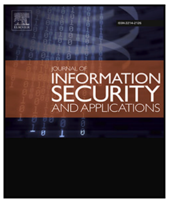
Journal of Information Security and Applications 82 (2024) 103736

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa



Automatic decision tree-based NIDPS ruleset generation for DoS/DDoS attacks

Antonio Coscia[a], Vincenzo Dentamaro[b], Stefano Galantucci[b,*], Antonio Maci[a], Giuseppe Pirlo[b]

[a] Cybersecurity Laboratory, BV TECH S.p.A., Milan, 20123, Italy
[b] University of Bari Aldo Moro, Bari, 70125, Italy

A R T I C L E I N F O

Keywords:
Distributed denial of service
Network-based intrusion detection prevention system
Decision tree
Suricata/snort
Hyperparameter tuning
Automatic rule generation

A B S T R A C T

As the occurrence of Denial of Service and Distributed Denial of Service (DoS/DDoS) attacks increases, the demand for effective defense mechanisms increases. Recognition of such anomalies in the computer network is commonly performed through network-based intrusion detection and prevention systems (NIDPSs). Although NIDPSs allow the interception of all known attacks, they are not robust to the continuing variation over time of DoS/DDoS anomalies. The machine learning (ML) paradigm provides algorithms that can effectively reduce concept drift due to the evolution of cyber threat data patterns. These methodologies can be exploited for creating effective rules suitable for popular NIDPS engines such as Suricata.

This paper proposes a new algorithm called *Anomaly2Sign*, which automatically produces rules for Suricata through an automatic Decision Tree (DT)-based generation process. The DT is trained on both anomalous and legitimate traffic, allowing the generation process to select anomaly features that can be mapped within the generated rule structure. Additionally, the DT hyperparameters are tuned at execution time to generate a minimal ruleset capable of detecting the largest number of anomalous packets. The proposed algorithm achieves classification metrics in the range of 99.7%–99.9% using the BOUN-DoS and BUET-DDoS datasets, outperforming the compared ML classifiers, i.e., Logistic Regression, Support Vector Machine, and Multi-Layer Perceptron. Furthermore, the leveraged DT model requires a shorter training and prediction time than the previously cited benchmark classifiers. To enforce the selection of the DT model, an analysis of model complexity is undertaken, including the evaluation of the Akaike Information Criterion (AIC) score. As a result of such an evaluation, the DT model achieved the lowest AIC score among the compared approaches denoting its low complexity. Finally, Anomaly2Sign has been compared with Syrius, i.e., an alternative state-of-the-art automatic NIDPS rules generator, obtaining better performance for detection rate and execution time.

1. Introduction

Nowadays, guaranteeing the security of information and communications technology (ICT) systems has emerged as a primary concern to defend both companies and national borders. Identifying, investigating, analyzing, and responding to cyber threats and attacks is imperative to ensure effective cyber security [1]. In cyberspace, malicious actors use several methods to assault their targets. According to the comprehensive review proposed in [2], some of the commonly deployed attacks are: (i) phishing; (ii) malware; (iii) man-in-the-middle; (iv) denial of service (DoS)/distributed denial of service (DDoS). Regarding the latter form of attack, it consists of flooding victims with a large amount of data that prevents the continuity of service provided by an ICT system. According to the report released by Cloudfare [3], DoS/DDoS attacks targeting different industries have increased in number and intensity in the first quarter of 2023. Detecting DoS/DDoS represents a topical issue for the scientific community as the temporal evolution of this cyberattack hinders the proper functioning of common defense mechanisms that are unable to handle this high volume of data [4]. The detection approaches employed to recognize DDoS attacks are generally categorized as signature- and anomaly-based [5]. These categories encompass intrusion detection and prevention systems (IDPSs), i.e., devices and applications designed to monitor networks and systems for detecting potential threats and performing preventive actions to avoid unauthorized access within the defended network perimeter [6]. Some IDPSs leverage signature engines that can match network traffic by searching for peculiar patterns within the analyzed network packet header or payload. These are known as network-based IDPSs (NIDPSs).

* Corresponding author.
E-mail addresses: antonio.coscia@bvtech.com (A. Coscia), vincenzo.dentamaro@uniba.it (V. Dentamaro), stefano.galantucci@uniba.it (S. Galantucci), antonio.maci@bvtech.com (A. Maci), giuseppe.pirlo@uniba.it (G. Pirlo).

https://doi.org/10.1016/j.jisa.2024.103736

Available online 6 March 2024

Many tools fall into this category, including Snort [7] and Suricata [8] open source engines [9]. In [10], the detection rate achieved by Snort and Suricata was evaluated for different cyber threats. Although the two engines result in a similar average detection rate, Snort outperforms Suricata in the DoS/DDoS detection rate. This evaluation has been confirmed in [11]. On the other hand, Suricata leads to a better result in terms of the computational performance required, such as CPU and memory usage, than Snort. To take advantage of the latter performance, it is preferable to generate Suricata DoS/DDoS rules that can obtain a better detection rate.

