This is the peer reviewd version of the followng article:

READ: Reverse engineering of automotive data frames / Marchetti, M., Stabili, D.. - In: IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. - ISSN 1556-6013. - 14:4(2019), pp. 1083-1097. [10.1109/TIFS.2018.2870826]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

07/06/2026 20:38

(Article begins on next page)

READ: Reverse Engineering of Automotive Data Frames

Mirco Marchetti and Dario Stabili

***Abstract*— Security analytics and forensics applied to in-vehicle networks are growing research areas that gained relevance after recent reports of cyber-attacks against unmodified licensed vehicles. However, the application of security analytics algorithms and tools to the automotive domain is hindered by the lack of public specifications about proprietary data exchanged over in-vehicle networks. Since the controller area network (CAN) bus is the de-facto standard for the interconnection of automotive electronic control units, the lack of public specifications for CAN messages is a key issue. This paper strives to solve this problem by proposing READ: a novel algorithm for the automatic Reverse Engineering of Automotive Data frames. READ has been designed to analyze traffic traces containing unknown CAN bus messages in order to automatically identify and label different types of signals encoded in the payload of their data frames. Experimental results based on CAN traffic gathered from a licensed unmodified vehicle and validated against its complete formal specifications demonstrate that the proposed algorithm can extract and classify more than twice the signals with respect to the previous related work. Moreover, the execution time of signal extraction and classification is reduced by two orders of magnitude. Applications of READ to CAN messages generated by real vehicles demonstrate its usefulness in the analysis of CAN traffic.**

***Index Terms*— In-vehicle networks, CAN bus, reverse engineering, automotive.**

I. Introduction

THE leading trend in the automotive industry is to bring self-driving capabilities and connected vehicles on the roads. While these innovations aim to enhance the driving experience and safety, they also increase the attack surface that cyber-attackers can exploit to compromise the security of connected vehicles. These novel attack opportunities, that in some cases can be (and have been) exploited remotely, create safety and security hazards for drivers, passengers and pedestrians. Several proof-of-concept attacks have already been illustrated by security researchers and described in papers, presentations and technical reports [1]–[3]. In all these cases the attacks were conducted by injecting malicious packets into the communication networks that connect all the computing elements that control many features of modern cars. These computing elements, usually called Electronic Control Units (ECUs) are microcontrollers deployed within the vehicle and exchange data through one or more in-vehicle networks realized through industrial communication buses. In all modern vehicles, the de-facto standard for implementing these in-vehicle networks is the Controller Area Network (CAN) bus. Physical or remote access to the CAN bus of a vehicle allows attackers to forge and inject any message, even those controlling safety-relevant features of the vehicle impacting its dynamic, such as acceleration, brakes and steering.

The complete specifications of the syntax of CAN messages for any given car model is included in the so-called *Communication Database for CAN*, also known as DBC. For each CAN message, the DBC of a vehicle defines its cycle time (many CAN messages are sent periodically), the ECU that generates the message, the ECUs for which the message is intended, and all other specifications required to interpret the message contents. To this aim the DBC includes complete descriptions of the different signals that are packed together within the payloads of all CAN bus messages, including their boundaries, encodings and units of measurement. Moreover, different vehicle configurations are described by different DBC files, thus preventing the applicability of the same DBC for different vehicle set-ups and models, even though they are produced by the same car maker. These formal specifications are kept confidential by car manufacturers. From the attacker's perspective, the lack of public specifications requires a reverse engineering effort to pinpoint the CAN messages that impact a specific feature. However, several researchers already published many cyber attacks on automotive ECUs based on reverse-engineering of undocumented CAN messages, thus confirming the shared opinion that any security approach that relies on obscurity will inevitably fail. While being of little effectiveness against attackers, the lack of public specifications hinders the work of security researchers trying to design and develop novel approaches for securing connected vehicles from cyber threats. Knowledge of the formal specification of CAN messages can improve the effectiveness of many security analytics algorithms that have already been applied to CAN traffic, and can pave the way for the application of novel approaches. Indeed, previous works in the field of CAN bus traffic analysis and anomaly detection [4]–[6] are only based

on the few standard features that do not vary among different car makers and models, and are part of the CAN standard itself [7]. On the other hand, more effective analysis techniques that require knowledge of the syntax of CAN messages to inspect the evolution of safety-relevant signals are still largely unexplored.

The first attempt towards autonomous reverse engineering of automotive data frames has been recently proposed in [8], where the authors describe an algorithm that automatically extracts signals and their boundaries from a sequence of CAN messages. While this work is promising, its effectiveness have never been measured against real CAN traffic.

This paper proposes several new contributions. First of all, we propose a novel algorithm for the extraction of signal boundaries within CAN messages called READ: Reverse Engineering of Automotive Data frames. Moreover, extracted signals are automatically associated to a label that describes the signal semantic. Finally, to the best of our knowledge this is the first paper including an experimental evaluation based on CAN messages recorded from a real, unmodified and licensed vehicle during several hours of driving in real roads and different traffic conditions.

Results of READ, as well as of previous work [8], are then compared against the formal specifications of CAN messages for our test vehicle, that were made available to us by a Tier 1 supplier of automotive electronic components. These data provide a ground truth that allows us to evaluate the performance of both approaches accurately and without biases introduced by simulation errors and wrong assumptions about the real nature of automotive ECUs, CAN networks and CAN messages.

Experimental results show that READ is able to extract a higher number of signals with better accuracy compared to previous proposals, and achieves this result with lower execution times.

The rest of the paper is organized as follows: in Section II we discuss related works, Section III provides the basic knowledge related to CAN bus and the traffic traces used in the paper. Section IV describes the proposed READ algorithm. Its performance evaluation is provided in Section V. Examples of real-life applications of the proposed algorithm are given in Section VI. Finally, Section VII draws conclusions and outlines future work.

II. Related Work

Cyber attacks to modern vehicles executed by injecting forged and malicious messages in the CAN bus [1]–[3], [9] spawned several research efforts aimed at improving the security level of modern vehicles. Some works aim to improve the security of communications over the CAN bus by applying cryptographic protocols [10], [11]. However similar solutions require to modify all the ECUs involved in secure communication and have profound impacts on the whole life-cycle of a vehicle [12]. Other less intrusive approaches apply anomaly detection [13] and security analytics algorithms to the traffic flowing on the CAN bus. Several algorithms for the identification of intrusions over the CAN bus have already been proposed, mainly by applying and adapting approaches borrowed from the IT and network security domain to the specific characteristics of the CAN bus and its messages.

The simplest approach leverages the relatively small number of legitimate message IDs that are generated by ECUs on board of modern vehicles (usually between 100 and 150) to detect malicious CAN bus messages injected by an attacker and having an invalid ID [14]. In this case, the normal model is represented by the set of valid IDs, either collected by inspecting the internal traffic of a given vehicle or derived from the vehicle specifications. In both cases, this approach allows easy and precise identification of attacks that inject CAN messages with an invalid ID, but can be easily evaded by injecting arbitrary messages with valid IDs and forged payloads.

Other approaches are based on the analysis of the frequency of periodic CAN messages [6], [15], [16]. Since most CAN messages are generated periodically, messages having the same ID and injected by an external attacker will necessarily cause anomalies in the message inter-arrival times. While theoretically sound, a similar proposal cannot be easily applied in practice. Inter-arrival times of periodic CAN messages extracted from real CAN traffic often exhibit an unexpectedly high variability, either caused by message contention and bus arbitration, or by messages whose cycle time may change over time or based on specific events. This variability leads to the generation of many false positives, that are not acceptable in the automotive domain. As an example, a false positive rate of 0 .00298 [15] over the CAN bus of a modern vehicles transmitting between 3000 and 4000 messages per second implies the generation of tens of false positive each second. Moreover, this approach is inapplicable to aperiodic messages for which a cycle time cannot be defined.

The idea to use more complex statistical features to model the normal behavior of legitimate CAN bus traffic has already been discussed in [5] and [17]. These approaches propose anomaly detectors based on entropy and achieved good detection results for attacks characterized by the injection of large batches of messages in a short time frame. However this approach is not effective against targeted attacks carried out by injecting only one or very few messages, as well as against slow attacks injecting many messages over an extended time frame.

A different approach has been shown in [4], in which a model of the normal behavior of legit CAN traffic is created by inspecting the possible transitions between consecutive message IDs. Despite achieving good detection results, this work is limited only to the inspection of the message IDs, behaves poorly against targeted injection of messages characterized by a very short cycle time, and can be evaded by attackers that manage to alter the payload of legitimate CAN messages.

Another approach is presented in [18], where the authors propose a method to detect anomalous messages by evaluating the hamming distance between consecutive payloads of the same CAN ID.

We can observe that the main limitation shared by all the aforementioned research efforts lies in the very few features that can be extracted and analyzed from a generic

traffic trace containing CAN messages. Indeed, message arrival time, ID and the binary blob of its payload only enable a very coarse-grained message classification, that can be useful in detecting simple attacks but is bound to fail in detecting more stealth intrusions comprising the injection of few well-designed malicious messages. This issue could be mitigated by having access to the complete formal specifications of CAN messages, including the list and boundaries of all signals encoded in their payload. Unfortunately these information are only available to car makers and their suppliers, and cannot be accessed by the general public, including the vast majority of academic researchers.

Knowledge of the semantic of CAN messages would also be extremely useful in reconstructing the state of the vehicle before a crash, possibly identifying driver mistakes, failures of the vehicle or anomalous activities attributable to a cyber attack. Nilsson and Larson [19] are the first to address this issue by proposing a list of requirements for detection, data collection and event reconstruction following a crash. This aspect has been further inspected by Mansor *et al.* [20], that proposed a reliable, secure, privacy-preserving and efficient mechanism to build a forensics data collection and storage system. Despite these solutions, it is clear that analysis of raw CAN messages requires to manually inspect high volumes of data to reverse-engineer messages syntax and semantic, reconstruct the vehicle dynamic and contextualize the messages and their contents.

