CAHOOT: a Context-Aware veHicular intrusiOn detectiOn sysTem

Davide Micale
University of Catania, Italy
davide.micale@phd.unict.it

Gianpiero Costantino
IIT-CNR, Pisa, Italy
gianpiero.costantino@iit.cnr.it

Ilaria Matteucci
IIT-CNR, Pisa, Italy
ilaria.matteucci@iit.cnr.it

Florian Fenzl
Fraunhofer SIT, Germany
florian.fenzl@sit.fraunhofer.de

Roland Rieke
Fraunhofer SIT, Germany
roland.rieke@sit.fraunhofer.de

Giuseppe Patanè
Park Smart Srl, Italy
giuseppe.patane@parksmart.it

***Abstract*—Software in modern vehicles is becoming increasingly complex and subject to vulnerabilities that an intruder can exploit to alter the functionality of vehicles. To this purpose, we introduce CAHOOT, a novel context-aware Intrusion Detection System (IDS) capable of detecting potential intrusions in both human and autonomous driving modes. In CAHOOT, context information consists of data collected at run-time by vehicle's sensors and engine. Such information is used to determine drivers' habits and information related to the environment, like traffic conditions. In this paper, we create and use a dataset by using a customised version of the MetaDrive simulator capable of collecting both human and AI driving data. Then we simulate several types of intrusions while driving: denial of service, spoofing and replay attacks. As a final step, we use the generated dataset to evaluate the CAHOOT algorithm by using several machine learning methods. The results show that CAHOOT is extremely reliable in detecting intrusions.**

***Index Terms*—Automotive, Intrusion Detection System, Context-aware, Machine learning.**

I. Introduction

Over the years, vehicles functionalities are managed by increasingly complex software. For instance, vehicles made by Volkswagen nowadays contain one hundred millions lines of code [1]. Level 5 autonomous vehicles will contain up to one billion lines of code [1] because all vehicles' functionalities will be electronically managed. Moreover, during the driving experience, a vehicle is able to collect a lot of information from its sensors, the Electrical Control Units (ECUs), and also from the environment. The driver can exploit the connectivity of the vehicle to read this information through OBD-II and a mobile connection, while the multimedia functionalities can be accessed via USB, disc, SD-card, Bluetooth and WiFi. The European Union Agency for Network and Information Security (ENISA) defines today's vehicles as smart car, i.e., vehicles that offer enhanced users experience and safety, and provide connectivity and added-value features [2].

Thus, as for Personal Computer years ago, nowadays, guaranteeing the security of vehicles is becoming a strong requirement. In the last decade, there are several papers in literature that present work on vehicle's attacks. The most famous one has been presented at the Black Hat USA 2015 by Miller and Valasek [3]. In particular, the two researchers were able to remotely control the wheels of a Jeep Cherokee by exploiting the Park Assist System (PAS) [4]. Vulnerabilities not only damage the reputation of car manufacturers but also their profits (the attack to the Jeep Cherokee forced the manufacturer Fiat-Chrysler to recall 1,4 million cars in the USA [5]).

To mitigate the occurrences of such kind of attacks, both the standard ISO/IEC 27039:2015 [6] and the United Nations delivered the regulation number 155 [7], UNECE R155, delivered in 2021, prescribe the introduction of Intrusion Detection and Prevention Systems (IDPS) as security mechanisms to monitor the target vehicle for intrusions. In particular, an Intrusion Detection System (IDS) is able only to alert when intrusions are detected, while an Intrusion Prevention System (IPS) tries also to prevent the detected intrusions.

This paper proposes a Context-Aware veHicular intrusiOn detectiOn sysTem (CAHOOT) paradigm that uses vehicle sensors to detect and mitigate intrusions in cars. CAHOOT uses contextual information taking into account the semantics of in-vehicle messages. For example, if a driver accelerates in front of an obstacle detected by a sensor, CAHOOT detects this strange behaviour as a possible intrusion. The sensors' values are a digital representation of the environment context.

The paper is structured as follows: the next section discusses about related work. Section III presents the vehicle's anatomy and the attack's model we refer to. Section IV describes the CAHOOT algorithm and Section V shows the results of our experiments. Section VI draws the conclusion of the paper and highlights possible future work.

II. Related Work

In either [8] and [9] the authors developed an IDS that establishes valid messages based on the road-context for autonomous vehicles. In particular, they combines CAN messages with images recorded from the camera to establish the validity of messages, i.e., to ecognize spoofing attacks on the steering wheel through two convolutional neural networks. Public available datasets are used for the evaluation.

Wasicek et al. [10] uses a Bottleneck Neural Network to read sensors' values and to determine a vehicle's anomalous

behaviour. The evaluation is made on a car with installed a chip tuning into the Engine Control Module (ECM) that changes its behaviour. Casillo et al. [11] present a Bayesian Network to detect malicious CAN messages. The dataset used for the training is generated by the autonomous driving CARLA simulator [12] whose AI periodically receives attacks.

