Real-Time Evasion Detection in Tree Ensemble Automotive Intrusion Detection Systems

Valency Oscar Colaco
Linköping University, Sweden
valency.colaco@liu.se

Simin Nadjm-Tehrani
Linköping University, Sweden
simin.nadjm-tehrani@liu.se

***Abstract*—Safety-critical functions in modern vehicles rely on electronic control units that communicate using the controller area network (CAN) protocol, which lacks vital security features. In this context, machine learning (ML) based intrusion detection systems (IDSs) were proposed as a solution to improve cyber resilience through real-time attack detection. However, these ML-IDSs must also withstand evasion attacks that could compromise vehicular safety. To this end, this paper addresses such attacks in misuse-based tree ensemble IDSs and proposes a method that detects evasion attempts. It uses the ordered set of reached leaf nodes activated by correctly classified training samples as a normality baseline. An autoencoder-based detector then identifies deviations as likely evasion attempts. Our approach does not modify the protected tree ensemble IDS, assumes no knowledge of the process for generating adversarial examples (ensuring generalisability), and works with any additive tree ensemble. We also prove that it is mathematically equivalent to the state-of-the-art, which we advance in terms of detection speed by replacing its Hamming distance-based deviation search with an autoencoder-based model of typical predictive behavior trained using our custom loss function. This enhancement results in a detection process that is orders of magnitude faster. Additionally, our method offers nuanced insights regarding the pre-evasion attack signature prior to the adversarial perturbation, thereby enriching the security analysis of the features targeted during evasion attempts. The prototype system we present, called *Maverick*, has a very low prediction latency, making it 85-563x faster than the current state-of-the-art while maintaining identical detection accuracy. Finally, *Maverick* predicts the pre-evasion attack signatures of the evasion samples with an accuracy of more than 93% and has an average prediction time well below the message transmission rate for CAN 2.0 and CAN FD, thereby satisfying the criteria for an evasion-hardened & real-time automotive IDS.**

I. Introduction

Modern automotive functions are controlled using a large number of electronic control units (ECUs) that communicate using the controller area network (CAN), which is the current standard for in-vehicular communications [21]. However, the CAN protocol lacks built-in security features, making it vulnerable to adversaries [21]. While solutions based on cryptography and encryption have been proposed to address some issues [23], their adoption has been limited as they only provide partial protection. As a result, they are often supplemented with Intrusion Detection Systems (IDSs) [7]. Furthermore, in this domain, an IDS must meet an additional requirement beyond detecting attacks: it must operate in real-time [26] due to the time-sensitive nature of automotive functions [22]. This real-time detection is essential to mitigate attacks as they occur, preventing potential harm to vehicle occupants and other road users. For instance, a DoS attack could overwhelm the CAN bus, delaying or blocking safety-critical functions like braking. Real-time detection allows for deploying swift countermeasures to ensure the vehicle's operational integrity.

While deep learning techniques like LSTMs or CNNs have been used for CAN-based intrusion detection tasks, these approaches often lack interpretability. To this end, tree ensembles emerge as a compelling alternative due to their interpretable (human-understandable) structures, making them particularly suitable for use in safety-critical applications where trust in the model's behavior is paramount [25]. Moreover, tree ensembles frequently outperform deep learning techniques like CNNs and LSTMs on intrusion detection tasks [27]–[30]. However, despite their proven effectiveness, they must also withstand evasion attacks through, for example, detection or prevention [28]. Therefore, we propose a method to detect evasion attacks in a misuse-based tree ensemble automotive IDS. Our method uses the ordered set of reached leaf nodes in the tree ensemble activated by the incoming sample, referred to as the Output Configuration (OC) [3]. Our work is based on the hypothesis proposed by Devos et al. [3] that adversarial examples produce relatively different OCs in contrast to those produced by normal samples when compared against the OCs generated using correctly classified training examples (referred to as the *reference set*). Note that we denote *normal* or *non-adversarial samples* as those belonging to either the attack or benign classes as long as an attacker has not perturbed them. Once an attacker perturbs a normal sample to evade detection (attack $\rightarrow$ benign) or to raise a false alarm (benign $\rightarrow$ attack), a large number of which could impact IDS availability, we deem such a sample as an *adversarial example* or simply, an *adversarial sample*.

The OC-Score method proposed by Devos et al. [3] uses a Hamming distance-based search across a reference set to detect evasions. While OC-Score is the current state-of-the-art (SOTA) in evasion detection, it has a critical flaw - extremely long prediction times (due to its exact search), making it unsuitable for real-time detection. Our initial attempts to address this by replacing the exact search with an approximate nearest neighbour (ANN) search method, such as Google's ScaNN, Meta's FAISS, or Spotify's ANNOY, yield unacceptable false positive rates; therefore, we use principles from anomaly detection, defining a custom loss function and utilizing the

reference set as a normality baseline to develop an autoencoder-based evasion detector for identifying evasion attempts. This enhancement results in a prediction time much lower than the transmission time per packet on the CAN bus, thereby enabling real-time detection. Our approach is orders of magnitude faster than OC-Score and works for any additive tree ensemble. It also does not modify the protected IDS and assumes no knowledge about the process for generating adversarial examples (so it generalises well). Finally, by leveraging manifold learning with an ensemble of small autoencoders, our method can also accurately predict the pre-evasion attack signatures (before the adversarial perturbation or manipulation) for incoming adversarial examples, providing evasion-informative insights to enrich the understanding of the specific features targeted by attackers in the evasion process. The contributions of this work are summarized as follows:

- We present a method that:
  - leverages the set of reached leaf nodes to detect evasion attacks in real-time for automotive tree ensemble IDSs.
  - uses manifold learning to predict the pre-evasion attack signatures of the evasions in order to provide nuanced insights into the attacker's perturbation strategy.
- We present *Maverick*, a prototype system that realizes an evasion-hardened and real-time IDS that maintains the decision accuracy of the original tree ensemble IDS.
- We demonstrate *Maverick's* effectiveness using two real-world case studies that highlight its real-time detection capabilities, achieving detection speeds that represent an 85-563x fold increase compared to the SOTA.