The network traffic analysis literature already includes many proposals aiming to automatically recognize the nature of a given network packet or flow, as an example by attributing a network communication to a specific application or protocol. These works are mostly based on three main approaches: matching of known signatures within network packets; analysis of source and destination port numbers at the transport layer; applying a classification algorithm to packet metadata [21]–[23]. We remark that these works have been designed to analyze TCP/IP network traffic and to identify only well known network applications (such as web browsing, email, chat and file transfer protocols). All these assumptions render these approaches inapplicable to the automotive domain characterized by in-vehicle networks that leverage completely different protocols and communication patterns. As an example, CAN messages do not include source and destination addresses nor port numbers, are broadcast communications that do not establish a bidirectional communication flow, and lack the clear separation between network, transport and application layers that are typical of IT networks.

Some information about semantic and syntax of CAN messages can be extrapolated through reverse engineering, as proposed in [1], [2], and [24]. However, all these approaches are based on manual inspection of a high number of CAN bus messages by a reverse engineer with experience in the automotive domain, that is a daunting and human-intensive task.

The first proposal toward automatic reverse engineering of signals conveyed in CAN messages, specifically aimed at providing more useful features for anomaly detection algorithms, can be found in [8]. This work proposes an algorithm that analyzes the payload of CAN messages and tries to extract signals and their boundaries by observing how the payloads of messages sharing the same ID evolve over time. However, the heuristics proposed in [8] have never been tested against real CAN messages, since their experimental evaluation is based on CAN traffic generated by a laboratory environment with simulated ECUs, rather than on real licensed vehicles.

This paper has three main novel contributions that differentiate it with respect to the state of the art. First, it proposes READ, a novel algorithm for the automatic identification of signals embedded in the payload of CAN messages that outperforms previous work [8] by detecting more than twice the number of correct signals and exhibiting much lower execution times. Rather than being an incremental improvement over [8], READ includes a completely novel set of heuristics and a completely different processing algorithm. These heuristics reflect the domain knowledge acquired by authors while manually reverse engineering CAN messages generated by several real and modern vehicles of different models and makers, and facilitate the reverse engineering process by automatically extracting and labeling individual signals from unknown CAN traffic traces.

Second, READ automatically associates a descriptive label to all extracted signals that helps human analysts in making sense of the data. The labels used by READ are specific to the automotive domain and convey a precise semantic meaning, while labels produced by [8] only depend on how the signal evolves over time and do not try to describe its meaning.

Third, this is the first paper in which results are validated against a ground truth. Thanks to cooperation with an industrial partner we applied both READ and the algorithm proposed in [8] to CAN traffic generated by a modern, licensed and unmodified vehicle, and compared their results against the complete formal specifications of the same vehicle. This original approach enables us to provide the first accurate and unbiased evaluation of the effectiveness of algorithms for automatic reverse engineering of CAN messages.

III. Background

The Controller Area Network (CAN) is a standard communication bus targeted to industrial applications and designed to allow data exchange among microcontrollers without requiring a host computer [7]. This technology represents the de-facto standard in the implementation of in-vehicle communication buses among Electronic Control Units (ECUs) deployed in modern cars.

The READ algorithm proposed in this paper focuses on the inspection of CAN *data frames*, a particular type of CAN message that carries data across the CAN bus and that is used by ECUs to send and receive arbitrary payloads. A graphical representation of the structures of the different CAN data frames is shown in Figure 1.

The main fields are *identifier (ID)*, *data length code (DLC)* and *data*. The *ID* is used to distinguish among different types of CAN data frame. Data frames characterized by a given ID are usually produced by only one ECU, and consumer ECUs use the ID to select the relevant data frames among

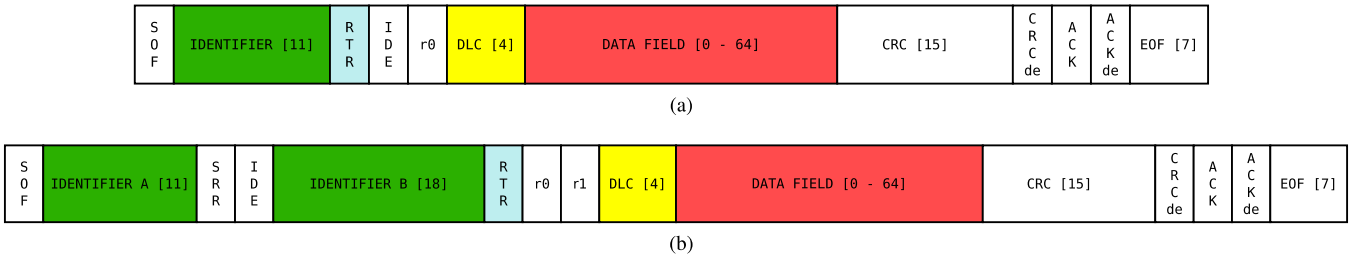


Fig. 1. Data frame types comparison. (a) Base format. (b) Extended format.

all CAN messages transmitted over the CAN bus. The ID is also used for arbitration of the CAN messages, where lower values of this field denote messages with higher priority. The ID is the only field that has different sizes depending on the type of CAN message. As shown in Figure 1a, the ID field for the *base format* has a fixed length of 11 bits, while in the *extended format* (shown in Figure 1b) the ID is 29 bits long. Also note that the extra 18 bits of the *extended format* are encoded separately from the 11 bits of the *basic format* to achieve backward compatibility.

The *DLC* field has a size of 4 bits and represents the number of bytes composing the *data* field. Since the maximum length of the data field is 8 bytes, valid DLC values only ranges between 0 and 8, thus values from 9 to 15 are unused.

The *data* field has a maximum size of 64 bits and is used to represent the actual payload of a CAN message. A generic data frame usually packs several different signals within the same data field, and the CAN standard leaves complete freedom to the car makers about the structure, number and meaning of signals. Hence, without having access to the formal specifications of CAN messages, their data field can only be interpreted as an opaque binary blob.

To design the READ algorithm and evaluate its performance we recorded CAN traffic traces from an unaltered licensed vehicle provided by our industrial partner. Collected data belongs to different CAN traffic traces, gathered during several hours of driving sessions in different times of the day and in different traffic conditions. While driving, different dashboard buttons and features were activated when necessary (e.g. windshield wipers while raining, turning indicator while changing lanes). The collected CAN traffic traces account for more than 126 million CAN data frames and include about 110 different message IDs, of which about 70 are in the base format while the remaining are in the extended format.

IV. The READ Algorithm

The Reverse Engineering of Automotive Data frame (READ) algorithm proposed in this paper extracts individual signals from CAN traffic by inspecting all the bits of the data field of all observed CAN messages and evaluating their evolution over time.

ECUs communicate by exchanging messages that deliver values gathered from different sensors to actuators that control several subsystems of the vehicle. In particular, most of the signals generated by sensors encode the current value of a given physical phenomena, such as the speed at which a wheel is rotating or the acceleration measured along a given axis. The evolution over time of similar signals is unpredictable, since it depends on the road and driving conditions. However it is clearly limited by physical constraints. Since many signals are issued on the CAN bus according to a predefined cycle time (such as a hundredth or a tenth of a second), the difference among two consecutive values of a signal representing a physical phenomena is necessarily small, and constrained by the cycle time and by the nature of the observed phenomena.

Let us consider as an example the CAN signal representing the rotation speed of the front left tire, and let us assume that this signal is sent over the CAN bus every 10 milliseconds. Consecutive values will necessarily be very similar, hence only their less significant bits will change. However, over time the tire rotational speed will change significantly, thus leading to the modification of more significant bits of the signal.

READ analyzes the ordered sequence of payloads of CAN messages having the same ID. Each bit of the payload is analyzed to determine the frequency of changes in the bit value among consecutive payloads (bit-flip). The READ algorithm applies several novel heuristics to:

- define and extract the boundaries of different signals within payload of messages having the same CAN ID;
- label extracted signals according to different classes representing the signal type.

The different phases of the READ algorithm are described in the following sections.

A. Data Preparation

The preliminary step required to execute READ is to create ordered lists of CAN messages having the same ID. Let us assume that the main input is a CAN traffic trace including all CAN messages exactly as they were transmitted over the CAN bus of a licensed vehicle. During this phase this single trace is split into several sub-traces, one for each different ID included in the original input. Each sub-trace only contains the payload of CAN messages having a single ID, in the same order as they appear in the original input. We remark that since READ analyzes each sub-trace independently, it is possible to have multiple instances of READ running in parallel on different sub-traces. An overview of the data preparation phase of the READ algorithm is given in Figure 2.

B. READ Processing Steps

The algorithm does not rely on any a-priori knowledge about the nature of the message payloads nor of the signals encoded

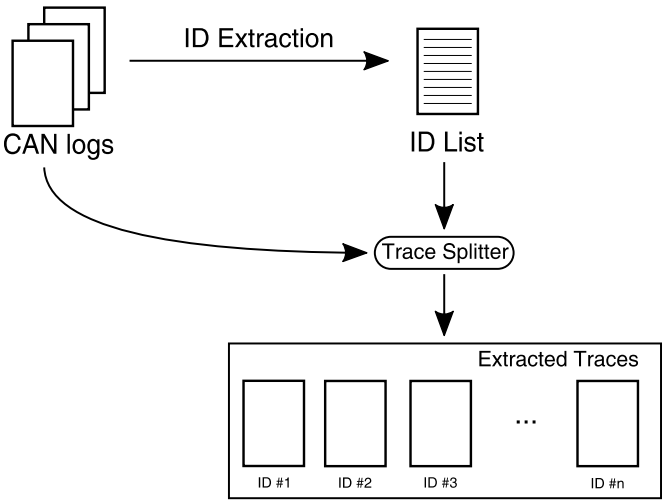


Fig. 2. Preparation phase of READ.