The methods for detection of sequence context anomalies comprise different approaches such as process mining that is used in the work of Rieke et al. [13], hidden Markov models which are used in the work of Levi et al. [14] and Narayanan et al. [15], OCSVM is used in the work of Theissler et al. [16], neural networks are used by Kang et al. [17], detection of anomalous patterns in a transition matrix are used by Marchetti et al. [18], and, frequency of appearance of a sequence of CAN messages is used by Taylor et al. [19] and Kalutarage et al. [20]. Grimm et al. [21] provide a comprehensive survey on context-aware security approaches in the vehicular and related domains, while Al-Jarrah et al. [22] extensively survey the current state of the art on IDS systems for in-vehicle networks. Al-Jarrah et al. conclude that currently the area of context-aware systems is still under-investigated.

CAHOOT extends the existing literature because it is the first IDS based also on context information able to detect replay and DoS attack in addition to the spoofing attack. Moreover, the simulation environment and activity we present is the only one that take into account simultaneously brakes, steering and throttle parameters.

III. The Attack Model

A lot of information circulate inside and outside vehicles by using ICT systems that are installed on it. An autonomous car contains various sensors to keep track of the environment and the vehicle status [23]. Inside the vehicle, there are also several ECUs that provide functionalities to the car. Such ECUs are connected one another through multiple buses, e.g., CAN, CAN-FD, FlexRay and Automotive Ethernet. Different partitions of these busses are connected each other through gateways. Thus, vehicles are computers on wheels and as normal computer can be subject to remote attacks. An intruder may exploit local or remote vulnerabilities of a vehicle to gain some digital access to it, either locally or remotely.

We consider an *intruder* able to run the following attacks:

- *DoS* attack: the intruder is able to deny the driver's input through the generation of CAN frames where payloads values are set to zero for steering, throttle and brakes.
- *Spoofing* attack: the intruder is able to generate a valid CAN frame. For example, the forged frame may generate a valid signal to active an ECU functionality.
- *Replay* attack: the intruder is able to re-use valid CAN frames with a malicious or fraudulent aim.

IV. CAHOOT algorithm

The CAHOOT algorithm aims to detect an intruder that performs both single or multiple attacks among the ones we listed in the previous section while a car is moving. It is also able to detect a possible intrusion also when both the intruder and the driver generated a CAN message with the same values.

CAHOOT uses machine learning (ML) techniques to generate a model capable to detect intrusions from the value of the vehicle sensors.

A. Intruder's Behaviour

To create a model that is as accurate as possible, we assume that the intruder is able to frequently change the attacks among the three attacks described in Section III. The duration of each attack is randomly chosen with an arbitrary minimum and maximum of steps duration. In addition, the type of attack is randomly chosen. This allows us to identify both single and multiple attacks within a target driving session. Listing 1 and Listing 2 describe intruder's behaviour model we consider.

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  if should_attack_change
4  {num_slots ← Select an integer number between min_duration and
       max_duration
5  stop_attack_time ← Current timestamp + num_slots * slot_time
6  current_attack = None}
7  (steering_hacked, throttle_brake_hacked, current_attack, index_history
       , prev_steering, prev_throttle_brake) =
       launch_attack(current_attack, steering_history,
       throttle_brake_history, index_history, prev_steering,
       prev_throttle_brake)
8  steering_history ← Append steering to steering_history
9  throttle_brake_history ← Append throttle_brake to
       throttle_brake_history
10 return (steering_hacked, throttle_brake_hacked, current_attack,
       stop_attack_time, steering_history, throttle_brake_history,
       index_history, prev_steering, prev_throttle_brake)
```

Listing 1 shows the algorithm *prepare_attack* that plans the duration of each vehicle intrusion. In detail, it checks if the attack in progress should continue or should be changed, i.e., the algorithm compares the current time with the time on which the attack must be suspended (line 2). In case the attack should end and be changed with a new type, the algorithm defines the duration of the new attack as slots of time. The algorithm randomly choose the number of slots between the minimum and maximum (line 4). Hence, the attack will stop at the sum between the actual time and the product between the number of slots and the length of each slot (lines 5). The attacks are periodically stopped and substituted with new ones to simulate multiple attacks in a single driving session.

Regardless of the attack should change or not, the function *launch_attack* is called (line 7) and returns the new forged messages alongside with the current type of attack, the index of the next messages that the replay attack must repeat, i.e., *index_history*, and the last forged messages that the spoofing attack must repeat, i.e., $prev\_steering$ and $prev\_throttle\_brake$. Next, the inputs steering and the throttle_brake of human/AI are registered in the arrays $steering\_history$ and $throttle\_brake\_history$ (lines 8 and 9). These arrays may be used later on for the replay attack. The attack inputs are never appended in the arrays because

the replay attack goal is to mimic the human/AI inputs so the attack should replay only human/AI inputs.

The algorithm returns the values of steering and throttle_brake generated by the intruder, the type of attack actually in progress, the time on which the attack will be suspended, the history values of steering and throttle_brake, the $index\_history$, $prev\_steering$ and $prev\_throttle\_brake$ (line 10).

Listing 2: Launch Attack

