CANtropy: Time Series Feature Extraction-Based Intrusion Detection Systems for Controller Area Networks

Md Hasan Shahriar, Wenjing Lou, and Y. Thomas Hou
Virginia Polytechnic Institute and State University, Blacksburg, VA, United States
{hshahriar,wjlou,thou}@vt.edu

Abstract—**A controller area network (CAN) connects dozens of electronic control units (ECUs), ensuring reliable and efficient data transmission. Because of the lack of security features of CAN protocol, in-vehicle networks are susceptible to a wide spectrum of threats, from simple injections at high frequencies to sophisticated masquerade attacks that target individual sensor values (signals). Hence, advanced analysis of the multidimensional time-series data is needed to learn the complex patterns of individual signals and their mutual dependencies. Although deep learning (DL)-based intrusion detection systems (IDS) have shown potential in such domain, they tend to suffer from poor generalization as they need optimization at every component. To detect such advanced CAN attacks, we propose CANtropy, a manual feature engineering-based lightweight CAN IDS. For each signal, CANtropy explores a comprehensive set of features from both temporal and statistical domains and selects only the effective subset of features in the detection pipeline to ensure scalability. Later, CANtropy uses a lightweight unsupervised anomaly detection model based on principal component analysis, to learn the mutual dependencies of the features and detect abnormal patterns in the sequence of CAN messages. The evaluation results on the advanced SynCAN dataset show that CANtropy provides a comprehensive defense against diverse types of cyberattacks with an average AUROC score of 0.992, and outperforms the existing DL-based baselines.**

I. Introduction

Vehicles are complex cyber-physical systems (CPS), running over a hundred million lines of code continuously [1], controlling everything from the power-train to driver assistance and infotainment. Such functionalities are executed by hundreds of distributed electronic control units (ECUs) that communicate through different in-vehicle communication buses. Controller area network (CAN) is a de facto standard in the in-vehicle network due to its efficiency and reliability. However, CAN lacks basic security requirements, and any access to the CAN bus makes the vehicles vulnerable to cyberattacks [2]. Moreover, as modern vehicles further connect to the outside world, the risk of being attacked remotely is rising. Hence, there are a lot of research efforts on building intrusion detection systems (IDS) to secure such critical in-vehicular communication.

Symposium on Vehicles Security and Privacy (VehicleSec) 2023
27 February 2023, San Diego, CA, USA
ISBN 1-891562-88-6
https://dx.doi.org/10.14722/vehiclesec.2023.23090
www.ndss-symposium.org

In recent years, deep learning (DL) has been used on various time-series tasks, including forecasting, anomaly detection, signal processing, etc. Given the optimal environments, DL models can learn complex data distribution and build efficient IDS. One of the core benefits of DL is that they do not need explicit feature representation as input, rather, they learn how to extract features from the raw time-series data. However, there are a few challenges in achieving efficient DL-based IDSs, such as finding the optimal model architecture, providing representative training data, ensuring effective hyperparameters, adequate computing resources, etc. [3]. Optimizing all of these is a challenging task in most applications, especially in the vehicular domain. First, there may not be enough historical data available representing all the possible states of the vehicles to train a DL model. Additionally, vehicular data are highly dynamic, with patterns and trends that change over time and space. Such limitations may lead to ineffective feature extraction and provide a sub-optimal performance of the DL-based IDS [4].

Feature engineering is the process of transforming raw data into meaningful and relevant information that better represents the patterns in the normal data and helps the IDS to learn effectively. Whereas in a few domains like computer vision, pre-defining feature extractor functions is quite challenging, it is much more straightforward and practical in time-series data. In critical CPS, such feature engineering is very essential as it ensures explainability and scalability [5]. There are many different approaches to feature extraction, and the specific approach that is most effective depends on the characteristics of the data. One option is to extract features from statistical and temporal domains. Statistical features capture statistical properties of the data, such as the mean, median, standard deviation, etc, whereas temporal features capture information about the temporal sequence of the data points, such as entropy, slope, energy, etc. [6]

Although extracting all the possible features for all the signals will give the maximum information about the time-series signal data, such an approach will increase the computational overhead for both feature extraction and model fitting/testing. We hypothesize that, for any signal, we need only a subset of feature functions that are sufficient to represent the property of that signal. Moreover, as different signals have different types, natures, and patterns, the effective feature subsets will be unique for each signal [7]. Hence, to make the whole detection pipeline efficient and scalable, feature engineering for individual signals needs to be customized based on the signal's property. Therefore, to address the challenges of extracting effective features from the raw input data, skip the unnecessary sets of signal-feature pairs, and build a lightweight and scalable CAN IDS, we propose CANtropy, a time series feature extraction-based IDS for CAN bus. We make the following contributions to this paper:

- CANtropy consists of a comprehensive set of feature functions from both 'statistical' and 'temporal' domains for time-series data. To select the most effective subset of features for individual signals, CANtropy implements a feature exploration phase that analyzes all the possible signal-feature pairs and creates a priority mapping based on their variances.
- To ensure scalability and efficiency during the detection phase, CANtropy extracts only the most effective signal-feature pairs and uses a lightweight unsupervised detector based on principal component analysis (PCA) to learn and analyze the mutual dependencies and correlations of different signals.
- We implement and evaluate CANtropy's performance on advanced signal-level CAN IDS dataset SynCAN [8] showing that CANtropy outperforms the baseline detectors in detecting all types of attacks. This work can also serve as a benchmark for future DL-based IDS, providing a lower bound on their expected performance.

The rest of the paper is organized as follows: We introduce necessary background information and modeling in §II. The technical details are shown in §III. We provide an experimental setup and implementation details in §IV. The evaluation results are analyzed in §V. The related works are discussed in §VI. Finally, we conclude the paper in §VII.