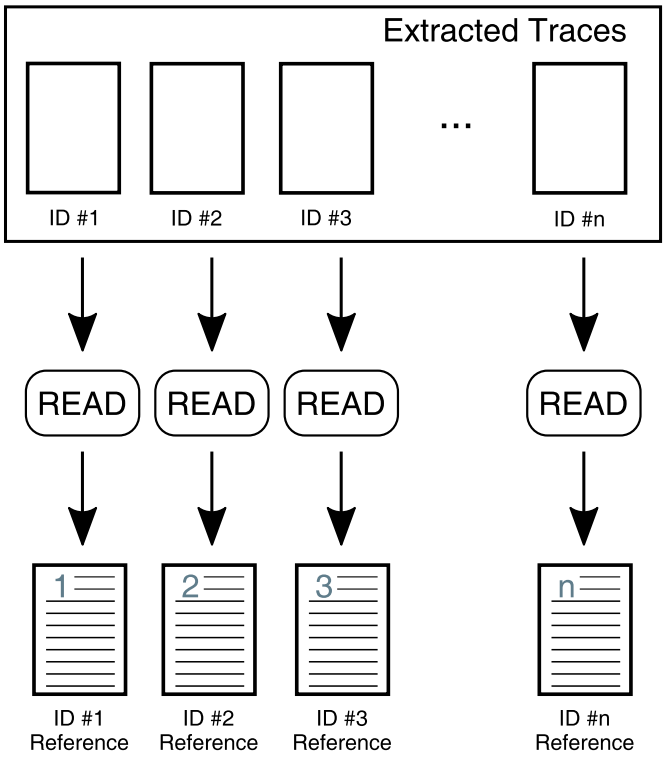


Fig. 3. READ work-flow.

within the payloads. The input of the algorithm is composed by a list of payload values composed by a number of bits dependent on the payload length as specified by the DLC field. The output is represented by the list of signals included in the payload, their boundaries, and a label describing the type of each signal. An overview of the READ algorithm work-flow is given in Figure 3.

Figure 4 shows the detailed steps performed by the READ algorithm. The pre-processing phase analyzes the messages payloads and computes the metadata used by the next phases. Phase 1 evaluates the preliminary references for the signals that are further refined in Phase 2.

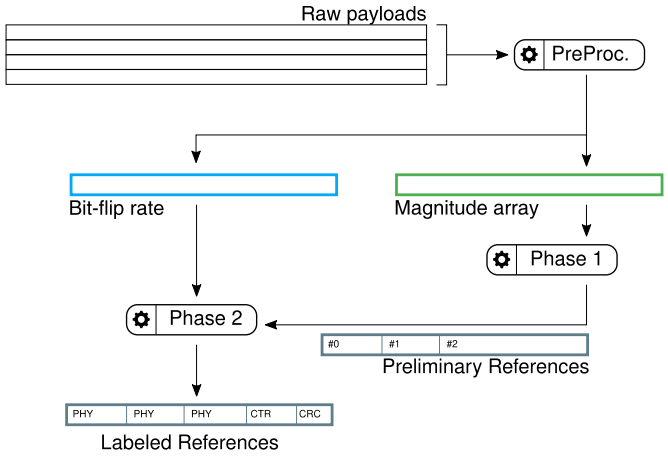


Fig. 4. Main processing steps of the READ algorithm.

1) Pre-Processing: The bit-flip rate is evaluated for each bit of the payload, independently of its neighbors. First READ counts the number of bit-flips (from 0 to 1 and vice versa) occurrences among consecutive messages. Then the bit-flip rate is obtained by dividing the number of bit-flips for the number of payloads. The result of this phase is an array of n elements, each representing the bit-flip rate for a single bit of the data field of a given ID, where n represents the number of bits included in the payload as defined by the value of the *DLC* field.

This intermediate result is an input for the computation of another array of n elements, defined as the *magnitude* array. The formula used to compute the magnitude array is shown in Equation 1, where M_i is the i_{th} element of the magnitude array and B_i is the i_{th} element of the bit-flip array.

$$M_i = \log_{10}(B_i), \quad 0 \leq i < n \tag{1}$$

Values of the magnitude array represent the different orders of magnitude of the bit-flip rate for each bit of the payload. The pseudo-code describing the preprocessing step of the READ algorithm is included in Algorithm 1.

Algorithm 1 Pseudo-Code of the Pre-Processing Step

```
1: function PRE-PROCESSING (messageList, DLC)
2:     payloadLen ← len(messageList)
3:     bitFlip ← array(DLC)
4:     magnitude ← array(DLC)
5:     previous ← messageList[0]
6:     while item in messageList do
7:         for ix in range(1..DLC) do
8:             if item[ix] = previous[ix] then
9:                 bitFlip[ix] + +
10:    for ix = 0; ix < DLC; ix + + do
11:        bitFlip[ix] ← bitFlip[ix]/payloadLen
12:        magnitude[ix] ← log10(bitFlip[ix])
13:    return bitFlip, magnitude
```

After computing the bit-flip rate and magnitude arrays, the algorithm inspects them to identify the signal boundaries.

This process involves two phases. The first phase only considers the magnitude array, and produces as output a preliminary list of signal boundaries. The second phase leverages these preliminary boundaries and the bit-flip rate to identify the precise boundaries of each signal and to label them according to their nature.

2) Phase 1: In this phase the magnitude array is used for the definition of preliminary signal boundaries. The algorithm scans the magnitude array looking for couples of consecutive bits in which the first bit is characterized by a bit-flip magnitude that is higher then the second one. Whenever a similar couple is found, a preliminary boundary is set between the two bits. This heuristic is effective for identifying boundaries of signals that represent physical values, since drops in the bit-flip magnitude are caused by a less significant bit of a signal immediately followed by the most significant bit of the adjacent signal. The pseudo-code describing the main processing steps of Phase 1 is given in Algorithm 2.

Algorithm 2 Pseudo-Code of Phase 1

```
1:  function PHASE 1(magnitude, DLC)
2:      ref ← list()
3:      prevMagnitude ← magnitude[0]
4:      ix S ← 0
5:      for ix in range(1..DLC) do
6:          if magnitude[ix] < prevMagnitude then
7:              refadd((ix S, ix − 1))
8:              ix S ← ix
9:          prevMagnitude ← magnitude[ix]
10:     refadd((ix S, DLC − 1))
11:     return ref
```

3) Phase 2: The second phase takes as input the preliminary boundaries identified in the previous phase together with the bit-flip rate array to identify and correctly label signals that do not represent physical values. Car manufacturers often include metadata in the payload of safety-critical messages to implement two naïve protection strategies that are effective against basic replay attacks [25]. Common solutions adopted by several car manufacturers include two additional types of fields: *Counters* and *CRCs*. Since these metadata are encoded in the payload of CAN messages together with other signals that convey physical values, it is helpful for an analyst to quickly identify them and tell them apart. This is the rationale behind the definition of heuristics specifically tailored to identify counters and CRCs.

Counters are signals whose value always increases by one with respect to the counter of the previous message with the same ID. Counters allow the receiving ECU to recognize a retransmission of a CAN frame that has already been received correctly, as well as messages that are received out of order. Similar conditions happen when an attacker that sniffed a message from the CAN bus performs a replay attack by injecting the same message over the bus. Counters exhibit two peculiar features that differentiate them from messages conveying the value of a physical phenomena:

1) the magnitude of the least significant bit equals 0, since it has a bit flip probability of 1;
2) the bit-flip rate of the elements doubles every step from the most to the least significant one, reaching 1 in the least significant bit.

By looking for similar patterns it is possible to identify counters within signals extracted in the previous phase. As example let us consider the case of a 4-bits counter having the following bit-flip rates: [0.125, 0.25, 0.5, 1]. By applying Equation 1 the magnitude array will be: [0, 0, 0, 0]. Since magnitude values do not change, Phase 1 will fail in identifying its boundaries, and the counter will remain embedded in other signals. However, since bit-flip rates vary according to the defined heuristic, the counter boundaries will be correctly identified by Phase 2.

CRCs are signals that contain the result of a cyclic redundancy check on the message payload to detect random transmission errors in safety-relevant signals. We remark that CRC signals are included within the payload of CAN messages, and do not replace (and should not be mistaken for) the CRC field that follows the payload in CAN data frames (see Figure 1). The algorithm for computing the CRC that follows the payload is public and described in the CAN bus specifications [7], while the algorithm used for evaluating the CRC signals within the message payload is proprietary. Empirical analyses of CAN messages including CRC signals led us to conclude that CRC signals exhibit the following distinctive features:

1) the magnitude of all bits is equal to 0;
2) the bit-flip rate of all bits is distributed according to a normal probability distribution centered in 0.5.

Similarly to the counter fields, boundaries of CRC fields are not detected in Phase 1, and are identified in this phase by looking for the aforementioned pattern of bit-flip rates. If this pattern is found, precise boundaries are set and the signal is labeled as CRC. Algorithm 3 shows the pseudo-code related to Phase 2.

The final output of READ is a list of all the IDs found in the input CAN traffic traces, in which each ID is associated to the number of signals identified by READ and their boundaries. Moreover, each signal is classified as a *Physical value*, a *Counter* or a *CRC*.

