Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Context-Aware on-board Intrusion Detection System

Davide Micale
davide.micale@phd.unict.it
University of Catania
Ilaria Matteucci
National Research Council
Florian Fenzl
Fraunhofer Institute for Secure Information Technology
Roland Rieke
Fraunhofer Institute for Secure Information Technology
Giuseppe Patanè
Park Smart SRL

Research Article

Keywords: Automotive, Intrusion Detection System, Context-aware, Machine learning

Posted Date: March 10th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2650857/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at International Journal of Information Security on March 28th, 2024. See the published version at https://doi.org/10.1007/s10207-024-00821-3.

Noname manuscript No.
(will be inserted by the editor)

A Context-Aware on-board Intrusion Detection System

Davide Micale · Ilaria Matteucci · Florian Fenzl · Roland Rieke · Giuseppe Patanè

Received: date / Accepted: date

Abstract Modern vehicles are becoming increasingly attractive from the perspective of possible intruders. The main reasons are twofold: modern vehicles are now connected to the outside world via Wi-Fi, Bluetooth, and mobile connection, such as LTE and 5G, and the increasing complexity of the on-board software enlarges the attack surface.

In this article, we introduce CAHOOTv2, a context-sensitive intrusion detection system (IDS) that uses the vehicle's sensors to determine driver habits and gather information about the environment to detect intruders. We use hyperparameter tuning to increase detection accuracy. To demonstrate the validity of the algorithm, we collected driving data from both an Artificial Intelligence (AI) and 39 humans. We include the AI driver to demonstrate that CAHOOTv2 is able to detect intrusions when the driver is both a human or an AI. The dataset is obtained using a modified version of MetaDrive simulator where we consider also the presence of an intruder able to perform the following types of intrusions: denial of service, replay, spoofing, additive and selective attacks. We make several experiments showing the benefits of hyperparameters tuning. The results of CAHOOTv2 are promising on detection of intrusions.

D. Micale
University of Catania, Italy
E-mail: davide.micale@phd.unict.it

I. Matteucci
IIT-CNR, Pisa, Italy
E-mail: ilaria.matteucci@iit.cnr.it

F. Fenzl
Fraunhofer SIT, Germany
E-mail: florian.fenzl@sit.fraunhofer.de

R. Rieke
Fraunhofer SIT, Germany
E-mail: roland.rieke@sit.fraunhofer.de

G. Patanè
Park Smart Srl, Italy
E-mail: giuseppe.patane@parksmart.it

Keywords Automotive · Intrusion Detection System · Context-aware · Machine learning

1 Introduction

Vehicles are increasingly connected to the outside world thanks to the introduction of several mobile technologies like LTE and 5G. In 2021 there were approximately 236 million connected vehicles worldwide [23]. Moreover, it is estimated that in 2035 the connected vehicles will increase up to 863 millions [23]. Inside vehicles there are also several Electrical Control Units (ECUs) that provide functionalities to the car [7]. ECUs are connected each other through multiple buses, e.g., Controller Area Network (CAN), CAN-FD, FlexRay and Automotive Ethernet. Different partitions of these busses are connected to each other via gateways.

Modern vehicles are also connected to other vehicles or the roadside units of the infrastructure via V2X communications. Using V2X communications, each vehicle is able to get information about the surrounding environment. These information may influence driving decisions, e.g., change route because of a traffic jam.

Also, many newer vehicles are connected through LTE or 5G to the carmakers' server. Carmakers collect information of the car to offer services, e.g., sensors' data, air conditioning management, route planning and history, insurance premium charges, maintenance history and battery management for electrical vehicles. In particular, carmakers can offer to third party the access to the sensors' data.

In addition, being connected to the World, vehicles start to resemble computer on wheels: on-board software is becoming increasingly complex. Nowadays, vehicles contain one hundred of millions of lines of code [4]. However, level 5 autonomous vehicles will contain up to one billion lines of code [4]. In fact, cars contain various sensors to keep track of the environment and the vehicle status [35]. The sensors' data can be accessed internally through the CAN bus protocol or from the external using an OBD-II diagnostic port [28]. In case of an autonomous car, sensors' data are processed by programmable components, such as, Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) [5], to improve the driver's experience.

In summary, the increasing of connected vehicles, in conjunction with the increase of complexity of vehicles software, may potentially facilitate vehicle intrusions.

In the last decade, the literature presents several examples of vehicle's attacks. In 2016, a vulnerability in the web browser of Tesla vehicles allowed an intruder to remotely send messages in the CAN bus [3]. For instance, researchers Ralf-Philipp Weinmann and Benedikt Schmotzle found a vulnerability in a software component of Tesla that allowed them to unlock the doors and trunk, change seat positions and change both steering and acceleration modes [32]. Also, using a privilege escalation exploit, it is possible to use the compromised vehicle to compromise surrounding vehicles.

The study of how to protect the CAN buses from in-vehicle vulnerabilities is extremely important. In fact, all the attacks in literature leverage the lack of confidentiality for data in transit on the intra-vehicle CAN bus network, which are, consequently, exposed to several threats. An intruder may exploit local or remote vulnerabilities of a car to gain some digital access to it, either locally or remotely. She may then modify the behaviour of a target vehicle by sending customized CAN frames that trigger a specific functionality on a receiving ECU.

In 2022 the EUROPOL has arrested 31 criminals that were selling a tool, marketed as a diagnostic tool, to replace the original software of the vehicle. The software replacing allowed the criminals to steal keyless cars from two French carmakers without using the original keys [6].

The standard ISO/IEC 27039:2015 [9] and the regulation number 155 of the UNECE (UNECE R155), delivered in 2021, of the United Nations [21] prescribed the use of Intrusion Detection and Prevention Systems (IDPS) to monitor the vehicles from intrusions. Under the IDPS umbrella, an Intrusion Detection System (IDS) merely reports an intrusion alert, while an Intrusion Prevention System (IPS) alerts and prevents the intrusions. In particular, vehicular context-aware IDSs use the semantic of the messages to detect intrusions.

In this article, we present CAHOOTv2, an improvement of CAHOOT [19], a context-aware IDS able to detect intrusions into a sequence of in-vehicle messages related to a driver's driving style. Indeed, CAHOOT is the first IDS based also on context information able to detect replay and DoS attack in addition to the spoofing attack.

Contextual information allows CAHOOTv2 to better detect intrusions. For example, if a driver accelerates and a sensor detects an obstacle in front of the vehicle, CAHOOTv2 classifies this behaviour as a possible intrusion. The environment context is digitally represented by the sensors' values.

1.1 State of the art

In RAIDS [10] and [14] the IDS detects intrusions exploiting the images from the on-vehicle camera and the CAN messages. Each work uses two Convolutional Neural Networks trained to detect spoofing attacks.

