Decision Tree-Based Rule Derivation for Intrusion Detection in Safety-Critical Automotive Systems

Lucas Buschlinger
Roland Rieke
Fraunhofer SIT, Germany
studium@lucas-buschlinger.de
roland.rieke@sit.fraunhofer.de

Sanat Sarda
Fraunhofer
Singapore
sanat.sarda@fraunhofer.sg

Christoph Krauß
Darmstadt University of Applied Sciences
Germany
christoph.krauss@h-da.de

***Abstract*—Intrusion Detection Systems (IDSs) are being introduced into safety-critical systems such as connected vehicles. Since the behavior and effectiveness of measures are validated before approval, the decisions made by an IDS are required to be traceable and the IDS also needs to work efficiently on resource-constrained embedded systems. These requirements complicate the direct use of Machine Learning (ML) approaches in IDS design. In this paper, we propose an approach to using ML to generate rules for an efficient rule-based IDS like Snort. Our approach eases the time-consuming and difficult process of creating a rule set. We use decision trees to generate rules that can be used by experts as a basis for creating a rule set for a specific safety-critical use case. In addition, we use long short-term memory methods to circumvent the problem of limited training data availability, a common limitation in safety-critical systems. Our implementation and evaluation shows the feasibility of our approach to derive specific IDS rules for such systems.**

***Index Terms*—intrusion detection, rule generation, machine learning, anomaly detection, decision trees, long short-term memory, automotive security.**

I. Introduction

A major technology shift is currently taking place in safety-critical systems. Commercial Off-The-Shelf (COTS) systems and connectivity, also with the Internet, are typical characteristics. For example, modern connected vehicles communicate with the outside world and the technologies used in the vehicle are also changing. Automotive Ethernet (AE), for example, is replacing or supplementing classic technologies like the Controller Area Network (CAN) bus. New protocols such as Scalable service-Oriented MiddlewarE over IP (SOME/IP) [1] are becoming standards for service-oriented communication in the automotive sector. SOME/IP messages are transported over UDP or TCP using an IP-based in-vehicle network.

However, cyber-attacks can now also take place via the communication interfaces and security gaps in these new protocols introduce new attack vectors that can eventually access safety-critical vehicle functions [2]. The use of known technologies and protocols also reduces the effort required by attackers. Since even the best security measures can never fully prevent attacks, IDSs are also being introduced into safety-critical systems. These usually work as Network Intrusion Detection System (NIDS) and monitor network communication for attacks.

However, integrating an IDS into safety-critical systems is a major challenge, as they have to meet certain requirements. Two aspects are of particular importance: traceability and efficiency. Traceability is fundamentally important for safety-critical systems, since behavior and effectiveness of measures are validated before getting approval. This also applies to the decisions made by an IDS when classifying attacks, as it is necessary to determine exactly why an alert was generated. Efficiency is important because safety-critical embedded systems mostly have strong resource constraints.

In classical networks, ML-based approaches are used on a regular basis. These often have the advantage that they can also detect unknown attacks by, e.g., learning the normal behavior. However, these frequently represent black box systems, which usually require a lot of resources and whose behavior is not comprehensible. For example, for an IDS in a modern connected vehicle, false positives should not occur. If, for example, an extremely rare case occurs that a message is sent to trigger the airbag (because an accident has actually occurred), the IDS must not classify this as an attack. It must be verifiable that the IDS classifies correctly, which is difficult with ML-based approaches. The resource limitations of embedded systems in a vehicle also make it difficult to use.

Rule-based systems are therefore an alternative, since their behavior is verifiable and the matching of rules requires very few resources (e.g., white-list policy in [2]). However, setting up rules can be very time-consuming and difficult, as in-depth knowledge is required and the rule set can quickly explode.

In this paper, we propose an ML-based IDS which combines rule generation with deep learning data set extension. Our approach is based on Decision Trees (DTs) to generate traceable Snort rules which can be used by an expert as a basis when creating a rule set for a specific safety-critical use case. Furthermore, we incorporate a Long Short-Term Memory (LSTM) network to extend the data set available to the rule generator with the goal of mitigating the common issue of lacking data. Our implementation and evaluation shows that the system is feasible and yields rules significantly outperforming the detection rate of the baseline.

The remainder of this paper is structured as follows. In Section II, we present background and discuss related work before we introduce our proposed IDS in Section III. The implementation is described in Section IV and the evaluation in Section V. Finally, we conclude the paper in Section VI.

II. Background and Related Work

ML-based IDS applications often provide better detection accuracy compared to hand-crafted rule-based IDS, but are rarely deployed in safety critical systems because a lack of insight into IDS results effectively prevents mitigation actions [3]. The most common IDSs (e.g., Snort [4]) use rules to detect intrusions. Rule generation is a laborious task requiring expert knowledge and time. It is, hence, a desirable goal to generate rules semi-automatically or automatically. This has been done by deducting rules from labeled traffic data or from system descriptions. For example, Monzer et al. [5] present a rule generator which converts a system model of a Cyber-Physical System (CPS) into anomaly-based IDS rules. A similar approach by Nivethan and Papa [6] converts behavioral descriptions into Snort rules. But these approaches require an expert describing the system in detail, leading to similar amounts of work as when generating IDS rules by hand. Also, neither of these works fully evaluate their system's effectiveness in detecting attacks.

A more advanced approach is presented by Naik et al. [7]. Using Snort's base rule set, the authors use Fuzzy Rule Interpolation (FRI) to generate rules that are similar to the base rules. This generates a vast number of rules, many of which are not effective in detecting attacks. They further use a Genetic Algorithm (GA) to filter the best new rules and detect attacks missed by Snort. Jin et al. also use FRI for intrusion detection based on Snort's baseline rule set [8]. However, no rules are directly generated and FRI is used for traffic classification. Both studies provide no comparisons of their system's effectiveness to a publicly available data set nor describe their actual rule generation in detail. Gomez et al. [9] present MOEA-Snort, a multi-objective evolutionary algorithm that uses a single aggregate objective function and Pareto-optimization to optimize automatic rule generation in order to detect anomalous traffic.