C. Analysis of the READ Algorithm

We now analyze three main aspects of READ: computational complexity, correctness and convergence requirements.

READ sequentially applies the three processing steps previously described, hence we evaluate the computational complexity of each step. The computational complexity of the pre-processing phase (see Algorithm 1) depends on two factors: the size of the payload of the analyzed CAN messages and the number of messages included in the analyzed trace. While the payload size varies among different CAN IDs it is constant for all messages having the same ID. Moreover, it is bounded by the constant value of 64. On the other hand, the number of iterations performed by the outer cycle grows linearly with the number of messages included in the analyzed traffic trace. Hence the computational complexity of the pre-processing step grows linearly with the number of messages. The computational complexity of Phase 1 (see Algorithm 2) only depends on the payload size, hence this step

Algorithm 3 Pseudo-Code of the Phase 2

```
1: function PHASE 2(ref, bitFlip)
2:     Ref ← list()
3:     for sign in ref do
4:         ixS, ixE, mgt ← sign        ixS and ixE are the starting and ending indexes of the signal, while mgt is an array of size ixE − ixS with the magnitude values of the signal
5:         mu ← mean(bitFlip[ixS:ixE])
6:         std ← stdDev(bitFlip[ixS:ixE])
7:         if mgt[ixE] = 0 then
8:             Sctr ← matchCounter(bitFlip[ixS:ixE])   matchCounter retruns the starting index of the matched counter pattern, −1 otherwise
9:             if sctr ≥ 0 then
10:                Ref.add((ixS, Sctr, PHYSVAL))
11:                Ref.add((Sctr, ixE, COUNTER))
12:            else
13:                exit ← False
14:                while Scrc in range(ixS.ixE) and not exit do
15:                    if all(mgt[Scrc : ixE]) = 0 and 0.5 − std ≤ mu ≤ 0.5 + std then
16:                        Ref.add((Scrc, ixE, CRC))
17:                        Ref.add((ixS, Scrc, PHYSVAL))
18:                        exit ← True
19:    return Ref
```

has a constant computational complexity. Finally, the computational complexity of Phase 2 (see Algorithm 3) depends on the number of signals extracted by Phase 1. While the number of signals is not known a-priori, it is limited by the payload size. Hence the computational complexity of Phase 2 is also constant. We can conclude that the overall computational complexity of READ grows linearly with the number of analyzed messages. An experimental evaluation of the execution times of READ over real CAN traffic traces is provided in Section V-C.

For READ, as for any data-driven reverse engineering approach, convergence and correctness are two closely related concepts. READ results are correct if they agree with the formal specification of CAN messages, that represent an arbitrary design choice rather than a theoretical correct or optimal solution. Moreover, READ can converge to correct results only after the analysis of a number of messages. In the worst case, all messages analyzed by READ have an identical payload value. Borrowing from the information theory, we can say that a similar sequence of messages has a conditional entropy [26] equal to 0. Hence READ (nor any other algorithm) can acquire any knowledge from analyzing the sequence of messages, independently of its length. We can conclude that in this worst, degenerate case all reverse-engineering algorithms will never converge to a correct solution. An example of this phenomena is included in Section V-B and shown in Figure 8c.

On the other hand, in the best case READ can converge to optimal results with a very low number of messages. For the correct extraction and labeling of *Physical* signals, READ requires only two consecutive values in which just the least significant bit flips while all other bits remain constant, independently of the signal size. For *CRCs* READ needs to analyze only two consecutive values in which all bits flip, independently of the signal size. For a *Counter* of size c bits, in the best scenario READ needs to analyze $2^c/2 + 1$ consecutive values. As an example, let us consider the realistic case of 4-bit counters. READ can properly extract and label these signals by observing 9 consecutive values that range from 0 to 8. Hence, in this optimistic scenarios we can conclude that the theoretical lower bound to the number of messages that READ has to analyze to converge to correct results is 9.

We stress that both the worst and the best case scenarios are not representative of real READ use cases. An evaluation of the convergence requirements in an "average" scenario would require us to make a-priori and arbitrary assumptions on the evolution over time of signal values, that in reality are influenced by many imponderable and unforeseeable factors such as driving path, traffic conditions and driving stile. Hence we prefer to provide an experimental evaluation of the correctness and convergence requirements over real data in Sections V-B and V-D, respectively.

V. Performance Evaluation

This section evaluates the performance of READ on two different datasets, described in Section V-A. In both cases, the output generated by READ and by an implementation of the algorithm proposed in [8] are compared with respect to the ground truth represented by formal specification of CAN messages. For a fair comparison of execution times, both algorithms are implemented in Python, run on the same hardware and analyze the same input CAN traffic traces.

The two algorithms have been evaluated according to three key performance indicators:

- *Correctness of signal extraction and labeling:* this performance indicator measures the number of signals that the algorithm is able to correctly extract and label (Section V-B);
- *Execution time:* the computational costs of the two algorithms are evaluated in terms of their execution time (Section V-C);
- *Convergence requirements:* the number of messages that the two algorithms need to analyze before achieving their best signal extraction and labeling performance (Section V-D).

A. Datasets Description

To distinguish among the two dataset used on the experimental evaluation, we refer to them with the terms *synthetic* and *real*.

The former dataset is composed by synthetic messages generated algorithmically following manually reverse-engineered boundaries publicly available online. The latter dataset includes real CAN messages recorded from an unmodified licensed vehicle together with their correct formal specifications provided by an undisclosed industrial partner.

The *synthetic* dataset includes CAN messages that were generated algorithmically. To guarantee a high level of

TABLE I

SIGNALS INCLUDED IN THE MESSAGE ID 158

Name	Type	Length (bit)
XMISSION_SPEED	Physical Value	16
ENGINE_RPM	Physical Value	16
XMISSION_SPEED2	Physical Value	16
ODOMETER	Physical Value	8
unlabelled	N/A (constant)	2
COUNTER	Counter	2
CHECKSUM	CRC	4
Total		64

TABLE II

SIGNALS INCLUDED IN THE MESSAGE ID 1*D*0

Name	Type	Length (bit)
WHEEL_SPEED_FL	Physical Value	15
WHEEL_SPEED_FR	Physical Value	15
WHEEL_SPEED_RL	Physical Value	15
WHEEL_SPEED_RR	Physical Value	15
CHECKSUM	CRC	4
Total		64

TABLE III

SIGNALS INCLUDED IN THE MESSAGE ID 201

Name	Type	Length (bit)
INTERCEPTOR_GAS	Physical Value	16
INTERCEPTOR_GAS2	Physical Value	16
unlabelled	N/A (constant)	2
COUNTER	Counter	2
CHECKSUM	CRC	4
Total		40

realism despite the lack of public formal specifications, we used as a reference the signal boundaries of the Acura ILX 2016 CAN messages, available online [27] and manually reverse-engineered by researchers of Comma.AI [28]. We remark that reverse engineering results are partial, and large portions of the payload have not been analyzed for many CAN IDs. For our evaluation we selected three of the CAN IDs for which the reverse-engineered process led to the definition of an almost complete message specification. The selected messages have IDs **158** (Powertrain data), **1D0** (Wheel Speeds) and **201** (Gas Sensor). A description of the signal boundaries used for the generation of the *synthetic* dataset is given in Tables I, II and III, for messages 158, 1D0 and 201, respectively.

The final *synthetic* dataset is composed by 1 , 500, 000 messages, 500, 000 for each of the three IDs. Signal values have been generated algorithmically.

The *real* dataset is composed by 25 different CAN traffic traces. These traces have been collected from the same unmodified licensed vehicle under different driving conditions on real roads and subject to real traffic conditions. The complete dataset represents more than 14 hours of data and more than 125 millions of CAN data frames. The longest trace lasts about 38 minutes, while the shortest is about 32 minutes long. A quantitative description of the *real* data set is given in Table IV, that summarizes the number of messages for each trace and its duration.

TABLE IV

NUMBER OF CAN MESSAGES AND DURATION OF THE 25 CAN TRAFFIC TRACES THAT COMPOSE THE *Real* DATASET

Trace	Number of CAN Messages	Duration (HH:MM:ss)
1	4,646,605	00:32:25
2	4,715,886	00:32:54
3	4,759,489	00:33:11
4	4,589,270	00:32:01
5	4,959,564	00:34:36
6	4,909,393	00:34:15
7	5,532,928	00:38:36
8	5,224,743	00:36:27
9	5,444,530	00:37:59
10	5,598,760	00:38:51
11	4,634,650	00:32:20
12	5,348,991	00:37:19
13	4,558,212	00:31:48
14	4,993,051	00:34:50
15	5,262,967	00:36:43
16	5,145,906	00:35:54
17	5,045,538	00:35:12
18	4,617,935	00:32:13
19	4,942,846	00:34:29
20	4,866,383	00:33:57
21	5,294,014	00:36:56
22	5,497,079	00:38:21
23	5,253,401	00:36:39
24	5,315,515	00:37:05
25	5,399,129	00:37:40
Total	126,523,785	14:42:41

B. Signal Extraction and Labeling

The main performance indicator of READ is the total number of signals correctly extracted from the CAN traffic traces included in the *synthetic* and *real* datasets.