The remainder of this paper is structured as follows. Section II compares this paper with related works. Section III presents the background knowledge. Section IV presents the threat model. Section V presents the proposed method. Section VI presents the experimental evaluations. Section VII presents the discussions, and Section VIII concludes this paper.

II. Related Works

Evasion attacks on automotive intrusion detection systems (IDSs) have become a growing concern for cybersecurity due to their direct impacts on vehicular safety [21]. Stefano et al. [22] assess the robustness of automotive IDSs against evasion attacks and assert that adversaries can significantly impair their detection capabilities. Chen et al. [15] propose a booster-fixer training framework for enforcing security properties in classifiers. Their approach suffers from the issue of scalability and follows a counterexample-guided inductive synthesis loop in which counterexamples generated during formal verification are used to optimize the models until the security property is eventually satisfied. Apruzzese et al. [17] propose an ensemble of application-specific IDSs called APPCON to defend organisational networks, where real-time detection is generally not a critical requirement. Vos et al. [18] propose a new class of robust tree ensembles called GROOT to detect evasion attacks. In another paper [19], they propose a robust relabeling scheme based on computing the minimum vertex cover of a bipartite graph, which optimally changes the prediction labels of decision tree leaves to maximize adversarial robustness. Lecuyer et al. [20] propose a certified defence based on differential privacy to detect adversarial examples. All of these approaches offer incremental detection improvements and are superseded by OC-Score, which we compare against in the next paragraph.

A different category of methods in the literature use an analysis of the internal prediction mechanisms within machine learning models to detect adversarial examples. Sperl et al. [16] highlight that the dense layers of deep neural networks carry security-sensitive information. They use this information to train a secondary neural network to detect adversarial examples in real-time. Devos et al. [3] propose a method called OC-Score to detect evasion attacks in tree ensembles by analyzing the set of leaves activated by the adversarial example in the ensemble's constituent trees. While OC-Score uses a Hamming distance-based approach to detect evasion attacks, it suffers from the critical limitation of high prediction times (due to its exhaustive search across a reference set), making it unsuitable for use in the automotive domain. To address this issue, our prototype, *Maverick*, replaces OC-Score's Hamming distance-based search mechanism with an autoencoder-based detector (trained using our custom loss function) to identify evasion attempts. This enhancement significantly reduces prediction times by several orders of magnitude while maintaining OC-Score's superior detection performance. Additionally, by leveraging small autoencoders in a class-wise ensemble, *Maverick* can accurately predict the malicious signature that the attacker(s) perturbed to evade detection. In this paper, we consider OC-Score to be the most relevant for comparison with our work and use it in our evaluation in section VI.

III. Background

In this section, we present the background knowledge on tree ensembles, the OCs, autoencoders, CAN, and the AUTomotive Open System ARchitecture (AUTOSAR) IDS Protocol.

A. Tree Ensembles

Tree ensembles use a collection of decision trees to make predictions. Let F be a tree ensemble consisting of N decision trees, as $F = \{T_1, ..., T_N\}$. The prediction function (f) of the tree ensemble is then defined as,

$$f(x) = p\left(\sum_{i=1}^{N} t_i(x)\right)$$

Where p is the post-processing function (e.g., softmax) and t_i is the prediction function implemented by tree T_i.

B. Output Configuration (OC)

Let F be a tree ensemble consisting of N individual decision trees, as $F = \{T_1, ..., T_N\}$. The Output Configuration (OC) of a sample x in an ensemble F is the ordered set of leaves in each of the trees visited due to x, defined as,

$$OC(x) = \langle l^1, ..., l^N \rangle$$

Where l^N are leaf identifiers.

C. Autoencoders

An autoencoder is an artificial neural network which is trained to reconstruct its input by learning the identity function of the original data distribution. In general, an autoencoder trained on data from distribution D gains the capability to reconstruct unseen instances from the same distribution. If an incoming instance does not belong to the concepts learned from D, then we expect a high reconstruction error [5].

D. Controller Area Network (CAN)

The Controller Area Network (CAN) is an automotive bus technology for communication within a vehicle [8]. Multiple CAN protocol variants exist that support different transmission speeds and payload sizes [9]. CAN 2.0 supports two transmission speeds [10] - High Speed (ISO 11898-2, up to 1 Mbit/s), and Low Speed (ISO 11898-3, up to 125 Kbit/s) with an average message size of 112 bits. CAN-FD (ISO 11898-1) supports transmission rates up to 5 Mbit/s with an average message size of 560 bits. Note that the average message size was calculated after considering minimal bit stuffing. Based on this information, the transmission rate for CAN Low Speed works out to 1,116 packets per second, and that of CAN High Speed and CAN-FD work out to 8,929 packets per second.

E. The AUTOSAR IDS Protocol

AUTOSAR is a global partnership between automotive stakeholders for the development of automotive software. The AUTOSAR R20-11 standard [11] sets out the specification of an automotive IDS protocol which is described in figure 1.