In more advanced ML rule-generation studies, Rastegari et al. [10] compare *if-then* detection rules generated by their GA implementation with those generated by DTs, k-Nearest-Neighbors (KNN), and GA-based classifiers. For generating IDS rules with ML, mostly DTs are used so far. Fallahi et al. build DTs based on the ISCX 2012 data set with the goal of deriving Snort rules [11]. Their derived rules can achieve high F-measures of up to 0.988. However, a comparison to Snort's baseline performance is missing, as well as details on rule derivation from a DT. A similar approach is taken by Soe et al. in [12], but performance is not evaluated.

Using probability-based ML techniques, Ganesan et al. propose a system for generating Snort rules based on Bayesian abductive reasoning [13]. They built a Bayesian model, which predicts how base Snort rules are most likely to change so that they detect malicious traffic that has not been observed yet. In their evaluation against a Netbios attack included in the MACCDC 2012 data set, the new rules detect more malicious packets. A straightforward comparison to baseline Snort is nonetheless missing.

In [14], Kim et al. present an IDS, in which multiple Deep Learning (DL) stages are combined into a single IDS. Suggestions on alerts generated by AI-IDS to improve or generate new Snort rules are also provided. Ren et al. [15] propose a Random Forest (RF)-based IDS where feature selection is implemented using a GA and its results are evaluated by a RF. Another RF-based anomaly detector is then trained on the optimized data set. This combined system is shown to outperform other ML-techniques like KNN, Support Vector Machine (SVM), and DT. In general, shallow ML-techniques have the advantage of preserving the original data, making results more interpretable. Shah and Issac [4] combine multiple ML-based anomaly detectors as Snort plugins. The plugins based on SVMs, DTs, Fuzzy Logic, and Bayes models analyze traffic in parallel to Snort's regular detector.

Automated rule generation also is combined with ML-based anomaly detection in a hybrid IDS. Kaur and Singh propose in [16] a second-stage LSTM for anomaly detection to feed a signature generator. The generated rules are then given to a first-stage signature-based IDS. Rule generation is done by finding the longest common substring, which appears at least a certain number of times in the malicious packets. However, their evaluation only considers the detection capabilities of the anomaly detector and not the impact of newly generated rules.

In [17], Khraisat et al. propose a hybrid IDS that combines the C5 decision tree classifier (signature) and one-class SVM (anomaly) that provides better results compared to the single algorithms. In [18], Golrang et al. present a hybrid approach where a multi-objective genetic method and an Artificial Neural Network (ANN) run simultaneously to extract feature subsets. Thereafter, the efficiency of the feature subsets is evaluated by a RF ensemble method.

To summarize, many publications lack descriptions of actual rule generation as well as a comparison of detection capabilities with the baseline Snort rules.

III. IDS Design

There are three main components of the proposed IDS. In conjunction, these provide a comprehensive ML-based IDS while combining state-of-the-art DL techniques with traceable rule generation.

a) Rule-Based Detector: The main reasons for using a rule-based detector are the very high traceability of its decisions as well as its reduced complexity and superior computing performance. This suits the typical requirements of safety-critical environments well.

b) Rule Generator: The rule generator is responsible for creating reliable and traceable rules for the rule-based detector and is based on shallow ML techniques. The generator is trained on a known labeled data set and rules are derived either from or by the resulting model. In regular operation, the rule generator is periodically retrained with an extended data set. In this study, the extended data set is obtained by combining the already available data with entries newly classified by the anomaly detector. The goal is to incrementally create better rules for the IDS' detector when a broader database becomes available, as well as rules for new attacks.

c) Anomaly Detector: The anomaly detector extends the data set available to the rule generator. In our study, the anomaly detector is DL-based and trained on known traffic representing the environment the IDS works in. This baseline data set is typically the same as the one used to train the rule generator, at the start. Then, during regular IDS operation, the anomaly detector records and classifies new data traffic intercepted in the IDS's environment. By combining the already available data with the newly classified data, this results in an extended data set for re-training the rule generator.

d) Traffic Logger: The traffic logger records all traffic within the IDS's operating environment. Logging traffic data is needed to extend the data set via the anomaly detector.

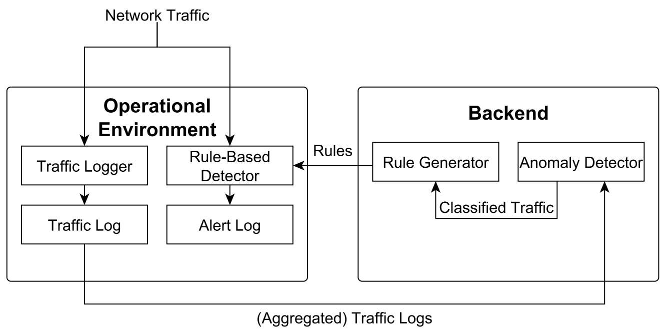

Fig. 1. High-level IDS architecture

In the proposed architecture (see Figure 1), the main components are distributed into two separate locations with varying resources. The detector and traffic logger are installed in the actual operating environment. The more resource-intensive ML-based components, are moved to a backend. This has the advantage that the relation between backend and operational environment (e.g., connected vehicle) is not necessarily 1:1, but can be 1:N. That is, a single backend instance can be responsible for the rule generation for an arbitrary number of detector instances and combine the traffic logs from multiple sources. This way, anomalies seen in one instance of the IDS can already be detected, corresponding rules generated, and published before others are affected.