```
1  function launch_attack (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  current_attack ← Randomly select one from "DoS", "Spoofing" and "Replay"}
6  if current_attack = "DoS"
7  {(steering, throttle_brake) ← dos_attack()}
8  if current_attack = "Spoofing"
9  {(steering, throttle_brake) ← spoofing_attack(bootstrap, prev_steering,
       prev_throttle_brake)
10 prev_steering ← steering
11 prev_throttle_brake ← throttle_brake}
12 if current_attack = "Replay"
13 {(steering, throttle_brake, index_history) ← replay_attack(bootstrap,
       steering_history, throttle_brake_history, index_history)}
14 return (steering, throttle_brake, current_attack, index_history,
       prev_steering, prev_throttle_brake)
```

Listing 2 depicts the algorithm *launch_attack*. It is in charge of maintaining active and in progress attack or decide which attack should be run. The Spoofing and Replay attack need the variable *bootstrap* that represents if the attack is in progress or not, i.e., the variable tracks if a new attack must be launched or a previous attack must continue. The variable is *False* in case the attack is in progress (line 2) and *True* when the attack is not running (line 4). In case an attack is not in progress, the type of attack is randomly chosen between DoS, Spoofing and Replay (line 5). Once the *bootstrap* variable is established, based on the current attack value, an attack is launched (lines 6 to 13). Keep note that in case of spoofing attack, the $prev\_steering$ and $prev\_throttle\_brake$ variables are updated with the most recent hacked messages generated (lines 10 and 11).

Finally, the *launch_attack* returns the steering and throttle_brake values chosen by the attack, the current type of attack, the *index_history* selected by the Replay attack function last time it is launched and the previous pair of steering and throttle_brake used by the Spoofing attack (line 14).

B. Instances Extraction Paradigm

To train the model to detect intrusion, CAHOOT requires a dataset that contains both legit and forged messages for each functionalities we aim to consider, i.e., $steering_{legit}$, $steering_{hacked}$, $throttle\_brake_{legit}$ and $throttle\_brake_{hacked}$, with also the sensors' values (Table I).

The instances of the dataset are extracted to generate the final dataset on which the messages are organized in pairs and, each pair is labelled as T when it is composed by $steering_{legit}$ and $throttle\_brake_{legit}$ or as F in all the other cases (Table II). The organization in pairs allows CAHOOT to detect possible intrusion that may happen when the intruder is going to send the same message sent by the driver. In fact, let us suppose that the driver wants to go straight, i.e., $steering_{legit}$ is equal to 0, and the intruder starts a DoS attack, i.e., $steering_{hacked}$ is equal to 0 (Table I, row 3). The steering message sent by the intruder is considered as legit because it is equal to the driver's one. However, the algorithm raises an alert based on the values of $throttle\_brake_{legit}$ and $throttle\_brake_{hacked}$ that should be different (Table II, rows 9 and 10). On the other hand, if both the messages in the pair are equal (Table I, row 4), for instance because the intruder is trying to perform a DoS attack, then CAHOOT only inserts into the dataset one instance labelled with T (Table II, row 11). In this way, it prevents the DoS by discarding the flow of not legit messages.

Hence, on the initial dataset we run the $instances\_extraction$ function (Listing 3) whose output is the dataset $ins_{extracted}$ that contains the final created dataset.

As first step, the algorithm reads each $instance$ of the initial dataset ins (line 3) to organize the messages in two arrays. The first array contains tuples composed by steering message alongside with a boolean value representing message's legitimacy. The second array contains tuples composed by throttle_brake message alongside with a boolean value representing message's legitimacy.

The two arrays are used to organize all the instances in the initial dataset in such a way that legit and hacked messages are clearly distinguishable: the legit messages are inserted in the arrays (lines 10 and 11), while the hacked messages are inserted only if they are other than the respective legit ones (lines from 12 to 15). From $instance$ the messages $steering_{legit}$, $steering_{hacked}$, $throttle\_brake_{legit}$ and $throttle\_brake_{hacked}$ are removed (line 16). Thus, $instance$ now contains the engine runtime and the sensors' values.

The algorithm creates several instances based on $instance$, one instance per each combination of the steering and throttle_brake messages present respectively in $steering\_array$ and $throttle\_brake\_array$ (lines 19 and 20). Then, each generated instance is labeled "T" in case it contains only messages from the driver or "F" in case it contains at least one message from the intruder (lines from 21 to 24). Next, each labeled instance is added to the $ins_{extracted}$ dataset (line 25). After all the instances present in ins are read, the algorithms return the dataset $ins_{extracted}$ (line 26).

Listing 3: Instances Extraction Paradigm