II. Preliminaries and Objectives

A. Controller Area Network

Robert Bosch GmbH introduced controller area network (CAN) as an automotive communication bus with the latest version (2.0) released in 1991 [9]. A CAN data frame can have up to 11 bits of arbitration ID (or CAN ID) and 64 bits of binary payloads. In the CAN bus, every ECU broadcasts its message periodically. If multiple ECUs attempt to transmit messages at the same time, the ECU with the lowest CAN ID gets the highest priority, and others wait until the channel is available again. Hence, as different CAN IDs have different priorities, they usually appear in the CAN bus at a wide range of frequencies. To build advanced CAN IDS, the binary payload needs to be decoded to corresponding real-valued decimal signal values that provide a time-series representation. Specific car's database for CAN (DBC) file contains the decoding instruction. Although such files are proprietary and are not publicly available, there are a lot of reverse engineering efforts, such as CAN-D [10], to generate approximate DBC files which can achieve reasonable performance in decoding. Hence, binary payloads can be decoded into a signal-level dataset using the instructions in the DBC file. However, such a dataset contains some missing values due to the arbitration process. Similar to [11], we also assume that the signals remain unchanged until new updated values are reported, and we fill the missing values with the last reported values using a forward filling mechanism.

B. Features Functions

This part summarizes the comprehensive list of feature functions that CANtropy considers for each of the signals in the CAN dataset. Table I shows the description and complexity of individual features both from 'statistical' and 'temporal' domains. Features 1-16 are from the 'statistical' domain, where 5 out of 16 feature functions have log complexity, and the rest have $constant$ complexity. Similarly, features 17-34 are from the 'temporal' domain, where 4 out of 18 have log complexity, and the rest are $constant$. More detail on the feature functions can be found in [6].

C. Attack Model

We consider the attacker can gain access to the CAN bus physically (OBD-II port) or remotely, such as through infotainment, ADAS systems, etc. We also consider the attacker can turn off any targeted ECU [12] and inject malicious messages. In general, we consider the attacker can launch the following three types of attacks.

- **Fabrication attack** is a type of injection attack where the attacker injects random IDs and payloads to flood the network. Usually, the CAN IDs are chosen to be very low to indicate a high priority. Due to the continuous injection of high-priority messages, legitimate ECUs cannot transmit their data and eventually end up in denial of service (DoS) types of attacks.
- **Suspension attack** is possible due to the limitation of the CAN protocol itself (error control). An attacker can target an ECU and force it to disconnect from the CAN bus, stopping the ECU from sending the corresponding signal values. This attack is well known as *bus-off attack* or *suppress attack*.
- **Masquerade attack** is the most advanced but difficult-to-launch attack, where the attacker needs to turn off the targeted ECU first and then spoof it by transmitting malicious data on its behalf. Based on the type of data transmitted, the attacker can achieve different levels of malicious goals.

We will evaluate CANtropy considering the well-known CAN attack dataset, SynCAN [8], which has all the above-mentioned attacks.

D. Design Objectives

We design CANtropy to achieve the following objectives:

- **Detecting advanced attacks.** The fundamental goal of CANtropy is to effectively extract important features from the time-series data and build a detection model to achieve high performance, with near-zero false positives, in detecting advanced attacks, particularly those advanced stealthy attacks, where DL-based methods have shown ineffective.
- **Near real-time detection.** CANtropy should extract the features and provide anomaly scores in near real-time. As the human driver response time ranges from 0.7s to 1.5s [13], the inference time of CANtropy should be lower than 0.7s.

III. CANtropy Detail Design

This section explains the proposed CANtropy framework along with the technical details. As is shown in Fig. 1,

TABLE I: List of Features from Statistical and Temporal Domain.

Index	Domain	Feature	Description	Complexity
1	Statistical	ECDF	Computes the values of ECDF (empirical cumulative distribution function) along the time axis.	log
2	Statistical	ECDF Percentile	Determines the percentile value of the ECDF.	log
3	Statistical	ECDF Percentile Count	Determines the cumulative sum of samples that are less than the percentile.	log
4	Statistical	Histogram	Computes histogram of the signal.	log
5	Statistical	Interquartile range	Computes interquartile range of the signal.	constant
6	Statistical	Kurtosis	Computes kurtosis of the signal.	constant
7	Statistical	Max	Computes the maximum value of the signal.	constant
8	Statistical	Mean	Computes the mean value of the signal.	constant
9	Statistical	Mean absolute deviation	Computes mean absolute deviation of the signal.	log
10	Statistical	Median	Computes median of the signal.	constant
11	Statistical	Median absolute deviation	Computes median absolute deviation of the signal.	constant
12	Statistical	Min	Computes the minimum value of the signal.	constant
13	Statistical	Root mean square	Computes root mean square of the signal.	constant
14	Statistical	Skewness	Computes skewness of the signal.	constant
15	Statistical	Standard deviation	Computes standard deviation of the signal.	constant
16	Statistical	Variance	Computes variance of the signal.	constant
17	Temporal	Absolute energy	Computes the absolute energy of the signal.	log
18	Temporal	Area under the curve	Computes the area under the curve of the signal computed with trapezoid rule.	log
19	Temporal	Autocorrelation	Computes autocorrelation of the signal.	constant
20	Temporal	Centroid	Computes the centroid along the time axis.	constant
21	Temporal	Entropy	Computes the entropy of the signal using the Shannon Entropy.	log
22	Temporal	Mean absolute diff	Computes mean absolute differences of the signal.	constant
23	Temporal	Mean diff	Computes mean of differences of the signal.	constant
24	Temporal	Median absolute diff	Computes median absolute differences of the signal.	constant
25	Temporal	Median diff	Computes median of differences of the signal.	constant
26	Temporal	Negative turning points	Computes number of negative turning points of the signal.	constant
27	Temporal	Neighbourhood peaks	Computes the number of peaks from a defined signal neighborhood.	constant
28	Temporal	Peak to peak distance	Computes the peak to peak distance.	constant
29	Temporal	Positive turning points	Computes number of positive turning points of the signal.	constant
30	Temporal	Signal distance	Computes signal traveled distance.	constant
31	Temporal	Slope	Computes the slope of the signal by fitting a linear equation to the observed data.	log
32	Temporal	Sum absolute diff	Computes the sum of absolute differences of the signal.	constant
33	Temporal	Total energy	Computes the total energy of the signal.	constant
34	Temporal	Zero crossing rate	Computes Zero-crossing rate of the signal.	constant