The network traffic within the IDS operating environment passes through the rule-based detector as well as the traffic logger. The detector applies its rule set to detect potential attacks and generates a log. Taking preventive actions would also be possible, but is not considered here. The traffic log is sent to the backend and pre-processed for the anomaly detector. The pre-processing is implementation-specific and out-of-scope here.

In the backend, the anomaly detector classifies all new traffic received from the traffic logger(s) to extend the available data set. The extended data set is then transferred to the rule generator, for retraining and the generation of a new set of rules. The anomaly detector should be highly accurate to avoid unnecessary False Positive (FP) rules. The newly generated rules are finally transferred to the rule-based detector. This cycle repeats continuously.

IV. IDS Implementation

The IDS implementation can be adapted to different types of data sets. Unfortunately, no suitable data set representing a safety-critical, e.g., automotive, environment was available. We aim to remedy this in a follow-up work and use this proof of concept to demonstrate the feasibility of the IDS's overall design. The UNSW-NB15 [19], a well-known and widely used data set, is thus used for training and evaluating the IDS. The base data set consists of 100 GB of network traffic. Within the data set, fuzzing, network analysis, backdoor, DoS, reconnaissance, shell code and worm attacks, as well as other generic malicious data and exploits are present. The data is split into two parts: the *initial* and *operational* training sets. The first is the data that is initially available to train the anomaly detector and generate the first set of rules. The latter represents data that would be collected by the traffic logger during the operation of the IDS and is used to simulate extending the data set. Hence, this part of the data would not be available in a real-life scenario at first but collected over time. All data labeled as benign (class 0) and malicious (class 1) is used, as we consider intrusion detection as a binary classification problem in this paper. Figure 2 shows the implemented components in this work in relation to Figure 1.

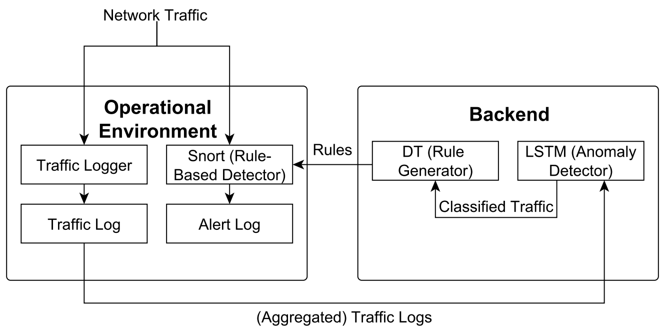

Fig. 2. Implemented High-level IDS architecture

A. Rule Generator

The current state of the art for automated rule generation comprises in particular DTs and GAs. GAs evolve an existing or random baseline rule set, e.g., Snort's standard rules, by changing features of the rules in each generation. Enhanced rules are kept in the current population and the process repeats until a stopping condition is reached. The results are well explainable as a GA changes only the rule's features, which are typically well defined [20]. The GAs can also be tweaked [21]. Overall, GAs are a good option for automated rule generation, but potential for overfitting is still present [20]. However, the rule generator in our implementation is designed using a DT. DTs are the easiest and most explainable method. Trained on labeled network traffic, a DT learns which feature values determine a malicious packet. As every branch is effectively an *if-then* decision, paths within DTs can easily be translated into IDS rules [11]. Mapping the considered features to the IDS's rule structure makes this even easier. The rule generator is implemented in Python using the scikit-learn framework [22].

a) Feature Selection: DTs generally compute the importance of each feature and, hence, perform feature selection automatically. However, using irrelevant features towards the final rule generation incurs an unnecessary increase in the consumption of resources and would also make the tree more prone to overfitting [23]. A mapping between the UNSW-NB15 data set's features and available options for Snort rules [24] is defined in Table I.

TABLE I
MAPPING OF UNSW-NB15 FEATURES AND SNORT RULE OPTIONS

UNSW-NB15 Feature	**Snort Rule Field**
Snort Rule Header	
proto	Protocol
srcip	Source IP address
sport	Source Port
dstip	Destination IP address
dport	Destination Port
Snort Non-Payload Options	
state	(TCP) Flags
smean	dsize
sttl	TTL
is_sm_ips_ports	sameip

As listed, only 9 of the 49 features within the UNSW-NB15 data set can be used for filling fields in Snort rules during rule generation. Among these are all the features necessary for filling the header of Snort rules. Additionally, the *label* feature denoting whether an entry is considered malicious or not has to be used as only malicious entries should lead to alert rules but both classes are present in the DT. Since we use both the malicious and benign data, no extra subclasses for various attack categories are considered for now. The non-payload features, *smean* and *sttl* denoting the mean size of packets from the source and the Time-to-Live (TTL) at the source are used. In the UNSW-NB15 data set, these features are also available for the destination. These were not considered as Snort generally works in a source-to-destination direction. The other remaining features of the data set could also not be matched to the options of Snort rules. These features are very limited in regard to the overall capabilities of Snort and its rule structure, e.g., the inspection of package contents.

b) Training: The data is pre-processed before passing it to the model using the Pandas library [25]. Prior to pre-processing the actual features, the data set is balanced to avoid a potential bias towards the benign class [26], if necessary. Balancing is performed by randomly sub-sampling the majority, i.e., benign class, to match the number of entries in the minority, i.e., malicious class. In this IDS implementation, 50000 entries are selected from the original data set. This is due to memory restrictions, as the DT implementation provided by scikit-learn transforms sparse arrays, used by Pandas, into dense arrays. Sparse arrays result from one-hot encoding categorical data (IP addresses, port numbers, the protocol identifier, and the state flag). This is a necessary step, as the DT implementation by scikit-learn does not support categorical values but only numerical ones, whereas DTs typically are able to handle mixed data (see e.g., [27]). Hence, all non-values, like NAs and empty strings, are replaced with zeros. The DT is then fitted and evaluated using ⅔ of the data for training and the remainder for testing, respectively. Predictions, i.e., classifications, of the testing data are obtained using the model's class `predict` function. Performance measures are calculated through scikit-learn's performance scoring functions based on the predicted and actual labels.