Rajapaksha et al. [25] propose an IDS that uses Gated Recurrent Unit neural network trained only using benign data over CAN messages. A minimum probability threshold is estimated to detect the intrusion. Authors evaluated the work on several public available datasets.

Xue et al. [33] introduced an IPS that uses the vehicle dynamics to detect intrusions. In particular, authors define policies starting from the specifications of the target vehicle, in-vehicle messages and onboard sensors to detect intrusion that could affect the safety of the driver.

The detection of sequence context anomalies can be made following different approaches. Rieke et al. [27] used process mining. Levi et al. [15] and Narayanan et al. [20] proposed works that use hidden Markov models. Theissler et al. [30] used a One Class Support Vector Machine (OCSVM), while in the Kang et al. [12] work neural networks are used. Marchetti et al. [18] used detection of anomalous patterns in a transition matrix. Taylor et al. [29] and Kalutarage et al. [11] used frequency of appearance of a sequence of CAN messages.

Karopoulos et al. [13] propose a new vehicular IDS taxonomy where each IDS belongs to multiple categories. Also the authors provide a survey of the publicly released datasets, simulation tools and IDSs.

The survey of Grimm et al. [8] focuses on the benefits of the context-aware approach on several security fields and the related work. Al-Jarrah et al. [1] provide a survey of IDSs and categorizing them. The authors

also note the importance of considering the semantics of data and context to detect anomalies.

Micale et al. [19] introduce CAHOOT, the context-aware IDS that detects intrusions either on AI and human drivings for several attacks types. The algorithm is tested using several machine learning algorithms in a dataset made by five humans on a simulator. Random Forest obtained the best results.

In the following, we describe the advantages of CAHOOTv2 with respect to CAHOOT and the related work.

1.2 Contribution

The advantages of CAHOOTv2 with respect to CAHOOT are:

- CAHOOTv2 is trained to detect two variants of spoofing attack.
- CAHOOTv2 improves intrusion detection accuracies with respect to CAHOOT. The Machine Learning algorithms present parameters that must be set before the training process starts and may influence the generated model. These parameters are called hyperparameters [34]. The process of searching the hyperparameters that improve the performance of the models is called hyperparameters tuning [34]. In CAHOOTv2, we design a paradigm that selects the best hyperparameters to use.
- To validate the performance of the algorithm, we also expanded the dataset collecting driving data from 39 humans.

Also, the advantages of CAHOOTv2 over related context-aware IDSs works are:

- CAHOOTv2 detects DoS and replay attacks in addition to spoofing attacks and variants.
- CAHOOTv2 detects intrusions that target steering, throttle and brake instead of only steering or steering and brake.
- CAHOOTv2 detects intrusions on both AI and human driving.

1.3 Article's Structure

The article is structured as follows: the next section presents the attackmodel. Section 3 describes the CAHOOTv2 algorithm. Section 4 shows the results of our experiments. Section 5 concludes the paper with suggestions of future improvements of the algorithm.

2 Attack Model

As attack model we consider an *internal intruder* that can be deployed in: a) ECUs that control the steering wheel, engine and brake b) sensors. The attacker is able to forge and sniff messages and performs the following attacks:

- *DoS* attack: the intruder is able to deny driver input by generating CAN frames in which payload values for steering, throttle and brakes are set to zero.
- *Replay* attack: the intruder is able to re-use valid CAN frames with a malicious or fraudulent aim.
- *Spoofing* attack: the intruder is able to generate a valid CAN frame. For example, the forged frame may generate a valid signal to activate an ECU functionality. We also consider two spoofing attack variants presented in [10]:
  - *Additive* attack: the intruder uses the current valid CAN frame payload and adds a random value in $\pm[0.2, 0.9]$ to simulate an abrupt steering, acceleration or brake.
  - *Selective* attack: the intruder introduces a CAN frame that contradicts the driver's will. The intruder uses the current valid CAN frame payload and flips the sign if the payload absolute value is greater than 0.3 or adds a random value in $\pm[0.5, 1]$.

3 CAHOOTv2 algorithm

The CAHOOTv2 algorithm is built on top of CAHOOT [19] and aims to detect more attacks and increases the accuracy on the older ones through the hyperparameters tuning.

CAHOOTv2 inherits from CAHOOT several characteristics:

- CAHOOT has the ability to detect intrusions while car is moving analyzing the semantic of CAN messages.
- The algorithm CAHOOT leverages machine learning (ML) techniques for the intrusion detection.
- The algorithm can also detect intrusions when the driver and the intruder generate the same CAN message value.
- The driver can be a human or an AI.
- CAHOOT detects three types of intrusions that target steering, throttle and brake.

In the following, we describe the paradigms that CAHOOTv2 and CAHOOT have in common, the pseudocodes of the new attacks and how we integrate them on the intruder's behaviour. Then, we explain the paradigm responsible for improving the accuracy. Note that

in each pseudocode we detail the differences between CAHOOT and CAHOOTv2. Also, new CAHOOTv2's functions are described in detail.

3.1 MetaDrive

CAHOOTv2 is evaluated using MetaDrive [16], a driving simulator written in Python capable of procedurally generating infinite driving scenarios. Also, the simulator provides a pre-trained AI.

We introduce and intruder into the MetaDrive simulation workflow (Fig. 1). The in vehicle communication are represented by a set of messages of two Python lists: the steering messages and the throttle/brake messages sent by both the intruder and the legit driver.

For each step of the intrusion workflow (Fig. 1):

- The legit driver sends driving inputs while an intruder forges fake ones.
- Messages are sent to the set of messages that are read by CAHOOTv2.
- The CAHOOTv2 algorithm distinguishes forged messages from the legit ones.
- The component responsible of the steering and the throttle/brake receives the steering wheel and the throttle/brake messages and runs them to the simulated vehicle.
- The simulator clears the set of messages to be able to fill it again in the next simulation step.

Keep note that in the detection phase CAHOOTv2 do not need both legit and forged messages. If the intruder does not forge messages, CAHOOTv2 receives only the legit messages and establishes their legitimacy.

3.2 Intruder's Behaviour

In CAHOOTv2, the intruder frequently changes the attacks randomly choosing among the five described in Section 2. The duration of attacks are randomly chosen in an arbitrary interval of steps duration.

Listing 1 and Listing 2 describe our model of the intruder's behaviour. In particular, Listing 1 shows the algorithm *prepare_attack* that plans the duration of each vehicle intrusion, while Listing 2 depicts the algorithm *launch_attack*.