CANtropy has three fundamental phases of implementation: i) P_1: exploration phase, ii) P_2: development phase, and iii) P_3: deployment phase. All these phases consist of some sequential tasks, where some of them are common in different phases, and some are unique. The tasks are i) T_1: data collection and preprocessing, ii) T_2: feature extraction, iii) T_3: feature analysis and configuration, iii) T_4: model fitting and threshold selection, and iv) T_5: model testing and attack detection. The exploration phase consists of tasks T_1, T_2, and T_3, the development phase consists of tasks T_1, T_2, and T_4, and the deployment phase runs tasks T_1, T_2, and T_5. Hence, whereas tasks T_1 and T_2 are common in all these phases, the last task differs based on the phase.

Whereas phases P_1 and P_2 are typically considered to be accomplished in a relatively powerful computer (local/cloud), phase P_3 can be operated on light-weighted devices/ECUs. CANtropy uses a time window to collect a sequence of CAN messages and determine if there is any attack within that time window. Therefore, CANtropy detects attacks at the window level rather than at the message level. The following subsections explain the technical aspects of CANtropy in detail.

A. Exploration Phase (P_1)

1) Data Collection and Preprocessing (T_1): The first task of the implementations of CANtropy is to set up the data collection and preprocessing pipeline. We assume CANtropy is installed on a computing device, such as a computer, Raspberry Pi, etc., connected to the CAN bus through the OBD-II Port or on a dedicated ECU with direct access to the CAN bus. Through this interface, CANtropy continuously reads and collects raw CAN messages bus and translates the binary payloads using the available decoding instructions. The data collection can be done using either open-sourced software such as Seeed CAN-BUS Shield and SocketCAN or commercial CAN data loggers such as CANalyzer, and VehicleSpy, etc. To translate binary payloads to meaningful multi-dimensional time-series signals, we assume CANtropy is either preloaded with the OEM's DBC file or relies on third-party decoding software, such as CAN-D [10]. Let us assume that CANtropy extracts a set of signals $\mathcal{S} = [s_1, s_2, \cdots, s_n]$, where n is the total number of decoded signals. Hence, CANtropy stores the multi-dimensional time-series signals data $\mathbf{X}_{explr} \in \mathbb{R}^{t_1 \times n}$ from the latest t_1 messages, where t_1 is the data collection period (in time steps). Once T_1 is complete, the data is stored in local storage for feature extraction (task T_2).

2) Feature Extraction (T_2): Feature extraction is the second sequential task, regardless of the phase. Before starting this step, we assume CANtropy collects a sufficient amount of data (in T_1) representing the different states of the vehicle. The features are extracted using a moving window approach, where smaller chunks of $\mathbf{X}$ of length w are selected to create the inputs ($\in \mathbb{R}^{w \times n}$), from which different features are extracted. Let us consider $\mathcal{F} = [f_1, f_2, \cdots, f_m]$ as a comprehensive set of feature functions that CANtropy can apply to the signals. CANtropy starts T_2 will a binary configuration mapping matrix $\mathbf{C} \in \mathbb{R}^{n \times m}$ indicating which signal-feature pairs will be considered during the feature extraction step. Hence, for any signal $s_i \in \mathcal{S}$ and feature function $f_j \in \mathcal{F}$, CANtropy, the extracted feature is considered as $s_i f_j$. The feature $s_i f_j$

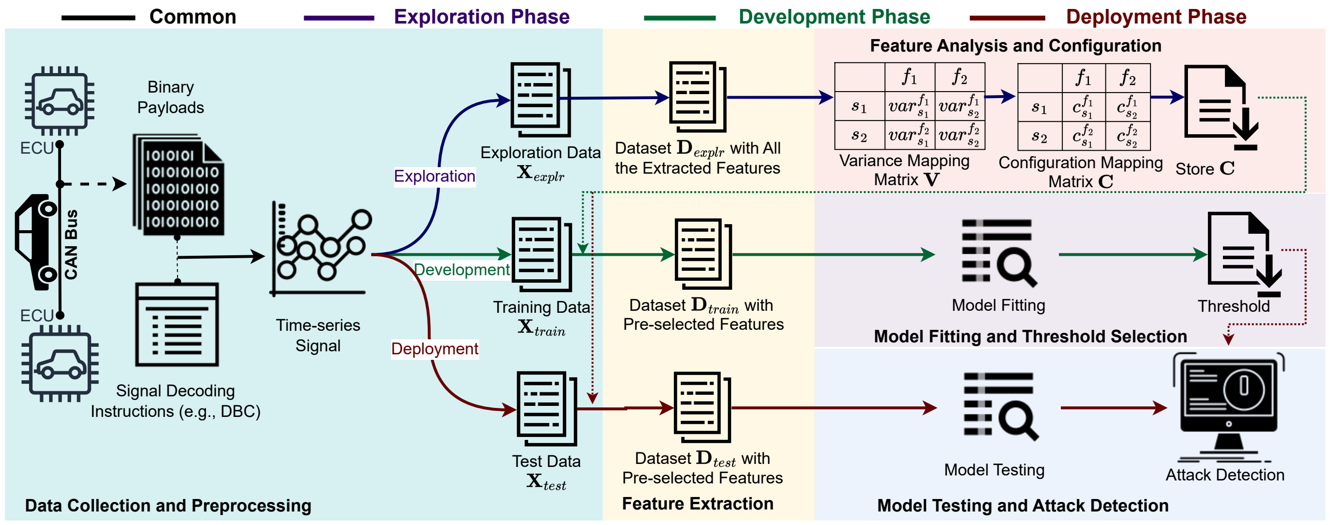
Common
Exploration Phase
Development Phase
Deployment Phase
Binary Payloads
ECU
CAN Bus
ECU
Signal Decoding Instructions (e.g., DBC)
Time-series Signal
Exploration
Development
Deployment
Exploration Data $\mathbf{X}_{explr}$
Training Data $\mathbf{X}_{train}$
Test Data $\mathbf{X}_{test}$
Data Collection and Preprocessing
Dataset $\mathbf{D}_{explr}$ with All the Extracted Features
Dataset $\mathbf{D}_{train}$ with Pre-selected Features
Dataset $\mathbf{D}_{test}$ with Pre-selected Features
Feature Extraction
Feature Analysis and Configuration
Variance Mapping Matrix $\mathbf{V}$
Configuration Mapping Matrix $\mathbf{C}$
Store $\mathbf{C}$
Model Fitting
Threshold
Model Fitting and Threshold Selection
Model Testing
Attack Detection
Model Testing and Attack Detection