Artificial intelligence (AI)-based solutions such as machine learning (ML) and deep learning (DL) algorithms can be employed to deal with cyber threat detection problems because they are promising in terms of the resulting detection rate [12]. According to the surveys proposed in [13], [14], ML algorithms can play a crucial role in this field, enabling scientists to address specific cyber security problems, such as DoS/DDoS detection tasks. In particular, ML approaches are used to learn DoS/DDoS attack patterns to identify these assaults before the disruption of targeted network services [15]. However, according to the review provided in [16], there is a need to combine the potentiality of DDoS detection approaches based on ML in a single cyber security platform, such as an NIDPS engine. This is achievable by explaining the results and output produced by the ML algorithms using a set of methods belonging to the field of eXplainable AI (XAI) [17]. Explaining an ML model aims to provide the logical decision rules inferred by the model itself in a form that is understandable to a human. This property is generally defined as the interpretability of an ML model [18]. In general, the higher the model interpretability, the better the reliability of the prediction system. Thus, high interoperability means being able to understand how an ML model makes predictions, namely, how each data feature contributes to the prediction. As a general rule, when choosing an explainable ML model, a simpler model wins according to Occam's Razor principle [19]. Furthermore, current approaches for explaining decisions made by DL algorithms, such as deep neural networks (DNNs), are unstable [17]. In [20], a Decision Tree (DT) is leveraged to interpret the decision rules extracted to write network intrusion classification rules. However, despite DT models being ideal in the sense of their capability in providing an understandable explanation of the prediction, as for each ML model, decisions inferred by the learner change by updating its hyperparameters. Therefore, given two distinct DT models achieving an equal detection rate, the only one resulting in the less number of extracted rules is preferable, since it would lead to the generation of an optimal ruleset in the sense of minimum number of rules generated.

This paper presents *Anomaly-to-Signature (Anomaly2Sign)*, i.e., an algorithm capable of automatically generating Suricata rules by leveraging the decision rules provided by a DT trained on a baseline containing both legitimate and anomalous traffic, such as DoS/DDoS attack network traffic dumps. To ensure a promising detection rate, the generation process begins if the DT used achieves suitable classification performance, i.e., a detection rate higher than those achieved by Snort rules in detecting DoS/DDoS in [10], otherwise the DT hyperparameters will be updated. Upon generating these rules, Anomaly2Sign checks that they are optimal, and if not, it optimally adjusts the DT hyperparameters so that the generation of a minimum number of rules with the highest detection rate is achieved.

The key contributions provided by this paper are:

- It extends the contribution proposed in [20] as follows:
  - It introduces a feature selection phase that prepares training data to make them suitable for the rule generation process.
  - It introduces a strategy for automatically updating the DT hyperparameters, having two main objectives:
    1. Ensure that the model achieves acceptable detection performance, i.e., comparable with the Snort detection performance on DoS/DDoS in [10].
    2. Generate the optimal ruleset, i.e., the one with the fewest rules and the highest detection rate, based on DT hyperparameter variation for a given hyperparameter space.
- It shows a benchmark analysis highlighting:
  - The timing and classification performance achieved by the same algorithms used for comparison in [20].
  - The complexity of the compared models by evaluating the Akaike Information Criterion (AIC).
  - The execution time required and detection rate achieved by Anomaly2Sign and a state-of-the-art methodology that automatically generates Suricata rules.

Table 1
List of acronyms used in this manuscript.

AI	Artificial Intelligence
AIC	Akaike Information Criterion
AUC	Area Under Receiver Operating Characteristic Curve
CBMP	Cluster-Based Majority undersampling Prediction
CIDR	Classless Inter-Domain Routing
DoS	Denial of Service
DDoS	Distributed Denial of Service
DLL	Data Link Layer
DNN	Deep Neural Network
DNS	Domain Name System
D_R	Detection Rate
DT	Decision Tree
FN	False Negative
FP	False Positive
FPR	False Positive Rate
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IDPS	Intrusion Detection and Prevention System
IP	Internet Protocol
IPS	Intrusion Prevention System
K-NN	K-Nearest-Neighbors
KPCA	Kernel Principal Component Analysis
LICIC	Less Important Components for Imbalanced multiclass Classification
LIME	Local Interpretable Model-Agnostic Explanations
LR	Logistic Regression
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NIDPS	Network-based Intrusion Detection and Prevention System
OS	Operating System
ROC	Receiver Operating Characteristic
RUS	Random UnderSampling
SDN	Software Defined Network
SHAP	SHapley Additive exPlanations
SMOTE	Synthetic Minority Oversampling TEchnique
SVM	Support Vector Machine
T-Link	Tomek-Links
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TPR	True Positive Rate
TTL	Time-To-Live
UDP	User Datagram Protocol
XAI	eXplainable Artificial Intelligence