Signal extraction results of both READ and the algorithm described in [8] (labeled as FBCA) for the *synthetic* dataset are represented in Figure 5. In all the three charts of Figure 5, the x-axis represents the single bits of the payload, ranging from 1 to n (with n being the size of the payload), while the y-axis represents outputs for READ (in the top row) and FBCA (in the bottom row). Vertical rows represents the *correct* boundaries of the signals as identified by the message specifications of Tables I, II and III. To better identify the signal boundaries extracted by the two algorithms, consecutive signals are colored with different shades. Signal extraction is correct if vertical rows overlap color changes that identify the boundaries of consecutive signals as computed by READ and FBCA. Figures 5a, 5b and 5c represent the signal boundaries that have been reverse engineered for IDs 158, 1*D*0 and 201, respectively. Since the payload of ID 201 is only 40 bits long, a white tail is used to represent the trailing unused bits of the payload.

From the analysis of Figure 5, it is clear that READ is able to correctly extract all the signals of the *synthetic* dataset. On the other hand, none of the boundaries identified by FBCA on this dataset is correct.

Another relevant metric is the number of labels that the READ algorithm correctly associates to the extracted messages. We highlight that READ and FBCA use two different sets of labels: READ labels signals as *Physical*, *Counter* and *CRC*, while FBCA labels signals as *Constant*, *Multi-Value* and

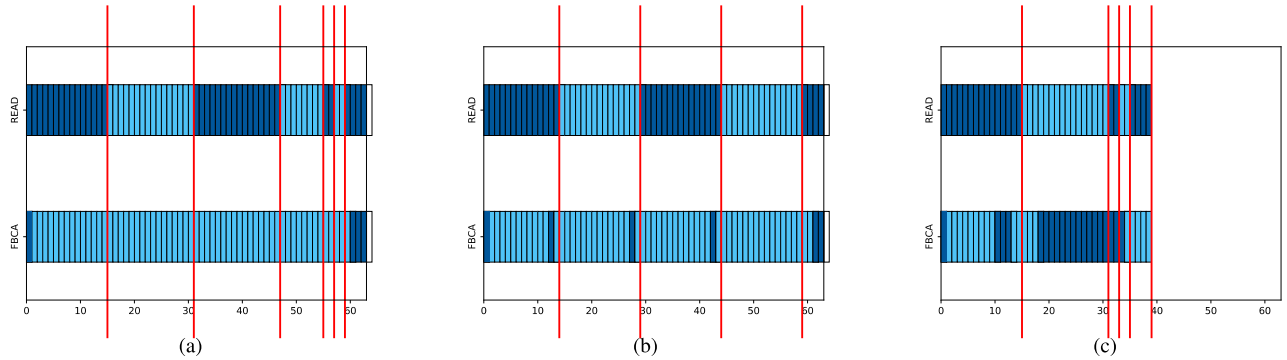


Fig. 5. Signal boundaries extracted from the synthetic dataset by READ and FBCA. (a) Extracted boundaries for ID 158. (b) Extracted boundaries for ID 1D0. (c) Extracted boundaries for ID 201.

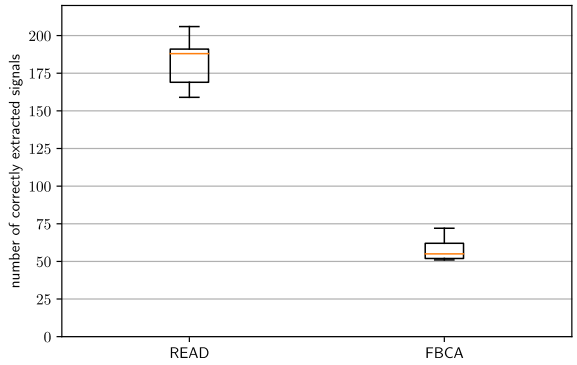


Fig. 6. Comparison of the number of signals correctly extracted by READ and FBCA from the analysis of the *real* dataset.

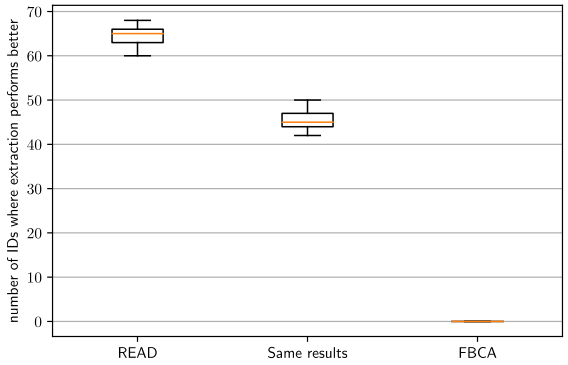


Fig. 7. Number of IDs for which one of the two algorithms (READ and FBCA) outperforms the other one in the *real* dataset.

Counter/Sensor. This difference prevents a direct comparison among the two algorithms. Moreover, while READ produces labels that imply a different field semantic, the labels produced by FBCA are only representative of the signal variability over time, and do not attempt to describe their meaning. As a result, only the labels produced by READ are comparable with respect to the ground truth represented by the formal specifications of CAN messages. Moreover, FBCA did not manage to correctly extract any signal from the synthetic dataset. Hence, we only evaluate labeling correctness for signals correctly extracted by READ. For the ID 158 (Table I) READ correctly labels the first four signals as *Physical*, the sixth as *Counter* and the seventh as *CRC*. For the ID 1*D*0 (Table II) READ correctly labels the first four signals as *Physical* and the fifth signal as *CRC*. For the ID 201 (Table III) READ correctly labels the first two signals as *Physical*, the fourth as *Counter* and the fifth as *CRC*. Since this evaluation is based on incomplete specifications we cannot verify the labeling of the fifth signal of ID 158 and of the third signal of ID 201.

Concerning the *real* dataset, signal extraction performance are shown in Figure 6. The two box plots represent the number of signals that were correctly identified by READ and FBCA across 25 experiments carried out over the available traffic traces (see Table IV).

From Figure 6 we observe that the overall number of correctly identified signals varies among different traces for both READ and FBCA. The main reasons behind this variability are the different number of CAN messages included within each trace and the different values that the same messages assumed in different traces. READ manages to extract a number of signals that ranges from 159 to 206 with a median of 188, whereas FBCA correctly identifies a number of signals ranging from 51 to 72, with a median of 55.

By comparing results of the two algorithms on the same traffic trace we observe that in all the 25 experiments READ correctly identifies more than thrice the number of signals detected by FBCA. Comparing the worst-case experiment for READ with the best case of FBCA, READ is always able to recognize more than twice the number of signals.

Besides counting the aggregate number of extracted messages, we evaluate how the two algorithms perform for different message IDs. This analysis is motivated by the fact that payloads of messages associated to different IDs may have a completely different structure in terms of number, position and types of embedded signals.

Results of this analysis are summarized in Figure 7. The first box plot represents the number of IDs for which READ extracts more correct signals than FBCA, the second box plot represents the number of IDs for which READ and FBCA extracts the same number of correct signals, while the third box plot represents the number of IDs for which FBCA extracts more correct signals than READ.

From Figure 7 we observe that READ is always able to match the results of FBCA, and to outperform it for the majority of IDs. Depending on the traffic traces, the two

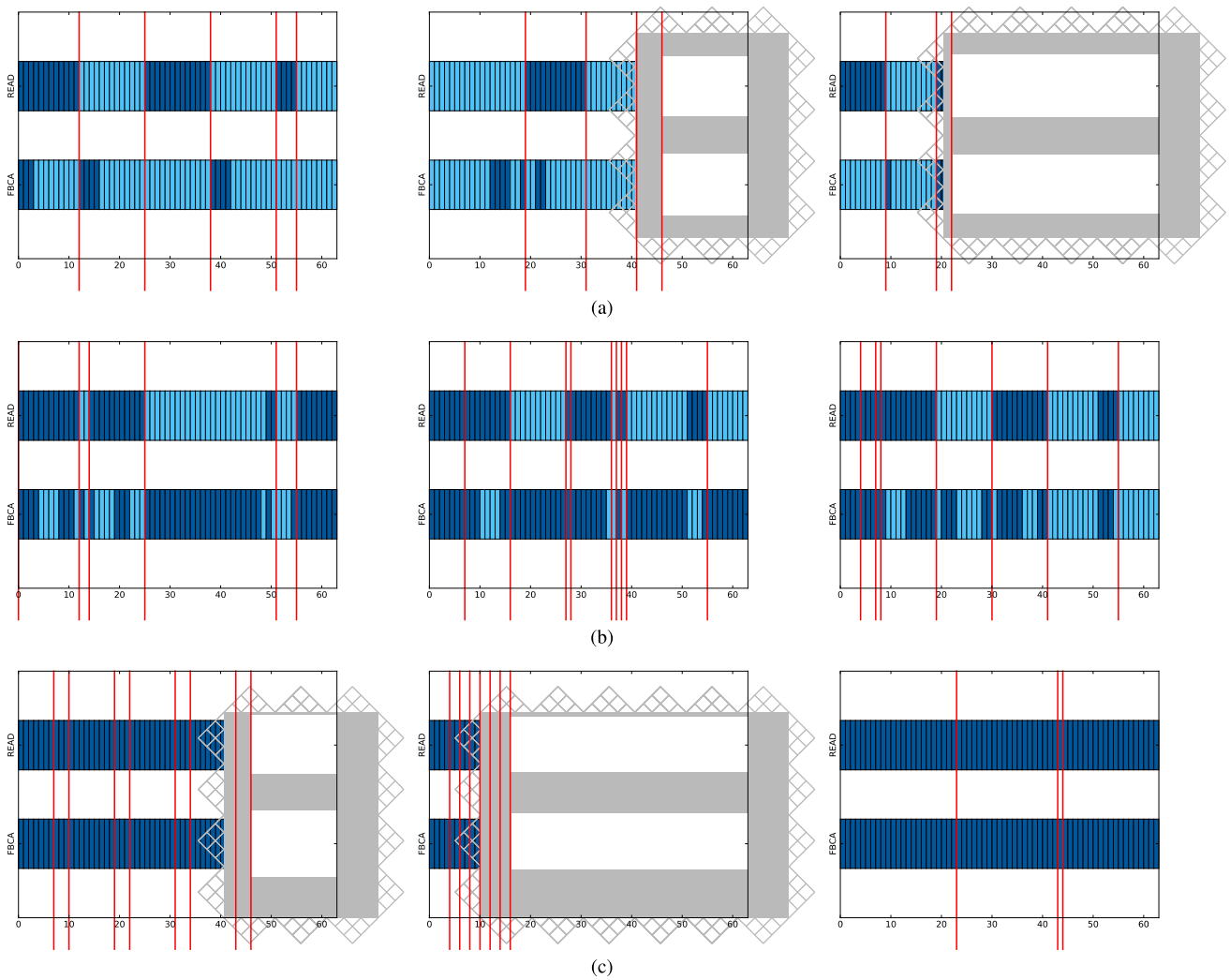


Fig. 8. Representative examples of signal boundaries extracted by READ and FBCA, compared with the ground truth for the *real* dataset. (a) Message IDs for which READ is able to correctly extract all the signals. (b) Message IDs for which READ is not able to correctly extract all signals. (c) Message IDs for which READ is not able to correctly extract any correct signal.