Listing 1 Prepare Attack

```
1  function prepare_attack(steering, throttle_brake,
       current_attack, steering_history,
       throttle_brake_history, index_history, prev_steering,
       prev_throttle_brake, stop_attack_time, min_duration,
       max_duration, slot_time)
2    should_attack_change ← stop_attack_time <= Current
       timestamp
3
4    if should_attack_change
5      num_slots ← Select an integer number between
         min_duration and max_duration
6      stop_attack_time ← Current timestamp +
         num_slots * slot_time
7
8      current_attack = None
9
10   (steering_forged, throttle_brake_forged, current_attack,
       index_history, prev_steering, prev_throttle_brake
       ) = launch_attack(steering, throttle_brake,
       current_attack, steering_history,
       throttle_brake_history, index_history,
       prev_steering, prev_throttle_brake)
11
12   steering_history ← Append steering to
       steering_history
13   throttle_brake_history ← Append throttle_brake to
       throttle_brake_history
14
15   return (steering_forged, throttle_brake_forged,
       current_attack, stop_attack_time, steering_history
       , throttle_brake_history, index_history,
       prev_steering, prev_throttle_brake)
```

The Listing 1 algorithm is the same of the *prepare_attack* presented in CAHOOT except for line 10 where $steering_{legit}$ and $throttle_brake_{legit}$ are sent to the function *launch_attack*. These values may be used to perform an additive or selective attack.

Listing 2 Launch Attack

```
1  function launch_attack(steering_legit,
       throttle_brake_legit, current_attack,
       steering_history, throttle_brake_history,
       index_history, prev_steering, prev_throttle_brake)
2    bootstrap ← False
3    if current_attack = None
4      bootstrap ← True
5
6      current_attack ← Randomly select one from
         "DoS", "Spoofing", "Replay", "Additive",
         "Selective"
7
8    if current_attack = "DoS"
9      (steering, throttle_brake) ← dos_attack()
10   if current_attack = "Spoofing"
11     (steering, throttle_brake) ←
         spoofing_attack(bootstrap, prev_steering,
         prev_throttle_brake)
12
13     prev_steering ← steering
14     prev_throttle_brake ← throttle_brake
15   if current_attack = "Replay"
16     (steering, throttle_brake, index_history) ←
         replay_attack(bootstrap, steering_history,
         throttle_brake_history, index_history)
17   if current_attack = "Additive"
```

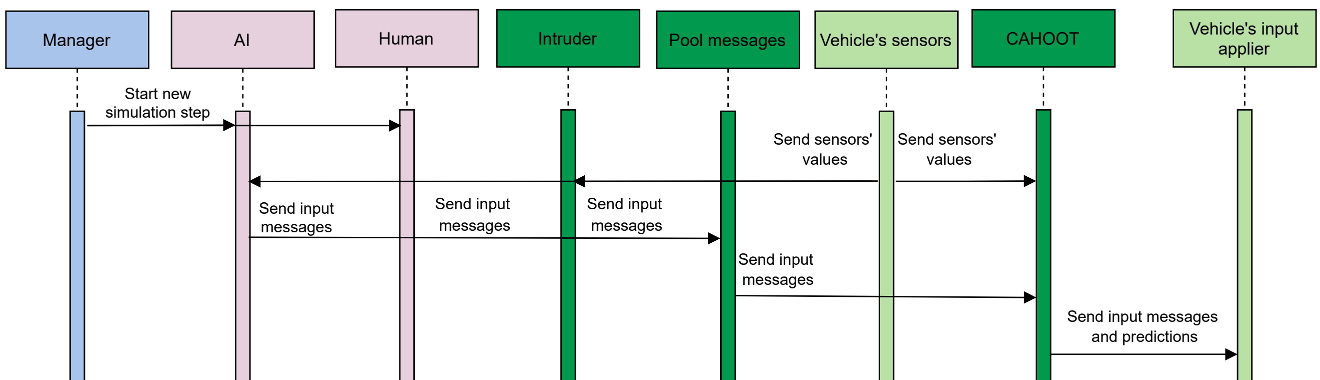

Fig. 1 Simulation sequence workflow of the vehicle

```
18        (steering, throttle_brake) ←
              additive_attack(steering_legit,
              throttle_brake_legit)
19    if current_attack = "Selective"
20        (steering, throttle_brake) ←
              selective_attack(steering_legit,
              throttle_brake_legit)
21
22    return (steering, throttle_brake, current_attack,
          index_history, prev_steering,
          prev_throttle_brake)
```

The Listing 2 algorithm is in charge of maintaining active and in progress attack or decide which attack should be run. In CAHOOTv2, *launch_attack* should randomly choose an attack between DoS, spoofing, replay, additive and selective (line 6). The additive and selective attacks need the $steering_{legit}$ and $throttle_brake_{legit}$ and apply to them mathematical operations to generate forged *steering* and *throttle_brake* (lines from 17 to 20).

3.2.1 Instances Extraction Paradigm

CAHOOTv2 requires a training dataset that contains both legit and forged messages. We label them as follows: $steering_{legit}$, $steering_{forged}$, $throttle_brake_{legit}$ and $throttle_brake_{forged}$, alongside with the sensors' values (Table 1).

The *instances_extraction* paradigm extracts the instances of the dataset to generate the final dataset. The new dataset contains messages organized in pairs, each one is labelled as T when it contains only legit messages, otherwise it is labelled as F (Table 2). With the organization in pairs, CAHOOTv2 is able to detect intrusions when intruder sends the same message sent by the driver. Let us suppose that the driver is not turning the steering wheel, i.e., $steering_{legit}$ is equal to 0, while the intruder starts a DoS attack, i.e., $steering_{forged}$ is equal to 0 (Table 1, row 3). The paradigm considers both the steering message forged by the intruder and the driver as legit since they are equal. However, based on the $throttle_brake_{legit}$ and $throttle_brake_{forged}$ the paradigm raises an alert (Table 2, rows 9 and 10). In case both the driver and the intruder send the same pair messages (Table 1, row 4), the algorithm inserts in the dataset only an instance labelled with T (Table 2, row 11).

3.2.2 New Attacks

Additive and selective attacks add a random value to $steering_{legit}$ and $throttle_brake_{legit}$. The sum operation may lead to a value that is not valid. Function *limit_value* (Listing 3) ensures that values greater than the *upper_bound* are changed in *upper_bound* (lines 5 and 6) and values lower than the *lower_bound* are changed in *lower_bound* (lines 7 and 8). In case the value is in [*lower_bound*, *upper_bound*], the function returns the value as it is (line 10). In MetaDrive, *upper_bound* and *lower_bound* are set to 1 and -1, respectively.