c) Rule Derivation: Rule derivation is divided into two tasks, the first is to get the decision paths from the tree by traversing it and the second to compose the rules from the information held by the decision paths. Note that we are only creating single-packet rules in this work. Nonetheless, multi-packet rules considering consecutive packets are an option within Snort and are of interest for our further developments. All possible decision paths are obtained by traversing the structure using scikit-learn's `decision_path` function on the DT in conjunction with the data used for fitting the tree, i.e., the training set. The decision paths list the indices of the nodes passed while traversing the tree during a decision process. Using the list of indices, the tree can now be traversed to obtain the values of the decision paths. Listing 1 sketches the algorithm used in pseudo-Python.

Listing 1
TRAVERSING THE DT

```
1 for node_index in node_indices[:-1]:
2   DP_features.append(tree.feature[node_index])
3   DP_thresholds.append(
        tree.threshold[node_index])
4   if DP[node_index+1] ==
          tree.children_left[node_index]:
5     DP_decisions.append(0)
6   elif DP[node_index+1] ==
          tree.children_right[node_index]:
7     DP_decisions.append(1)
8 DP_verdict = max(tree.value[node_indices[-1]])
```

In lines 1 through 7, the algorithm iterates over the node indices of the decision path, except the last node, i.e., the leaf, as it does not contain a decision. In line 2, the features of all nodes along the path are stored (e.g., *proto* or *sttl*) and in line 3 the thresholds to either go left or right are stored. The decision in the current node is then checked in lines 4-5 and 6-7, respectively. If the decision is to go to the left child (i.e., the feature is less than or equal to the threshold), this is denoted with a zero and if it is to go right (i.e., the feature is greater than the threshold), its denoted with a one. Lastly, the verdict of the decision path, i.e., whether or not this path leads to a classification as benign (class 0) or malicious (class 1), is stored. The information obtained from the decision paths is converted to a dictionary containing a mapping between the features and their decision. The mapping has the form `feature name : [threshold value, decision]`. An unordered dictionary is sufficient here as the order of decision is not important for deriving rules. To end the traversal step, paths with a benign verdict are dropped. As the last step, Snort rules are composed from the decision paths by mapping the features and their decisions in the DT's nodes to the possible options in the rules. Numerical value features like *TTL* and *dsize* can easily be mapped, as they only appear once per path and the decision of whether this feature is less-equal or greater is supported by Snort (e.g., *ttl:<55;*). Categorical features can either be mapped to their equivalents in Snort rules or combined to lists. The protocol and state features have corresponding equivalents in Snort, although the UNSW-NB15's proto field contains more diverse identifiers. TCP is simply set as the protocol in the rule, when the value from the decision path is not directly supported by Snort. Similarly, state flags are only supported for TCP and are ignored if the decision path contains a negated or unsupported flag.

Most importantly, every Snort rule needs to fill the rule header with the appropriate values (see also Table I) for the source and destination addresses and ports. As IP addresses and port numbers are identifiers, these are also categorical. Negated addresses and ports are supported by Snort. Therefore, all IPs are joined to lists for the source and destination IPs. The same is implemented for ports. In case no value for one of these four fields is present in the decision path, Snort's *any* wildcard is used. Full wildcard rules consisting only of *any* values in the header are removed, as they would alert on every packet. Lastly, the generated rule set is filtered for duplicates and written to a rule file to be passed to Snort. Listing 2 shows two exemplary rules derived from DTs. Note that non-detecting options like the msg, sid, and priority are simply set to default values.

Listing 2
EXEMPLARY RULES DERIVED FROM DTS

```
alert tcp [175.45.176.2] [63167] ->
    [149.171.126.14] [179] (msg:"Alert rule
    derived from Decision Tree"; sid:3000006;
    priority:1;)

alert tcp [175.45.176.1] [34180, !21170] ->
    [149.171.126.17, !10.40.182.255,
    !149.171.126.13] [179] (msg:"Alert rule
    derived from Decision Tree"; sid:3000034;
    priority:1; dsize:>52; ttl:>61;)
```

Figure 3 shows a small examplary DT ready for rule derivation. As can be seen, the first path to the right would lead to class `y[0]`. This path is considered to be benign and will, hence, not be converted into a rule. Consequently, only paths ending in a leaf node with `class = y[1]` will result in an alert rule. In Figure 3, this means the resulting rule would be a combination of source IP not being 175.45.176.2, destination port being 63167, and destination IP not being 175.45.176.3. As a Snort rule this would be `alert tcp [!175.45.176.2] any -> [!175.45.176.3] [63167] (msg:"...")` (source port is set to `any` as none was given in the DT's path).

