Vehicular Communications 9 (2017) 43–52

Contents lists available at ScienceDirect

Vehicular Communications

www.elsevier.com/locate/vehcom

Field classification, modeling and anomaly detection in unknown CAN bus networks ☆

Moti Markovitz, Avishai Wool *

School of Electrical Engineering, Tel Aviv University, Israel

ARTICLE INFO

Article history:
Received 10 July 2016
Received in revised form 1 January 2017
Accepted 23 February 2017
Available online 2 March 2017

Keywords:
CAN bus
Anomaly detection
Network layer issues
Security and privacy
Communication architecture

ABSTRACT

This paper describes a novel domain-aware anomaly detection system for in-car CAN bus traffic. Through inspection of real CAN bus communication, we discovered the presence of semantically-meaningful *Constant* fields, *Multi-Value* fields and *Counter* or *Sensor* fields. For CAN networks in which the specifications of the electronic control units (ECUs) are unknown, and hence, the borders between the bit-fields are unknown, we developed a greedy algorithm to split the messages into fields and classify the fields into the types we observed. Next, we designed a semantically-aware anomaly detection system for CAN bus traffic. In its learning phase, our system uses the classifier to characterize the fields and build a model for the messages, based on their field types. The model is based on Ternary Content-Addressable Memory (TCAM), that can run efficiently in either software or hardware. During the enforcement phase our system detects deviations from the model. We evaluated our system on simulated CAN bus traffic, and achieved very encouraging results: a median false positive rate of 1% with a median of only 89.5 TCAMs. Finally we evaluated our system on the real CAN bus traffic. With a sufficiently long period of recording, we achieved a median false positive rate of 0% with an average of 252 TCAMs.

1. Introduction

1.1. Motivation

Modern cars have multiple dedicated computers under the hood called “electronic control units” (ECUs). These ECUs control all aspects of the car’s operation: from the engine, breaking and steering controls to the car’s entertainment systems. The ECUs are connected to each other in a network that typically uses the CAN bus protocol. CAN bus is an extremely simple protocol, with absolutely no security components: it was designed under the assumption that all ECUs are legitimate, trustworthy, and operating according to their specifications. However, over the last few years researchers have shown that many ECUs are vulnerable to attack. Since CAN bus in itself is so naive, any attack on one ECU can immediately become an attack on all other ECUs; the subverted ECU can trivially masquerade as any other ECU and cause significant damage. Replacing CAN bus with a more robust technology is probably a good idea. However, due to the huge investment made by manufacturers, and the decades it takes until old cars are scrapped, it is an important goal to improve the security stance of cars within the limitations of CAN bus. An anomaly detection system for CAN bus traffic is a major step in this direction.

1.2. Related work

The research into CAN bus security has grown recently, due primarily to several demonstrations of the insecurity of existing in-car networks. Koscher et al. [1] were first to implement practical attacks on cars. Using CAN bus network sniffing, fuzzing and reverse engineering of ECU’s code, they succeeded to control a wide range of the automotive functions, such as disabling the brakes, stopping the engine, and so on. Later Checkoway et al. [2] showed that a car can be exploited remotely, without prior physical access, via a broad range of attack vectors, such as Bluetooth, cellular radio and even TPMS (Tire Pressure Monitoring System). As a result, Checkoway et al. pointed to the need of using security practices in cars for restricting access and improving code robustness, similar to those in general purpose IT networks. Valasek and Miller [3] demonstrated actual attacks on Ford Escape and Toyota Prius cars via the CAN bus network. They affected the speedometer, navigation system, steering, braking and more. Recently it was reported [4,5] that they remotely disabled Jeep’s brakes during driving, and caused Chrysler to recall 1.4M vehicles.

☆ A preliminary version of this paper has appeared in ESCAR’2015.

* Corresponding author.
E-mail addresses: motimark@gmail.com (M. Markovitz), yash@eng.tau.ac.il (A. Wool).

http://dx.doi.org/10.1016/j.vehcom.2017.02.005
2214-2096/

In response to these attacks, much research has been done in order to enhance the CAN bus network security. Larson and Nilsson [6,7] discussed five layers of defense-in-depth for securing vehicles; prevention, detection, deflection, countermeasures, and recovery, and presented their approach for each layer. Surveys of the research into the security of the in-vehicle networks were done by Kleberger et al. [8] and Studnia et al. [9].

Van Herrewege et al. [10] suggested to add a message authentication protocol to the CAN bus messages. They showed that the standard authentication protocols are not suitable to the CAN bus, and presented CANAuth; a new backward compatible lightweight message authentication protocol for the CAN bus. However, all the nodes that should be able to verify messages need to know a pre-shared key.

Lin et al. [11] suggested a security mechanism to prevent masquerade and replay attacks on CAN bus network. Their proposed mechanism requires to store some elements, including shared secret keys for each pair of nodes (or groups).

Matsumoto et al. [12] proposed a method of preventing unauthorized data transmission in CAN. Each protected ECU monitors all the data on the bus, and broadcasts an error message if it recognizes spoofed messages with it own ID, before the unauthorized transmission is completed. Dagan and Wool [13] suggested a defense method that intentionally causes CAN bus collisions between masquerading attack messages and carefully constructed defense messages, causing the attacker to permanently shut down by going into "bus-off" mode.

Larson et al. [14] used a specification-based approach for attack detection in in-vehicle network. They used a set of security specifications from information extracted from the CAN v2.0 and CANopen v3.01 specifications, and created Protocol-level Security Specifications and ECU-behavior Security Specifications. Due to its properties, the detector can not be placed on the network and it should be placed on each ECU.

Erez and Wool [15] applied field classification in SCADA systems. In our research we used similar methods.

There are several companies attempting to address various aspects of attacks on in-car networks [16–19] — some are still young and provide minimal details about their specific offerings. Among them, Berg et al. of Semcon [20] suggested a secure gateway concept for protecting the CAN bus network from the infotainment domain. The concept is to use three layers: a network layer, a messaging layer and a service layer. The secure gateway is based on standard IP protocols with standard encryptions, and the communication with the CAN bus network is handled using vehicle network adaptors.

Using ternary content addressable memories (TCAMs) has become the *de facto* standard for performing packet classification for high-speed routers on the Internet [21]. An exhaustive overview of the current research status can be found in [22].

1.3. Contributions

This paper describes a novel domain-aware anomaly detection system which detects irregular changes in CAN bus network traffic. A serious challenge is that there are tens of ECUs in a given car, and their CAN bus message formats are proprietary and not publicly documented. Thus finding automatic methods to parse these messages, without any a-priori semantic knowledge, is critical.

Our first contribution is that through inspection of real CAN bus communication, we were able to identify several field types with clear semantics, without any prior knowledge of the message formats. We discovered the presence of *Constant* fields, *Multi-Value* fields and *Counter* or *Sensor* fields.