Fig. 1: An overview of CANtropy's workflow.

is extracted if $\mathbf{C}[i, j]$ is equal to 1, otherwise, the feature is skipped. In phase P_1, CANtropy extracts all the possible features from all the signals; hence, $\mathbf{C}$ is initialized as an all-one matrix where every element is equal to one. Upon extraction, feature $s_i f_j$ is added to the dataset $\mathbf{D}_{explr}$, and once all the features are extracted, $\mathbf{D}_{explr} \in \mathbb{R}^{k_1 \times l_1}$ is ready to be used for feature analysis tasks (T_3). Here k_1 is the total number of considered windows, and $l_1 = m \times n$ is the number of total extracted features. Algorithm 1 summarizes the steps of feature extraction step from the decoded signal data.

3) Feature Analysis and Configuration (T_3): Feature analysis and configuration is the last step of P_1. In T_3, CANtropy calculates the importance of all the extracted features and determines the subsets of the most useful features for each signal. One way to understand the effectiveness of the signal-feature pairs is to observe their variances. A very low (or zero) variance indicates there is no change in the corresponding features of the considered signal; hence, extracting such a feature and considering it in the detection pipeline will be useless. Therefore, in the feature exploration phase, CANtropy calculates the variances of all the extracted features. For any signal $s_i \in \mathcal{S}$ and feature $f_j \in \mathcal{F}$, the variance of extracted feature $s_i f_j$ is considered as $var_{s_i}^{f_j}$ and stored in a variance mapping matrix $\mathbf{V} \in \mathbb{R}^{n \times m}$. To minimize the extraction of unnecessary features with low variances during phases, P_2 and P_3, the configuration mapping matrix $\mathbf{C}$ is updated based on the minimum variance threshold var_{th}. defined by the system operator. Algorithm 2 summarizes the steps of feature analysis and update of a configuration mapping matrix $\mathbf{C}$. The goal is to exclude those signal-feature pairs from the feature extraction and detection pipeline to increase the computational and detection performance. Hence, the the value of $\mathbf{C}[i, j]$ is set to 1 if $var_{s_i}^{f_j}$ is greater than var_{th}, otherwise it is set to 0. Once all the signal-feature pairs are checked, the updated configuration mapping matrix $\mathbf{C}$ is stored to be used in the next phases to generate only the relevant features for the detection pipeline.

B. Development Phase (P_2)

The development phase has similar tasks T_1 and T_2 as discussed in §III-A1 and §III-A2 except some changes in the hyper-parameters.

1) Data Collection and Preprocessing (T_1): This part CANtropy collects data to build the detection model. Data collected in P_1-T_1 could be reused in T_1 of P_2. Besides, CANtropy can append more data to ensure completeness of the training data and generate $\mathbf{X}_{train} \in \mathbb{R}^{t_2 \times n}$, where t_2 is the total number of messages collected.

2) Feature Extraction (T_2): Unlike P_1-T_2 where all the features are extracted, P_2-T_2 considers only the pre-defined signal-feature pairs as indicated in configuration matrix $\mathbf{C}$ (explain in §III-A2) that have high variance. Similar to P_1-T_2, the same moving window w is used to extract the features and stored in the dataset $\mathbf{D}_{train} \in \mathbb{R}^{k_2 \times l_2}$, where k_2 is total windows and l_2 ($\leq l_1$) is the total number of selected features.

3) Model Fitting and Threshold Selection (T_4): IDS are used to detect and alert potential malicious activity on any networked systems. Any model that can learn the normal pattern and predict the abnormality can serve as the detector. The detector can be trained in a supervised or unsupervised manner. The unsupervised methods can detect novel, unknown, and emerging attacks by learning only from benign data. While collecting benign vehicular data needs very limited effort, generating realistic attack traces on the vehicular network is quite difficult, risky, and needs a lot of expertise. Hence, due to the limited amount of malicious traces, unsupervised learning-method-based models are widely used in CAN IDS research. Therefore, CANtropy also considers an unsupervised model $\mathcal{M}$ as the detector. The unsupervised models mostly work on the reconstruction of the input data through a lossy process. The mean absolute value of the reconstruction loss is considered as the anomaly score. The hypothesis is that the reconstruction error of malicious data would be statistically higher than that of benign data.