Listing 3 Limit value

```
1  function limit_value(value)
2      upper_bound ← maximum acceptable value
3      lower_bound ← minimum acceptable value
4
5      if value > upper_bound:
6          return upper_bound
7      if value < lower_bound:
8          return lower_bound
9
10     return value
```

The *additive_attack* function sets the *steering* and *throttle_brake* with random values (Listing 4). First, two values are randomly generated in $\pm[0.2, 0.9]$ (lines 2 and 3). Then, these values are added to $steering_{legit}$ and $throttle_brake_{legit}$. Next, *steering* and *throttle_brake* are sent as input to the *limit_function* (lines 8 and

Table 1 Example of instances before run Instances Extraction Paradigm [19]

timestamp	$steering_{legit}$	$steering_{forged}$	$throttle_brake_{legit}$	$throttle_brake_{forged}$	...
01/01/2022 12:00:00.000	0,695	0,403	0,020	-0,001	...
01/01/2022 12:00:00.100	0,045	0,494	-0,042	-0,533	...
01/01/2022 12:00:00.200	0,0	0,0	-0,042	0,0	...
01/01/2022 12:00:00.300	0,0	0,0	0,0	0,0	...

Table 2 Example of instances after run Instances Extraction Paradigm [19]

timestamp	*steering*	*throttle_brake*	...	*label*
01/01/2022 12:00:00.000	0,695	0,020	...	T
01/01/2022 12:00:00.000	0,695	-0,001	...	F
01/01/2022 12:00:00.000	0,403	0,020	...	F
01/01/2022 12:00:00.000	0,403	-0,001	...	F
01/01/2022 12:00:00.100	0,045	-0,042	...	T
01/01/2022 12:00:00.100	0,045	-0,533	...	F
01/01/2022 12:00:00.100	0,494	-0,042	...	F
01/01/2022 12:00:00.100	0,494	-0,533	...	F
01/01/2022 12:00:00.200	0,0	-0,042	...	T
01/01/2022 12:00:00.200	0,0	0,0	...	F
01/01/2022 12:00:00.300	0,0	0,0	...	T

9). Finally, the function returns the limited *steering* and *throttle_brake* values (line 11).

Listing 4 Additive Attack

```
1  function additive_attack(steering_legit,
       throttle_brake_legit)
2     random_value_1 ← random value in ±[0.2, 0.9]
3     random_value_2 ← random value in ±[0.2, 0.9]
4
5     steering ← steering_legit + random_value_1
6     throttle_brake ← throttle_brake_legit +
         random_value_2
7
8     steering_limited ← limit_value(steering)
9     throttle_brake_limited ← limit_value(throttle_brake)
10
11    return (steering_limited, throttle_brake_limited)
```

The *selective_attack* function creates a *steering* and *throttle_brake* pair based on the value of the legit ones (Listing 5). In case, $steering_{legit}$ is in $\pm[0, 0.3]$, a random value in $\pm[0.5, 1]$ is added to $steering_{legit}$ (lines from 2 to 4). In case $steering_{legit}$ is not in $\pm[0, 0.3]$, the forged *steering* is the legit one with the sign flipped (lines 5 and 6). Similarly, the forged *throttle_brake* is generated (lines from 8 to 12). Then, *limit_value* is launched on *steering* and *throttle_brake* (lines 14 and 15). Finally, the limited forged *steering* and *throttle_brake* are returned (line 17).

Listing 5 Selective Attack

```
1  function selective_attack(steering_legit,
       throttle_brake_legit)
2     if steering_legit in ±[0, 0.3]
3        random_value ← random value in ±[0.5, 1]
4        steering ← steering_legit + random_value
5     else
6        steering ← -steering_legit
7
8     if throttle_brake_legit in ±[0, 0.3]
9        random_value ← random value in ±[0.5, 1]
10       throttle_brake ← throttle_brake_legit +
            random_value
11    else
12       throttle_brake ← -throttle_brake_legit
13
14    steering_limited ← limit_value(steering)
15    throttle_brake_limited ← limit_value(throttle_brake)
16
17    return (steering_limited, throttle_brake_limited)
```

3.3 Hyperparameters Tuning Paradigm

Listing 6 and Listing 7 describe how the model is trained using the best hyperparameters.

While the parameters of a model are learned from the dataset in the training phase through the machine learning technique, the hyperparameters should be set manually by the data scientist before starting the training phase. In most cases, the default hyperparameters present in the ML frameworks works well. However, the hyperparameters can be tuned to find a model that performs better [24]. In Random Forest, the ML algorithm used in CAHOOTv2, the hyperparameters types are about the structure of each tree present in the forest, the structure of the forest and the randomness.

The Model Generation paradigm in CAHOOTv2 differs with respect to the paradigm in CAHOOT starting from line 9: based on the gain ratio rankings, the worst features are removed (lines 9 and 10) from both the train and test sets. Then, the *hyperparameters_tuning* function is called (line 12). Next, a random forest classifier is initialized using the hyperparameters received (line 14). Finally, a random forest model is trained using the train dataset ins_bf_{train} which returns a trained model (line 16).

Listing 6 Model Generation

```
1  function generate_model(ins_labelled, num_iterations,
       cross_validation, params_dist_random_search)
2     (ins_train, ins_test) ← split randomly the instances as
          training and testing sets from ins_labelled
3     ins_extracted_train ← generate_dataset(ins_train)
4     ins_extracted_test ← generate_dataset(ins_test)
5
6     ranking ← GR(instances)
7     features_>0 ← discard features with rank = 0 from
          ranking
8
9     ins_bf_train ← ins_extracted_train with features
          features_>0
10    ins_bf_test ← ins_extracted_test with features
          features_>0
11
12    params_best ← hyperparameters_tuning(ins_bf_train,
          ins_bf_test, num_iterations, cross_validation,
          params_dist_random_search)
13
14    rf ← initialize a Random Forest using params_best
15
16    model ← train rf using ins_bf_train
17    return model
```

Listing 7 depicts *hyperparameters_tuning* paradigm. Because there are several possible combinations of hyperparameters, it is not feasible to try all the possible combinations to find the best one. In the first phase, the paradigm creates several random forests with random combinations of hyperparameters and searches a subset of the best hyperparameters (lines from 2 to 23). Then, it tries every combinations of hyperparameters present in the subset to find the hyperparameters with the best accuracy (lines from 25 to 34). Each combination is tested using the cross validation technique to ensure that the hyperparameters are valid for the entire dataset and not only for a specific test set. The random forests generated in the first phase are trained and tested using the training dataset. Instead, in the second phase the random forests are trained using train and test set. Although the first phase is performed on a limited number of hyperparameter combinations, this phase is computationally very onerous especially for large datasets. To speed up the computation, we apply the first phase only to the train set. There is only a minority of data in the test set, so discarding the test set has a limited impact on the search for the best hyperparameters.

In the following, we explain in detail the first and the second phase. In the inputs of *hyperparameters_tuning* is present $params_dist_{random_search}$, a bi-dimensional array that contains for each type of hyperparameter a list of possible values that should be tried by the paradigm. First, the paradigm creates a list with the name of the hyperparameters that will be tested (line 2). Then, the array $params_accuracies_{random_search}$ containing the pairs of hyperparameters chosen and the accuracy obtained by the random forest algorithm, is defined (line 4). The *num_iterations* variable defines how many random combinations of hyperparameters are tested in the first phase.