algorithms correctly extract the same number of signals for a number of IDs that ranges from 42 to 50. On the other hand, the number of IDs for which READ extracts more correct signals than FBCA ranges from 60 to 68. We highlight that FBCA never achieves better performance than READ.

To give a better understanding of the performance achieved by READ and FBCA we provide a graphical representation of the outputs of both algorithms in Figure 8.

In all the nine charts of Figure 8, the x-axis represents the single bits of the payload, ranging from 1 to n (with n being the size of the payload for that particular ID), while the y-axis represents outputs of READ (in the top row) and FBCA (in the bottom row). Vertical rows represent the *correct* boundaries of the signals as mandated by the formal specifications. To highlight the signal boundaries extracted by the two algorithms, consecutive signals are colored with different shades. Signal extraction is correct if vertical rows overlap color changes that identify the boundaries of consecutive signals as computed by READ and FBCA. Since for some payloads the number of bits n is lower than 64 bits, a white tail is used to represents the trailing unused bits.

Figures are grouped in three different rows based on the effectiveness of READ in extracting correct signal boundaries.

Row 8a includes three representative examples of message IDs for which READ correctly extracts all signals. For these three different message IDs the signals extracted by FBCA have wrong boundaries. We observe that many errors in FBCA's output are caused by the incorrect identification of spurious signals that do not exist in the formal specifications.

Row 8b shows three representative examples of message IDs for which both READ and FBCA are not entirely accurate. In particular, in all the three message IDs it is clear that both algorithms exhibit suboptimal performance in the extraction of small signals. We remark that similar signals usually convey information that is unrelated to physical phenomena, and are mainly composed by bitmasks used for sensing and controlling the state of specific functionalities of the vehicle, such as fan speed and air conditioning.

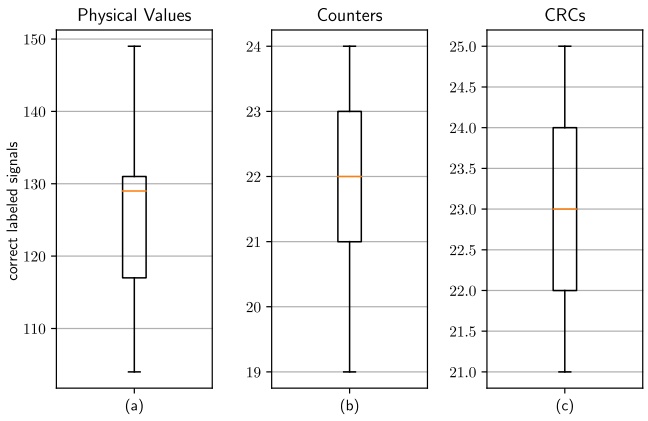


Fig. 9. Signals that are correctly extracted and correctly labeled by READ.

Finally, row 8c shows three representative examples of real message IDs for which both READ and FBCA are not able to extract a single correct signal. This happens for the message IDs in which payloads never change over the collected traffic traces (bit-flip rate is always equal to 0 for all bits). This final result shows that the quality of the collected data impacts the quality of results generated by both algorithms. However it should be noted that these messages do not convey safety-critical information related to the vehicle dynamics, and are mostly related to optional features.

We remark that Figure 8 does not contain any example in which FBCA extracts all signals from an ID. This is due to the fact that FBCA has never been able to achieve full accuracy for any of the IDs included in the tested traffic traces.

Besides signal extraction, we also evaluated the correctness of signal labeling. We remark that this evaluation has only been performed on signals for which READ managed to correctly identify the boundaries, and we recall that a similar evaluation cannot be performed for FBCA since it adopts labels that only describe the signal evolution over time and do not have any semantic meaning. Results of this analysis are summarized in Figure 9. The three box plots of this figure refer to signals that where correctly labeled as Physical values (Figure 9a), Counters (Figure 9b) and CRCs (Figure 9c). In all three box plots the y-axis represents the number of correctly labeled signals. Please note that for better readability the three y-axis use a different scale.

Figure 9 shows that the number of Counter and CRC signals that are correctly labeled by READ exhibit a small variability among all the different traces. This implies that the heuristics used to recognize these fields lead to stable results.

On the other hand, results shown in Figure 9 highlights that the correct labeling of signals representing Physical values exhibit a higher variability among the analyzed traffic traces, ranging from 104 in the worst case to 149 in the best case, with a median of 129. This variability is motivated by the different driving styles, road conditions and lengths associated to each traffic trace.

Signal labeling performance have been further analyzed and the results have been summarized in Table V, which compares the number of signals that are correctly extracted and labeled by READ with respect to the vehicle specifications in the best and worst case scenarios.

TABLE V
CORRECT LABELING OF SIGNALS EXTRACTED FROM THE *Real* DATASET

	Label	Extracted	Correctly labeled	Percentage
Worst Case	Physical	115	104	90.43%
	Counter	28	19	85.71%
	CRC	25	21	96.00%
Best Case	Physical	151	149	98.01%
	Counter	28	24	85.71%
	CRC	25	25	96.00%

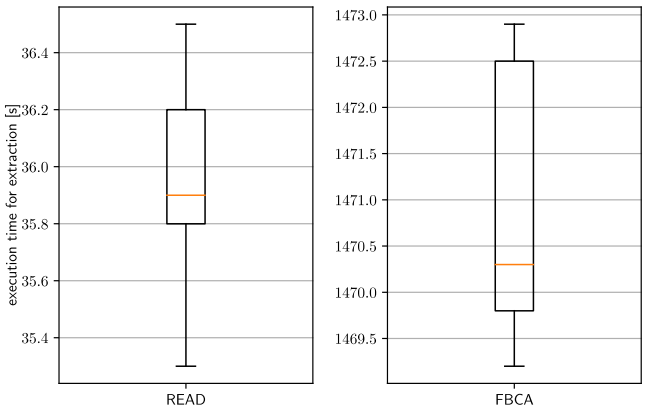


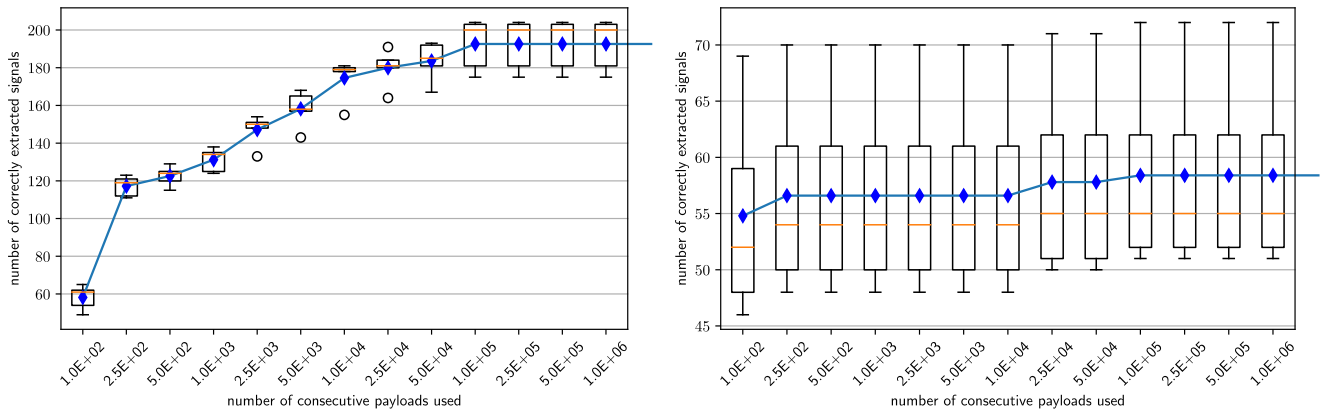
Fig. 10. Execution time of READ and FBCA over all CAN traffic traces.

Table V shows that, even in the worst scenario, READ correctly labels more than the 90% of the extracted Physical signals, while in the best case it is able to correctly label more than 98% of them. Results for both Counter and CRC labels are exactly the same in both cases (85 .71% and 96.00% correct labeling percentage, respectively), meaning that the heuristics used for labeling those particular signals are strong and consistent.

With those final considerations in mind, we can state that READ is better with respect to the state of the art [8] at identifying signals within the payload of CAN data frames by determining their exact boundaries. Moreover, for all messages extracted correctly, READ performs labeling with a very high accuracy. These results hold for both the *synthetic* and the *real* datasets.