In this experiment, we utilize principal component analysis (PCA) as the unsupervised detection model [14]. PCA works

Algorithm 1 Feature Extraction

```
Input: List of signals $\mathcal{S} = [s_1, s_2, \cdots, s_n]$
List of features, $\mathcal{F} = [f_1, f_2, \cdots, f_m]$,
Configuration matrix, $\mathbf{C} \in \mathbb{R}^{n \times m}$, CAN signal dataset, $\mathbf{X} \in \mathbb{R}^{t \times n}$
Output: Generated dataset, $\mathbf{D}$
1: Initialize empty 2D dataset $\mathbf{D}$ = [ , ]
2: for signal $s_i \in \mathcal{S}$ do
3:     for feature $f_j \in \mathcal{F}$ do
4:         if C[i,j] == 1 then
5:             Generate feature $s_i f_j$ shifting a of window $w$ over $\mathbf{X}$
6:             Add new feature $s_i f_j$ to dataset $\mathbf{D}$
7:         end if
8:     end for
9: end for
10: Save new dataset $\mathbf{D}$ for development or deployment phase
```

by transforming the original dataset into a new set of variables called principal components. These principal components are linear combinations of the original features, which capture the maximum amount of variance in the data. By reducing the number of dimensions, PCA can help to simplify the data and remove noise, which can make it easier to identify anomalies. Here, the minimum cumulative explained variance ratio (MCEVR) indicates how much variances will be retained after the PCA. Hence, after normalizing the $\mathbf{D_{train}}$, CANtropy fits the model $\mathcal{M}$ on it. Once fitted, CANtropy stores the model and analyzes the anomaly scores on the benign training data to determine the detection threshold. CANtropy considers the r-th percentile of the anomaly scores of the training data $\mathbf{p_{train}} = \mathcal{M}(\mathbf{D_{train}})$ as the threshold of anomaly p_{th}.

C. Deployment Phase (P_3)

The final and last phase of the implementation is the deployment phase. At the beginning of this phase, CANtropy loads the trained model $\mathcal{M}$ and threshold score p_{th}. Whereas phases P_1 and P_2 are run only once, the tasks in phase P_3 run continuously and sequentially to collect new messages and check for anomalies. During the data collection and pre-processing task (P_3-T_1) of the deployment phase, CANtropy collects the latest messages and only keeps the data of the last w messages as dataset $\mathbf{X}_{test} \in \mathbb{R}^{w \times n}$. The feature extraction process (P_3-T_2) is exactly as same as as P_2-T_2 (§III-B2. As the input data has the length of the window size w, the extracted features dataset $\mathbf{D}_{test} \in \mathbb{R}^{1 \times l_2}$, contains just a single row indicating the most recent features. Model testing and anomaly detection (P_3-T_5) is the last task of P_3, where the anomaly score of such a test sample is calculated using $\mathbf{p_{test}} = \mathcal{M}(\mathbf{D_{test}})$. If the anomaly scores $\mathbf{p_{test}} > p_{th}$, CANtropy considers this as an anomaly and raises the alarm; otherwise, considers that as a normal condition.

IV. Implementation

This subsection explains the dataset, evaluation setting, software implementation, evaluation metrics, and baselines for evaluation of CANtropy.

A. Dataset and Attacks

We implement CANtropy on the SynCAN (Synthetic CAN Bus Data) dataset [8], which is the widely accepted publicly available signal level CAN attack dataset released by

Algorithm 2 Feature Analysis and Variance Matrix

```
Input: List of signals $\mathcal{S} = [s_1, s_2, \cdots, s_n]$
List of features, $\mathcal{F} = [f_1, f_2, \cdots, f_m]$,
Variance threshold $var\_th \in \mathbb{R}$
CAN signal dataset, $\mathbf{X} \in \mathbb{R}^{t \times n}$
Variables: Configuration matrix $\mathbf{C} \in \mathbb{R}^{n \times m} \leftarrow \mathbf{0}$
Output: Variance matrix, $\mathbf{V} \in \mathbb{R}^{n \times m}$, Configuration matrix, $\mathbf{C}$
1: for signal $s_i \in \mathcal{S}$ do
2:     for feature $f_j \in \mathcal{F}$ do
3:         Generate feature $s_i f_j$ by shifting a of window $w$ over $\mathbf{X}$
4:         Calculate variance $var_{s_i}^{f_j}$ of $s_i f_j$, assign it to $\mathbf{V}[i, j]$
5:         if $\mathbf{V}[i, j] > var_{th}$ then
6:             $\mathbf{C}[i, j] = 1$
7:         end if
8:     end for
9: end for
10: Store configuration matrix $\mathbf{C}$ for future feature generation
```

ETAS. It contains stealthy signal-level CAN attacks along with comprehensive normal driving traces. The SynCAN dataset is generated based on the characteristics of real CAN traces, and it contains hundreds of advanced attack events. The dataset has, in total, 20 different signals and 10 different CAN IDs. The SynCAN dataset contains one type of *fabrication attack* (i.e., *flooding attack*), one type of *suspension attack* (i.e., *suppress attack*), and the types of advanced *masquerade attacks* (i.e., *plateau attack*, *continuous attack*, and *playback attack*). The *flooding attacks* is launched by frequently transmitting high-priority messages over a short period of time to jam the legitimate ECUs' transmission. The *suppress attacks* turns off the targeted ECU and prevents it from sending messages in the CAN bus for a while. The masquerade attacks are launched first by turning off the targeted ECU and sending malicious messages on its behalf of it. During the *plateau attacks*, the targeted ECU/signal start broadcasting a constant value over the attack periods. However, this attack can cause a fake jump or freeze the actual value. The attack impact depends on the jump and duration of the attack events. Similarly, the *continuous attacks* and *playback attacks* are launched by injecting continuously changing values and pre-recorded values on behalf of the targeted signals. As the signals start to deviate slowly from the actual ones, initially, the attack can resemble the realistic situations but eventually will create noticeable deviations.

B. Implementation and Evaluation Setup

CANtropy Software Implementation. We use Python 3.9.7 with the python library *tsfel*-0.1.4 (time series feature extraction library) [6] to extract the temporal and statistical features from the time-series data. We use *scikit-learn* library to implement principal component analyses (PCA).