A combination of hyperparameters is randomly generated (from line 6 to 9). For each type of hyperparameter present in $params_dist_{random_search}$, an hyperparameter is uniformly randomly chosen between all the possible values. Then, a random forest rf is initialized using the hyperparameters chosen. Next, rf is trained using the cross validation technique on the training dataset with the best features. The *cross_validation* variable defines the number of folds. The average accuracy is registered, alongside the list of the hyperparameters, in *params_accuracy* ["accuracy"] (lines 9 and 12). Then, the tuple is appended to the array of pairs hyperparameters/accuracy $params_accuracies_{random_search}$ (line 14).

Once the $params_accuracies_{random_search}$ is populated, the paradigm looks for a subset of the best features. First, the array $params_dist_{exhaustive_search}$ that will contain the subset of the best hyperparameters is defined (line 16). Then, the paradigm selects the best hyperparameters of each type. For each hyperparameter type $param_{name}$, the accuracies present in $params_accuracies_{random_search}$ are grouped by $param_{name}$ to obtain the average accuracy of each group (line 18). Then, the third quartile [17] is calculated on the average accuracies of the groups (line 20). The hyperparameters that have an average accuracies greater or equal to the third quartile are inserted in $params_dist_{exhaustive_search}$ (line 23). Hence, about 25 percent of the highest accuracies are selected for each type of hyperparameter.

Next, the train and test set are combined to obtain the entire dataset (line 25). The variables that will contain the best hyperparameters and the relative accuracy are defined (lines 27 and 28). Then, each possible combination of hyperparameters in $params_dist_{exhaustive_search}$ is tested using cross validation on the

entire dataset (lines from 29 to 31). In case the accuracy obtained is greater than the value currently in $accuracy_{best}$, the best hyperparameters and accuracy variables are updated (lines from 32 to 34). Finally, the paradigm returns the hyperparameters that obtained the best accuracy.

Listing 7 Hyperparameters Tuning Paradigm

```
1  function hyperparameters_tuning(ins_bf_train,
       ins_bf_test, num_iterations, cross_validation,
       params_dist_random_search)
2    params_name ← get the names of parameters in
         params_dist_random_search
3
4    params_accuracies_random_search ← empty array
5    for num_iterations:
6        params ← Empty array
7        for each param_name in params_name:
8            params[param_name] ← choose uniformly
                 random a hyperparameter in
                 params_dist_random_search[param_name]
9        params_accuracy["params"] ← params
10
11       rf ← initialize a Random Forest with params
             as hyperparameters
12       params_accuracy["accuracy"] ← train rf using
             cross_validation-fold cross validation
             applied to ins_bf_train
13
14       params_accuracies_random_search ← append
             params_accuracy in
             params_accuracies_random_search
15
16   params_dist_exhaustive_search ← empty array
17   for each param_name in params_name:
18       grouped_accuracies ← group
             params_accuracies_random_search by
             param_name and calculate the average
             accuracy of each group
19
20       third_quartile ← calculate the third quartile
             on grouped_accuracies["accuracy"]
21
22       params_accuracies_best_subset ← get the
             elements in grouped_accuracies on which
             grouped_accuracies["accuracy"] ≥
             third_quartile
23       params_dist_exhaustive_search[param_name] ←
             params_accuracies_best_subset["params"]
24
25   ins_bf ← ins_bf_train ∪ ins_bf_test
26
27   params_best ← None
28   accuracy_best ← 0
29   for each params combination in
         params_dist_exhaustive_search:
30       rf ← initialize a Random Forest with params
             as hyperparameters
31       accuracy ← train rf using cross_validation
             -fold cross validation applied to ins_bf
32       if accuracy > accuracy_best:
33           params_best ← params
34           accuracy_best ← accuracy
35
36   return params_best
```

4 CAHOOTv2 Evaluation

Hereafter, we describe how we evaluate the CAHOOTv2 algorithm on a dataset we generated by the MetaDrive simulation workflow depicted in Section 3.1.

4.1 Dataset

The dataset is generated using the driving simulator MetaDrive. Table 3 shows the features present in the dataset generated by the MetaDrive simulator.

We aim to obtain an IDS able to work with either humans and AI. Therefore, we decided to collect a dataset that contains data made by an AI and 39 humans. In particular, one human uses a keyboard while the remaining 38 use a Thrustmaster TMX [31]. Regarding the gender of the drivers, four drivers are females while the remaining 35 are males.

Figure 2 shows the ages grouped by the gender. Female drivers ages are between 19 and 27 in average 22,25 years old and median of 21,5 while male drivers ages are between 20 and 44 in average 24 years old and median of 22. Overall, the drivers' ages are between 19 and 44 with an average of 23,82 and median of 22.

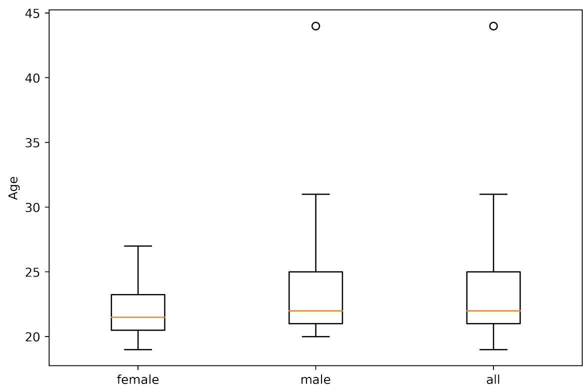

Fig. 2 Boxplots of genders

AI driving data is collected using the pre-trained AI present in MetaDrive to maintain the consistency of the same simulated vehicle in use between the AI and human drivers.

Table 3 Features description [19]

Feature	Description	Example	Unit
Speed	Speed of the vehicle	55	km/h
Throttle_brake	Amount of throttle or braking	0,55	N/A
Steering	Rotation of the steering wheel	-0.25	N/A
Last_position_x/y	Position of the vehicle at coordinate x/y	125	N/A
Dist_to_left/right_side	Distance from the left/right lane	0,423	m
Fuel_consumption	Fuel consumption since the start of the driving session	33,12	N/A
Engine_runtime_minute / second / millisecond	Minutes / seconds / milliseconds elapsed from engine start	39	minutes / s / ms
Yaw_rate	Angular acceleration on vertical axis	0.089661	N/A
Project_distance / velocity_to_vehicle_n_x / y	Vehicle's projection distance / velocity to the n-th nearest vehicle on coordinate x / y	0.187	N/A

4.2 Machine Learning algorithms

The CAHOOTv2 paradigm is implemented by using python-weka-wrapper3 [26] for the feature selection algorithm GainRatio and scikit-learn [22] that efficiently implements Random Forest (RF) [2].