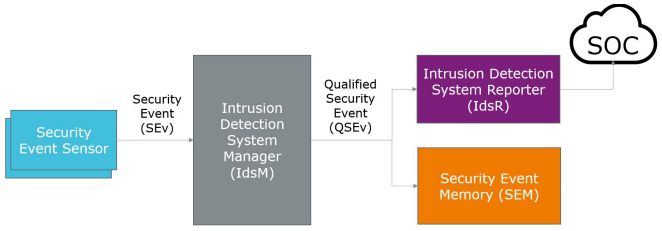

Fig. 1: Structure of an Automotive IDS [12]

An automotive IDS basically consists of four parts - the Security Event Sensor, the Intrusion Detection System Manager (IdsM), the Intrusion Detection System Reporter (IdsR), and the Security Event Memory (SEM). The security sensors are software algorithms that detect a potential security event (SEv) and report it to the IdsM. At the IdsM, the SEv then passes through qualification filters, and becomes a Qualified Security Event (QSEv) when certain criteria are met. This QSEv is then passed to the IdsR which forwards it to the vehicle manufacturer's security operating center (SoC) which deploys appropriate countermeasures. The SEM enables the secure storage of security events.

IV. Threat Model

We adopt a threat model that is consistent with Apruzzese et al. [1] and Biggio et al. [2]. According to Biggio et al., the threat model can be represented using an attacker's goal, knowledge, capability, and strategy. In this paper, the threat model is specific to attacks against the automotive IDS, thus indirectly threatening the underlying network flows and the vehicular functions. Regarding the underlying system, we assume the attacker goal to be safety violations like uncontrolled braking, acceleration, or steering wheel lock-ups.

Regarding the attacks against the automotive IDS, we assume the attacker goals to be integrity violations that happen when the core IDS functionality of "detecting attacks" is tampered with. We assume a grey box level of knowledge where an attacker has knowledge about the type of detection model along with the feature set. We also assume the attacker has compromised one of the ECUs in the vehicle as in Zenden et al. [7], enabling them to collect CAN messages and train a surrogate model. Finally, since the attacker strategy is to perform evasion attacks, we assume that the attacker is capable of performing indiscriminate exploratory integrity attacks using the surrogate model at the raw traffic level, also known as the problem space [1].

V. Proposed System

In this section, we present ***Maverick***, our prototype system, along with its architecture, components, and workflow. We also describe how *Maverick* can be embedded into an automotive networking system in line with global standards.

A. Theoretical Analysis

We use the hypothesis proposed by Devos et al. [3] that tree ensembles produce relatively different OCs for normal samples in contrast to adversarial examples when compared to a reference set, as follows:

1) Notations, Assumptions, and Definitions:

- **Assumption 1** (Tree Ensemble & Training): We assume a tree ensemble of N decision trees $T_1, ..., T_N$, with a prediction function f. We assume that this tree ensemble is trained using labelled samples $(x, y) \in X_{train}$ where x is the sample/vector and y is the label.
- We denote x_n as a normal (or unperturbed) vector, and x_a as an adversarial vector crafted to evade detection.
- **Definition 1** (Output Configuration or OC): We recall that in a tree ensemble of N decision trees $T_1, ..., T_N$, the OC of a sample x is the ordered set of leaves visited by x in each tree of the ensemble, defined as,

$$OC(x) = \langle l^1, ..., l^N \rangle$$

  Where l^N are leaf identifiers, and $l^N \in \mathbb{N}$.
- We denote the reference set, which is a set of OCs constructed using correctly classified training samples (both attack and benign) as $R = \{OC(x) : (x, y) \in X_{train}, f(x) = y\}$.
- **Definition 2** (Hamming Distance): Given two vectors u and v, the Hamming distance between them, denoted by $h(u, v)$ is defined as the number of places where u and v differ.

$$h(u, v) = \sum_{i=1}^{n} \mathbb{1}_{u_i \neq v_i}$$

- **Lemma 1** [3] (Samples & OCs): Given a normal sample x_n and its output configuration, $o_n = OC(x_n)$ and an adversarial example x_a along with its output configuration, $o_a = OC(x_a)$, we know that,

$$\min_{o_{nn} \in R} h(o_n, o_{nn}) < \min_{o_{an} \in R} h(o_a, o_{an})$$

Where o_{nn} and o_{an} are the OCs *closest* (in terms of Hamming distance) to o_n and o_a respectively.

- **Definition 3** (Clipped Root Mean Square Error or CRMSE): The clipped root mean square error is measured using the square root of the average squared clipped absolute difference between the corresponding elements of two vectors u and v, defined as,

$$\text{CRMSE}(u, v, a, b) = \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n} min(max(|u_i - v_i|, a), b)\right)^2}$$

Where n is the dimensionality of these vectors. a and b are the lower and upper bounds of the clipping operation, also known as the clipping co-efficients.

Now that we have explained our notations, assumptions, and definitions, we can prove our theoretical assertion.

2) Equivalence of Leaf-Based Distances: We now describe the principle idea behind our evasion detector as follows:

Theorem 1: The CRMSE for normal OCs is lower than that of adversarial OCs when compared to their respective closest OCs in the reference set with the clipping co-efficients set to 0 and 1.