Evaluation Settings. To evaluate CANtropy, we consider w as 500 as this window size ensures that all the signals are at least reported once and also allows the features to have a minimum amount of variance. While generating the features, we consider 'temporal', 'statistical', and 'both' as the type of domain to analyze the contribution of each domain. To reduce the number of features, we consider a list of variance threshold var_{th} as $\{0.0000, 0.0025, 0.0050, 0.0075, 0.0100\}$ to find the optimal point of the threshold. Besides, for PCA, we

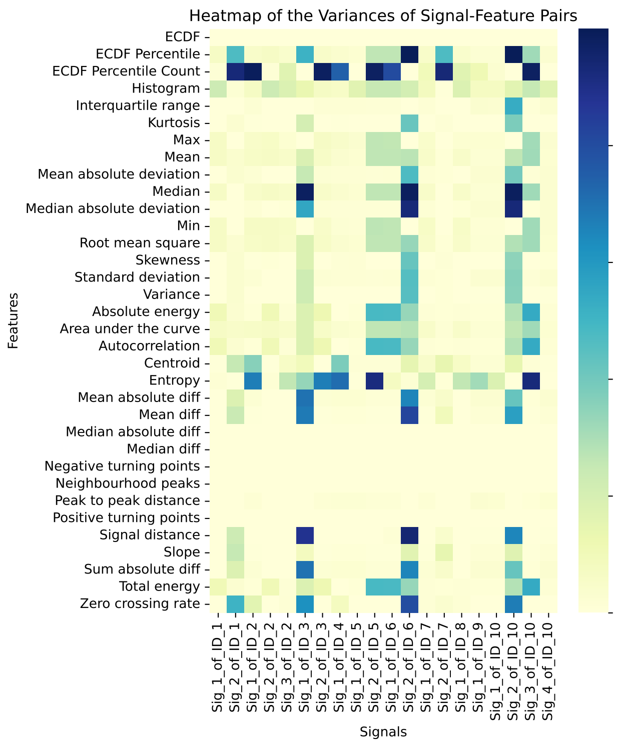

Fig. 2: Heatmap of variance mapping matrix $\mathcal{V}$.

try $\{99.9, 99.99, 99.999, 99.9999, 99.99999\}$ as the minimum cumulative explained variance ratio (MCEVR) to study how much we need to reconstruct to have a better separation between the benign and malicious samples.

Evaluation Metrics. For any binary classifier, there are four possible outcomes. True positive (TP) and true negative (TN) are the outcomes where the model correctly predicts the positive (attack) and negative (benign) classes, respectively. A false positive (FP) and false negative (FN) are the outcomes where the model incorrectly predicts the positive classes, and negative classes, respectively. Based on these outcomes, we evaluate CANtropy's performance using the following metrics:

- *True Positive Rate (TPR)* is calculated as the ratio between the number of positive views correctly classified as positive to the total number of actual positive views ($\frac{TP}{TP+FN}$).
- *False Positive Rate (FPR)* is the proportion of negative views incorrectly identified as positives ($\frac{FP}{FP+TN}$). FPR is the probability that false alerts will be raised.
- *ROC Curve, and AUC Scores* indicate the classifiers performance with varying discrimination thresholds. The ROC curve plots TPR and FPR for different thresholds. The area under the ROC curve is represented as AUROC, which indicates the robustness of the detectors. AUCROC is the probability that a random positive view will have a higher anomaly score than a random benign view. An ideal detector has an AUROC score of 1.00.

Baseline Models. This part describes the baseline models that we consider for the performance comparison.

- *CANShield [11]* uses multiple convolutional neural networks (CNN)-based AEs to analyze the time-series signal data with multiple time-scales. The ensemble model combines the prediction scores from different AEs and achieves robust performance.
- *CANet* [8] utilizes individual LSTM models for individual CAN IDS and finally merges their output to form a fully connected AE network. Both CANShield and CANet use DL-based models and use SynCAN datasets to evaluate their IDSs. As we are also considering the SynCAN dataset to evaluate CANtropy, these two models become the most relevant baselines.

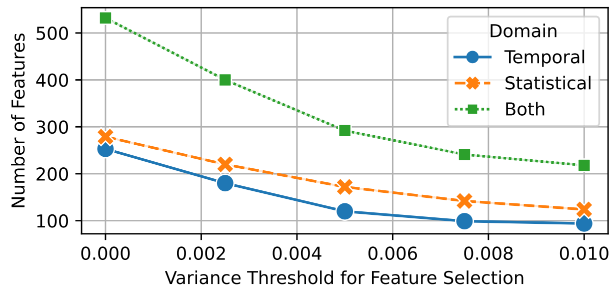

Fig. 3: Impact of variance threshold on the number of features.

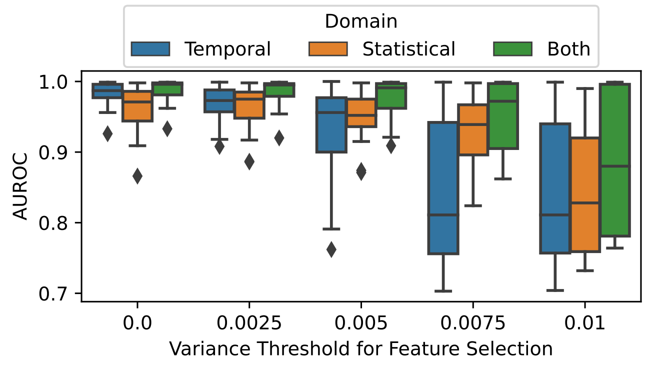

Fig. 4: Impact of variance threshold on AUROC scores.

V. Results

We evaluate CANtropy in the following aspects:

Visualization of Variance Mapping Matrix. Fig 2 shows the variance mapping matrix $\mathcal{V}$ for every pair of signals and features in the SynCAN dataset. As shown in the figure, three types of features exist; some have high variances for most of the signals, some do not have variance at all, and some have high variances for specific sets of signals. Hence, analyzing which signal-feature pairs have high variances and only working on those features will make IDS more effective and scalable than blindly extracting all the signal-features pairs.