Next we developed a classifier that automatically identifies the boundaries and types of these fields. In its learning phase, our anomaly detection system uses the classifier to characterize the fields and build a model for the messages, based on their field types. The model is based on Ternary Content-Addressable Memory (TCAM), that can run efficiently in either software or hardware. During the enforcement phase our system detects deviations from the TCAM-based model.

In order to evaluate our methods we needed traces from ECUs whose field structure is known, i.e., we could not use the live traces as a "ground truth". Instead we developed a synthetic ECU traffic simulator, that may be of independent interest. Our simulator emits random messages with realistic message formats whose field semantics are based on the field types we discovered in the live traffic.

We evaluated our system extensively on synthetic CAN bus traffic simulating 10 different message IDs, and achieved very encouraging results: a median false positive rate of 1% with a median of only 89.5 TCAMs.

Finally, we tested our system on the real traffic traces, with the unknown field structure and semantics. On this data set, we saw that a long training period was required (up to 44 seconds) — but only a small sample of messages was truly necessary. With random message selection, we achieved a median of 0% false positive rate with an average of 252 TCAMs.

2. Preliminaries

2.1. CAN bus

A controller area network bus (CAN bus) is a bus standard widely used in the automotive industry. Modern cars have a few dozen electronic control units (ECUs) that communicate over a CAN bus network. The ECUs broadcast the messages to the entire network, and each ECU determines which of the broadcast messages it handles. Every message contains the sent message ID, but there is no destination ID.

There are 4 types of CAN frames: *Data frame*, *Remote frame*, *Error frame* and *Overload frame*. Each frame consists of the fields *Identifier*, *Data*, *CRC*, *ACK* and few others. In this paper we only considered Data frames, and focused only on the *Identifier* and *Data* fields. The identifier field (the message ID) is an 11 or 29 bit value, and the data field is a 64-bit value, whose semantics are per ECU and generally proprietary.

2.2. TCAM

Ternary Content Addressable Memory (TCAM) is a special type of high-speed memory usually used by modern switches and routers for fast look-up tables and packet classification. The TCAM holds a database D of patterns $d_1, \ldots, d_k$ each consisting of *three* symbols: 0, 1 and * ("don't care"). When presented with a message m (consisting only of 0 and 1 bits), the TCAM identifies (in parallel) whether the database includes a matching pattern d_i, i.e., whether for every bit position j, if $d_i[j] \neq *$ then $d_i[j] = m[j]$.

Hardware TCAMs have high power consumption and are relatively expensive, thus they can usually store a limited number of patterns in their database. Because of this limitation, much research has been done to optimize the number of TCAMs (patterns) required to match a class of messages.

We selected TCAMs as a useful formalism to represent the normal traffic emitted by the ECUs: for each message ID we construct a TCAM database. TCAM patterns are very suitable to describe CAN bus messages because CAN messages have *positional* bit fields that are easy to represent as TCAMs. However, note that our approach does not rely on the availability of true TCAM hardware within the anomaly detection unit: Our enforcement module can easily be implemented by firmware, using efficient software data structures and algorithms that can work on standard hardware.

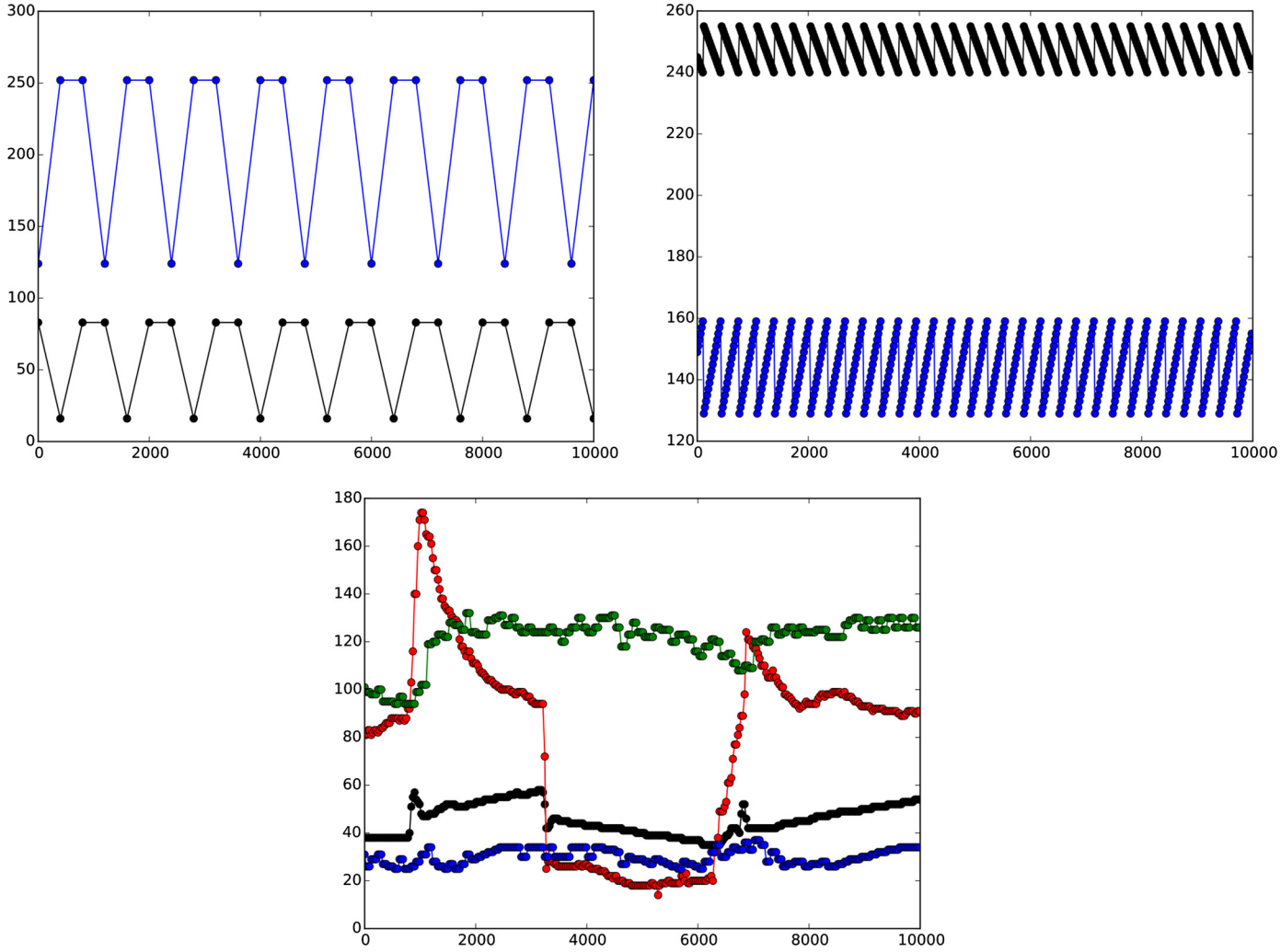

Fig. 1. Values of byte-aligned 8-bit "fields" (in the range 0–255) as a function of the message sequence number in the trace. (Top left) Two multi-value fields; (Top right) Counter fields: The top curve shows a down-counter cycling over 16 values, and the bottom curve shows an up-counter over 32 values. (Bottom) Four sensor fields.