The rest of this paper is organized as follows. Section 2 provides a theoretical framework for IDPSs and DT models. Section 3 reports a literature survey on: (i) DoS/DDoS anomaly detection solutions based on DT models; (ii) methodologies used to automatically generate NIDPS rules. Section 4 describes the main modules of the proposed algorithm showing its high-level architecture. The methods and materials used during the experimental phase are discussed in Section 5. Section 6 illustrates the experimental results and their critical evaluations. The Anomaly2Sign feasibility is discussed in Section 7. Section 8 presents the conclusions with the main findings and insights. Table 1 summarizes the list of abbreviations used in the manuscript.

2. Background

2.1. Network intrusion detection and prevention system

Network intrusion detection and prevention system (NIDPS) solutions are generally used as a defense line that can efficiently discriminate between legitimate and anomalous traffic. The general architecture of the NIDPS engine presented in [9] consists of a sequence of modules. In particular, the first module is delegated to collect network packets. This type of operation varies depending on the operating mode of the engine. In fact, whether the prevention mode is enabled, packet acquisition is performed in the inline mode because real-time processing must be guaranteed. On the other hand, if the system acts as a detector, the port mirroring (passive) mode is enabled; this allows temporary buffering of packets awaiting processing. Once a packet is captured, a decoder module analyzes the traffic to ensure compliance with network communication standards. Otherwise, the packet would be discarded, generating an alert. Afterwards, a preprocessor module performs defragmentation, reassembly, and session conformance at low-level protocols, whereas high-level plugins validate packets based on several application protocols. In addition to the previous modules, the detection engine module compares the analyzed network packets with the patterns defined in the engine's knowledge base, i.e., the involved ruleset. This component plays a key role as it is the core part of the decision-making process. Therefore, the higher the detection rate guaranteed by the involved ruleset, the better the overall performance of the process. Finally, according to the decision made by the detection engine, an action will be performed, which again depends on the operating mode of the engine. The most common open source NIDPS engines investigated in [9] and [10] are:

- Zeek [21]. This engine supports only detection mode; thus, it acquires network packets in passive mode and provides alerts when a rule matches a message. It is composed of several workers that interact with a manager module that consists of two main modules [9]: (i) an event engine; (ii) a policy script interpreter. The first transforms network packets into event logs to redirect them to the policy interpreter, which analyzes them using the Zeek rules.
- Snort [7]. In contrast to Zeek, Snort supports both detection and prevention operating modes. First, the network packets are sniffed using the Libcap framework; then, the packet is decoded and normalized to make it suitable for the analysis performed by the Snort rules [9]. The first versions of Snort were single-thread [10]. However, as of version 3, such an NIDPS engine supports multithreading [9].
- Suricata [8]. It shares several characteristics with Snort, such as the possibility of operating as both IDP and IPS and multithreading. In addition, it can support multiple detection engines [9].

The NIDPS engine chosen for this study is Suricata, which will be analyzed in depth in the next section.

2.1.1. Suricata

Each detection engine uses a proper detection ruleset to perform network packet analysis. Suricata employs the set of rules provided by the Emerging Threats community [22]. In this section, we focus on rule syntax because the goal of this research is to automatically generate Suricata rules. It uses syntax very similar to Snort. An example of a rule is given below:

```
alert tcp any any -> any any (msg:"A TCP packet was
    detected."; sid:1)
```

According to such an example, it can be divided into three main parts according to the official documentation of Suricata [23]:

- The rule action highlighted in cyan. It determines the action to be performed if the rule is triggered. Generally, if the inline mode is enabled, this field can be set to `drop`, otherwise it is set to `alert`.
- The rule header consists of a series of information such as the protocol (highlighted in red), traffic direction (highlighted in blue), source and destination addresses, and ports (on the left and right sides of the traffic direction are highlighted in green). As a general rule, the fields on the Suricata rule header can assume the following values:
  - Protocol:
    - Application layer, such as `http`, `dns`, `ftp`, `ssh`, etc.
    - Transport layer, i.e., `tcp`, `udp`.
    - Network layer such as `icmp`, `ip` (that stands for all).
  - Source(Destination):
    - Host or subnet (CIDR notation) `IP address`.
    - `any`, that is a general value matching all admissible values.
    - The so-called group variables, such as `$HOME_NET` (`$EXTERNAL_NET`), which include the series of IP addresses typically employed in private(public) networks.
  - Source(Destination) port:
    - A single port or a range of ports.
    - `any` as the same as the above described.
    - A group of ports when they are not contiguous.
  - Traffic direction:
    - `->` to indicate inbound traffic.
    - `<-` to indicate outbound traffic.
    - `<>` to indicate bidirectional traffic.