Impact of Feature Selector Variance Threshold. Fig 3 shows how the number of features reduces with the increasing variance threshold during feature selection. At the same time, Fig. 4 shows the impact of variance threshold on the AUROC of different types of attacks for different domains. From the figure, it is evident that although the temporal and statistical features are a little sensitive to the variance threshold, combining both types of features increases the

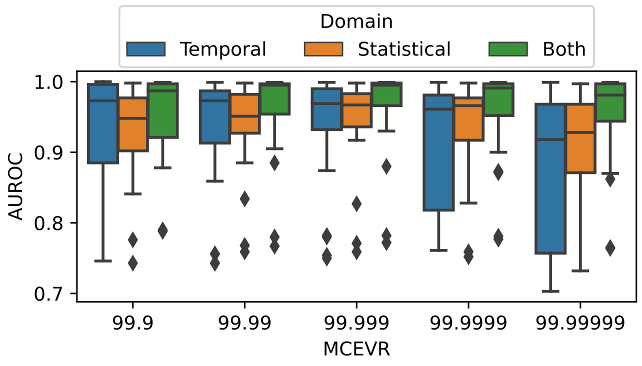

Fig. 5: Impact of MCEVR of PCA on AUROC score.

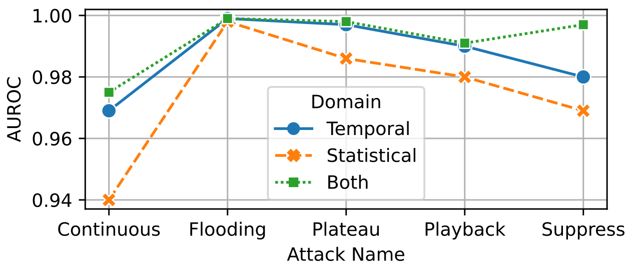

Fig. 6: Attack-wise performance of different domains

model's detection performance and is robust against the higher variance threshold (up to 0.005). However, considering all the features (threshold as 0.0) from both domains provides the best detection performance.

Impact of MCEVR of PCA. Fig. 5 shows the impact of different MCEVR of PCA on AUROC scores for different domains. It is provident that retaining 99.999% variance during the PCA provides the best representation of the data and maximizes the detection performances (AUROC) of different types of attacks. Although the statistical and temporal domain performs well, considering both features in a single detector makes the detection efficient (high AUROC scores) and robust.

Attack-wise Detection Performance. Fig. 6 shows the attack-wise performance of CANtropy with features from different domains. Here we consider a variance threshold of 0.00 and MCEVR of 99.999% to show the upper bound of CANtropy's performance on the SynCAN dataset. As shown in the figure, temporal features have better representation and can detect flooding, plateau, and playback attacks with a very high AUROC ($> 98\%$). On the other hand, statistical features are mainly effective against flooding attacks. However, if CANtropy combines both the temporal and statistical features, it can detect continuous and suppress attacks with better AUROC scores.

Baseline Comparison. Fig.7 shows the performance comparison of CANtropy with the baselines CANet [8] and CANShield [11]. The figure illustrates that baselines work well on one or two attacks, but none offer a comprehensive and steady performance. For example, CANet fails to detect suppress attacks, and CANShield struggles against the masquerade attacks. On the other hand, CANtropy outperforms the baseline models with a large margin in four out of five attacks and provides an average AUROC score of 0.992. However, the current version of CANtropy considers a step size that is the same as the length of the window during the feature extraction process. Hence, we consider a step size of 500 in these evaluation, whereas CANet and CANShield considered a step size of 1. Thus, the evaluation result of CANtropy might differ if a step size of 1 is used.

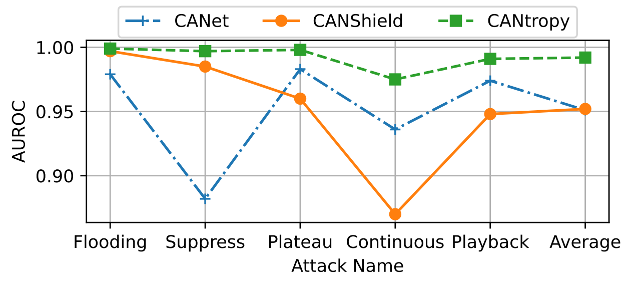

Fig. 7: Baseline comparison

Scalability and Inference Time. CANtropy only needs to fit the PCA model with the training data, which is done within a second, making such a framework scalable, lightweight, and easily transferable. On a computer with an 8-Core Intel i9 processor with 32 GB of RAM, the complete feature extraction takes, on average, 80ms per window, and the inference only takes 0.52 ms. Hence the total upper limit for a single deployment interference is 80.52ms, which is still under the human perception time of 0.7s to 1.5s. However, although the model inference is very fast, the feature extraction of CANtropy takes longer time compared to the baselines, especially CANShield. As the standard transmission time interval in CAN bus is around 2ms, the current version of CANtropy needs to wait for approximately 40 messages to run the inference again, giving the attacker a window to launch a successful attack.

VI. Related Works

There has been a good amount of work on CAN IDS, which can be divided into the following categories in general.

Physical Characteristics-based IDS. There have been few research on fingerprinting the ECUs based on their physical layer data, such as clock skew, voltage profile, signal characteristics, etc., on verifying the source of each message [15]–[18]. However, such approaches have shown ineffective [19] as an attacker can manipulate the physical attributes and impersonate the targeted ECU.

CAN ID-based IDS. Such IDS learns the patterns and extracts attributes from the sequences of CAN IDs and detects any abnormal patterns. Different supervised and unsupervised machine learning-based approaches are used in such pattern recognition [20], [21]. However, these IDSs can only detect injection attacks only when they alter the sequence of IDs. Moreover, they fail to detect any advanced masquerade attacks, as those only change the payloads, not the IDs.