C. Execution Time

We compared the time required to execute both algorithms on the traffic traces of the *real* dataset (see Table V-A). These evaluations were carried out on a server equipped with an Intel® Core™ i7-7700HQ CPU @3.8 GHz and with 16 GB of RAM running Fedora 24 x64. For this experiment, the *running time* is evaluated as the time an algorithm needs to generate its final output starting from a raw CAN bus traffic log. Both algorithms have been implemented in the Python programming language and leverage the same boilerplate code for low-level operations (such as reading files from memory and parsing the fields of a CAN message). Execution times of both READ and FBCA are shown in Figure 10, where the two box plots represent the time (expressed in seconds) needed for the complete signal extraction and labeling over the 25 traffic



(a) (b)

Fig. 11. Convergence of READ and FBCA. (a) READ convergence. (b) FBCA convergence.

traces by the two different implementations. Please note that for the sake of readability the two box plots refer to a different scale on the y-axis.

The execution times of READ are two orders of magnitude lower with respect to the execution times of FBCA. The minimum time required for a complete extraction of FBCA is 1469.2 seconds (24 minutes) for a traffic trace of 5 .5 million messages, equivalent to approximately 38 minutes of driving. On the other hand, the maximum execution time of READ on the same traffic traces equals to 36 .5 seconds.

D. Convergence Requirement

The final metric that we evaluate in this paper is the number of consecutive messages that the two algorithms need to analyze to produce their best results. Since both algorithms learn the message boundaries by observing the evolution of payloads over time, it is important to train them with a sufficient number of messages in order to achieve good and stable classification results. To perform an unbiased and realistic evaluation these experiments refer to the *real* dataset. This number is evaluated by extracting the first N messages for each ID from the available CAN traffic traces, applying the extraction algorithms to them and evaluating the number of correctly extracted signals. The convergence results for READ and FBCA are shown in Figure 11. Figure 11a and Figure 11b represent convergence results of READ and FBCA, respectively. In both figures the x-axis represents the number N of consecutive payloads analyzed by the reverse engineering algorithm and the y-axis represents the number of correctly extracted signals through the gathered traces. Please note that, for the sake of readability, the y-axis of Figures 11a and 11b refer to different scales.

Results show that both READ and FBCA converge to their best results with 100 , 000 or more consecutive payloads. We remark that authors of FBCA claim that their algorithm only requires about 100 messages for each ID in order to produce stable results. We experimentally verified that FBCA is capable to achieve near-optimal results after the analysis of 100 messages, however to achieve its peak performance over real CAN traffic it requires 100 , 000 or more messages for each ID. Moreover, even with as few as 100 messages in the training set, READ is able to correctly extract a median of 61 correct messages, with respect to 52 of FBCA.

To summarize, the experimental evaluation shows that READ achieve better performance than FBCA, with lower execution times and comparable convergence requirements.

VI. Real-Life Examples of Automotive Forensics Enabled by READ

To show how the application of READ can help analysts in quickly identifying important signals that are useful to determine the state of a vehicle, without the need for proprietary specifications and time-consuming manual reverse engineering efforts, we applied READ to one of the traces we collected.

A subset of representative signals that were correctly extracted and labeled by READ is shown in Figure 12. In particular, Figures 12a, 12b, 12c and 12d include time series that represent the evolution over time of groups of signals extracted from the payload of a single ID. We refer to them as ID#1 (Figure 12a), ID#2 (Figure 12b), ID#3 (Figure 12c) and ID#4 (Figure 12d). In all time series, the x-axis represents time and is expressed in seconds. The y-axis represent the raw value of the signal as extracted from the message payload by READ.

By observing these time series, a domain expert can easily draw educated guesses about their semantic. Consider as an example Figure 12a. These four time series are included within a single ID, and represent four physical phenomena with a very similar evolution over time. This behavior can be associated to an ECU that emits periodic messages including the rotational speed of the four wheels of the vehicle. Small differences in these time series can be caused by sharp turns, sudden braking, wheel skidding and can motivate the activation of electronic stability systems.

Also Figure 12b shows two correlated signals. A domain expert can associate these signals to the acceleration of the vehicle (top) and to the pressure exerted by the driver to the gas pedal (bottom).

Figure 12c shows two strongly correlated signals that resemble step functions and only assume values between 0 and 15.

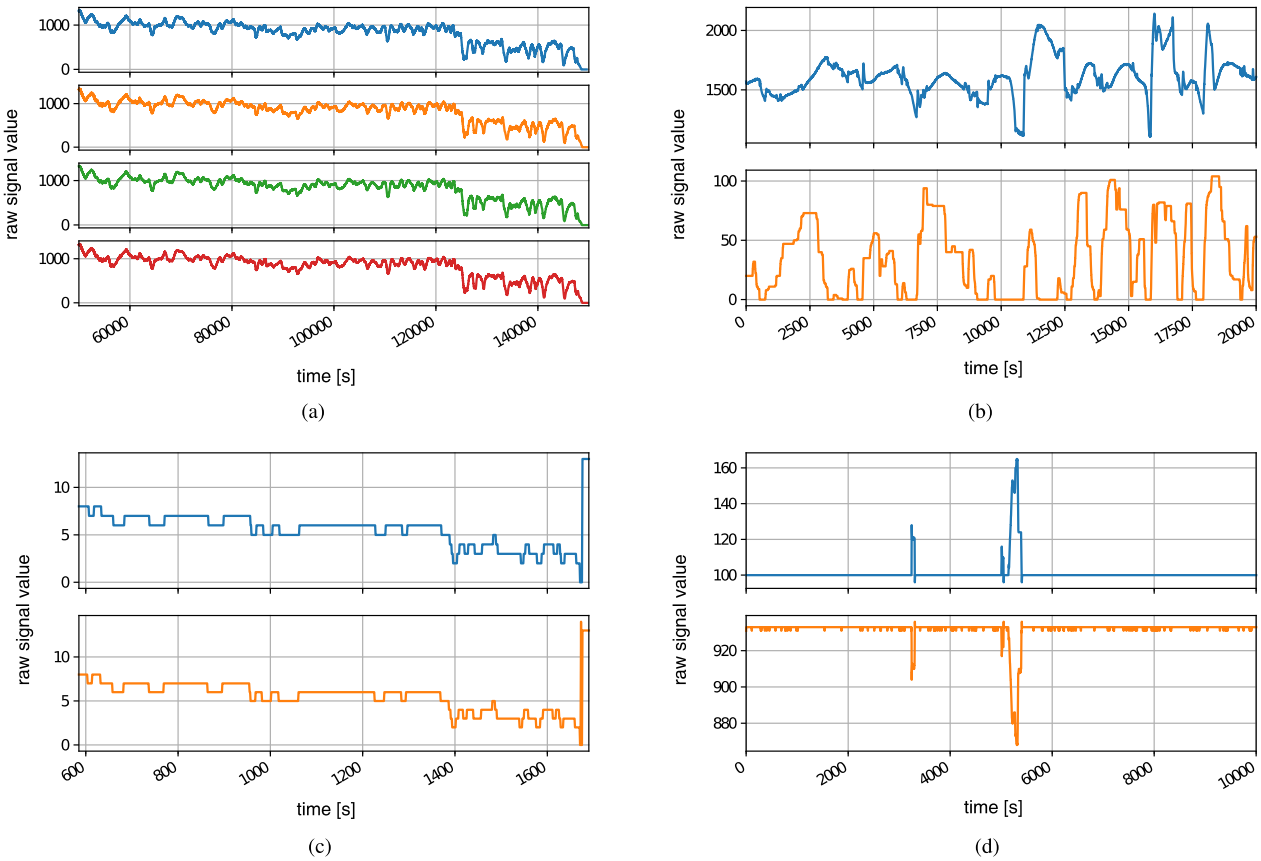


Fig. 12. Time evolution of Physical signals extracted by READ from four different message IDs. (a) Physical signals extracted from ID#1. (b) Physical signals extracted from ID#2. (c) Physical signals extracted from ID#3. (d) Physical signals extracted from ID#4.

Since the vehicle that generated the messages is equipped with an automatic transmission having 13 different gears, it is possible to associate this signals to the gear that is currently engaged.

As a last example, consider the two signals represented in Figure 12d. It can be easily noticed that the two signals are strongly and inversely correlated. By comparing the evolution of these signal to the events occurred while driving it is possible to associate these signal with the position of the brake pedal. For safety reasons, in the vehicle under test this information is encoded and transmitted in two redundant signals. In the top signal the lowest value means no pressure at all, and higher values mean that the pedal is pressed in a lower position. In the second signal the highest value means no pressure at all, and lower values mean that the pedal is in a lower position.

Since READ facilitates reverse engineering of many safety-relevant signals, it also enables the reconstruction of a comprehensive vehicle state starting from a raw CAN traffic trace. As an example, consider Figure 13, in which the time series representing the evolution of different signals are aligned with respect to the x-axis (time, expressed in seconds).

From top to bottom we have the rotational speed of one wheel, the vehicle acceleration, the position of the throttle pedal, the position of the brake pedal and the engaged gear. We only show a time window of 200 seconds for ease of readability. This example shows that by analyzing signals

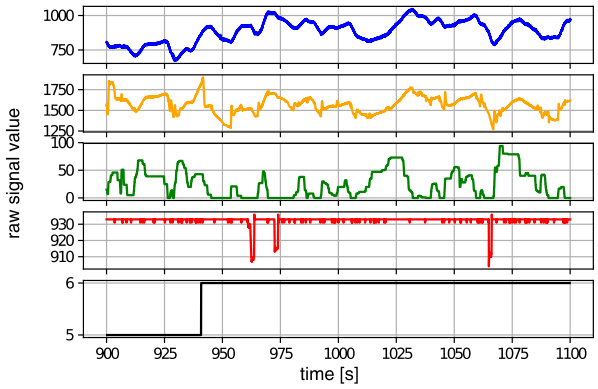


Fig. 13. Reconstruction of driver activities and vehicle behavior.

extracted by READ it is possible to precisely reconstruct the vehicle dynamics and to correlate it with relevant activities performed by the driver. Similar information are useful in the reconstruction of the events that led to a car crash, as well as for investigating possible vehicle malfunctions or anomalies caused by attackers injecting forged CAN messages.