Models generated using Random Forest technique obtain good results. However, tuning the hyperparameters, RF is able to achieve the best results. In the first experiment, we run the *hyperparameters_tuning* paradigm on CAHOOT algorithm with the dataset present in [19], hereafter called α. This dataset contains data made by the MetaDrive's AI and 5 humans using a Thrustmaster TMX. In the second experiment, we run CAHOOT and CAHOOTv2 on the dataset presented in the previous section, hereafter called β.

4.3 Experiments setup

To evaluate CAHOOTv2, we use several metrics, such as: *Accuracy*, *Precision* and *Recall*.

Accuracy represents how often the model is making a correct prediction.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

where:

- TP (True Positive) is the number of instances where at least one sensor's value is forged that are correctly predicted.
- TN (True Negative) is the number of instances where all the sensors' values are legit that are correctly predicted.
- FP (False Positive) is the number of instances where all the sensors' values are legit but incorrectly predicted.
- FN (False Negative) is the number of instances where at least one sensor's value is forged but incorrectly predicted.

Precision measures the ability of the classifier not to predict as forged an instance that is legit. It is calculated as follows:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

Recall measures the ability of the classifier to find all forged instances. It is calculated as follows:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

The dataset is randomly splitted in a training set that contains 85% of instances and a test set that contains the remaining 15%.

The intruder sends forged *steering* and *throttle_brake* messages while the driver is driving the simulated vehicle. Also, multiple attacks on each driving session are simulated through the setting of maximum and minimum duration of an attack respectively to 2 and 1 slots.

Table 4 shows the hyperparameters that we test in *hyperparameters_tuning* paradigm. We use 100 as number of iterations in the first phase.

We aim to detect the instances that contain at least one sensor's value forged from the *steering* and the *throttle_brake*.

4.4 Evaluation of *hyperparameters_tuning*

In the following, we compare the model that is trained by using the default hyperparameters with the one that

Table 4 Hyperparameters tested in hyperparameters_tuning paradigm

Hyperparameter	Description	Values
num_estimators	The number of trees that make up the forest	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
max_features	The number of features considered for the split	["log2", "sqrt"]
min_samples_split	The minimum number of samples required to split an internal node	[2, 7, 12, 18, 23, 28, 34, 39, 44, 50]
min_samples_leaf	The minimum number of samples required to be at a leaf node	[1, 6, 11, 17, 22, 28, 33, 39, 44, 50]
bootstrap	Whether to use the entire dataset to build each tree or a bootstrap sample	[true, false]
criterion	The function used to measure the quality of a split	["gini", "entropy"]

Table 5 Features selected by CAHOOT on α (percentage of each rank with respect to the sum of the ranks of the features)

Features	Rank percentage
steering	46,7%
throttle_brake	32,4%
speed	7,4%
yaw_rate	6,6%
fuel_consumption	2,3%
last_position_y	1,3%
last_position_x	0,9%
engine_runtime_minute	0,5%
engine_runtime_second	0,5%
dist_to_left_side	0,4%
project_distance_to_vehicle_1_y	0,3%
dist_to_right_side	0,2%
project_velocity_to_vehicle_0_y	0,2%

is trained using the best hyperparameters. The experiment is conducted on the same train and test set on dataset α.

Table 5 contains the list of features selected for the two models. To better distinguish features rankings, each feature rank is shown as a percentage of the sum of all the ranks.

The *steering* and *throttle_brake* messages are the most important features. The worse features are the distance from the right lane and the projection of velocity of the nearest vehicle in the y axis. The engine runtimes minutes and seconds are at the half of the table while the engine runtime milliseconds was discarded.

Table 6 shows that the search of hyperparameters increase the accuracy of 1,5%. The recall is 0,3% lower than the model trained with the best hyperparameters, but the precision is 0,9% higher, i.e., the false negative are slightly increased but false positive are decreased.

To better understand on which circumstances the customized hyperparameters best perform, we calculated the accuracy grouped by entity, i.e., human or the MetaDrive's AI is driving the car, and by type of attack, i.e., DoS, spoofing and replay. The model trained with custom hyperparameters is 1,2% more accurate with respect to the model trained with default hyperparameters on the MetaDrive's AI drivings. The attack that obtains the best accuracy increase is spoofing attack, i.e., 0,7%. On the other hand, the accuracy of replay attack increases only of 0,1%.

4.5 Evaluation of CAHOOTv2

In the following experiment, we compare three models: a model trained using CAHOOTv2 paradigm, i.e., a model trained to detect DoS, spoofing, replay, additive and selective attacks using the best hyperparameters, a model trained using CAHOOTv2 with the default hyperparameters and a model trained using CAHOOT paradigm, i.e., a model trained to detect only DoS, spoofing and replay attacks using the default hyperparameters. The experiment is conducted on the dataset β.

Table 7 contains the list of features selected for the three models. Keep note that CAHOOTv2 uses the same features regardless the hyperparameters selected. The table show that CAHOOTv2 and CAHOOT discard only *engine_runtime_millisecond*. While in CAHOOTv2 *steering* and *throttle_brake* together represent the 55,35% of the entire feature set, in CAHOOT *steering* and *throttle_brake* together represent the 82,62% of the entire feature set. Consequently, the remaining features are more important in CAHOOTv2. In all the models, the most important features are *steering*, *throttle_brake* and *speed*. While in CAHOOTv2 *dist_to_left_side* and *yaw_rate* are respectively the fourth and fiveth most important features, in CAHOOT they are only the ninth and the eighth most important features. In CAHOOT, the fourth and fiveth most

Table 6 Accuracy, precision and recall comparison on α of CAHOOT with default and best hyperparameters

CAHOOT with best hyperparameters			CAHOOT with default hyperparameters		
Accuracy	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**
96%	96,9%	97,6%	95,5%	96,0%	97,9%
Test only human drivers					
97,6%	98,2%	98,5%	97,2%	97,6%	98,6%
Test only MetaDrive's AI driver					
83,9%	88,1%	90,7%	82,7%	85,5%	92,5%
Test only Replay attack					
93,5%	96,2%	94,8%	93,4%	95,3%	95,5%
Test only DoS attack					
96,8%	96,6%	98,8%	96,3%	95,8%	98,9%
Test only Spoofing attack					
97,4%	97,7%	98,9%	96,7%	96,6%	99,1%

important features are *energy_consumption* and *last_position_x*.

In this case, Tables 8 and 9 show that CAHOOTv2 tuning the hyperparameters obtains the best accuracy, i.e., 0,3% of accuracy higher than the default hyperparameters and 8,2% of accuracy higher than CAHOOT. The model trained with the best hyperparameters increases the precision of 0,3% while maintaining equal the recall with respect to default hyperparameters.

We also calculated the accuracy grouped by entity, i.e., human or the MetaDrive's AI is driving the car, and by type of attack, i.e., DoS, spoofing, replay, additive and selective. Grouping allows us to better understand under what circumstances the model works best.