```
1  function instances_extraction (ins)
2  ins_extracted ← empty array
3  for each instance in ins
4  {steering_legit ← instance["steering_legit"]
5  steering_hacked ← instance["steering_hacked"]
6  throttle_brake_legit ← instance["throttle_brake_legit"]
7  throttle_brake_hacked ← instance["throttle_brake_hacked"]
8  steering_array ← empty array
9  throttle_brake_array ← empty array
10 steering_array ← steering_array ⋃ (steering_legit, True)
11 throttle_brake_array ← throttle_brake_array ⋃
       (throttle_brake_legit, True)
12 if steering_legit != steering_hacked
13 {steering_array ← steering_array ⋃ (steering_hacked, False)}
14 if throttle_brake_legit != throttle_brake_hacked
15 {throttle_brake_array ← throttle_brake_array ⋃
       (throttle_brake_hacked, False)}
16 remove from instance the columns "steering_legit", "steering_hacked",
       "throttle_brake_legit", "throttle_brake_hacked"
17 for each (steering, is_steering_legit) in steering_array
```

TABLE I: Example of instances before run Instances Extraction Paradigm

$timestamp$	$steering_{legit}$	$steering_{hacked}$	$throttle\_brake_{legit}$	$throttle\_brake_{hacked}$	...
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 II: Example of instances after run Instances Extraction Paradigm

$timestamp$	$steering$	$throttle\_brake$	...	$label$
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

```
18 {for each (throttle_brake, is_throttle_brake_legit) in
       throttle_brake_array
19 {instance["steering"] ← steering
20 instance["throttle_brake"] ← throttle_brake
21 if is_steering_legit == True and is_throttle_brake_legit == True
22 {instance["label"] ← "T"}
23 else
24 {instance["label"] ← "F"}
25 ins_extracted ← ins_extracted ⋃ instance}}}
26 return ins_extracted
```

C. Model generation

The Model Generation paradigm uses the Instances Extraction paradigm to generate the training and the test datasets (Listing 4). Going more into detail, once the dataset is randomly split in a training set and a test set (line 2), the instances are extracted for the training and test (lines 3 and 4). We run the extraction paradigm separately on the training set and the test set to make sure that all combinations of steering and throttle_brake messages from the same original instance are not distributed between the training set and the test set, but remain in the same set. The appearance of extracted instances of the same original in both training and test sets causes a data leakage [24]. Data leakage happens when information present in the training set is unexpectedly present also in the test set. Next, the best features are selected using a Feature Selection (FS) paradigm that ranks all features applying the *Gain Ratio* [25] (GR) approach (line 5). Those features with rank equal to zero are discarded (line 6), the other are passed to the ML algorithm which returns a trained model (line 7).

Listing 4: Model Generation