3. Identifying the fields

3.1. Recording the communication

For our research we first recorded a fairly large amount of CAN bus communication from a 2012 Ford Focus. We used Peak-System's PCAN-USB [23] device for recording, connected to the car with DB9-OBD2 cable on one side and connected to a laptop with USB cable on the other side. On the laptop we used PEAK-System's data acquisition application. Although the CAN message identifier may be either 11 or 29 bits long, in our case we saw only 11-bit long identifiers. Also, according to the specification the CAN message data field may be shorter than 8 bytes, yet in our recording all the messages had 8-byte data. We recorded the communication of different scenarios, such as braking, steering the wheel, using the turn indicators, etc. We collected 19 recordings, each with 100,000 messages and approximately 43 seconds long. We identified 57 different message IDs in the recorded traffic. The scenarios we recorded are listed in Table 1.

Table 1
Recorded scenarios.

Recording	Scenario
#1–#3	engine on, car standing
#4–#5	engine on, car standing, pushing gas pedal
#6–#7	operations of gas pedal, brakes, gear and parking brake
#8–#9	operations of lights and blinkers
#10–#13	driving
#14–#16	steering
#17–#18	turning engine on and off
#19	opening/closing vehicle windows

3.2. Field types

Our starting hypothesis was that the CAN 64-bit messages of each message ID have an internal structure: they are a concatenation of *positional* bit fields, with fixed lengths. To test this hypothesis, we had to analyze our CAN bus recordings and learn the types of the messages. Since we didn't have the specification of the message IDs and we didn't know the field boundaries, our method was to inspect every 8 bits separately. During this process, we identified four main classes of fields with clear semantics:

1. *Constant*: we found 274 different 8-bit fields (out of the $57 \cdot 8 = 456$ possible) whose values did not change throughout the entire recording.
2. *Multi-Value*: we found 35 fields that exhibit only a few values throughout the entire recording. Fig. 1 (top left) shows two fields that have only two different values each. Note that these two 8-bits fields are synchronized: they may actually be one longer field, possibly not byte-aligned, that exhibits only three different values.

3. *Counter*: we found 29 fields whose value behaves like a cyclic counter with some minimal and maximal value. Fig. 1 (top right) shows two counter fields: one with a range of 128–160 and the other with a range of 240–256. Note that the former may be a 5-bit field cycling over 32 values and the latter may be a 4-bit field cycling over 16 values.
4. *Sensor*: we found 48 fields whose values seem to represent a measurement of a physical quantity during the car trip. These fields exhibit a value that is continuous and noisy with jitter. It seems that different fields behave differently and they don't represent the same physical measurement. Fig. 1 (bottom) shows four sensor fields.

In addition, 70 fields did not match any of the above types. Note that a very similar classification of fields was observed by Erez and Wool [15] in SCADA traffic.

3.3. Field boundary and classification algorithm

Based on the identification of the field types, we developed a field splitting and classification algorithm. The algorithm receives a trace of 64-bit messages with the same message ID, and analyzes it. Its output is a list of disjoint fields, each characterized by a leftmost-bit and a length, and labeled by one of three types *const*, *multi-value*, *counter/sensor*. Note that our algorithm does not distinguish between counters and sensors and classifies such fields as "*counter/sensor*". We leave a more refined classification to future work. The pseudo code of the Field Boundary and Classification Algorithm is described in Fig. 2. The algorithm has 4 phases as follows:

1. *Split into Fields*: The algorithm starts without prior knowledge on the field division. We assume that every division may be valid, including fields that are not byte-aligned or nibble-aligned. At this stage, the algorithm defines 64×64 lower triangular matrix f that represents all the candidate fields: the column represents the index of the left bit of the fields, and the row represents the field length, both in $[1, 64]$. Obviously the field length depends on the index of the left bit, so that $left\ bit\ index + field\ length - 1 \leq 64$, thus the matrix contains only $(64+1) * 64/2 = 2080$ valid cells. Notice that the matrix also represents overlapping fields.
2. Procedure COUNT_VALUES in Fig. 2: For each field candidate defined in the matrix, the algorithm examines the n messages in the trace and counts the number of unique values that were found in the field range. For instance, for $f_{1,1}$, the field in range $[1, 1]$, the algorithm checks the first bit in all the n messages and sets $f_{1,1}.n\_values = 2$ if both 0 and 1 were found, and 1 if only 0 or only 1 was found. For $f_{64,1}$, the field candidate that represents the whole message, the algorithm sets $f_{64,1}.n\_values$ to be the number of different unique message that were found.
3. Procedure TYPE_AND_SCORE in Fig. 2: The algorithm determines the field type of each field candidate as *Const*, *Multi-Value* or *Counter/Sensor*, and calculates the candidate's score as follows:
   (a) *Const*: If $f_{l,c}.n\_values = 1$ then set $f_{l,c}.type = Const$. The score of a *Const* field candidate is the field length, l.
   (b) *Multi-Value*: Intuitively, a field candidate is classified as *Multi-Value* if it contains only a "small number" of unique values, and is not "too short". For a field with a length of l to be classified as *Multi-Value*, the maximum number of allowed unique values is $min(\sqrt{2^l}, T_{mv})$, where T_{mv} is a configuration parameter. In Section 6.2.1 we explain how T_{mv} was chosen. We used a minimal length of 4 bits. The score of *Multi-Value* field is also the field length.

```
procedure COUNT_VALUES
    for l_bit in [1, 64] do
        for field_len in [1, 64] ∧ l_bit + field_len ≤ 65 do
            f[l_bit, field_len].n_values ←
                #_unique_values(messages[l_bit : field_len])
        end for
    end for
    return f
end procedure

procedure TYPE_AND_SCORE(f)
    for field in f do
        if field.n_vals = 1 then
            field.type ← Const
            field.score ← field.len
        else if field.n_vals < T_Vmax ∧ field.len ≥ T_Lmin then
            field.type ← Multi − Value
            field.score ← field.len
        else
            field.type ← Counter/Sensor
            field.score ← k^2 / 2^l
        end if
    end for
end procedure

procedure CHOOSE_FIELDS
    chosen_fields = ∅
    while f ≠ ∅ do
        best_field ← GET_FIELD_WITH_MAX_SCORE(f)
        chosen_fields.append(best_field)
        f ← REMOVE_OVERLAPPING_FIELDS(f, best_field)
    end while
    return chosen_fields
end procedure

procedure GET_FIELD_WITH_MAX_SCORE(f)
    procedure COMPARE_FIELDS(f1, f2)
        if f1.type = f2.type then return f1.score > f2.score
        else return f1.type > f2.type
        end if
    end procedure
    return max(f, compare_func = COMPARE_FIELDS())
end procedure
```

Fig. 2. Field boundary and classification algorithm.