Considering tests only on humans, the model with the best hyperparameters obtains accuracy and precision scores greater than the ones obtained by the default hyperparameters and CAHOOT. Considering tests only on the MetaDrive's AI instances, the model with best hyperparameters has an accuracy slightly lower with respect to default hyperparameters, i.e., 0,1%, but the model is more balanced. The difference between precision and recall with the best hyperparameters is 3,5% while in the default hyperparameters is 5,5%.

Tables 8 and 9 show that that the model easily detects intrusions on instances where the human is driving the vehicle. On the other hand, it is more difficult detect intrusions on instances where the Metadrive's AI drives the vehicle. Humans tend to make gradually driving adjustments, whereas Metadrive's AI makes continuous and sudden changes. Graduality allows the model to detect more precisely an intrusion in progress for human drivings.

The replay attack is the most difficult to detect but CAHOOTv2 increases the accuracy up to 0,4% sacrificing some of the recall to increase the precision. The DoS attack is better identified by the model with the best hyperparameters, i.e., 0,3% more accuracy. However, CAHOOT is 0,1% more accurate but precision and recall are more unbalanced with respect to the best hyperparameters. The spoofing attack is the easiest to detect. All three algorithms obtain really high results, in particular CAHOOTv2 with the best hyperparameters, i.e., up to 0,3% more accurate. The additive attack and selective attack are easy to detect for CAHOOTv2 regardless the hyperparameters. However, the best hyperparameters allow the accuracy to increase up to 0,4%. CAHOOT is able to detect these attacks but with lower scores with respect to CAHOOTv2.

Table 7 Features selected by CAHOOTv2, with default and best hyperparameters, and CAHOOT on β (percentage of each rank with respect to the sum of the ranks of the features)

Features	Rank percentage	
	CAHOOTv2	CAHOOT
steering	31,83%	52,31%
throttle_brake	23,52%	30,31%
speed	9,0%	3,91%
dist_to_left_side	4,93%	0,4%
yaw_rate	4,47%	1,16%
last_position_y	3,92%	1,66%
last_position_x	3,33%	1,95%
energy_consumption	3,27%	2,1%
dist_to_right_side	3,07%	1,89%
project_distance/velocity_to_vehicle_n_x/y	from 1,24% to 0,14%	from 0,39% to 0,05%
engine_runtime_second	0,69%	0,18%
engine_runtime_minute	0,56%	0,17%

Table 8 Accuracy, precision and recall comparison on β between CAHOOTv2 and CAHOOTv2 with default hyperparameters

CAHOOTv2			CAHOOTv2 default hyperparameters		
Accuracy	Precision	Recall	Accuracy	Precision	Recall
97,9%	98,8%	98,2%	97,6%	98,5%	98,2%
Test only human drivers					
98,0%	99,0%	98,3%	97,8%	98,7%	98,3%
Test only MetaDrive's AI driver					
87,3%	89,9%	93,4%	87,4%	88,7%	95,2%
Test only Replay attack					
94,8%	96,9%	95,4%	94,5%	96,3%	95,6%
Test only DoS attack					
96,5%	97,1%	97,4%	96,3%	96,8%	97,4%
Test only Spoofing attack					
99,6%	99,5%	99,9%	99,4%	99,3%	99,9%
Test only Additive attack					
97,7%	99,5%	97,3%	97,3%	99,2%	97,1%
Test only Selective attack					
99,6%	99,7%	99,8%	99,5%	99,5%	99,8%

5 Conclusions and future work

The high complexity of newer vehicles increases the attack surfaces on which a vulnerability could be present. An intrusion while the vehicle is in motion could endanger the lives of the driver and passengers.

In this article, we introduced CAHOOTv2 that improves the ability on intrusion detection of CAHOOT generating more balanced models thanks to the best hyperparameters used for the training phase. We also expanded the dataset with additional drivers to better validate the results.

Security solutions are strongly linked to safety, especially when considering the automotive domain. CAHOOT and CAHOOTv2 are designed to be an IDS, so malicious event are just identified and no active reactions are implemented to avoid that they may impact the vehicle's safety. While driving a car, events may occur that require exceptional responses from the driver, e.g., a cat suddenly crossing the road forcing an abrupt stop. If not properly trained, an IDS may interpret these events as malicious. CAHOOT and CAHOOTv2 are already trained to identify dangerous situations, e.g., one of the simulated cars performed sudden overtaking or congested traffic that forced the driver to make abrupt decisions.

In future, we will design an algorithm that is able to detect intrusions and also is able to identify drivers while preserving their privacy. Rather than endangering the lives of the driver and passengers in the vehicle, the intruder could be interested on introduce CAN messages to misleading the driver identification system

Table 9 Accuracy, precision and recall comparison of CA-HOOT on β

Accuracy	**Precision**	**Recall**
91,7%	92,7%	95,9%
Test only human drivers		
91,8%	92,8%	96,0%
Test only AI drivers		
83,6%	85,4%	94,1%
Test only Replay attack		
94,4%	95,9%	95,7%
Test only DoS attack		
96,6%	96,6%	98,0%
Test only Spoofing attack		
99,3%	99,1%	99,9%
Test only Additive attack		
83,5%	87,1%	91,3%
Test only Selective attack		
86,7%	87,9%	95,4%

present in the vehicle to impersonate an authorized driver. To prevent this, the intrusion detection component of the algorithm may identify the forged messages and prevent them to reach the driver identification component.

5.0.1 Data Availability Statements

The data that support this research activity have been collected in a compliant way with respect to etichal and privacy regulation. Data can be made disclosed after the participant consent.

5.1 Declarations

5.1.1 Compliance with Ethical Standards

The authors declare that they have no conflict of interest.