More specifically, given normal and adversarial examples x_n, x_a, along with their output configurations $o_n = OC(x_n)$, $o_a = OC(x_a)$, and their closest OCs $o_{nn}, o_{an} \in R$ according to lemma 1 then,

$$\text{CRMSE}(o_n, o_{nn}, 0, 1) < \text{CRMSE}(o_a, o_{an}, 0, 1)$$

Or simply,

$$\text{CRMSE}(o_n, o_{nn}) < \text{CRMSE}(o_a, o_{an})$$

Proof. From Lemma 1, we know that,

$$h(o_n, o_{nn}) < h(o_a, o_{an})$$

From definition 2, we know that the Hamming distance between two vectors is the number of places where the vector elements differ as,

$$|\{i : o_{n,i} \neq o_{nn,i}\}| < |\{i : o_{a,i} \neq o_{an,i}\}|$$

Where $|\cdot|$ is the set cardinality. For integer vectors, the above expression can be written as,

$$\sum_{i=1}^{n} min(max(|o_{n,i} - o_{nn,i}|, 0), 1) < \sum_{i=1}^{n} min(max(|o_{a,i} - o_{an,i}|, 0), 1)$$

Where $|\cdot|$ is the absolute difference between the OC elements. Since both sides of the expression are non-negative, we can square them while preserving the inequality relationship as,

$$\left(\sum_{i=1}^{n} min(max(|o_{n,i} - o_{nn,i}|, 0), 1)\right)^2 < \left(\sum_{i=1}^{n} min(max(|o_{a,i} - o_{an,i}|, 0), 1)\right)^2$$

Dividing both sides by n and taking the square root,

$$\sqrt{\frac{1}{n}\left(\sum_{i=1}^{n} min(max(|o_{n,i} - o_{nn,i}|, 0), 1)\right)^2} < \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n} min(max(|o_{a,i} - o_{an,i}|, 0), 1)\right)^2}$$

From definition 3, the above expression can be written as,

$$\text{CRMSE}(o_n, o_{nn}, 0, 1) < \text{CRMSE}(o_a, o_{an}, 0, 1)$$

Or simply,

$$\text{CRMSE}(o_n, o_{nn}) < \text{CRMSE}(o_a, o_{an})$$ ■

We use this theorem for the design of *Maverick*, the architecture of which will be described in the next subsection.

B. Maverick Architecture

Our method (embodied in a prototype system called *Maverick*) relies on the core principle of anomaly detection, which, in general, identifies events that deviate from some notion of normality. Note that *Maverick* detects evasions based on principles from anomaly detection, while the underlying IDS is misuse-based. If we recall from section III, autoencoders are trained to reconstruct the input x as $h_\theta(x) \approx x$ where h_θ is the hypothesis function with model parameter θ. By using the set of OCs produced using correctly classified training samples, i.e., R as our baseline for normality, we deploy an autoencoder-based evasion detector to identify deviations. These deviations come in the form of OCs with a high reconstruction error that correspond to evasion attacks. Figure 2 provides an overview of our proposed system architecture.

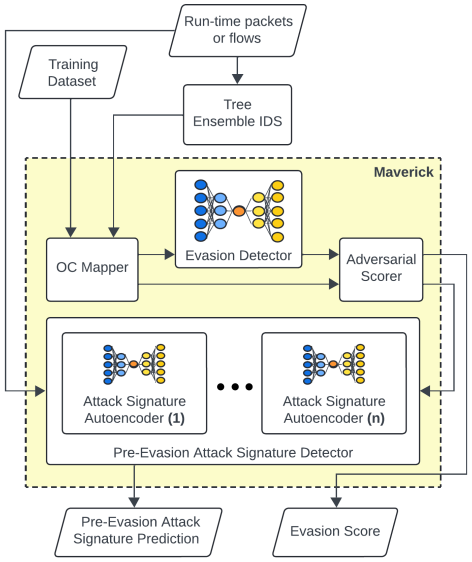

Fig. 2: *Maverick* Architecture

The OCs generated using correctly classified training samples represent typical predictive behaviour of the tree ensemble IDS. Since adversarial examples exploit unusual decision paths in tree ensembles [3] to cause intentional misclassifications, the OCs produced from such examples deviate from typical predictive behavior. This deviation is identified using the evasion detector and scored in the adversarial scorer. The output of *Maverick* is an evasion score, which can be considered as

the output from the security sensor (e.g., AUTOSAR R20-11). Alternatively, one can simply set a threshold on this score to output class predictions directly. Once the samples have been identified as likely adversarial, they are passed to the pre-evasion attack signature detector, which adds additional information pertaining to the attack signature before the adversarial perturbation. This enriches the security analysis by providing crucial insights into which features of the CAN messages are being targeted by attackers to evade detection. This way, our added components make the (initially defenseless) tree ensemble evasion-hardened and evasion-informative. We will now explain the components of *Maverick* in more detail.

C. OC Mapper

This component is responsible for generating the OCs both for the training samples, and for the runtime samples. Formally, if x an input sample, and L_i the reached leaf ID corresponding to the i^{th} tree in the ensemble then,

$$OC\text{-}mapper(x) = \langle L_1, ..., L_N \rangle$$

Where N is the number of trees in the ensemble.

D. Evasion Detector

This component is responsible for detecting OCs associated with evasion attacks. It consists of an autoencoder trained on the reference set, which serves as the baseline for typical IDS predictive behavior. We use this autoencoder-based detector to identify deviations from the baseline in the form of high reconstruction error OCs that correspond to evasion attacks. From theorem 1, we know that the CRMSE for OCs from normal samples is less than that of adversarial examples when compared to R. We leverage this result to define a custom loss function, training our autoencoder to specifically minimize the CRMSE loss on R. This way, samples that deviate from normality (learned using R) can be flagged as likely adversarial. If we consider A as our autoencoder reconstruction function, o as an input OC, and o_r as our reconstructed OC, then, $A(o) \approx o_r$. This re-constructed OC is then passed on to the adversarial scorer where an evasion score is computed.