   (c) *Counter/Sensor*: Field candidates that are not classified as *Const* or *Multi-Value* are classified as *Counter/Sensor*. We chose the score of *Counter/Sensor* fields to be $k^2/2^l$, when k is the number of the unique values for the field, and l is the field length.
   A more intuitive score for *Counter/Sensor* fields is the "density" $k/2^l$ of the unique values in the field: a field that utilizes a large fraction of its possible values is likely to be a counter or a sensor. But during our experimentation we found that the exponential factor 2^l is too aggressive, and using the density as a score lets short fields dominate long fields, which did not match our intuition. Therefore, we enlarged the weight of the number of unique values and we chose the score to be $k^2/2^l$. Note that there are many other potential scores, we leave their evaluation to future research.
4. Procedure CHOOSE_FIELDS in Fig. 2: In the final stage, the field classification algorithm iteratively chooses the field candidates with the best fit. The fit is determined by the field type and the score: *Const* candidates have the highest priority, followed by *Multi-Value* and then *Counter/Sensor*. The priorities within each type are ordered by the candidate's score. In each iteration, the algorithm chooses the field with the best fit, and discards field candidates that overlap the selected field candi-

date. The algorithm iterates this step with the remaining fields in the matrix until no field candidates remain.

At the end of this process, the field classification algorithm returns a set of disjoint fields for the given trace of messages. Note that step 4 (CHOOSE_FIELDS) is a greedy algorithm which does not necessarily guarantee that the optimum set of fields was found. One may design other, possibly slower, algorithms, to find a high-quality field splitting out of the exponential number of choices.

4. Building the TCAM model

At the end of the learning phase, via the field classification described in section 3.3 we have a set of fields that represents the model of a specific message ID. Each field has a type, and specific properties according to the type: the constant value for *Const*, the list of all the observed values for *Multi-Value*, and the minimal and maximal observed values for *Counter/Sensor*.

As mentioned above, our approach uses TCAMs for the enforcement stage. For each message ID we build a set of TCAMs that only match messages that fit the properties of all the message ID's fields. Thus, any message that does not match the TCAMs is flagged as an anomaly. The set of TCAMs is built in two steps:

Step 1: Define the pattern of bits that match each field. The patterns for *Const* and *Multi-Value* are obvious: one or more patterns, each matching one value from the list of valid values. For a *Counter/Sensor* we need to create a set of patterns that match the *range* between the field's minimal and maximal values.

We create the set of patterns that matches a range of values by the following greedy algorithm: Assuming that there are t values in the range we first define a set of t patterns, each matching a unique value in the range (without using *"don't care"* symbols). Then the algorithm merges every two patterns that have the same prefix and only one different bit, using *"don't care"* instead of that bit. The algorithm keeps trying to merge patterns until there remains no pair of patterns that can be merged. The result is a small set of prefix-optimized patterns that matches the required range.

Note that we are using a fairly simple method: many authors have suggested more sophisticated methods to optimize the number of TCAMs required for range matching. For example, Meiners et al. [24] propose a non-prefix compression scheme to reduce the number of required TCAM rules, and other approaches are suggested by [21,22,25]. In the future we may improve our TCAM set using one or more of these methods.

Step 2: Given the set of patterns for each field, we define the set of TCAMs for the whole message ID by the Cartesian product of all the patterns for each field.

5. The ECU simulator

As mentioned in Section 3.1, we recorded a large amount of CAN bus communication for our research. However, since we don't know the actual field structure of the recorded messages, we could not evaluate the quality of our model using this data. In order to deal with this issue, we developed a simulator of CAN bus communication. The simulation allows us to control the CAN bus communication properties and simulate various scenarios that we can not get from the recordings, such as rare messages, long or short field lengths, etc. In the future we will be able to use the simulation to implement attack models.

The ECU simulator generates random CAN bus messages according to pre-configured parameters of the simulated ECU. Each generated message contains several fields, and each field can have a different length and type. The supported field types are: *const*, *multi-value*, *counter* and *sensor*. Note that a simulated counter field over l bits always counts between 0 and 2^l, and not between arbitrary min and max values.

The ECU simulator maintains a message structure consisting of a number of fields, each field with its own length l. For each field type the simulator is configured with additional data: *Const*: A single value in the range $[0, 2^l]$; *Multi-Value*: An ordered list of values in the range $[0, 2^l]$. *Counter*: The start value in the range $[0, 2^l]$. *Sensor*: We model a sensor field as a noisy sine wave whose negative values are truncated, thus we need the following parameters configured: The wave length in the configured range, the DC and amplitude in the range $[0, 2^l]$ and the phase in the range $[0, 2\pi]$.

During its run, the ECU simulator generates the messages for each message ID, according to the selected message structure. For each field in the message, the ECU simulator generates randomized data according to the fields type and the properties of the field:

- *Const*: Return the single value.
- *Multi-Value*: Randomly choose one of the list of values. The randomization assigns higher probabilities to values with lower indexes: We use the absolute value of a normal $N(0, 1)$ distribution, mapping the distribution's range between $[0, 3]$ to the list of the values. This way the values in the set are chosen non-uniformly, with values at the start of the list selected more frequently than those towards the end.
- *Counter*: Return consecutive numbers starting at the start value modulo 2^l.
- *Sensor*: Return a noisy sine with the configured wave length, DC, amplitude (amp) and phase ϕ. The value V_i for the sample i is: $V_i = max(0, DC + amp \cdot sin((2\pi \cdot i)/(wave\ length) + \phi) + U[-\Delta, \Delta])$, when U is a uniform distribution, and $\Delta = 0.2 \cdot amp$.

For our research we generated simulated CAN bus communication for 10 message IDs having randomly chosen fields, with 500 messages per message ID, split into 5 100-message traces. The field structure of message ID #1 is shown in Table 2.

6. Performance evaluation

6.1. Field classification evaluation

For our research we implemented the field classification algorithm in Python. The application ran on a PC with Intel(R) Core(TM) i5-3550 CPU @ 3.30 GHz and 8 GB of RAM running Windows 7 Pro 64-bit. After we developed the field classification algorithm, we wanted to evaluate the quality of the results over the simulated ECU data. For this purpose, we defined two metrics, as follows:

6.1.1. Field classification distance

The first metric measures the distance between the field structure extracted by the algorithm and the actual field structure. For the simulated message IDs of Section 5, we know the actual field structure so we can calculate this distance. The distance between the extracted field structure and the actual field structure is defined as the sum of two scores:

1. The number of bits that were classified into a different field type than the actual field type. The range of this score is $[0, 64]$.
   This score is not enough, since a bit may be classified into the same field type, but in a different field, if the extracted field boundaries are incorrect.
2. The number of extracted field boundaries that are different from the actual field boundaries. The range of this score is

Table 2
Field structure of simulated message ID #1. The message has 11 bit fields, including 2 sensor fields, 2 counter fields, 2 constant fields, and 4 multi-value (mv) fields.