  Note that it is possible to use logical operators that impose a condition to be met by values instantiated in the rule header.
- The rule options (highlighted in yellow) represent the series of peculiar information to search within the header or the payload of the analyzed packet. There are several optional keywords that can be inserted into a Suricata rule, such as `msg` and `sid`, which represent the description and identifier of the rule, respectively. The option part of the rule can include different keywords depending on the cyber threats it attempts to identify. Because this paper provides an algorithm capable of generating rules to deal with DoS/DDoS anomalies, the following keywords must be considered:
  - `classtype` indicates the category of the rule. For example, in the case of DoS/DDoS, it can be set to `attempted-dos`.
  - `flow` determines whether the network flow should be initiated from the client or the server.
  - `threshold (or detection_rule)`:
    - `count` that is a counter of packets intercepted by the same rule.
    - `seconds` that denotes the time interval to leave the counting active.
    - `type` that denotes the operating mode of threshold keywords. In particular, if set to `threshold`, the rule alerts every count packet in the defined time interval. Otherwise, if set to `limit` the rule alerts at most count times. However, it can be set to `both`, meaning that the two modes discussed above are activated.

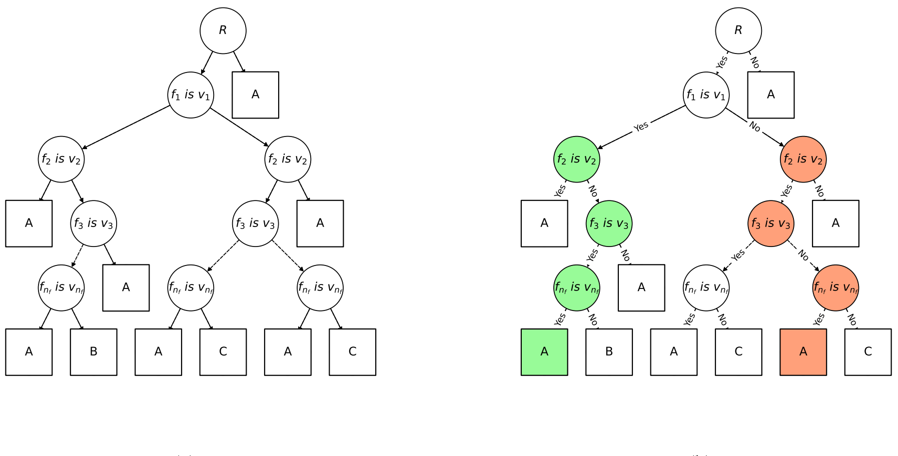


Fig. 1. (a) Decision Tree induction process. (b) Two examples of conditional control statements.

On the other hand, some option fields are not problem-specific:

- `dsize` that denotes the byte size of the packet payload at the transport layer.
- TCP `flags` that can be `S` (SYN), `A` (ACK), `R` (RST), etc.
- `ttl` that quantifies a specific Time-to-Live (TTL) value in the header of the network layer packet.

2.2. Decision Tree

Machine learning (ML) models can be divided into parametric and nonparametric algorithms. An ML algorithm is said to be parametric when the learned function is simplified to a known form consisting of a fixed number of parameters. On the other hand, a nonparametric model makes no assumptions about how the mapping function will be derived [24]. A Decision Tree (DT) model belongs to the class of nonparametric models based on the supervised learning paradigm [20]. This type of model employs a tree-like structure to illustrate the probability of an event occurring according to the training data given as input to the model. From the root node, decisions and leaves are arranged in a top-down tree structure to represent the decision model. Each level of nodes represents a feature in the dataset, while a branch represents a possible value or set of values. In addition to the root nodes providing significant predictors, the leaf nodes determine the final classifications [20]. The aforementioned process that allows the building of DTs from training data is called DT induction [25] and is shown in Fig. 1(a). Such a process can be represented using a conditional control statement, which provides an easy-to-understand and intuitive set of rules for making decisions [20]. Each statement, as highlighted in Fig. 1(b), can be modeled as follows [26]:

IF $\{f_1$ is $v_1 \wedge \ldots \wedge f_{n_f}$ is $v_{n_f}\}$ THEN class A ELSE class B(C)

where f_i with $i = 1, \ldots, n_f$ is a feature within the training set, v_i represents the value assigned to the feature itself, where the chain of conditions to be satisfied is called the antecedent of the rule, while the prediction made is called the consequent. Decision rules generated by a DT are strongly affected by its hyperparameters; therefore, given two DTs tuned in a different manner, these could lead to a different set of decision rules.