E. Adversarial Scorer

This component is responsible for generating an evasion score that quantifies the adversarial nature of the input sample. If $o = OC\text{-}Mapper(x)$ is an input OC, and $o_r = A(o)$ is a reconstructed OC, then the evasion score, $\alpha = \text{CRMSE}(o, o_r)$. This score can be thresholded and converted into class predictions, or it can be attached to the corresponding security event (e.g., the SEv in the AUTOSAR R20-11 protocol before it makes its way to the IdsM qualification filters).

F. Pre-Evasion Attack Signature Detector

Once the incoming sample is identified as likely adversarial in the adversarial scorer, it is passed to the pre-evasion attack signature detector, which is responsible for predicting the attack signature prior to the perturbation. This component consists of an ensemble of small autoencoders, each trained on specific attack signatures in the training dataset. The idea is that each autoencoder learns the manifold of a specific attack signature, and when adversarial samples are passed through the ensemble, the attack signature with the lowest reconstruction error is returned. This approach works because adversarial examples of a given class, even after being perturbed into another class, still retain significant features of their original manifold, thereby returning a lower reconstruction error.

G. Workflow

In this subsection, we present the workflow of our proposed method. For ease of understanding, the workflow is split into two phases: pre-deployment and post-deployment. Figure 3 shows our proposed workflow in action.

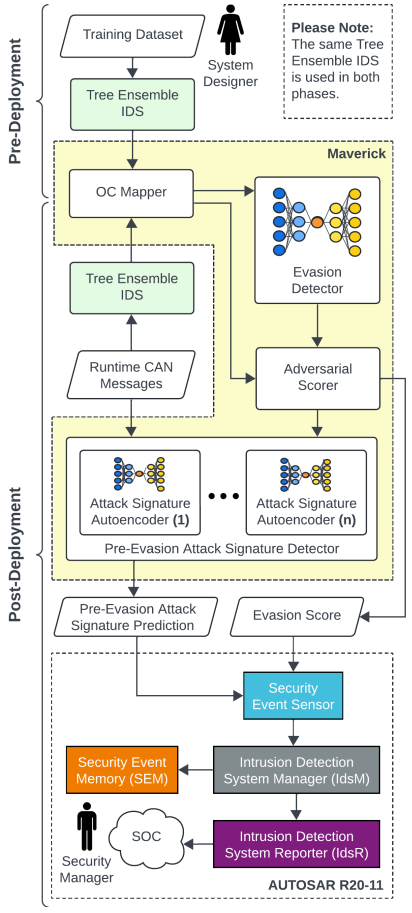

Fig. 3: *Maverick* Workflow in an Automotive Setting

In the pre-deployment phase, the system designer constructs the reference set using OCs of correctly classified training examples. This reference set is used to train the autoencoder-based evasion detector. In the post-deployment phase, the runtime samples are mapped to OCs using the OC Mapper. These samples are then passed to the evasion detector and scored using the adversarial scorer. Once the sample is identified as likely adversarial, additional information about the pre-evasion attack signature is added to the security event. The final evasion score and the security event are then sent to the IdsM & IdsR that forward it to the vehicle manufacturer's SOC in line with the AUTOSAR R20-11 IDS Protocol.

VI. Experimental Evaluation

We present the experimental evaluation of *Maverick* in two real-world case studies, and compare our method to the current state-of-the-art method in evasion detection, i.e., OC-Score [3]. Our evaluation addresses three questions:

Q1. Can *Maverick* detect adversarial examples with a similar or better accuracy than OC-Score?
Q2. What is each approach's average prediction time associated with detecting adversarial examples?
Q3. How accurately can *Maverick* discover the source of attacker perturbations to understand evasion strategies?

In our experiments, we use VoTE [6], a formal verification engine for tree ensembles based on abstract interpretation as an oracle to generate adversarial examples. We use the scikit-learn [13] implementation for Random Forests and the DMLC implementation [14] for XGBoost Gradient Boosting Machines (GBM). All experiments are conducted on a Windows 11 Machine running Ubuntu 20.04 in Windows Subsystem for Linux mode. The machine comes equipped with an Intel Core i7-10875H CPU and 16 GB RAM. Due to space constraints, additional experimental settings are not shown here. However, these settings, along with the code and data files for the experimental outcomes presented in the tables and charts in this section, are freely available for future repeatability at https://github.com/va-co/maverick.

A. Datasets

In this work, we use two datasets, namely CIC-IoV-2024 [4] developed by the Canadian Institute for Cybersecurity (CIC), and the HCRL-CAN-FD[1] developed by the Hacking and Countermeasure Research Lab (HCRL) in Korea. The first dataset includes real CAN 2.0 data from a 2019 Ford car and focuses on spoofing & DoS attacks. The second dataset includes real CAN-FD data from a 2021 Genesis G80 car and focuses on flooding, fuzzing & malfunction attacks. Note that the proportion of benign to attack samples within both these datasets is heavily imbalanced in favor of the benign class, mimicking real-world conditions where benign traffic vastly outnumbers attack traffic.

B. Experimental Setup (CIC-IoV-2024)

We pre-process this dataset as follows. The *category* and *specific_class* columns are dropped as these are typically used for multi-class classification problems. In the *label* column, we map attack and benign to 1 and 0 respectively. We use 5-fold cross-validation in all our experiments for a robust evaluation. For each iteration, the dataset is divided into five folds: the first three folds are used to train a random forest IDS with 254 trees, depth 10, while the fourth fold is used to train a surrogate model that generates adversarial examples using VoTE. The fifth fold, designated as the test set, is used exclusively for evaluation. In subsequent iterations, these folds are shuffled cyclically, and the process is repeated four more times. The results are then summarized. To prevent any experimental bias, the evaluation was conducted using an equal number of randomly selected test samples and adversarial examples.