5.1.2 Competing Interests

The authors declare that they have no known competing financial interest or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Al-Jarrah, O.Y., Maple, C., Dianati, M., Oxtoby, D., Mouzakitis, A.: Intrusion detection systems for intra-vehicle networks: A review. IEEE Access **7**, 21,266–21,289 (2019). DOI 10.1109/ACCESS.2019.2894183. URL https://ieeexplore.ieee.org/document/8642311
2. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001). DOI 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324
3. "CVE": Cve-2016-9337 (2016). URL:https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9337 [retrieved: 11, 2022]
4. Diess, H.: Levers to unleash value (2020). URL:https://www.volkswagenag.com/presence/investorrelation/publications/presentations/2020/01-januar/January_2020_VWAG_Investor_Roadshow.pdf [retrieved: 11, 2022]
5. Donzellini, G., Garavagno, A.M., Oneto, L.: Microprocessor systems on FPGA, pp. 439–553. Springer International Publishing, Cham (2022). DOI 10.1007/978-3-030-87344-8_5. URL https://doi.org/10.1007/978-3-030-87344-8_5
6. "EUROPOL": 31 arrested for stealing cars by hacking keyless tech (2022). URL:https://www.europol.europa.eu/media-press/newsroom/news/31-arrested-for-stealing-cars-hacking-keyless-tech [retrieved: 11, 2022]
7. Gmiden, M., Gmiden, M.H., Trabelsi, H.: An intrusion detection method for securing in-vehicle can bus. In: 2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), pp. 176–180 (2016). DOI 10.1109/STA.2016.7952095
8. Grimm, D., Stang, M., Sax, E.: Context-aware security for vehicles and fleets: A survey. IEEE Access **9**, 101,809–101,846 (2021). DOI 10.1109/ACCESS.2021.3097146
9. "International Organitation for Standardization: Iso/iec 27039:2015, information technology — security techniques — selection, deployment and operations of intrusion detection and prevention systems (idps) (2015). URL:https://www.iso.org/standard/56889.html [retrieved: 11, 2022]
10. Jiang, J., Wang, C., Chattopadhyay, S., Zhang, W.: Road Context-Aware Intrusion Detection System for Autonomous Cars. Lecture Notes in Computer Science p. 124–142 (2020). DOI 10.1007/978-3-030-41579-2_8. URL http://dx.doi.org/10.1007/978-3-030-41579-2_8
11. Kalutarage, H.K., Al-Kadri, M.O., Cheah, M., Madzudzo, G.: Context-aware anomaly detector for monitoring cyber attacks on automotive can bus. In: ACM Computer Science in Cars Symposium, CSCS '19. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3359999.3360496. URL https://doi.org/10.1145/3359999.3360496
12. Kang, M.J., Kang, J.W.: A novel intrusion detection method using deep neural network for in-vehicle network security. In: 2016 IEEE 83rd Vehicular Technology Conference (VTC Spring) (2016)
13. Karopoulos, G., Kambourakis, G., Chatzoglou, E., Hernández-Ramos, J.L., Kouliaridis, V.: Demystifying in-vehicle intrusion detection systems: A survey of surveys and a meta-taxonomy. Electronics **11**(7) (2022). DOI 10.3390/electronics11071072. URL https://www.mdpi.com/2079-9292/11/7/1072

14. Kondratiev, V., Kuznetsov, A.: An algorithm for intrusion detection into the control system of an unmanned vehicle. In: 2021 International Conference on Information Technology and Nanotechnology (ITNT), pp. 1–5 (2021). DOI 10.1109/ITNT52450.2021.9649295
15. Levi, M., Allouche, Y., Kontorovich, A.: Advanced analytics for connected cars cyber security. CoRR **abs/1711.01939** (2017)
16. Li, Q., Peng, Z., Xue, Z., Zhang, Q., Zhou, B.: Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. arXiv preprint arXiv:2109.12674 (2021)
17. Mann, P.S.: Introductory Statistics. Wiley (2009)
18. Marchetti, M., Stabili, D.: Anomaly detection of CAN bus messages through analysis of id sequences. In: 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 1577–1583 (2017)
19. Micale, D., Costantino, G., Matteucci, I., Fenzl, F., Rieke, R., Patanè, G.: Cahoot: A context-aware vehicular intrusion detection system. In: editor (ed.) In Proceedings of Trust-Comm 2022 (2022). Accepted to be published
20. Narayanan, S.N., Mittal, S., Joshi, A.: Obd securealert: An anomaly detection system for vehicles. In: IEEE Workshop on Smart Service Systems (SmartSys 2016) (2016)
21. "Official Journal of the European Union": Uniform provisions concerning the approval of vehicles with regards to cybersecurity and cybersecurity management system (2021). URL:`http://data.europa.eu/eli/reg/2021/387/oj` [retrieved: 11, 2022]
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
23. Placek, M.: Connected car fleet by region 2021-2035 (2022). URL:`https://www.statista.com/statistics/1155517/global-connected-car-fleet-by-market/` [retrieved: 11, 2022]
24. Probst, P., Wright, M., Boulesteix, A.L.: Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9** (2019). DOI 10.1002/widm.1301
25. Rajapaksha, S., Kalutarage, H., Al-Kadri, M.O., Madzudzo, G., Petrovski, A.V.: Keep the moving vehicle secure: Context-aware intrusion detection system for in-vehicle can bus security. In: 2022 14th International Conference on Cyber Conflict: Keep Moving! (CyCon), vol. 700, pp. 309–330 (2022). DOI 10.23919/CyCon55549.2022.9811048
26. Reutemann, P.: python-weka-wrapper3 (2020). URL:`https://fracpete.github.io/python-weka-wrapper3/index.html` [retrieved: 11, 2022]
27. Rieke, R., Seidemann, M., Talla, E.K., Zelle, D., Seeger, B.: Behavior analysis for safety and security in automotive systems. In: Parallel, Distributed and Network-Based Processing (PDP), 2017 25nd Euromicro International Conference on, pp. 381–385. IEEE Computer Society (2017)
28. "Snap-on Incorporated": Global obd vehicle communication software manual (2013). URL:`https://www.snapon.com/Files/Diagnostics/UserManuals/GlobalOBDVehicleCommunicationSoftwareManual_EAZ0025B43.pdf` [retrieved: 11, 2022]
29. Taylor, A., Japkowicz, N., Leblanc, S.: Frequency-based anomaly detection for the automotive CAN bus. In: 2015 World Congress on Industrial Control Systems Security (WCICSS), pp. 45–49 (2015)
30. Theissler, A.: Anomaly detection in recordings from in-vehicle networks. In: Proceedings of Big Data Applications and Principles First International Workshop, BIGDAP 2014 Madrid, Spain, September 11-12 2014 (2014)
31. "Thrustmaster": TMX Force Feedback. URL:`https://www.thrustmaster.com/products/tmx-force-feedback/` [retrieved: 11, 2022]
32. Weinmann, R.P., Schmotzle, B.: TBONE: for public release on 2021-04-28 (2021). URL:`https://kunnamon.io/tbone/` [retrieved: 11, 2022]
33. Xue, L., Liu, Y., Li, T., Zhao, K., Li, J., Yu, L., Luo, X., Zhou, Y., Gu, G.: SAID: State-aware defense against injection attacks on in-vehicle network. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 1921–1938. USENIX Association, Boston, MA (2022). URL `https://www.usenix.org/conference/usenixsecurity22/presentation/xue-lei`
34. Zhang, A., Lipton, Z.C., Li, M., Smola, A.J.: Dive into Deep Learning (2020). `https://d2l.ai` [retrieved: 11, 2022]
35. Zheng, B., Liang, H., Zhu, Q., Yu, H., Lin, C.W.: Next generation automotive architecture modeling and exploration for autonomous driving. In: 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 53–58 (2016). DOI 10.1109/ISVLSI.2016.126