```
1 function generate_model(ins_labelled)
2 (ins_train, ins_test) ← split randomly the instances as training and testing sets
      from ins_labelled
3 ins_extracted_train ← instances_extraction(ins_train)
4 ins_extracted_test ← instances_extraction(ins_test)
5 ranking ← GR(instances)
6 features_>0 ← discard features with rank = 0 from ranking
7 model ← MLAlgorithm(ins_extracted_train with features features_>0,
      ins_extracted_test with features features_>0)
8 return model
```

V. CAHOOT Evaluation

To evaluate CAHOOT, we exploited the driving simulator MetaDrive [26]. It is a driving simulator written in Python to train a neural network for autonomous driving through Reinforcement Learning [27]. MetaDrive is able to generate infinite driving scenarios with procedural generation of maps and different traffic flows. Inside the simulator is present a pre-trained Artificial Intelligence (AI).

We modify the MetaDrive simulation workflow with the introduction of an intruder. The in vehicle communication is simulated by a set of messages made of two different Python lists: the first one contains the steering messages while the second list contains the throttle/brake messages sent. Both lists represent messages sent by the intruder and the driver. The intrusion workflow for each step of the simulation works as follows: 1) While the driver sends the inputs, an intruder forges fake messages of steering wheel and throttle/brake; 2) The steering wheel and the throttle/brake messages of the intruder and the driver are sent to the set of messages; 3) CAHOOT reads from the set the messages and establishes which ones are the legit messages and which ones are the hacked messages; 4) Steering wheel and throttle/brake messages from the set are transmitted to the wheels and the vehicle component responsible for applying the throttle/brake; 5) The set of messages is emptied and ready to be filled with messages from the next step.

Keep note that even if in the intrusion workflow are present both the messages forged by the intruder and the messages legit, CAHOOT do not need both legit and forged messages for the detection phase. In case the intruder stops forging messages, CAHOOT would receive only the legit messages and establishes their legitimacy.

The dataset generated using the MetaDrive simulator contains the features in Table III.

A. Machine Learning algorithms

The CAHOOT paradigm is implemented by using several Python libraries to implement different ML algorithms. We test Random Forest and Neural Network Multi-Layer Perceptron (MLP). Random Forest do not require any settings of parameters. Even with the default ones, the performance obtained by these methods could be satisfactory. However, MLP requires that some parameters must be set and fine-tuned to obtain the best results, e.g., the architecture of the layers, the number of batches and so on.

We normalize training and test set to speed up the model training process using the z-score normalization [28] procedure. To improve the neural network performance, we use the embeddings for categorical values as explained in [29].

TABLE III: Features description

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/second/millisecond 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

Categorical values are the engine runtime milliseconds, engine runtime seconds and the engine runtime minutes. The remaining features are continuous. We then create data loaders for training set and test set with batches of size equal to 2048.

The architecture of the MLP contains 4 layers and the sizes of the hidden layers are respectively 2048, 1024 and 512. We then search the best learning rate using the algorithm LRFinder present in FastAI [30]. Finally, we use this learning rate in the model training. We trained the model for 480 epochs.

B. Experiments setup

The experiments run on a Virtual Machine with an Intel(R) Xeon(R) using 16 threads, 157 GB of RAM and CentOS Linux 7 as OS. To evaluate CAHOOT, in the experiments we use several metrics: *Accuracy*, *Precision* and *Recall*. *Accuracy* represents how often the model is making a correct prediction.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

TP (True Positive) is the number of instances where at least one sensor's value is hacked 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 hacked but incorrectly predicted.

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

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

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

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

We randomly split the dataset in a training set of 85% of instances and a test set of the remaining 15%. We have fed each ML method with the same training set and tested with the same test set. The dataset contains drivings made by an AI and 5 human drivers using a Thrustmaster TMX [31]. In the dataset are present 107 driving sessions made by humans. To demonstrate the validity of CAHOOT, we also simulated further human drivings using data augmentation techniques. Data augmentation are methods to generate synthetic patterns starting from a dataset [32].

TABLE IV: Features selected by CAHOOT (percentage of each rank w.r.t. the sum of the ranks of the features)

Features	Rank percentage	
	Train Human and AI	Train Human
$steering$	46,7%	43,0%
$throttle\_brake$	32,4%	37,3%
$speed$	7,4%	7,3%
yaw_rate	6,6%	5,6%
$fuel\_consumption$	2,3%	2,0%
$last\_position\_y$	1,3%	1,2%
$last\_position\_x$	0,9%	0,9%
$engine\_runtime\_minute$	0,5%	0,2%
$engine\_runtime\_second$	0,5%	0,5%
$dist\_to\_left\_side$	0,4%	1,1%
$project\_distance\_to\_vehicle\_1\_y$	0,3%	-
$dist\_to\_right\_side$	0,2%	0,5%
$project\_velocity\_to\_vehicle\_0\_y$	0,2%	0,3%

While the driver is driving the simulated vehicle, the intruder sends $steering$ and $throttle\_brake$ messages. We decided to simulate attacks with several success rates, i.e., 0%, 20% and 40%. Also, to simulate multiple attacks on each driving session, we set the maximum and duration of an attack respectively to 2 and 1 slots.

We aim to detect the instances that contain at least one sensor's value hacked from the $steering$ and the $throttle\_brake$.

C. Evaluation without data augmentation