Field type	Length	Properties
Sensor	4	wave length = 44, DC = 10, amplitude = 1, phase = 1.14
Counter	8	start value = 151
Const	3	constant value = 4
Const	12	constant value = 1717
mv	7	set of values = [51, 113, 90, 37, 20, 22, 53, 115, 35, 32, ...]
Sensor	14	wave length = 30, DC = 9000, amplitude = 1047, phase = 3.6
Const	2	constant value = 2
mv	9	set of values = [151, 1, 484, 323, 426, 45, 113, 28, 276, 274, ...]
mv	3	set of values = [1, 7, 2, 1, 6, 0, 1, 0, 2, 2, ...]
Counter	1	start value = 1
mv	1	set of values = [0, 0, 1, 0, 1, 0, 0, 0, 0, 1, ...]

[0, 63]. For example, if the simulated field structure includes only a single field [0–63], and the extracted field structure includes two fields [0–31], [32–63], the field boundary between bits 31 and 32 is different, and the score will be 1.

The field classification distance is the sum of the two scores, normalized by 127 to the range [0, 1]. Note that a lower field classification difference score represents better quality.

6.1.2. Field classification false positive rate

The second metric for the quality of the field classification algorithm is the false positive rate of the anomaly detection (enforcement) stage. In this stage we check the created TCAM model against the simulated CAN bus communication. Since all the simulated messages are valid CAN bus messages, every message that is categorized by the model as abnormal is a false positive error. The score is the number of false positive errors, normalized by the number of all the messages to the range [0, 1].

For calculating the field classification false positive rate, we used 5-fold cross-validation. In this method, we use the 5 equal sized traces of simulated messages, when one trace is used for the learning stage, and the other 4 traces are used as messages for the anomaly detection stage. This process is repeated 5 times, so each trace is used for the learning stage once. The final false positive rate is the average of the 5 rates.

6.2. Calibration

6.2.1. The maximal number of allowed unique values for multi-value field — T_{mv}

As mentioned in Section 3.3, the number of allowed unique values for a field to be classified as a *Multi-Value* field is limited by a configuration parameter T_{mv}. In order to calibrate T_{mv}, we measured the false positive rate for T_{mv} in range [8, 32]. The results are shown in Fig. 3 (top). It can be seen that the false positive rate jumps dramatically when $T_{mv} = 16$ and again when $T_{mv} = 32$, so we chose the working point $T_{mv} = 12$.

6.2.2. Number of messages for learning

As mentioned in section 6.1.2, the learning stage of the field classification algorithm requires a trace of messages. The number of messages in the trace creates a trade-off between the learning time and the false positive rate. A small number of messages yields short learning time, but causes a higher false positive rate and vice versa. In order to calibrate a good working point, we measured the false positive rate for different numbers of messages in the learning phase; 10, 50, 100, 200, 500, for the ten simulated message IDs. Fig. 3 (bottom) shows the results of the runs. It can be seen that the false positive rate of the anomaly detection drops dramatically around 100 messages, and this is the length of the learning trace that was chosen for the remainder of the evaluation on the simulated traces.

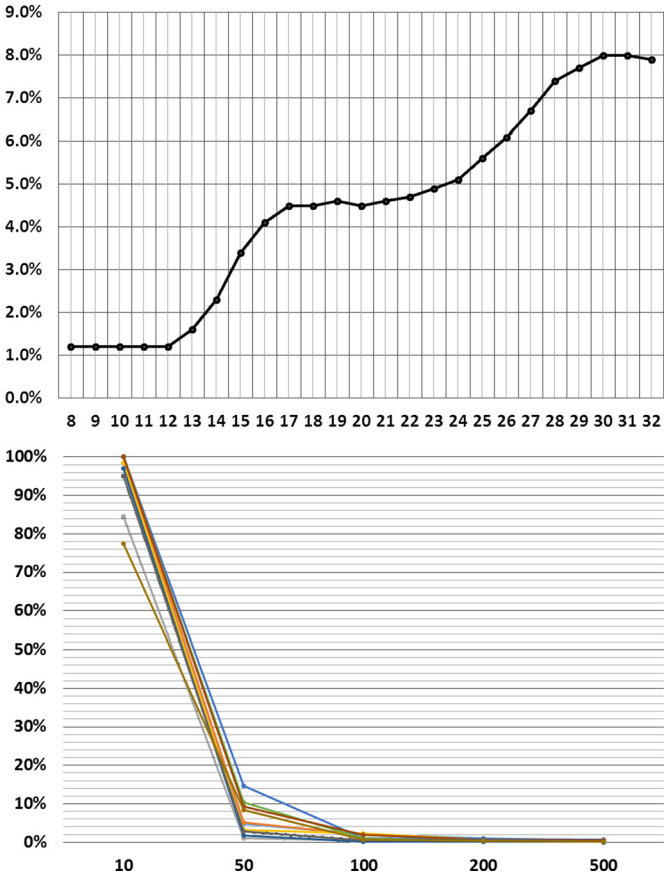

Fig. 3. (Top) False positive rate as a function of T_{mv}; (Bottom) False positive rate by number of messages in traces.

6.3. Results and discussion

We tested our model with the 10 simulated message IDs and the metrics of Section 6.1. Fig. 4 (top) shows the false positive rate with a median value of 1.0% and maximum of 2.2%. Fig. 4 (middle) shows the field classification distance with a median value of 26.1% and maximum of 40.8%.

It is somewhat surprising that despite the seemingly poor field classification distance, the false positive rate is so good. We can explain this phenomenon in two ways. First, the field classification distance penalizes an incorrect field classification by its *length* since it counts misclassified *bits*, hence a single misclassified field can increase the distance by several percentage points. Therefore the field classification distance seems to paint an overly pessimistic picture. Second, some misclassifications yield TCAM models that are tolerant and do not raise false alarms. For example, if a *Multi-*