2.2.1. Hyperparameters

This section discusses two of the more relevant DT hyperparameters analyzed in this study:

- Splitting Criterion ($\mathcal{S}_c$). By evaluating a statistical metric, this hyperparameter determines how well a given feature value separates training examples building pure partitions according to the target classification. One of the following two indexes is commonly selected for such a purpose:
  - Entropy. This index can be used to determine the impurity of any set of data X corresponding to a DT node. Let n be the different values that the target variable can assume, and the entropy of X corresponding to the n-wise classification is defined as follows [20]:

    $$H(X) = -\sum_{z=1}^{n} p_z \log_2(p_z) \tag{1}$$

    where p_z indicates the likelihood that the set of data considered belongs to class z. This measure is typically used to define the information content of a tree node to continue (or not) the splitting procedure. Specifically, the most informative attribute search continues until nodes resulting in null entropy are reached, i.e., the lower the entropy, the higher the information gain.
  - Gini. This index defines the purity of a set of data after being split along a particular attribute. In particular, the better the splitting, the more pure the data within the resulting sets will be. It is calculated as follows [27]:

    $$G(X) = 1 - \sum_{z=1}^{n} p_z^2 \tag{2}$$

    Given an attribute on which a split results into two new sets X_1 and X_2 are obtained, the reduction of the impurity due to such a split can be evaluated as:

    $$\Delta G(v) = G(X) - \left(\frac{|X_1|}{|X|} G(X_1) + \frac{|X_2|}{|X|} G(X_2) \right) \tag{3}$$

    where $|X|$, $|X_1|$ and $|X_2|$ represent the number of samples in X, X_1 and X_2 respectively.

- Maximum depth of DT (d_{MAX}). This hyperparameter defines the maximum depth that the DT can reach. In a limited case, e.g., no null entropy nodes are met before, the splitting process is performed until the maximum depth of the tree is reached. Let $|D_{TR}|$ be the number of samples within the training data, d_{MAX} is limited to the maximum theoretical size $|D_{TR}| - 1$. However, it is reasonable to expect that this limit will never be reached, since if this condition is satisfied, the model suffers from overfitting, as shown by Mantovani et al. [28]. In fact, the more a tree grows in depth, the more complex the model becomes, since more branches will occur; therefore, it will acquire more information about the data and there will be a growth in the overall variance.

3. Related work

3.1. Decision Tree for DoS/DDoS detection

DT is an ML model widely used in the literature to deal with the DoS/DDoS detection task. In [29], an analysis of several ML algorithms that perform the DoS/DDoS classification task was conducted, showing that among the benchmarked methods, DT can achieve the highest classification accuracy score. A similar analysis is performed in [30], where the DT model was compared with a K-Nearest-Neighbor (K-NN) model. As a result of this analysis, DT outperforms K-NN in both classification accuracy and execution time, i.e., training and prediction time. In [31], authors have taken advantage of a DT model to present a low computational overhead system capable of detecting DDoS flooding attacks. This was achieved by selecting a minimal number of features using a low-variance filter. The results obtained denote the effectiveness in minimizing the CPU load required while retaining a promising accuracy value. In [32], several tree-based classifiers were evaluated in the DDoS attack detection task using the CICDS2017 dataset. Using an efficient feature selection strategy, the best solution among the compared has been the partial DT algorithm, both in terms of execution time and classification accuracy. In [33], a DT model driven by information gain is used to detect DDoS attacks in real time. The experimental phase shows the effectiveness of the model in building a DT capable of identifying the source of an attack among a large volume of distributed traffic. This is achieved because of the pruning mean tree strategy, which guarantees robustness of the model even in the case of noise. In [34], a DT model is leveraged to address DDoS detection tasks. In particular, the DT is trained using the CICDDoS2019 dataset, achieving a higher classification accuracy than other ML classifiers. In [35], a DDoS detection framework for a wireless network based on random forest and J48 is proposed. Saikat et al. [36] try not only to provide a defense mechanism for DDoS but also to support detection by providing decision explanations. The interpretable models used are Local Interpretable Model-Agnostic Explanations (LIME) and SHapley Additive exPlanations (SHAP). To build them, an ML model is required for learning purposes, such as DT, random forest, or logistic regression. LIME using DT as an explainable model has one of the highest likelihood scores. This result denotes the effectiveness of the interpretability property of DT. The methodology proposed by Ahmim et al. in [37] introduces an NIDS mechanism that combines tree-based classifiers. It is a multi-stage methodology within which the first and second classifiers receive the network traffic dataset as input and classify it as malicious or legitimate. The third classifier refines the result of the detection mechanism using the same data, features, and results as the other two classifiers. The final result provides an effective alerting mechanism but at the cost of using three classifiers and a more complex application architecture. In [38], a DT classifier was selected to perform the intrusion detection task combined with the feature selection strategy proposed by the authors. The leveraged DT, i.e., an ID3, has been evaluated using the KDD Cup 99 dataset, which consists of a series of network anomalies, including DoS attacks. The experimental evaluation highlights the benefit of the overall approach in terms of timing and classification performance. In [39], the authors investigated the application of DT to detect DDoS attacks by capturing traffic from a software defined network (SDN) and using two state-of-the-art datasets. The experiments revealed that the DT model performs better than the support vector machine (SVM) and naïve bayes classifiers. In [40], a DDoS protection module based on a modified version of the DT is presented. The system collects network information by selecting significant traffic characteristics to pass to the DT model trained using the Gini impurity. Furthermore, the system incorporates a pessimistic error pruning strategy with the main objective of reducing model execution time. The experiments were carried out in an SDN environment, demonstrating the effectiveness of the system in detecting DDoS attacks. The model proposed in [41] employs a DT algorithm to detect DDoS attacks in an SDN-based cloud system. The evaluation performed on the GureKDDcup6 dataset confirms the efficiency of this DT-based approach.