Regarding the spoofing attack, adversarial examples were generated by only perturbing the data bits (as this attack typically targets a specific CAN arbitration ID). In the DoS attack, both the CAN arbitration ID & data bits are perturbed.

C. Experimental Setup (HCRL-CAN-FD)

We pre-process this dataset as follows. All the hexadecimal values are converted to decimal. The *timestamp* column is replaced by a new column called *dTIME* that measures the time difference between CAN messages of the same arbitration ID. The R and T labels are mapped to 0 and 1 respectively, and the *DLC* column is dropped. In this case study, we use the same 5-fold cross-validation strategy as mentioned in the previous case study. The only difference is that the tree ensemble IDS is an XGBoost GBM with 254 trees with a maximum depth of 10.

Regarding the flooding, fuzzing, and malfunction attacks, adversarial examples were generated by only perturbing the data bits. This is because the malfunction attack was targeted at a specific CAN arbitration ID, the flooding attack already used a high-priority CAN arbitration ID, and the fuzzing attack already used random CAN arbitration IDs.

D. Evaluation Metrics

We evaluate the detection performance of *Maverick* and OC-Score using the following metrics - Accuracy (Acc.), F1-Score (F1-Sc.), Area under the receiver-operating characteristic curve (AUC), Matthews's Correlation Coefficient (MCC), True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), & False Negative Rate (FNR). The evaluation of the pre-evasion attack signature detection is done using balanced accuracy along with macro-averaged precision, recall and F1-Score. Finally, we measure the minimum, maximum, average, and standard deviation of the prediction times per method in μs for 10 experimental runs.

E. Detection Performance

The task is to correctly classify samples in both adversarial and non-adversarial scenarios. First, we present the summarized detection performance of the tree ensemble IDSs on their test sets in table I. This serves as a reference evaluation criterion, assessing OC-Score and *Maverick* on their ability to maintain the accuracy of the original IDS decisions whilst being subjected to both adversarial & non-adversarial scenarios.

Case Study	Tree Ensemble IDS	Acc.	TPR	TNR	FPR	FNR
CIC-IoV-2024	Random Forest	1.000	1.000	1.000	0.000	0.000
HCRL-CAN-FD	XGBoost GBM	1.000	1.000	1.000	0.000	0.000

TABLE I: Baseline Tree Ensemble IDS Performance

From the results in tables II and III, we see that both OC-Score and *Maverick* maintain similar detection performance in both adversarial and non-adversarial scenarios in the context of the two case studies.

In Table III, *Maverick* demonstrates slightly lower performance in iteration 4, which is attributed to the autoencoder

[1] https://ocslab.hksecurity.net/Datasets/can-fd-intrusion-dataset

OC-Score								
Iteration	**Acc.**	**F1-Sc.**	**AUC**	**MCC**	**TPR**	**TNR**	**FPR**	**FNR**
1	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
2	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
3	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
4	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
5	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
Précis	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Maverick								
Iteration	**Acc.**	**F1-Sc.**	**AUC**	**MCC**	**TPR**	**TNR**	**FPR**	**FNR**
1	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
2	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
3	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
4	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
5	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
Précis	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

TABLE II: Detection Performance Comparisons (CIC-IoV-2024)

not having fully converged. While training the autoencoder for an additional five epochs could have achieved convergence, we chose not to unfairly influence or bias the experimental outcomes through this additional training, ensuring consistency in our evaluation process. Furthermore, the lower performance is evident only in the third decimal place, indicating that it should not have a significant impact on the overall results.

OC-Score								
Iteration	**Acc.**	**F1-Sc.**	**AUC**	**MCC**	**TPR**	**TNR**	**FPR**	**FNR**
1	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
2	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
3	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
4	1.000	1.000	1.000	0.999	1.000	0.999	0.001	0.000
5	1.000	1.000	1.000	0.999	1.000	0.999	0.001	0.000
Précis	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Maverick								
Iteration	**Acc.**	**F1-Sc.**	**AUC**	**MCC**	**TPR**	**TNR**	**FPR**	**FNR**
1	1.000	1.000	0.999	0.999	1.000	0.999	0.001	0.000
2	1.000	1.000	1.000	0.999	1.000	0.999	0.001	0.000
3	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
4	0.998	0.998	0.997	0.995	0.998	0.997	0.003	0.002
5	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
Précis	0.999 ± 0.001	0.999 ± 0.001	0.999 ± 0.001	0.998 ± 0.002	0.999 ± 0.001	0.999 ± 0.001	0.001 ± 0.001	0.000 ± 0.001

TABLE III: Detection Performance Comparisons (HCRL-CAN-FD)

In the next subsection, we show that this maintained level of performance comes with significant timing improvements.

F. Runtime Performance

The prediction times for both methods in their respective case studies is shown in table IV. Note that a ✅ indicates compliance with the detection-timing requirements on an automotive CAN-based IDS, while a ❌ signifies non-compliance.