In the following, we first evaluate CAHOOT training it by using the human and AI driving sessions. Then, the training is done by using only human driving sessions. Table IV contains the list of features selected by CAHOOT. To better distinguish features rankings, each feature rank is shown as a percentage of the sum of all the ranks.

The training CAHOOT using human and AI drivings leverages the $steering$ and $throttle\_brake$ messages. 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.

The MLP is trained with a learning rate of 0,00023.

In Table V, we make a comparison among Random Forest and MLP. When CAHOOT is trained using human and AI drivings, the table shows that Random Forest (Table V (a)) obtained the best accuracy while MLP (Table V (b)) is the most balanced model, obtaining similar Precision and Recall.

To better understand on which circumstances Random Forest best performs, we calculated the accuracy grouped by

entity, i.e., human or the AI is driving the car, and by type of attack, i.e., DoS, spoofing and replay. Table V (a) shows that the model has difficulty in the identification of the instances where the AI drives the car. On the other hand, the model has an excellent accuracy on instances where the human is driving. AI makes continuous and sudden driving adjustments, whereas humans tend to make gradual changes. Graduality makes human drivings predictions more accurate. The most difficult type of attack to identify is the replay attack while the spoofing is the most easiest to identify.

Because on human drivings the algorithm obtains high accuracy, we tried to improve the results by training and testing using only drivings made by humans.

Table IV contains the list of features selected by CAHOOT. As before, the first six highest ranked features are $steering$, $throttle\_brake$, $speed$, yaw_rate, $fuel\_consumption$ and $last\_position\_y$. However, engine runtimes seconds and minutes are no longer at the half of the table and ranking fourth to last and last respectively. Also, the projection with the nearest vehicles obtained a rank equal to zero, except for the projection of velocity to the nearest vehicle in the y axis.

The MLP is trained with a learning rate of 0,00016 obtained using the LRFinder algorithm. Table V shows that Random Forest obtained the best accuracy. Testing only the human drivings, the model trained with both human and AI drivings obtains slightly better accuracy w.r.t. the model trained using only human drivings. The AI reacts almost instantly to intrusions making it the ideal driver. Thus, the model trained with also AI drivings is better at detecting legitimate messages. Moreover, Table V (a) shows that replay attack is the most difficult to recognize, while spoofing attack is the simplest to recognise with a Recall nearly perfect, i.e., 99,78%.

D. Evaluation with data augmentation

There are several data augmentation techniques on literature [32]. However, some techniques may produce dataset not realistic. For example, jittering, i.e. the application of noise to the dataset, may produce driving sessions in which the driver never comes to a complete stop at stop signs. To synthesize additional human driving sessions, we use data augmentation techniques which guarantee that at each driving session the fuel consumed by a vehicle since the start of the driving session can not decrease over time, i.e., at the i-th instance of the session $fuel\_consumption[i] \geq fuel\_consumption[i-1]$. Hence, we simulate additional human driving sessions using two data augmentation techniques: *time warping* uses a cubic spline that stretches or contracts the temporal dimension of the driving session [33], *window warping* stretches by 2 or contracts by $\frac{1}{2}$ a random window of the time series [34].

Hereafter, the procedure to generate the augmented test set.

1) Preparation of the dataset for Data Augmentation: because we want to augment human drivings to generate new synthetic human drivings, every AI drivings from the entire dataset are discarded. Either the training set and the test set contain hacked messages of $steering$ and $throttle\_brake$. We do not need to augment the hacked messages because we can simply randomly generate new hacked messages. Hence, the hacked messages are discarded and the resulting dataset saved in $dataset\_legit$. New hacked messages of the augmented data will subsequently be randomly generated. The data augmentation methods that will be used contract and/or stretches the time. Hence, even the features which represent the engine runtime will be consequently altered. The stretching and contracting is made by the data augmentation to generate new driving sessions that represent respectively a longer and smaller driving route in a time frame equal to the time before the data augmentation occurs. The engine runtime stretched and contracted will report to the machine learning method these time changes defeating the purpose of the data augmentation in the first place. To address this issue, the original engine runtime features are stored in a separate array and subsequently be used for the augmented datasets. Finally, the function returns the $dataset\_legit$, the array of engine runtimes and the index of the driving sessions.

2) Augmentation of the dataset: the data augmentation method $augmented\_function$ will be applied to the original dataset, i.e., either training and test set. The augmented training set will be used to simulate the replay attacks, but will not be used to train the model. First, an array with the augmented datasets is created. To increase further the dataset, each data augmentation method can be performed multiple times. At each repetition, an array $dataset\_augmented$ that will contain the dataset augmented is created. Then, at each driving session is applied the $augmented\_function$ and the augmented driving session is appended to $dataset\_augmented$. Next, the augmented engine runtimes are substituted with the original engine runtimes. The augmented dataset is added to the array of augmented datasets. Once the augmented function is repeated $repeat$ times, the array of augmented datasets is returned.