3.2. Methodologies for automatic NIDPS rule generation

In addition to the existing or manually written signatures, automatic NIDPS rule generators produce complementary signatures. Specifically, they can rely on a larger amount of data from which to automatically (by exploiting some algorithmic procedure) derive the rule (in a specific syntax) that can intercept the desired anomaly [42]. Vollmer et al. [43] propose a multimodal genetic algorithm for the automatic generation of Snort rules, based on an offline process that acts downstream of intrusion detection. The generation process is performed directly on the ICMP packets. Each rule represents an individual as a candidate solution, and each field of the ICMP protocol represents an individual gene. The rules are then sorted according to the fitness score to minimize the reaction time of the filtering system. Anomalies, such as duplicate rules, are removed to streamline the results. Finally, depending on the exploration phase performed by the genetic algorithm for the overall evolutionary cycle, the optimal rule may not be generated. The authors of [44] propose an alternative evolutionary approach. In this case, the fitness value is given by the detection rate, i.e., the ratio of the number of intrusions detected by the rule to the total number of intrusions. The selection criterion for the ruleset is the best overall fitness value, which is based on the average of individual fitness values. The first genetic operator is proportional to fitness, whereas the recombination phase generates perturbations to refine the solution. The population of the new rules is tested using Snort to compute the overall fitness according to the detection rate cited above. In [44], both TCP and ICMP DoS attacks are considered. Kao et al. [45] developed an automatic rule generation algorithm for Snort called, which is capable of classifying malicious HTTP traffic and preventing application layer DDoS attacks. This algorithm does not require legitimate traffic as the input data. Given some samples of malicious HTTP traffic, they are matched against existing rules, and if not matching, a rule generation process ignoring the initial ruleset is started. Failure to consider legitimate traffic in this process could increase the number of false positives. The limitation of this algorithm is that it focuses only on a narrow range of anomalies related to the HTTP protocol. The research proposed in [46] aims at automating the Snort rule generation process using data mining algorithms such as Ripper and C5.0. The algorithm starts by processing and extracting some features from a network traffic dump. Then, it generates new features, and the obtained dataset is analyzed using data mining techniques. The generated rules are then selected according to their detection rate, which is based on the performance achieved by these rules when they are tested using different data containing HTTP DoS/DDoS and SSH brute force attacks. The authors of [47] propose an algorithm for automatic NIDPS rule generation called Syrius. It takes as input one or more attack examples as input and produces a list of candidate rules, i.e., rules that capture malicious traffic according to the data captured. Initially, it identifies options that allow the detection mechanism to completely intercept malicious traffic according to the input. The rule generated so far by Syrius is called the seed rule, and because only malicious traffic is considered, this could lead to false negatives. Furthermore, Syrius uses legitimate traffic to generate alternative rules that weaken the constraints expressed in the seed rule while still capturing malicious traffic. Finally, Syrius uses heuristic functions to rank the generated candidate rules according to their similarity to existing ones. The rule generated at the end of this process is called the golden rule.