Other benefits of READ include the ability to quickly recognize counters and CRCs encoded in safety-relevant CAN messages. Consider as an example the IDs 158 and 201 of the *synthetic* dataset (see Tables I and III). In this car model (Acura ILX 2016) counters are 2 bits long and CRCs are

4 bits long. On the other hand, for the licensed vehicle whose CAN messages are included in the *real* dataset, counters are 4 bits long and CRCs are 8 bits long. Similar differences among car models and producers are common, and thanks to READ they can be automatically discovered.

Without READ, all previous analyses would require access to the proprietary formal specification of CAN messages, or a time consuming, error prone and manual reverse engineering activity based on trial and error [1], [2], [24]. On the other hand, by applying READ it is possible to show a domain expert many time series that represent the evolution over time of different metrics. The domain expert is still required to pinpoint the exact nature of each signal, but the output generated by READ can greatly reduce the overall reverse engineering effort. We also envision the possibility of introducing novel heuristics that are able to automatically recognize the specific value of a signal and propose more precise labels to the domain expert. Some of these heuristics may be simple incremental evolutions over READ. Consider as an example the message including the speed of the four wheels. A similar message is present in all the vehicles we analyzed, hence an additional heuristic that recognizes a message including four strongly correlated physical values of the same size as the wheel speed message can be easily integrated in READ. If external information sources (such as a GPS sensor) are also available, they can be correlated to the output of READ to automatically identify safety relevant physical values that describe the vehicle dynamics (such as speed and acceleration), as well as driver activities (such as actioning the steering wheel and the brake and throttle pedals). Further extensions of READ include the possibility of automatically visualizing the values of all extracted signals in the form of time series, and highlight strong positive and negative correlations, as well as point and contextual anomalies. All these exploitations of READ output are out of scope for this paper and left as future works.

As a final note, we remark that the performance evaluation of READ presented in this paper assumes that the CAN traffic traces used for training only contain licit messages. If training CAN traffic contains illicit CAN messages injected by an attacker, the performance of READ may decrease, depending on the nature of the attack. We recall that READ requires the analysis of about 100.000 CAN messages for each CAN ID to achieve its peak performance (see Section V-D). If these traces contain targeted attacks consisting in the injections of few illicit CAN messages, then their effect on the bit-flip probabilities computed over all messages of the training traces will be negligible, hence READ performance will not be affected by similar attacks. On the other hand, if the attacker manages to inject many thousands of illicit messages, they may cause significant alterations of the bit-flip probabilities, with possible detrimental effects on the accuracy of READ. However, we highlight that massive attacks involving the injection of thousands of messages in a short time frame can be easily detected through several different intrusion detection approaches [4]–[6], [18]. Moreover, to avoid the effect of possible attacks, an analyst can apply READ to a known-good vehicle of the same maker and model to extract accurate signal boundaries. These boundaries can be safely used for the extraction of signal values even from traffic traces containing massive attacks.

VII. Conclusions

This paper presents READ, a novel algorithm designed to extract signals boundaries from generic CAN bus traffic traces without any prior knowledge of their syntax and semantic. The proposed algorithm is based on the evaluation of the bit-flip rate of each bit composing the payloads associated to consecutive CAN messages. Moreover, it proposes novel heuristics to identify signal boundaries and labels extracted signals according to three different classes that reflect different types of signals.

Extensive experimental evaluations against the ground truth represented by the full specifications of a modern, unmodified, licensed vehicle show that READ extracts more than twice correct signals with respect to previous proposals [8]. Moreover, it is characterized by lower execution times and comparable convergence requirements.

The results of READ can reduce considerably the human effort required to reverse engineer the syntax and semantic of the payload of CAN messages, thus helping academic researchers that do not have access to proprietary vehicle specifications in the development of more effective solutions for detecting and reacting to cyber attacks that involve the injection or manipulation of CAN messages. Moreover READ helps in quickly identifying signals that represent relevant features of the dynamics of a vehicle, and easily reconstruct a precise view of the evolution over time of the state of the vehicle and of the main driver activities.

Acknowledgments

This work has been carried out by the authors in conjunction with an undisclosed industrial partner that allowed us to compare the results of READ with formal vehicle specifications that are not available to the public. The authors would like to thank the industrial partner for helping in the realization of the experimental evaluation.

References

[1] C. Miller and C. Valasek. (2015). *Remote Exploitation of an Unaltered Passenger Vehicle. White Paper of Black Hat US Conference*. [Online]. Available: http://illmatics.com/Remote%20Car%20Hacking.pdf

[2] C. Miller and C. Valasek. (2014). *Adventures in Automotive Networks and Control Units*. [Online]. Available: https://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf

[3] Keen Security Lab of Tencent. (2016). *Car Hacking Research: Remote Attack Tesla Motors*. [Online]. Available: http://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/

[4] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 1577–1583.

[5] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *Proc. IEEE 2nd Int. Forum Res. Technol. Soc. Ind. Leveraging Better Tomorrow (RTSI)*, Sep. 2016, pp. 1–6.

[6] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *Proc. World Congr. Ind. Control Syst. Secur. (WCICSS)*, Dec. 2015, pp. 45–49.

[7] Robert Bosch GmbH. (1991). *CAN Specification Version 2.0*. [Online]. Available: http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf

[8] M. Markovitz and A. Wool, "Field classification, modeling and anomaly detection in unknown CAN bus networks," *Veh. Commun.*, vol. 9, pp. 43–52, Jul. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214209616300869

[9] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2016, pp. 1044–1055, doi: 10.1145/2976749.2978302.

[10] A.-I. Radu and F. D. Garcia, "LeiA: A lightweight authentication protocol for CAN," in *Computer Security—ESORICS*, I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds. Cham, Switzerland:: Springer, 2016, pp. 283–300.

[11] B. Groza, S. Murvay, A. van Herrewege, and I. Verbauwhede, "LiBrA-CAN: A lightweight broadcast authentication protocol for controller area networks," in *Cryptology and Network Security*, J. Pieprzyk, A. R. Sadeghi, and M. Manulis, Eds. Berlin, Germany: Springer, 2012, pp. 185–200.

[12] D. Stabili, L. Ferretti, and M. Marchetti, "Analyses of secure automotive communication protocols and their impact on vehicles life-cycle," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, Jun. 2018, pp. 452–457.

[13] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009, doi: 10.1145/1541880.1541882.

[14] C. Ling and D. Feng, "An algorithm for detection of malicious messages on CAN buses," in *Proc. Nat. Conf. Inf. Technol. Comput. Sci.* Paris, France: Atlantis Press, 2012.

[15] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell, "Modeling inter-signal arrival times for accurate detection of CAN bus signal injection attacks: A data-driven approach to in-vehicle intrusion detection," in *Proc. 12th Annu. Conf. Cyber Inf. Secur. Res. (CISRC)*. New York, NY, USA: ACM, 2017, pp. 11-1–11-4, doi: 10.1145/3064814.3064816.

[16] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *Proc. 25th USENIX Conf. Secur. Symp. (SEC)*. Berkeley, CA, USA: USENIX Association, 2016, pp. 911–927. [Online]. Available: http://dl.acm.org/citation.cfm?id=3241094.3241165

[17] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2011, pp. 1110–1115.

[18] D. Stabili, M. Marchetti, and M. Colajanni, "Detecting attacks to internal vehicle networks through Hamming distance," in *Proc. AEIT Int. Annu. Conf.*, Sep. 2017, pp. 1–6.

[19] D. K. Nilsson and U. E. Larson, "Conducting forensic investigations of cyber attacks on automobile in-vehicle networks," in *Proc. 1st Int. Conf. Forensic Appl. Techn. Telecommun., Inf., Multimedia Workshop*, 2008, pp. 8-1–8-6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1363217.1363228

[20] H. Mansor, K. Markantonakis, R. N. Akram, K. Mayes, and I. Gurulian, "Log your car: The non-invasive vehicle forensics," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 974–982.

[21] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. 6th Int. Conf. Passive Act. Netw. Meas. (PAM)*. Berlin, Germany: Springer-Verlag, 2005, pp. 41–54, doi: 10.1007/978-3-540-31966-5_4.

[22] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Proc. IEEE Conf. Local Comput. Netw. 30th Anniversary (LCN)*, Nov. 2005, pp. 250–257.

[23] A. Callado *et al.*, "A Survey on Internet traffic identification," *IEEE Commun. Surveys Tuts.*, vol. 11, no. 3, pp. 37–52, 3rd Quart., 2009.

[24] C. Smith. (2016). *The Car Hacker's Handbook*. [Online]. Available: http://opengarages.org/handbook/ebook/

[25] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," in *Proc. Comput. Secur. Found. Workshop VII*, Jun. 1994, pp. 187–191.

[26] R. M. Gray, *Entropy and Information Theory*. New York, NY, USA: Springer, 2011.

[27] Comma.AI. (2018). *Comma Cabana*. [Online]. Available: https://community.comma.ai/cabana/?demo=1

[28] Comma.AI. (2018). *Comma.AI—Ghostriding for the Masses*. [Online]. Available: https://comma.ai/