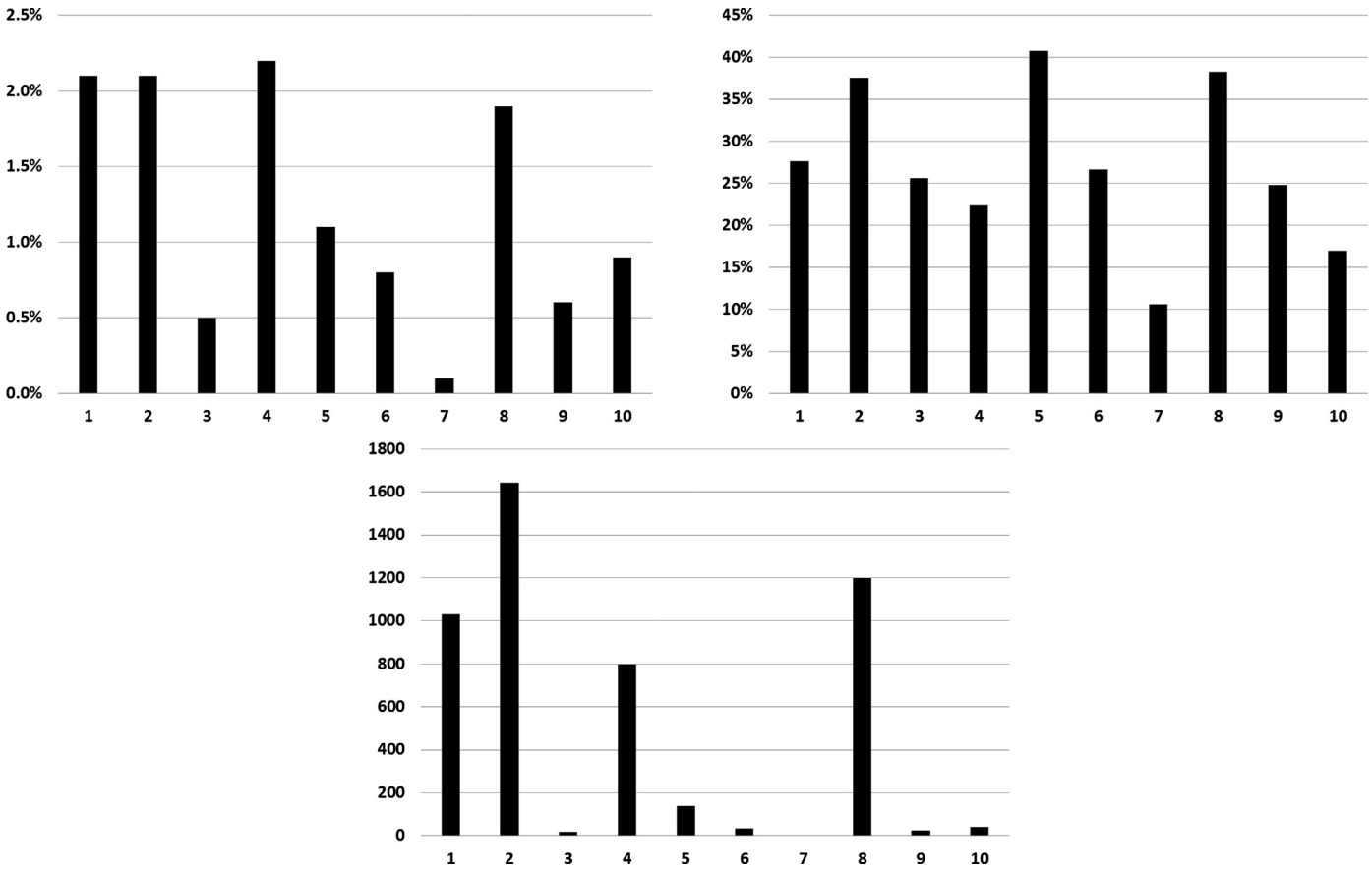

Fig. 4. Performance using traces of the 10 simulated message IDs: (Top left) False positive rate; (Top right) Field classification distance; (Bottom) Model size in TCAMs.

Value field is classified as *Counter/Sensor* field, this will probably not increase the false positive rate: typically the Multi-Value field's allowed values will fall inside the enforced range of the Counter/Sensor.

Another measurement of the quality of our model is the size of the model in TCAMs. Hardware TCAMs are limited resources, and a model with a large number of TCAMs can be impossible to use. Fig. 4 (bottom) shows that our model contains a median of 89.5 and at most 1642 TCAMs (despite the relatively simple TCAM optimization technique we used). These are very practical database sizes for TCAM hardware.

7. Back to the recorded data

After we developed and calibrated our model using the simulated data, we wanted to check our system with the previously recorded data, for which we did not have the fields structure or semantics. In order to evaluate the quality of the system, we used the following methods:

7.1. Visual examination

In order to get a general impression whether the field classification fits, we arbitrarily split each 64-bit message into 8 8-bit "fields". We created a graph for each 8-bit field, and classified the field by visual inspection to one of the types: *Constant*, *Multi-Value*, *Counter* or *Sensor*. Then we classified the same ECU messages using our field classification algorithm, and we checked if the two classifications match. A message ID was considered a match if the classification of each of its 8 fake "fields" agreed with the algorithm's classification of all the fields intersecting the fake field. We visually classified 47 message IDs, and found 40 matching message IDs. The mismatches can be explained mostly by mistakes in the visual examination. For example, in Fig. 5 we visually classified fields 16–23 and 24–31 as *Multi-Value*, field 32–39 as *Sensor*, and the other fields as *Constant*. Our algorithm classified fields 0–15 as *Constant*, 16–27 as *Multi-Value*, 28–31 as *Constant*, 32–35 as *Sensor* and 36–63 as *Constant*: note how the algorithm splits the bits 24–63 into fields that do not align with byte boundaries – we believe that the algorithm is more accurate than our visual inspection here.

7.2. False positive rate

We ran our classification algorithm with the recorded data, and we measured the false positive rate as with the simulated data. Initially we didn't get good results: Although we found many message IDs with a false positive rate of 0%, there were several message IDs with high false positive rate, up to 80%. We tried to omit 20%, 30% and 50% of the first messages from the learning stage, but we got the same results. When we examined this issue, we found that the mismatch occurred due to fields that changed slowly. These fields may represent a slow counter like gas gauge, odometer, etc. With a short learning stage the algorithm does not process all the different valid values of these fields, and wrongly classifies them as *Constant* or *Multi-Value*, instead of *Counter* or *Sensor*. To address this limitation, a long learning stage is required. However, we found that it is not necessary to capture every single message to build a good model: Our traces were for 44 seconds and included up to 4363 messages per message ID. We randomly sampled 100 messages for each message ID (as little as 2.3% of the messages) and used the random sample to train the model. Fig. 6 shows the results for two of the 19 recordings: A false positive rate with a median of 0% and maximum of 3% (left) and a false positive rate

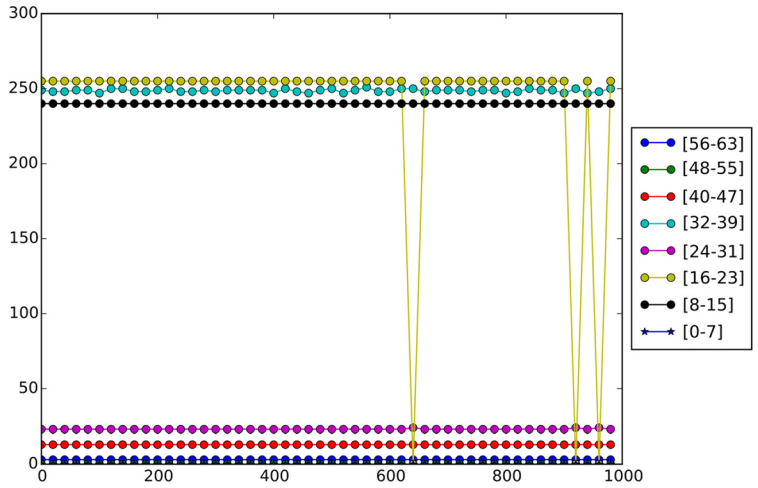

Fig. 5. Values of 8 fake 8-bit fields (in the range 0–255) as a function of the message sequence number in the trace. These were visually classified into two *Multi-Value*, one *Sensor* and five *Constant* fields.