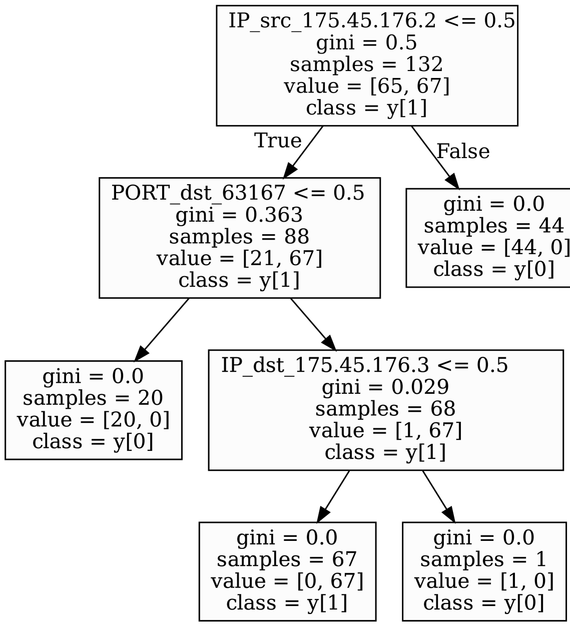

Fig. 3. Examplary DT ready for rule derivation

B. Anomaly Detector

Although in Deep Neural Networks (DNNs) in general the inner workings of the model remain hidden and traceability of results is difficult, the performance in intrusion detection outperforms that of shallow techniques [28], [29]. Convolutional Neural Network (CNN)-based anomaly detectors perform slightly better than regular DNNs [28], [29]. Recurrent Neural Networks (RNNs) include cycles and can capture context, e.g., temporal dependencies. It has been shown by Taylor et al. [30] that with respect to detection of automotive cyber-attacks RNNs are vastly superior to the other methods with LSTMs having the overall best average performance. Since we aim for intrusion detection in the area of safety-critical systems such as networked vehicles, we assume that LSTMs is the most suitable type of RNNs for this. Thus, for the evaluation of our concept, a LSTM is implemented in Python using the Keras [31] and Tensorflow [32] libraries.

a) Feature Selection: For the LSTM, only the *attack_cat* feature (nominal name of each attack category) is dropped from the data set as anomaly detection is considered as a binary classification problem here. Besides that, the LSTM automatically selects the most relevant features and weighs them accordingly. Although selecting features would reduce the complexity of the model and resource consumption, this becomes less significant, as this component is designed to be implemented in the backend with vast resources.

b) Training: Training the LSTM is straightforward and performed on the *initial* training data set. After evaluation, the LSTM's architecture of a single hidden layer with 128 LSTM units, together with Sigmoid as the activation function, binary cross-entropy as the loss function, and Adam as the optimizer was selected. Before passing the data set to Keras for the actual training, the data set is loaded and pre-processed using Pandas. However, pre-processing is kept to a minimum, and only the IP addresses, the protocol identifier, the service identifier, and the state flags contained within the UNSW-NB15 data set are transformed to integer values and non-integers replaced with zeros. During training, 10-fold cross-validation is used to validate the model, the performance of which is retrieved using Keras `evaluate` function.

c) Traffic Data Classification: For classification, the new unseen traffic data collected in the operational environment is passed through the LSTM. We assume that the raw traffic data has already been processed and prepared (e.g., removing data already recognized as malicious by the IDS by filtering based on the alert log) for use in the anomaly detector. In this implementation, this is simulated by using the *operational* training data set. This data is loaded and prepared in the same way as described above but not filtered.

To classify the data, the stored LSTM model is loaded from memory and the data, without the labels, passed to the model's `predict` function. This returns a value between 0 and 1 due to the nature of the Sigmoid function. Hence, predicted labels are derived by rounding to either 0 or 1. As the actual labels for this data are available, which would not be the case in a real implementation, the performance of the final model is also evaluated on this unseen data. Finally, the predicted labels are put into the *operational* training data set, replacing the actual labels, and the data set is written to a CSV file as the *LSTM classified* data set. This latter and the *initial* training set represent the extended data.

C. Rule-Based Detector

As the actual component responsible for detecting malicious traffic was not the main focus of this IDS implementation, Snort version 3 [33] was deployed. Snort has the advantage of being a well-performing rule-based IDS, even in resource-constraint devices [34], and fits our use case well. Many basic rule sets are already available, making the evaluation of the detector's baseline performance straightforward.

To assess the performance of Snort, either with the baseline rule set as well as the generated rules, the alerts have to be matched with the ground truth. Hence, some kind of information about whether or not a packet is malicious has to be encoded within the data. As Snort's detection mechanisms work on and above Layer 3, Media Access Control (MAC) addresses can be used for this. Hence, the original PCAP files of all data sets were augmented with MAC addresses indicating malicious and benign packets. We implement this by matching the IP addresses, ports, and timestamps of packets with the ground truth table provided in the UNSW-NB15 data set. If a packet matches an attack or malicious record in the table, its destination MAC address is set to the `DE:AD:BE:EF:00:00` magic value. The address is zeroed, in case no match is found.

The logs are generated using Snort's CSV alert mode with the corresponding features. For evaluation, all duplicate alerts are removed first. Then, all alerts containing the magic value `DE:AD:BE:EF:00:00` MAC address are counted as True Positives (TPs), as they are alerts for malicious packets. In contrast, all other alerts are counted as FPs, since these alerts were generated for benign packets. Furthermore, False Negatives (FNs) are calculated by getting the difference between the known total number of malicious packets and the TPs ($|FN| = |malicious| - |TP|$). The True Negatives (TNs) are the remainder of all unaccounted packets, i.e., the difference between the total number of packets and the sum of the TPs, FPs, and FNs ($|TN| = |packets| - (|TP| + |FP| + |FN|)$).

V. IDS Evaluation

This section presents the evaluation of the IDS design and its proof-of-concept implementation.

A. ML System Accuracy

In the following, we analyze how well our individual ML components generalize to the data at hand. The ML-specific accuracies generally fall in line with those that the related works (e.g., [11], [28], [29]) have observed. The accuracies have been evaluated on the following testbed, simulating the backend: For hardware, an AMD Ryzen 5 2600X with 16 GB of main memory, the testbed's operating software is Windows 10, Python 3.8.5 within Windows 10's Linux subsystem (WSL) in version 2 is used to run the Python code and the Python libraries viz. Pandas 1.1.5, Numpy 1.18.5, scikit-learn 0.23.2, Keras 2.4.3, and Tensorflow 2.3.1 are used.

a) DT Rule Generator Accuracy: To evaluate the DT, each combination of training, traversing, and deriving rules was repeated 10 times. The results are the averages of these individual outcomes. Full k-fold validation was not feasible due to the aforementioned memory restrictions.