Payload-based Detection. To detect advanced attacks, these IDSs analyze the maliciousness within the content of the messages by looking at the payloads instead of the IDs. As the binary payloads are up to 64 bits of binary values, these IDSs consider the sequence of payloads as a 64-dimensional boolean dataset and learn how the binary bits change over time [22]–[24]. However, as the payloads are obfuscated, in most cases, this representation is ineffective in learning normal driving patterns, especially against advanced masquerade attacks, while the attacker can toggle only a single signal and still cause huge damage to the system.

Signal-level Detection. These IDSs provide the most comprehensive defense against a wide range of cyberattacks- from naive flooding attacks to advanced masquerade attacks. These IDSs utilized correlation and clustering [25], machine learning, such as CNN, RNN, LSTM, etc. [8], [11], [26], [27], etc. However, such DL models lack explainability and suffer in extracting effective features from multi-dimensional time-series data, thus cannot provide a comprehensive solution against a wide range of cyberattacks.

On the contrary, our proposed CANtropy consists of a comprehensive set of feature extraction functions, where the list of features for each signal is defined based on the signal itself. Such optimal selection of effective sets of features allows a lightweight detector to detect advanced cyberattacks effectively.

VII. Conclusion

With the capability of handling a high-dimensional CAN data stream, CANtropy extracts the most relevant features and utilizes a lightweight detection model based on PCA, to detect advanced cyber attacks. The evaluation results on the SynCAN dataset show CANtropy provides an average AUROC score of 0.992, outperforms the existing baselines by a large margin, and advances the state-of-the-art CAN IDS research.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation (NSF) and the Department of Homeland Security (DHS) under NSF grant CNS-1837519, CNS-1916902, and the Virginia Commonwealth Cyber Initiative.

References

[1] Robert N Charette. This car runs on code. *IEEE spectrum*, 2009.

[2] Yang Xiao, Shanghao Shi, Ning Zhang, Wenjing Lou, and Y Thomas Hou. Session key distribution made practical for can and can-fd message authentication. In *Annual Computer Security Applications Conference*, pages 681–693, 2020.

[3] Francois Chollet. The limitations of deep learning. *Deep learning with Python*, 2017.

[4] Mayra Macas, Chunming Wu, and Walter Fuertes. A survey on deep learning for cybersecurity: Progress, challenges, and opportunities. *Computer Networks*, page 109032, 2022.

[5] Siti-Farhana Lokman, Abu Talib Othman, and Muhammad-Husaini Abu-Bakar. Intrusion detection system for automotive controller area network (can) bus system: a review. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–17, 2019.

[6] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. Tsfel: Time series feature extraction library. *SoftwareX*.

[7] Paul Agbaje, Afia Anjum, Arkajyoti Mitra, Gedare Bloom, and Habeeb Olufowobi. A framework for consistent and repeatable controller area network ids evaluation.

[8] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. Canet: An unsupervised intrusion detection system for high dimensional can bus data. *IEEE Access*, 8:58194–58205, 2020.

[9] Marco Di Natale, Haibo Zeng, Paolo Giusto, and Arkadeb Ghosal. *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media.

[10] Miki E. Verma, Robert A. Bridges, Jordan J. Sosnowski, Samuel C. Hollifield, and Michael D. Iannacone. Can-d: A modular four-step pipeline for comprehensively decoding controller area network data. *IEEE Transactions on Vehicular Technology*, 70(10):9685–9700, 2021.

[11] Md Hasan Shahriar, Yang Xiao, Pablo Moriano, Wenjing Lou, and Y Thomas Hou. Canshield: Signal-based intrusion detection for controller area networks. *arXiv preprint arXiv:2205.01306*, 2022.

[12] Kyong-Tak Cho and Kang G Shin. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.

[13] Paweł Droździel, Sławomir Tarkowski, Iwona Rybicka, and Rafał Wrona. Drivers' reaction time research in the conditions in the real traffic. *Open Engineering*, 10(1):35–47, 2020.

[14] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[15] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 911–927, 2016.

[16] Kyong-Tak Cho and Kang G Shin. Viden: Attacker identification on in-vehicle networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1109–1123, 2017.

[17] Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee. Identifying ecus using inimitable characteristics of signals in controller area networks. *IEEE Transactions on Vehicular Technology*, 67(6):4757–4770, 2018.

[18] Marcel Kneib and Christopher Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 787–800, 2018.

[19] Rohit Bhatia, Vireshwar Kumar, Khaled Serag, Z Berkay Celik, Mathias Payer, and Dongyan Xu. Evading voltage-based intrusion detection on automotive can. In *NDSS*, 2021.

[20] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[21] Araya Kibrom Desta, Shuji Ohira, Ismail Arai, and Kazutoshi Fujikawa. Rec-cnn: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots. *Vehicular Communications*.

[22] Min-Joo Kang and Je-Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*.

[23] Aiguo Zhou, Zhenyu Li, and Yong Shen. Anomaly detection of can bus messages using a deep neural network for autonomous vehicles. *Applied Sciences*, 9(15):3174, 2019.

[24] Florian Fenzl, Roland Rieke, Yannick Chevalier, Andreas Dominik, and Igor Kotenko. Continuous fields: enhanced in-vehicle anomaly detection using machine learning models. *Simulation Modelling Practice and Theory*, 105:102143, 2020.

[25] Pablo Moriano, Robert A Bridges, and Michael D Iannacone. Detecting can masquerade attacks with signal clustering similarity. In *Workshop on Automotive and Autonomous Vehicle Security (AutoSec)*, 2022.

[26] Javed Ashraf, Asim D Bakhshi, Nour Moustafa, Hasnat Khurshid, Abdullah Javed, and Amin Beheshti. Novel deep learning-enabled lstm autoencoder architecture for discovering anomalous events from intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[27] Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha. Latte: L stm self-att ention based anomaly detection in e mbedded automotive platforms. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–23, 2021.