3) Insert of hacked messages: the augmented instances that are part of the test set need hacked messages. Hence, we will create a test set from each augmented dataset on which the instances have attached new hacked messages. First, the array that will contains the augmented test set is created. To populate this array, the procedure must pick from each augmented dataset in $datasets\_augmented$ the corresponding augmented instance of each test set instance. Note that the procedure use instances from the test set of human drivings only. Then, the index of the augmented instance is obtained and passed to the function *generate_intrusion* responsible for the generation of new hacked messages for the augmented instance. The function is explained later on. Then, an $instance\_augmented\_attacked$ is defined and contain the augmented values alongside with the hacked messages. Hence, $instance\_augmented\_attacked$ can be now appended to the augmented test set. Once all the augmented test set instances are appended, the original test set is appended to the augmented test set.

The last function to explain is *generate_intrusion*. The function randomly generates hacked messages for the augmented test set instance. First, an attack between DoS, replay and

TABLE V: Accuracy, precision and recall comparison of CAHOOT using different ML methods

Random Forest					
Acc Train Human and AI drivers	Prec	Rec	Acc Train Human drivers	Prec	Rec
95,50%	95,98%	97,87%	97,03%	97,30%	98,60%
Test only human drivers					
97,25%	97,57%	98,64%		N/A	
Test only AI drivers					
82,70%	85,54%	92,46%		N/A	
Test only Replay attack					
93,36%	95,34%	95,46%	95,49%	96,69%	97,05%
Test only DoS attack					
96,26%	95,83%	98,90%	97,15%	97,15%	98,74%
Test only Spoofing attack					
96,73%	96,62%	99,11%	98,24%	97,91%	99,78%

(a) Results on Random Forest

MLP					
Acc Train Human and AI drivers	Prec	Rec	Acc Train Human drivers	Prec	Rec
93,81%	95,74%	95,70%	95,30%	96,84%	96,63%
Test only human drivers					
95,73%	97,23%	96,83%		N/A	
Test only AI drivers					
79,86%	85,60%	87,80%		N/A	
Test only Replay attack					
90,32%	94,59%	91,83%	92,14%	95,55%	93,42%
Test only DoS attack					
94,70%	95,57%	96,85%	95,58%	96,68%	96,87%
Test only Spoofing attack					
96,12%	96,79%	98,07%	97,77%	97,97%	99,08%

(b) Results on MLP

spoofing attacks is randomly chosen. Then, the chosen attack is launched. In particular in the spoofing attack, the attack is launched obtaining a random steering and a random throttle_brake values. In case the attack is Replay attack, the arrays with driver's history of previous steering and throttle_brake values must be built. The procedure determines the index start of the current driving session based on the variable $index$, i.e., the index of the augmented test instance. The previous augmented instances for the current driving session are the instances starting from the instance with the index $start\_index$ and ending with the instance that has index $index$ minus 1. Then, the procedure collects the previous steering and throttle_brake values of the augmented instances for the current driving session. Next, the procedure execute the function "replay_attack" and returns the result. The function choose randomly an instance index from the current driving session.

We evaluate CAHOOT using a test set augmented by 3x, 5x, 7x and 9x, i.e., the test set is made by the original test set and the augmented test sets using the window and time warp methods repeated respectively 1 time, 2 times, 3 times and 4 times. The number of human driving session presents in the test sets augmented by 3x, 5x, 7x and 9x are respectively 321, 535, 749 and 963 human driving sessions. We use Random Forest as ML algorithm because it obtained the best accuracy in all the previous tests.

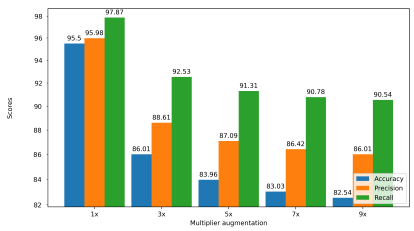

Fig. 1: Comparison of Attack Identification test bed with test set augmented.

In the first experiment, CAHOOT is trained using the human drivings and the AI drivings (Figure 1). The bar plot shows that the test set without data augmentation, i.e., 1x, loose 9,49% of accuracy with the respect to the test set augmented 3x. The use of data augmentation amplify noises present on the dataset which leads to a deterioration in identification. On the other hand, the accuracies, the precisions and recalls of the 5x, 7x and 9x are similar. Hence, CAHOOT's accuracy degradation is less affected by noise as the dataset grows. The attack type that obtained the lowest accuracy is replay attack with a minimum accuracy of 75,08% and a maximum of 79,92%. The attack type with the highest accuracy is spoofing attack with an accuracy that ranges between 86,72% and 89,39%.

In the last experiment, CAHOOT is trained and tested using only the human drivings (Figure 2).

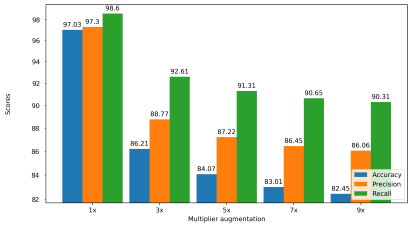

Fig. 2: Comparison of Attack Identification test bed trained using only human drivings with test set augmented.

The bar plot shows that the test set without data augmentation, i.e., 1x, loose 10,82% of accuracy with the respect to the test set augmented 3x. Accuracy, the precision, and recall measures of the 5x, 7x and 9x are similar as previously seen in the previous experiments. The attack type that obtained the lowest accuracy is replay attack with a minimum accuracy of 74,87% and a maximum of 80,26%. On the other hand, the attack with the highest accuracy is once again the spoofing attack in a range between 86,76% and 89,86%.

VI. Conclusion

In this paper, we presented CAHOOT, a context-aware IDS able to detect intrusions into a sequence of in-vehicle

messages related to a driver's driving style. We evaluated the performance of CAHOOT using several metrics. CAHOOT obtained high scores in all the configuration, especially using Random Forest. Compared respectively to the lowest and the highest score of the main context-aware IDSs, CAHOOT performed on spoofing attack the best and second best score proving its reliability. Moreover, we adopted data augmentation techniques to increase the number of human drivings to demonstrate that CAHOOT performs well with larger datasets.

As future work, we will improve CAHOOT to work on more complex scenarios and to identify how many data should be collected from the human and AI drivers. In addition, we will consider new driving scenarios, e.g., the drivers will be instructed to drive fast like being late for an appointment. Furthermore, we aim to refine the algorithm to detect which sensor of the car is being attacked.