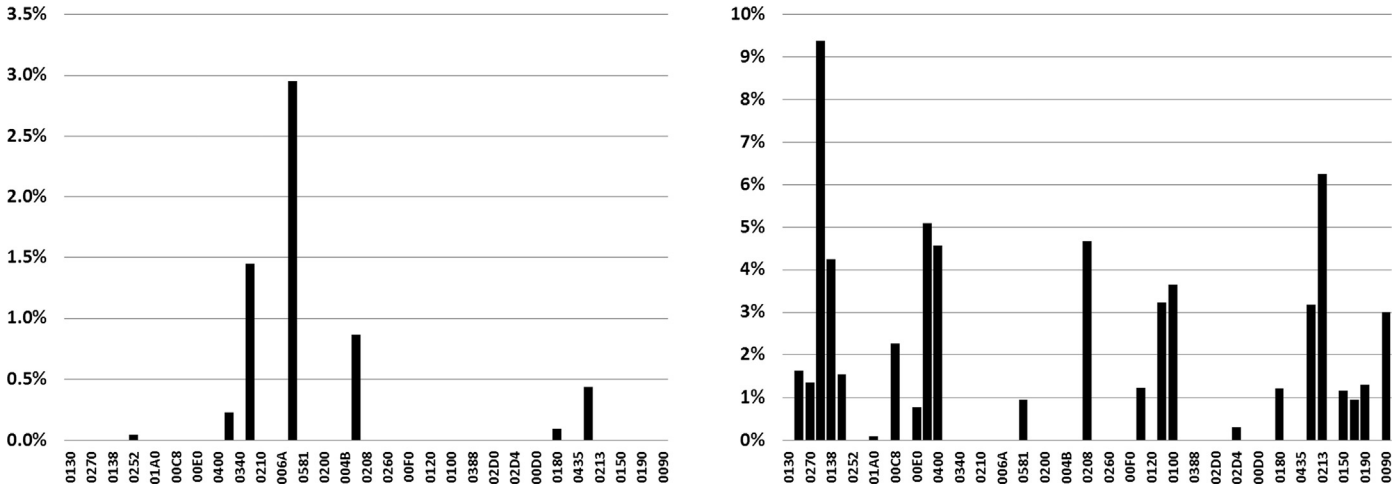

Fig. 6. Performance of the model using traces of recorded message IDs with learning based on 100 random messages: False positive rate for each message ID for recording #1 (left); and for recording #13 (right).

with a median of 0% and maximum of 9.4% (right). Other recordings had similar results. The meaning of the results is that a long period of recording is required, in order that all the different values will appear in the learning stage—but random subsampling is sufficient to produce a good model.

In order to check the effect of the size of the random sample, we also randomly sampled 500 messages for each message ID, and used the samples to train the model. Fig. 7 shows the results for the same two recordings as Fig. 6, but with 500 random samples. The false positive rate was significantly reduced in both recordings: A false positive rate with a median of 0% and maximum of 0.52% in recording #1 (down from a maximum of 3% with 100 samples), and a false positive rate with a median of 0% and maximum of 1.6% for recording #13 (down from a maximum of 9%). Comparing Figs. 6 and 7 also shows that most of the ECUs for which we observed false alarms with 100 samples now exhibit zero false alarms with 500 samples. Thus we see that a 5-fold increase in the sample size—still with only 500 sampled messages—produces an effective improvement to the model's accuracy.

Fig. 8 shows the averages of the false positive rates over all the ECUs for each recorded scenario. The figure shows that the average false positive rate varies across the various car usage scenarios: it is higher for the scenarios of driving (recordings #10–#13), turning engine on and off (recordings #17–#18) and combination of gas, brakes, gear and parking break operations (recordings #6–#7). The average false positive rate is lower for the scenarios when the engine is on but the car is standing (recordings #1–#5), steering and opening or closing the windows (recordings #14–#16 and #19). Operations with lights and turn indicators (recordings #8–#9) have minor effects. We believe that this variation is caused by the diversity of messages in the different scenarios: e.g., we observe more messages types and field values in a driving scenario than in an idle scenario, and this greater diversity requires a larger training set to eliminate false positives.

8. Conclusion and future work

In this paper we presented the design and evaluation of a domain-aware anomaly detection system for CAN bus networks. Although the CAN bus message formats are proprietary and not publicly documented, we were able to identify the semantics of several classes of message fields, by detailed analysis of captured CAN bus traffic. We developed a classifier that automatically identifies the boundaries and types of these fields. Based of the field classification, our anomaly detection system builds a model for the normal messages. The model is formulated as a database of TCAM symbols, that be matched against messages efficiently in either software or hardware. We evaluated our system on the simulated CAN bus traffic, using 10 different message IDs, and achieved very encouraging results: a median false positive rate of 1% with a me-

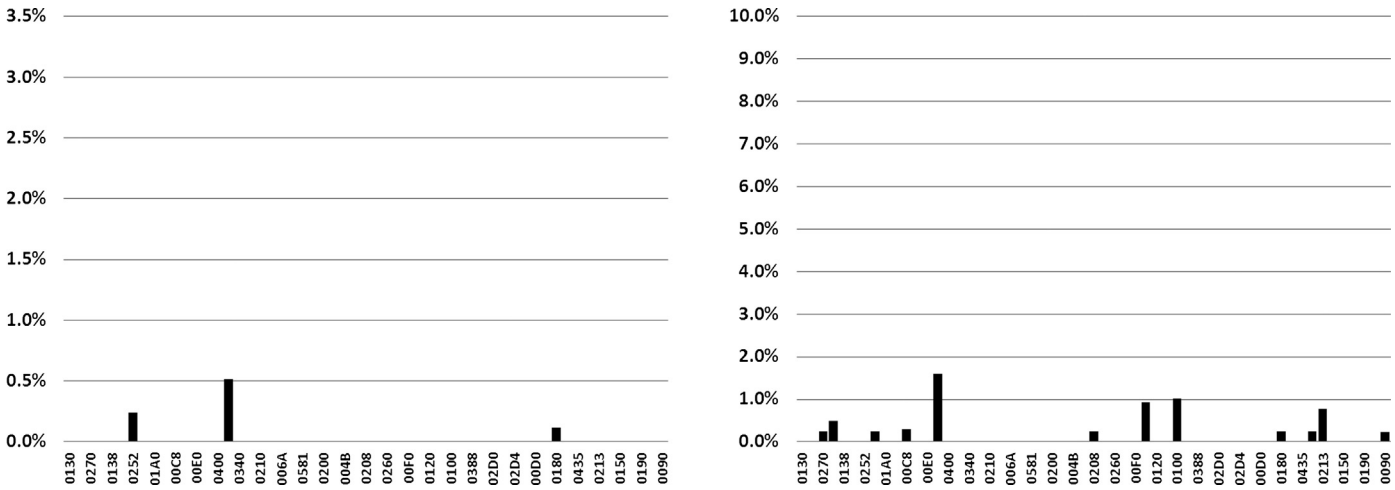

Fig. 7. Performance of the model using traces of recorded message IDs with learning based on 500 random messages: False positive rate for each message ID for recording #1 (left); and for recording #13 (right).