Method	**Case Study**	**Min. (μs)**	**Max. (μs)**	**Avg. (μs)**	**SD (μs)**	**CAN LS**	**CAN HS**	**CAN FD**
OC-Score	CIC-IoV-2024	6084.27	6412.64	6298.64	126.15	❌	❌	❌
Maverick		70.39	79.07	**74.32**	2.58	✅	✅	✅
OC-Score	HCRL-CAN-FD	29795.60	33560.07	31538.58	1138.43	❌	❌	❌
Maverick		52.56	63.06	**55.77**	3.00	✅	✅	✅

TABLE IV: Prediction Time Comparisons

Recall from section III, that an automotive IDS should be able to process CAN 2.0 Low Speed (LS) packets at $896\mu s$ per packet, and CAN 2.0 High Speed (HS) packets at $112\mu s$ per packet. Regarding CAN-FD, this timing requirement is $112\mu s$ per packet. From table IV, we see that *Maverick's* average prediction times of $74\mu s$ and $56\mu s$ meet this real-time prediction requirement of automotive IDSs. In comparison, the average prediction times for OC-Score are $6299\mu s$ and $31539\mu s$ respectively, and that would make *Maverick* 85-563x faster. We believe the iteration over the reference set (for OC-Score) likely explains the significantly longer prediction times.

G. Pre-Evasion Attack Signature Detection

The task is to predict the pre-evasion attack signatures in a multi-class setting given detected adversarial examples.

Case Study	**Accuracy**	**Precision**	**Recall**	**F1-Score**
CIC-IoV-2024	0.933	0.960	0.933	0.938
HCRL-CAN-FD	1.000	1.000	1.000	1.000

TABLE V: Pre-Evasion Attack Signature Detection Performance

From the results in table V, we see that *Maverick* can predict these signatures with an accuracy of more than 93%.

VII. Discussions

In this section, we present some insights into *Maverick's* construction scalability, threats to validity & applicability.

A. Construction Scalability

Although *Maverick* offers good, real-time detection, it comes at the cost of fairly high construction times (21-199 minutes) due to the training of the autoencoder. However, we would like to point out that this cost is one-off and happens offline. Moreover, the autoencoder training in our experiments was done without a GPU (which typically has hundreds of computational cores for parallelization or specialized tensor cores to dramatically speed up the training process).

B. Resource Demands and Threats to Validity

Firstly, when using *Maverick*, it is important to note that it uses a single large autoencoder which consumes a fair bit of RAM. This issue can be mitigated by utilizing an ensemble of smaller autoencoders as demonstrated by Mirsky et al. [5]. Secondly, an adversary could poison the reference set during *Maverick's* autoencoder training, thereby distorting the notion of normality for the evasion detector. This can be addressed by implementing data provenance techniques to ensure the integrity & authenticity of the reference set. Thirdly, an adversary could attempt to compromise the latent space of *Maverick's* autoencoder to undermine its detection capabilities. Although this attack is less likely, as it requires administrative access to the autoencoder's internal mechanisms, we believe it is important for potential users to be aware of this risk.

C. Autoencoder Re-training

As new attack signatures are added to the training set and the tree ensemble IDS is re-trained, the autoencoder in *Maverick* must also be re-trained. Similarly, re-training becomes necessary in response to concept drift or when the notion of normality in the system changes. Techniques such as *Transcend* [24] used in conjunction with *Maverick* can help optimally determine when the autoencoder needs to be re-trained vis-à-vis the aforementioned challenges.

D. Sensitivity to Variations in Traffic Compositions

For answering evaluation question Q1 in section VI, we used an equal ratio of adversarial to non-adversarial samples to prevent any experimental bias. This balanced approach was crucial, as a system overly sensitive to adversarial samples could result in a surge of false positives, while a system

heavily optimized for non-adversarial samples might fail to detect adversarial samples leading to a surge in false negatives. However, in realistic security scenarios, adversarial evasion attacks may account for a smaller proportion of the total traffic, with the remaining traffic consisting of non-adversarial or normal (benign and attack) samples. To check the sensitivity, we gradually vary the proportion of adversarial samples from 0% to 10% in increments of 1%, while systematically reducing the proportion of non-adversarial samples from 100% to 90%, and confirm that *Maverick's* detection performance remains consistent[2] across these varying traffic compositions, validating its effectiveness in realistic deployment scenarios.

VIII. Conclusion & Future Works

This paper explores how to detect evasion attacks in tree-ensemble-based automotive IDSs. Our approach doesn't modify the protected IDS, is independent of the evasion attack generation process, and works with any additive tree ensemble. It also preserves the detection accuracy of the underlying IDS and detects evasions with a very low overhead in terms of real-time performance, 85-563x faster than the state-of-the-art. Additionally, it provides key insights for an enriched security analysis regarding the evasion strategies used by attackers.

Future works include exploring this idea for different classes of machine learning models beyond tree ensembles. For instance, analyzing the activation patterns in neural networks to detect anomalies and their application to IDSs in other contexts is worth exploring. Also, extending this work to detect adversarial sequences is an idea worth investigating.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] Apruzzese, G., Andreolini, M., Ferretti, L., Marchetti, M., & Colajanni, M. (2022). Modeling realistic adversarial attacks against network intrusion detection systems. Digital Threats: Research and Practice (DTRAP).

[2] Biggio, B., Corona, I., Nelson, B., Rubinstein, B.I., Maiorca, D., Fumera, G., Giacinto, G., & Roli, F. (2014). Security evaluation of support vector machines in adversarial environments. Support vector machines applications, 105-153.

[3] Devos, L., Perini, L., Meert, W., & Davis, J. (2023, September). Detecting evasion attacks in deployed tree ensembles. In Joint European conference on machine learning and knowledge discovery in databases.