The literature review revealed the lack of a generation algorithm capable of handling a wider range of protocols to provide a more effective defense methodology for network and application layer DoS and DDoS attacks. In addition, it is desirable that the generated rules are optimal in terms of both the number of rules produced and the number of anomaly cases intercepted.

4. The proposed contribution: Anomaly2Sign

Anomaly2Sign is an innovative rule generation algorithm that takes advantage of the interpretability property of a DT. It consists of three main phases. First, the input baseline is transformed by the algorithm using an ad hoc feature selection strategy. Second, the DT model trained on the processed baseline is validated by computing its classification performance that reflects the same performance of the rules that it would generate. If the model is considered inefficient, its hyperparameter configuration is changed until an acceptable performance is reached, starting with the rule generation phase. If the model is considered inefficient, its hyperparameter configuration is changed until acceptable performance is reached, starting with the rule generation phase. Therefore, the rule-building phase starts if the DT rules can accurately model the target anomaly, discriminating it from legitimate traffic. Finally, the optimal ruleset, i.e., the one with the fewest rules and the highest detection rate based on DT hyperparameter variation over a given hyperparameter space, is generated.

4.1. Data preparation

Since the process of automatically generating NIDPS rules is performed using an ML model, the baseline given as input for Anomaly2Sign must be accurately pre-processed in the sense of noise removal and data encoding. Furthermore, as such a baseline contains samples from different network traffic categories (or classes), it is important to compute the current number of samples within each class to identify the class skew due to a possible imbalance between classes. Such scenario occurrence is particularly suffered from DT models driven by information gain [26], therefore, it is essential to deal with it. Note that the pre-processing strategies described so far can be deliberately adopted on the basis of the addressed problem and, despite their execution being a requirement that the input baseline must meet, these do not represent a key point of the proposed contribution as is the feature selection procedure described below.

4.1.1. Feature selection

This process aims at selecting features that will be used during the learning phase by the DT model. In particular, the feature selection strategy is performed according to the goal of the entire algorithm, i.e., leveraging the decision rules inferred by the DT model to generate a NIDPS rule. Therefore, the DT must learn the relationships between only the variables that can be part of the rules that the model will generate. The identification of such a class of features consists of two phases, as described below. The first step is performed according to the following definition.

Definition 4.1 (*Pre-selected Feature*)**.** Given a baseline D, a feature f_i, with $i = 1, \dots, n_f$ is pre-selected for the learning process of the DT model leveraged by Anomaly2Sign if it satisfies at least one of the following properties:

1. There is a direct match for the feature with a Suricata syntax keyword, i.e., there exists a direct mapping.
2. There is a transformation resulting in a semantic match of a feature with a Suricata syntax keyword. This case is an indirect mapping.

Features satisfying the above definition will be included in the set of pre-selected features Y so that $|Y| \leq n_f$. Suppose to have a baseline composed of some network packets having the following features:

In Table 2, $n_f = 8$. According to the pre-selection strategy stated in 4.1, $|Y| = 7$ as Suricata's syntax lacks a keyword that identifies the operating system of the network traffic source. On the other hand, TTL, frame length, and SYN can be used. In particular, both TTL and SYN can be directly mapped to `ttl` and `flags:S` Suricata network-layer protocol keywords (see Section 2.1.1 in rule options). On the contrary, frame length, that is, packet size in bytes at the data link layer (DLL) of the protocol stack, cannot be directly mapped into a Suricata keyword. However, it is possible to infer the dimension of the payload size at the transport layer. For example, suppose that TCP is the protocol at the transport layer and assume that the packet has the minimum header size at both the network and transport layers (20 bytes each). Under such a hypothesis, the payload at the transport layer, i.e. `dsize` in Suricata, can be computed as the difference between the frame length and the summation of header sizes at the network and transport layers. Therefore, in this case, the frame length has mapped to the `dsize` indirectly. Focusing on the remaining features in Table 2 (highlighted in bold), these can be part of the Suricata rule at the header level. However, except for the protocol field that will be set by the model decision rules, the other will be discarded from the final set of selected features Λ because all the rules generated by Anomaly2Sign have the following fixed header structure:

Table 2
An example of D.