Overall, the DT works well on the UNSW-NB15 data set. As shown in Table II, accuracy, precision, and recall are all over 98%. The F_1-scores are continuously larger than 0.99, too. Extending the initial data set with data classified by the anomaly detector demonstrates a beneficial effect on the overall accuracy. This largely is because of a better overall distribution of the data selected for training. Selecting all features as mapped instead of only those matching the rule header provides minor differences. Since the IPs and ports have to be one-hot encoded, these typically already comprise up to 32000 features. Adding four more features is, therefore, only a minor difference.

While these scores are excellent, FPs are still present. This potentially results in FP rules being created and benign packages being classified as malicious. In principle, the same is true for FNs, but no rules for letting benign traffic pass are created. Nonetheless, FNs can result in missing rules for data wrongly classified as benign. These points emphasize the need for a highly accurate model as the basis for rule generation.

In terms of computational performance, listed in Table III, the training and processing times of the DT during rule derivation are good for a proof-of-concept implementation. Note that these performance numbers represent the systems

TABLE II
ML-PERFORMANCE OF THE DT WITH DIFFERENT UNSW-NB15 DATA SETS AND FEATURES

Features	Data set	Accuracy	Precision	Recall	F_1-Score
Header	Initial	99.183%	98.573%	99.809%	0.991
Header	Extended	99.795%	99.732%	99.860%	0.998
All	Initial	99.210%	98.744%	99.685%	0.992
All	Extended	99.832%	99.857%	99.808%	0.998

that are intended to run on a powerful backend. These numbers can thus only indicate the potential load on said backend and do not represent the computational performance of the IDS within its operational environment, e.g., a safety-critical automotive controller. Nonetheless, the overall performance of the backend directly influences, for example, the minimum period in which rules can be updated. Interestingly, using all of the mapped features increases the times taken significantly. Moreover, training the DT with the extended data set results in a broader but shallower tree. This means that the DT also generates more decision paths. This explains the improved performance discussed above, as the DT generalizes better. Nonetheless, it has to be noted that this has to be due to a better distribution within the data itself.

TABLE III
COMPUTATIONAL AND TEMPORAL PERFORMANCE OF THE DT ON UNSW-NB15

Features	Data Set	Training	Traversal	Depth	Leaves
Header	Initial	3.896s	1.952s	45	52
Header	Extended	4.202s	1.877s	23	79
All	Initial	28.900s	8.841s	62	49
All	Extended	35.400s	10.604s	24	61

b) LSTM Anomaly Detector: The LSTM works well on learning and classifying the UNSW-NB15 data set. Table IV shows the performance of the LSTM model during training and when classifying the unseen *operational* training set. During training, accuracy, precision and recall is high, over 98%, and loss is very low at just 0.022. The excellent F_1-score is 0.993. When classifying new data, i.e., the *operational* training set, accuracy is very good with 98.671%. Recall is excellent at 99.996%, which implies that only 8 out of 215260 malicious entries were not detected. In contrast, precision of 87.432% depicts that a larger number of benign entries, 30941 out of 2081800, were mis-classified as malicious. Overall, the F_1-score of 0.933 indicates that the LSTM is working well for anomaly detection based on binary classification.

B. Detection Performance of Generated Rules

For evaluation, two experiments are performed. In each experiment, rules are generated using the *initial* training data set and its extension with the *LSTM classified* data set. For each of the data sets, rules with and without negated field values, are generated. This results in four types of rules for

TABLE IV
LSTM PERFORMANCE ON UNSW-NB15 DATA SET

Mode	Accuracy	Precision	Recall	F_1	Loss	Time
Train	99.279%	98.625%	99.952%	0.993	0.022	475.100s
Classify	98.671%	87.432%	99.996%	0.933	-	996.748s

each experiment: initial training set without negated fields, initial training set with negated fields, extended classified data set without negated fields, and extended classified data set with negated fields. Rules are generated and evaluated five times per experiment, as the DT implementation is not deterministic.

a) Baseline Rules: To measure the baseline rule set performance, the network traces are evaluated using Snort's community rules [35]. Only a fraction of malicious packets is detected with a recall of 2.228%, precision of 2.237%, and a F_1-score of 0.022. A high accuracy of 98.150% shows that this metric alone is not very meaningful.

b) Header-Only Rules: In a first experiment, rules are only generated by considering the IP, port, and protocol fields of Snort, i.e., the header fields. Figure 4 shows that a high accuracy of over 97% is achieved for each of the four rule set types. Except for the extended data set with negated rules, precision, recall, and F_1-scores are over 60%, 65%, and 0.63, respectively. The fourth type of rule performs slightly worse, with a precision of 41.219% and an F_1-score of 0.45. Recall is equal and, hence, the same number of malicious packets are detected but more benign packets are being misclassified. Overall, these types of rules provide a significantly better detection performance than the baseline community rule set but extending the classified data set is not beneficial.

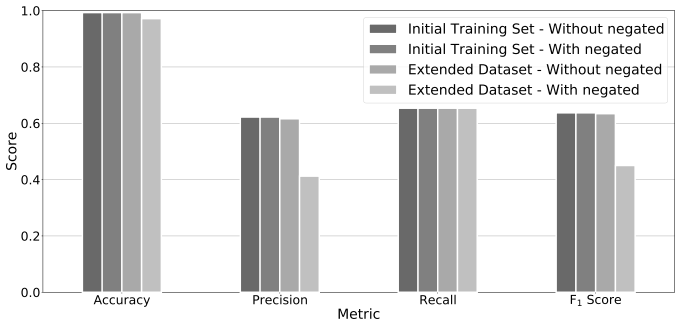

Fig. 4. Classification results - Header-Only Rules

c) Full Rules: In a second experiment, all features in Table I are considered. However, this leads to a significant decrease in performance compared to header-only rules. One possible reason is that the non-payload rule options are too generic for precise detection and simply fit too many packets.

d) Comparison to Other Works: Other works implementing intrusion detection for the UNSW-NB15 dataset have achieved similar results. In their work [36], Faker et al. use several techniques, DNN, RF, and Gradient Boosted Tree.