ACKNOWLEDGMENT

The project leading to this work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 883135 (E-Corridor).

REFERENCES

[1] H. Diess, "Levers to unleash value," 1 2020, URL:https://www.volkswagenag.com/presence/investorrelation/publications/presentations/2020/01-januar/January_2020_VWAG_Investor_Roadshow.pdf [retrieved: 09, 2022].

[2] E. U. A. for Cybersecurity, *Cyber security and resilience of smart cars : good practices and recommendations*. European Network and Information Security Agency, 2017.

[3] A. Drozhzhin, "Black hat usa 2015: The full story of how that jeep was hacked," 8 2015, URL:https://usa.kaspersky.com/blog/blackhat-jeep-cherokee-hack-explained/5749/ [retrieved: 09, 2022].

[4] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," 8 2015, URL:http://illmatics.com/Remote\%20Car\%20Hacking.pdf [retrieved: 09, 2022].

[5] BBC, "Fiat chrysler recalls 1.4 million cars after jeep hack," 07 2015, URL:https://www.bbc.com/news/technology-33650491 [retrieved: 09, 2022].

[6] I. O. for Standardization, "Iso/iec 27039:2015, information technology — security techniques — selection, deployment and operations of intrusion detection and prevention systems (idps)," 2 2015, URL:https://www.iso.org/standard/56889.html [retrieved: 09, 2022].

[7] O. J. of the European Union, "Uniform provisions concerning the approval of vehicles with regards to cybersecurity and cybersecurity management system," 3 2021, URL:http://data.europa.eu/eli/reg/2021/387/oj [retrieved: 09, 2022].

[8] J. Jiang, C. Wang, S. Chattopadhyay, and W. Zhang, "Road context-aware intrusion detection system for autonomous cars," *Lecture Notes in Computer Science*, p. 124–142, 2020. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-41579-2_8

[9] V. Kondratiev and A. Kuznetsov, "An algorithm for intrusion detection into the control system of an unmanned vehicle," in *2021 International Conference on Information Technology and Nanotechnology (ITNT)*, Sep. 2021, pp. 1–5.

[10] A. Wasicek, M. Pesé, A. Weimerskirch, Y. Burakova, and K. Singh, "Context-aware intrusion detection in automotive control systems," in *Proc. 5th ESCAR*, 06 2017, p. 1–14.

[11] M. Casillo, S. Coppola, M. De Santo, F. Pascale, and E. Santonicola, "Embedded intrusion detection system for detecting attacks over can-bus," in *2019 4th International Conference on System Reliability and Safety (ICSRS)*, Nov 2019, pp. 136–141.

[12] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[13] R. Rieke, M. Seidemann, E. K. Talla, D. Zelle, and B. Seeger, "Behavior analysis for safety and security in automotive systems," in *Parallel, Distributed and Network-Based Processing (PDP), 2017 25nd Euromicro International Conference on*. IEEE Computer Society, Mar 2017, pp. 381–385.

[14] M. Levi, Y. Allouche, and A. Kontorovich, "Advanced analytics for connected cars cyber security," *CoRR*, vol. abs/1711.01939, 2017.

[15] S. N. Narayanan, S. Mittal, and A. Joshi, "Obd securealert: An anomaly detection system for vehicles," in *IEEE Workshop on Smart Service Systems (SmartSys 2016)*, May 2016.

[16] A. Theissler, "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.

[17] M. J. Kang and J. W. Kang, "A novel intrusion detection method using deep neural network for in-vehicle network security," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, 2016.

[18] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of id sequences," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1577–1583.

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

[20] H. K. Kalutarage, M. O. Al-Kadri, M. Cheah, and G. Madzudzo, "Context-aware anomaly detector for monitoring cyber attacks on automotive can bus," in *ACM Computer Science in Cars Symposium*, ser. CSCS '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3359999.3360496

[21] D. Grimm, M. Stang, and E. Sax, "Context-aware security for vehicles and fleets: A survey," *IEEE Access*, vol. 9, pp. 101 809–101 846, 2021.

[22] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, "Intrusion detection systems for intra-vehicle networks: A review," *IEEE Access*, vol. 7, pp. 21 266–21 289, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8642311

[23] B. Zheng, H. Liang, Q. Zhu, H. Yu, and C.-W. Lin, "Next generation automotive architecture modeling and exploration for autonomous driving," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 53–58.

[24] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 4, dec 2012. [Online]. Available: https://doi.org/10.1145/2382577.2382579

[25] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[26] Q. Li, Z. Peng, Z. Xue, Q. Zhang, and B. Zhou, "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning," *arXiv preprint arXiv:2109.12674*, 2021.

[27] Y. Li, "Deep reinforcement learning," 2018. [Online]. Available: https://arxiv.org/abs/1810.06339

[28] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*, ser. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. [Online]. Available: https://books.google.it/books?id=pQws07tdpjoC

[29] Rachel Thomas, "An introduction to deep learning for tabular data," Apr. 2018. [Online]. Available: https://www.fast.ai/2018/04/29/categorical-embeddings/

[30] L. N. Smith, "Cyclical learning rates for training neural networks," 2017.

[31] Thrustmaster, "TMX Force Feedback," URL:https://www.thrustmaster.com/products/tmx-force-feedback/ [retrieved: 09, 2022].

[32] B. K. Iwana and S. Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *PLOS ONE*, vol. 16, no. 7, pp. 1–32, 07 2021. [Online]. Available: https://doi.org/10.1371/journal.pone.0254841

[33] T. T. Um, F. M. J. Pfister, D. Pichler, S. Endo, M. Lang, S. Hirche, U. Fietzek, and D. Kulic, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks," *CoRR*, vol. abs/1706.00527, 2017. [Online]. Available: http://arxiv.org/abs/1706.00527

[34] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data Augmentation for Time Series Classification using Convolutional Neural Networks," in *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, Riva Del Garda, Italy, Sep. 2016. [Online]. Available: https://halshs.archives-ouvertes.fr/halshs-01357973