[4] Neto, E. C. P., Taslimasa, H., Dadkhah, S., Iqbal, S., Xiong, P., Rahman, T., & Ghorbani, A. A. (2024). CICIoV2024: Advancing realistic IDS approaches against DoS and spoofing attack in IoV CAN bus. Internet of Things, 26, 101209.

[5] Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089.

[6] Törnblom, J., & Nadjm-Tehrani, S. (2019). An abstraction-refinement approach to formal verification of tree ensembles. In Computer Safety, Reliability, and Security: SAFECOMP 2019 Workshops, WAISE, Turku, Finland, September 10, 2019, Proceedings 38 (pp. 301-313).

[7] Zenden, I., Wang, H., Iacovazzi, A., Vahidi, A., Blom, R., & Raza, S. (2023, April). On the Resilience of Machine Learning-Based IDS for Automotive Networks. In 2023 IEEE Vehicular Networking Conference (VNC) (pp. 239-246). IEEE.

[2] Tables confirming this are available with our code.

[8] Specification, C. A. N. (1991). Bosch. Robert Bosch GmbH, Postfach.

[9] Oberti, F., Savino, A., Sanchez, E., Casasso, P., Parisi, F., & Di Carlo, S. (2024). CAN-MM: Multiplexed Message Authentication Code for Controller Area Network Message Authentication in Road Vehicles. IEEE Transactions on Vehicular Technology.

[10] Shen, Y., Cui, J., Zhong, H., Zhang, J., Bolodurina, I., & He, D. (2024). A Two-Layer Dynamic ECU Group Management Scheme for In-Vehicle CAN Bus. IEEE Transactions on Intelligent Transportation Systems.

[11] Specification of an Intrusion Detection System Protocol. (2000). www.autosar.org/fileadmin/standards/R20-11/FO/AUTOSAR_PRS_IntrusionDetectionSystem.pdf

[12] Vector. (2020). Detecting Cyber-Attacks With Automotive Intrusion Detection Systems. https://www.vector.com/at/en/know-how/security/automotive-intrusion-detection-systems

[13] Pedregosa, F. (2011). Scikit-learn: Machine learning in python Fabian. Journal of machine learning research, 12, 2825.

[14] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[15] Chen, Y., Wang, S., Qin, Y., Liao, X., Jana, S., & Wagner, D. (2021, November). Learning security classifiers with verified global robustness properties. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (pp. 477-494).

[16] Sperl, P., Kao, C. Y., Chen, P., Lei, X., & Böttinger, K. (2020, September). DLA: dense-layer-analysis for adversarial example detection. In 2020 IEEE European Symposium on Security and Privacy (EuroS&P).

[17] Apruzzese, G., Andreolini, M., Marchetti, M., Colacino, V. G., & Russo, G. (2020). AppCon: Mitigating evasion attacks to ML cyber detectors. Symmetry, 12(4), 653.

[18] Vos, D., & Verwer, S. (2021, July). Efficient training of robust decision trees against adversarial examples. In International Conference on Machine Learning (pp. 10586-10595). PMLR.

[19] Vos, D., & Verwer, S. (2022, September). Adversarially robust decision tree relabeling. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 203-218).

[20] Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., & Jana, S. (2019, May). Certified robustness to adversarial examples with differential privacy. In 2019 IEEE symposium on security and privacy (SP) (pp. 656-672). IEEE.

[21] Cerracchio, P., Longari, S., Carminati, M., & Zanero, S. (2024). Investigating the Impact of Evasion Attacks Against Automotive Intrusion Detection Systems. In Symposium on Vehicles Security and Privacy (VehicleSec) 2024 (pp. N-A).

[22] Longari, S., Noseda, F., Carminati, M., & Zanero, S. (2023, June). Evaluating the Robustness of Automotive Intrusion Detection Systems Against Evasion Attacks. In International Symposium on Cyber Security, Cryptology, and Machine Learning (pp. 337-352).

[23] Wolf, M., Weimerskirch, A., & Wollinger, T. (2007). State of the art: Embedding security in vehicles. EURASIP journal on embedded systems.

[24] Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., & Cavallaro, L. (2017). Transcend: Detecting concept drift in malware classification models. In 26th USENIX security symposium.

[25] Colaco, V., & Nadjm-Tehrani, S. (2023). Formal Verification of Tree Ensembles against Real-World Composite Geometric Perturbations. In the Workshop on Artificial Intelligence Safety 2023 (SafeAI 2023) co-located with the Thirty-Seventh AAAI Conference on Artificial Intelligence.

[26] Rajapaksha, S., Kalutarage, H., Al-Kadri, M. O., Petrovski, A., & Madzudzo, G. (2023, February). Improving in-vehicle networks intrusion detection using on-device transfer learning. In Symposium on vehicles security and privacy (Vol. 10).

[27] Mehta, J., Richard, G., Lugosch, L., Yu, D., & Meyer, B. H. (2023). DT-DS: CAN intrusion detection with decision tree ensembles. ACM Transactions on Cyber-Physical Systems, 7(1), 1-27.

[28] Apruzzese, G., Colajanni, M., Ferretti, L., & Marchetti, M. (2019, May). Addressing adversarial attacks against security systems based on machine learning. In 2019 11th international conference on cyber conflict (CyCon) (Vol. 900, pp. 1-18). IEEE.

[29] Resende, P. A. A., & Drummond, A. C. (2018). A survey of random forest based methods for intrusion detection systems. ACM Computing Surveys (CSUR), 51(3), 1-36.

[30] Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F., & Yang, A. (2022). Comparative research on network intrusion detection methods based on machine learning. Computers & Security, 121, 102861.