For binary classification, they are able to achieve accuracies of 99.19%, 98.86%, 97.92%, respectively. Unfortunately, no other metrics were reported. In [15], Ren et al. implement an RF-based IDS with multiclass class classification. Their system reaches accuracies of over 93% for all classes. However, precision, recall, and F_1-scores vary widely between the attack classes and range from less than 10% for the analysis attack class to over 95% for the generic attack class. Averaged, the system reaches precision, recall, and F_1-scores of around 60%, similar to what is presented in this work.

C. Discussion

In general, our proof of concept demonstrates that the IDS works as intended. While a full *live* demonstration using alternate network traffic captured during operation was not possible within the scope of this first proof of concept, we intend to deploy a further prototype in a real-world scenario (e.g., a connected vehicle) to validate the aspect of updating rules in the long run. Our implementation also shows that distributing more resource-intensive components to a vastly more powerful backend makes sense. The performance shows that the designed IDS is a relatively good classifier but using the generated rules, in a realistic scenario, may not be as straightforward as in this proof-of-concept implementation. Whereas the baseline rule set is not performing well, with a very low detection rate, the designed ML-IDS generated rules alert on far more malicious packets. This is shown by the up to 30 times higher F_1-score and recall values compared to baseline performance. Such an increase can be expected, as the community rules are very general and our generated rules are tailored specifically to the data.

However, the precision scores of maximum 60% show a large number of false positives are also generated. This is also reflected in the F_1-scores, which reach a maximum of 0.63 when using the header-only rules. The inclusion of negated rule options (e.g., IPs or ports) is also not beneficial. As it appears that Snort ORs the entries within an array of, e.g., IPs, having just a single IP matched fulfills the condition for the array. This is easily the case with negated values, as only one such IP or port can be present in a packet's header. Thus, incorporating negated values is not helpful.

So far, the additional data classified by the LSTM anomaly detector does not increase performance. This is attributed to the fact, that the training data for the anomaly detector (LSTM) and rule generator (DT) come from the same data set and, hence, have the same characteristics. In the future development of our system, we have to investigate training the components with differing data. Furthermore, the translation from the ML components into rules has to be vastly improved. Both the LSTM and DT are performing exceptionally well on their own, but the resulting rules do not reach similar levels yet. The overall performance hence suggests to conduct a more precise analysis of the generated rules before deploying, especially considering the requirements of safety-critical systems.

The most prevalent limitation of the IDS is definitely the too low performance. The detection and FP rates shown in Section V-B would not be suitable to the targeted safety-critical environment. Observed limiting factors are the, comparably, low number of usable features from the data set and the lack of payload-based features. We, hence, suggest looking into GAs for payload-based rules as they can work on this type of data but also on the categorical data used herein. Similarly, training the anomaly detector (in this case the LSTM) as a binary classifier limits it to only recognizing the attacks it has been trained with. Training the anomaly detector to only know legitimate traffic, which is known especially in safety-critical environments, can therefore be a better option. For example, this can be realized by deploying an autoencoder as the anomaly detector and training it on the benign traffic.

In comparison to the related work, the IDS provides a novel approach but shares similarities with some of the works. The overall design is similar to the one presented by Kaur and Singh [16]. In their design, the LSTM is a second stage used for detecting anomalies that have slipped past the first rule-based component. Furthermore, their rule generation uses substring matching to generated *if-then* rules instead of using ML for generating Snort rules. Offloading the computationally more expensive ML-based operations to a backend is similar to the approach presented by Loukas et al. [37]. However, their system fully moves intrusion detection to the cloud, making it dependent on the car being online. The system proposed in this paper can also perform intrusion detection when the car is offline.

VI. Conclusion

We propose a multi-stage ML-based IDS which combines a traceable DT-based rule generator with a DL-based anomaly detector for extending the data set used by the generator. It is evaluated with the well-known UNSW-NB15 data set. Our approach is adapted to the requirements of safety-critical systems, particularly traceability and efficiency. As an example, we had the safety-relevant systems in, e.g., a vehicle using automotive Ethernet in mind. Overall, the performance shows the IDS can be implemented and effectively deployed for the target use-cases. The production-ready IDS solution, will however be more complex and require deeper analysis of created rules. Experts can use the generated rules as input when creating a new rule set, e.g., in scenarios like vehicular Ethernet networks where no rules are available yet. The easy interpretability of DTs also allows traceable results to be generated. Traceable decisions will allow developers to determine which rules fired and how they were generated. The use of a DL-based approach to expand the data set of the rule generator could, however, be viewed as too opaque, since errors spread to the generated rules and ultimately lead to incorrect classifications by the IDS. As future work, we intend to demonstrate the adaptability of our solution to different types of data sets, such as traces from automotive Ethernet networks.

ACKNOWLEDGMENT

This research work has been partly funded by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE and by the BMBF project VITAF (ID 16KIS0835). Additionally, the project leading to this application has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 883135 (E-Corridor).

REFERENCES

[1] AUTOSAR. (2019, 11) SOME/IP protocol specification. Last accessed on 2021-07-29. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/foundation/19-11/AUTOSAR_PRS_SOMEIPProtocol.pdf

[2] T. Gehrmann and P. Duplys, "Intrusion detection for some/ip: Challenges and opportunities," *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pp. 583–587, 2020.

[3] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 305–316.

[4] S. A. R. Shah and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to snort system," *Future Generation Computer Systems*, vol. 80, p. 157–170, 03 2018.

