. He has been actively involved inside UPT with the development of laboratories by Continental Automotive and Vector Informatik. Besides regular participation in national and international research projects in information security, he lead the CSEAMAN (2015-2017) and PRESENCE (2018-2020) projects, two research programs dedicated to the security of vehicular ecosystems funded by the Romanian National Authority for Scientific Research and Innovation.