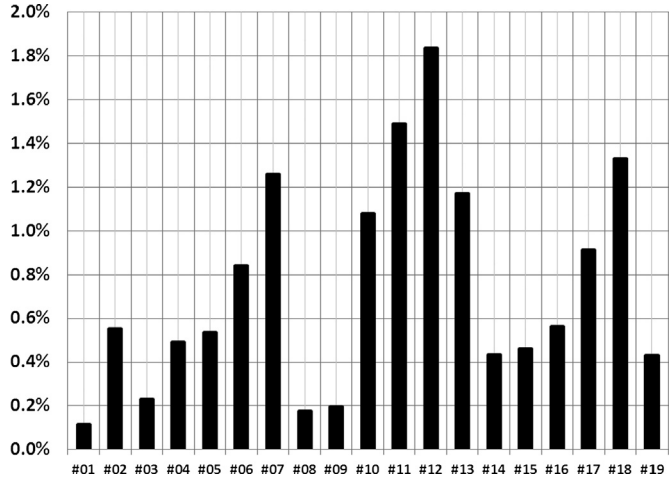

Fig. 8. The averages of the false positive rates over all the ECUs for each recording #1–#19.

dian of only 89.5 TCAMs. Afterwards we evaluated our system on the real CAN bus traffic. With a sufficiently long period of recording, we achieved a median false positive rate of 0% with an average of 252 TCAMs.

We believe there are a number of directions for future work beyond the results reported here. Firstly, we would like to evaluate the sensitivity of our system to detect attacks and evaluate the trade-off between false alarm and true positive rates. Such an evaluation may involve actual attacks or modified simulated traffic that includes anomaly examples representative of attacks. Secondly, beyond evaluating our system against simulated message IDs, we would like to evaluate it against real CAN bus messages—with the relevant ECU real specifications. Finally, in a usable system the false-alarm rate needs to be very close to zero: we believe that with larger training sets our system can produce such results, but this requires further experimentation.

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, Experimental security analysis of a modern automobile, in: IEEE Symposium on Security and Privacy, 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, Comprehensive experimental analyses of automotive attack surfaces, in: Proceedings of the 20th USENIX Conference on Security, SEC'11, USENIX Association, Berkeley, CA, USA, 2011, p. 6.

[3] C. Miller, C. Valasek, Adventures in automotive networks and control units, http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf, 2014 [Online; accessed 22-July-2015].

[4] A. Greenberg, Hackers remotely kill a Jeep on the highway—with me in it, http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, 2015.

[5] A. Greenberg, After Jeep hack, Chrysler recalls 1.4M vehicles for bug fix, http://www.wired.com/2015/07/jeep-hack-chrysler-recalls-1-4m-vehicles-bug-fix/, 2015.

[6] U.E. Larson, D.K. Nilsson, Securing vehicles against cyber attacks, in: Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research, CSIIRW '08, ACM, New York, NY, USA, 2008, pp. 30:1–30:3.

[7] D.K. Nilsson, U.E. Larson, A defense-in-depth approach to securing the wireless vehicle infrastructure, J. Netw. 4 (7) (2009) 552–564.

[8] P. Kleberger, T. Olovsson, E. Jonsson, Security aspects of the in-vehicle network in the connected car, in: IEEE Intelligent Vehicles Symposium (IV), 2011, pp. 528–533.

[9] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, Y. Laarouchi, Survey on security threats and protection mechanisms in embedded automotive networks, in: 2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop, DSN-W, IEEE, 2013, pp. 1–12.

[10] A. Van Herrewege, D. Singelee, I. Verbauwhede, CANAuth—a simple, backward compatible broadcast authentication protocol for CAN bus, in: ECRYPT Workshop on Lightweight Cryptography, 2011.

[11] C.-W. Lin, A. Sangiovanni-Vincentelli, Cyber-security for the controller area network (CAN) communication protocol, in: International Conference on Cyber Security, 2012, pp. 1–7.

[12] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, K. Oishi, A method of preventing unauthorized data transmission in controller area network, in: 75th IEEE Vehicular Technology Conference, VTC Spring, 2012, pp. 1–5.

[13] T. Dagan, A. Wool, Parrot, a software-only anti-spoofing defense system for the CAN bus, in: 14th Embedded Security in Cars, ESCAR'16, Munich, Germany, 2016.

[14] U. Larson, D. Nilsson, E. Jonsson, An approach to specification-based attack detection for in-vehicle networks, in: IEEE Intelligent Vehicles Symposium, 2008, pp. 220–225.

[15] N. Erez, A. Wool, Control variable classification, modeling and anomaly detection in modbus/tcp SCADA systems, Int. J. Crit. Infrastruct. Prot. 10 (2015) 59–70.

[16] Arilou, http://ariloutech.com, 2015 [Online accessed 22-July-2015].

[17] TowerSec, http://tower-sec.com, 2015 [Online accessed 22-July-2015].

[18] Argus Cyber Security Ltd, http://argus-sec.com [Online accessed 22-July-2015].

[19] Security inMotion, http://www.security-inmotion.com [Online accessed 22-July-2015].

[20] J. Berg, J. Pommer, C. Jin, F. Malmin, J. Kristensson, Secure gateway – a concept for an in-vehicle IP network bridging the infotainment and the safety critical domains, https://www.escar.info/images/Datastore/2015_escar_usa/ESCAR2015USA_FullPaper_JonasBergEtAl_SecureGateway.pdf.

[21] A. Liu, C. Meiners, E. Torng, TCAM razor: a systematic approach towards minimizing packet classifiers in tcams, IEEE/ACM Trans. Netw. 18 (2) (2010) 490–500, http://dx.doi.org/10.1109/TNET.2009.2030188.

[22] A. Bremler-Barr, D. Hendler, Space-efficient TCAM-based classification using gray coding, IEEE Trans. Comput. 61 (1) (2012) 18–30, http://dx.doi.org/10.1109/TC.2010.267.

[23] PEAK-System, PCAN-USB: CAN Interface for USB, http://www.peak-system.com/PCAN-USB.199.0.html?&L=1 [Online; accessed 22-July-2015].

[24] C. Meiners, A. Liu, E. Torng, Bit weaving: a non-prefix approach to compressing packet classifiers in TCAMs, IEEE/ACM Trans. Netw. 20 (2) (2012) 488–500, http://dx.doi.org/10.1109/TNET.2011.2165323.

[25] H. Liu, Efficient mapping of range classifier into ternary-CAM, in: Proceedings. 10th Symposium on High Performance Interconnects, 2002, 2002, pp. 95–100.