[5] M. H. Monzer, K. Beydoun, and J. Flaus, "Model based rules generation for intrusion detection system for industrial systems *," in *2019 International Conference on Control, Automation and Diagnosis (ICCAD)*, 2019, pp. 1–6.

[6] J. Nivethan and M. Papa, "Dynamic rule generation for scada intrusion detection," in *2016 IEEE Symposium on Technologies for Homeland Security (HST)*, 2016, pp. 1–5.

[7] N. Naik, R. Diao, and Q. Shen, "Dynamic fuzzy rule interpolation and its application to intrusion detection," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 4, pp. 1878–1892, 2018.

[8] S. Jin, Y. Jiang, and J. Peng, "Intrusion detection system enhanced by hierarchical bidirectional fuzzy rule interpolation," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 6–10.

[9] J. Gómez, C. Gil, R. Baños, A. Márquez, F. Montoya, and M. Montoya, "A pareto-based multi-objective evolutionary algorithm for automatic rule generation in network intrusion detection systems," *Soft Computing*, vol. 17, 07 2012.

[10] S. Rastegari, P. Hingston, and C.-P. Lam, "Evolving statistical rulesets for network intrusion detection," *Applied Soft Computing*, vol. 33, pp. 348 – 359, 2015.

[11] N. Fallahi, A. Sami, and M. Tajbakhsh, "Automated flow-based rule generation for network intrusion detection systems," in *2016 24th Iranian Conference on Electrical Engineering (ICEE)*, 2016, pp. 1948–1953.

[12] Y. Soe, Y. Feng, P. Santosa, R. Hartanto, and K. Sakurai, "Rule generation for signature based detection systems of cyber attacks in iot environments," *Bulletin of Networking, Computing, Systems, and Software*, vol. 8, no. 2, 2019.

[13] A. Ganesan, P. Parameshwarappa, A. Peshave, Z. Chen, and T. Oates, "Extending signature-based intrusion detection systems with bayesian abductive reasoning," 2019.

[14] A. Kim, M. Park, and D. H. Lee, "AI-IDS: Application of deep learning to real-time web intrusion detection," *IEEE Access*, vol. 8, pp. 70 245–70 261, 2020.

[15] J. Ren, J. Guo, W. Qian, H. Yuan, X. Hao, and H. Jingjing, "Building an effective intrusion detection system by using hybrid data optimization based on machine learning algorithms," *Security and Communication Networks*, vol. 2019, p. 7130868, 06 2019.

[16] S. Kaur and M. Singh, "Hybrid intrusion detection and signature generation using deep recurrent neural networks," *Neural Computing and Applications*, vol. 32, no. 12, pp. 7859–7877, 06 2020.

[17] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, and A. Alazab, "Hybrid intrusion detection system based on the stacking ensemble of c5 decision tree classifier and one class support vector machine," *Electronics*, vol. 9, no. 1, 2020.

[18] A. Golrang, A. M. Golrang, S. Yildirim Yayilgan, and O. Elezaj, "A novel hybrid ids based on modified nsgaii-ann and random forest," *Electronics*, vol. 9, no. 4, 2020.

[19] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.

[20] R. H. Gong, M. Zulkernine, and P. Abolmaesumi, "A software implementation of a genetic algorithm based approach to network intrusion detection," in *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Network*, 2005, pp. 246–253.

[21] J. Gómez, C. Gil, R. Baños, A. L. Márquez, F. G. Montoya, and M. G. Montoya, "A pareto-based multi-objective evolutionary algorithm for automatic rule generation in network intrusion detection systems," *Soft Computing*, vol. 17, no. 2, pp. 255–263, 02 2013.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, "Intrusion detection systems for intra-vehicle networks: A review," *IEEE Access*, vol. 7, pp. 21 266–21 289, 2019.

[24] Cisco - The Snort Project. Snort users manual - rules headers. [Online]. Available: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node29.html

[25] The pandas development team. (2020, Feb.) pandas-dev/pandas: Pandas. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[26] M. Gharib, B. Mohammadi, S. H. Dastgerdi, and M. Sabokrou, "Autoids: Auto-encoder based method for intrusion detection system," 2019.

[27] The Mathworks Inc. Splitting categorical predictors in classification trees. [Online]. Available: https://mathworks.com/help/stats/splitting-categorical-predictors-for-multiclass-classification.html

[28] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, p. 102419, 2020.

[29] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences*, vol. 9, no. 20, p. 4396, 10 2019.

[30] A. Taylor, S. P. Leblanc, and N. Japkowicz, "Probing the limits of anomaly detectors for automobiles with a cyber attack framework," *IEEE Intelligent Systems*, vol. PP, no. 99, pp. 1–1, 2018.

[31] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[32] TensorFlow Developers. (2021, Dec.) Tensorflow. [Online]. Available: https://doi.org/10.5281/zenodo.5799851

[33] The Snort development team. Snort3. Last accessed on 2022-01-24. [Online]. Available: https://github.com/snort3/snort3

[34] T. Zitta, M. Lucki, L. Vojtech, M. Neruda, and L. Mejzrova, "Experimental load test statistics for the selected ips tools on low-performance iot devices," *Journal of Electrical Engineering*, vol. 70, no. 4, pp. 285–294, 08 2019.

[35] The Snort community. Snort community rules. Last accessed on 2021-01-12. [Online]. Available: https://www.snort.org/downloads/registered/snortrules-snapshot-3034.tar.gz

[36] O. Faker and E. Dogdu, "Intrusion detection using big data and deep learning techniques," in *Proceedings of the 2019 ACM Southeast Conference*, ser. ACM SE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 86–93.

[37] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, and D. Gan, "Cloud-based cyber-physical intrusion detection for vehicles using deep learning," *IEEE Access*, vol. 6, pp. 3491–3508, 2018.