Src IP	Src OS	**Dst IP**	**Dst Port**	TTL	Frame length	Protocol	SYN
10.10.1.5	Windows	10.10.1.18	443	128	174	TCP	Set
10.10.1.17	Windows	10.10.1.26	53	128	60	UDP	Not set
10.10.1.20	Linux	10.10.1.22	None	64	42	ICMP	Not set
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
10.10.1.25	Linux	10.10.1.21	None	64	1301	HTTP	Not set

Src stays for source.
Dst stays for destination.

```
$ZERO_TRUST any -> $NET_TO_PROTECT $PORT_GROUP
```

where the usage of group variables described in Section 2.1.1 has been exploited. In particular, the fixed header attributes are populated as follows:

Attribute 4.1 (*Source → $ZERO_TRUST*)**.** *The Zero Trust approach considers any traffic insecure, regardless of whether it comes from internal or external sources. This criterion has two major advantages:*

- *The application of a methodical strategy for network security that protects an organization by eliminating implicit trust and constantly validating every stage of a digital transaction;*
- *Not differentiating traffic sources minimizes the number of produced rules. In particular, $ZERO_TRUST = [$INTERNAL_NET, $EXTERNAL_NET], where $INTERNAL_NET is the list of private subnets, while $EXTERNAL_NET is any other network, i.e., $EXTERNAL_NET != $INTERNAL_NET.*

Attribute 4.2 (*Source Port → any*)**.** *Any source port because it is assumed to be random.*

Attribute 4.3 (*Destination → $NET_TO_PROTECT*)**.** *Relevant destination address(es) to protect.*

Attribute 4.4 (*Destination Port → $PORT_GROUP*)**.** *List of ports that may be affected by the particular protocol defined in the rule.*

Therefore, given the simplified baseline in Table 2, the resulting selected feature set Λ comprises TTL, frame length, and protocol.

4.2. Rule generation

Given D composed of $|\Lambda|$ features, before starting with the training phase, the set of model hyperparameters Ω within which to vary them is defined to select the resulting DT capable of generating the lowest number of rules characterized by the highest detection rate value.

4.2.1. *From Decision Tree rules to Suricata rules*

Since the decision rules produced by the DT model will be made applicable via their transposition into Suricata rule [20], the first step consists of validating the model performance. In fact, it is not necessary to start with the rule generation process if the model on which they are based is inefficient. Consequently, the rule-building process begins if the DT achieves an acceptable detection rate calculated according to the formula proposed in [48]:

$$D_R = \frac{A_D}{T_A} \tag{4}$$

where A_D represents the number of detected attacks while T_A represents the total number of attacks. To perform such a computation, the baseline D is initially split into train (D_{TR}) and test (D_T) sets consisting of all selected Λ features. Then, we defined the detection rate threshold so that the rule-building phase starts if $D_R \geq 97.5\%$ is met. Such a percentage threshold has been selected to generate rules achieving performance on DoS/DDoS attacks that can outperform DoS/DDoS Snort rules in [10].

4.2.2. *Rule-building phase*

The rule-building phase can be divided into two sequential steps:

- First, Anomaly2Sign generates rules that have: (i) an action that can be `drop` or `alert`, making the generated rule suitable for both IDS (`alert`) and IPS (`drop`); (ii) a fixed header structure, according to how determined in the feature selection phase, so except for the protocol field that will be set by the model decision rules; (iii) an optional pattern matching rule built according to the values extracted by the DT trained on D_{TR}. As an example, suppose that the baseline D shown in Table 2, consists of both legitimate and malicious network packets. Furthermore, suppose that the induction process of the DT results (for $H(X) = 0$, where $X \subset D_{TR}$) in the following inference:

  – IF Protocol is TCP $\wedge$ SYN is set $\wedge$ TTL is 128 $\wedge$ frame length $\leq$ 200 THEN Malicious.

  In this step, such a decision rule becomes the following Suricata rule:

  ```
  alert tcp $ZERO_TRUST any -> $NET_TO_PROTECT
      $PORT_GROUP (msg:"Malicious"; sid:1;
      dsize:<160; ttl:128; flags:S;)
  ```

  As can be seen, no fixed rule fields have been set exploiting the XAI methodology proposed in [20]. Note that the larger Λ, the greater the number of fields that can be valued according to DT decisions. For example, in the case of DoS/DDoS anomalies, the number of samples for which the inference was generated can be used to value `count` in the `threshold` keyword. In addition, if a temporal feature is included in Λ, `seconds` can be set according to the attack time frame, which is calculated as the difference between the last and first time values assigned to the anomaly packets. Furthermore, keywords related to the direction of traffic flow are populated according to whether the traffic targets the same server (`track by dst`, `flow: to server`).
- Second, Anomaly2Sign leverages an optimality criterion based on the fact that by updating the DT hyperparameters, the DT inferences change; therefore, the Suricata rules change accordingly. In particular, a ruleset must be selected based on the DT hyperparameter configuration that creates the optimal ruleset according to the following definition.

Definition 4.2 (*Optimal Ruleset*). Given two Suricata rulesets R_1 and R_2, it is said that R_2 is the optimal ruleset, if the following conditions are both met:

1. $|R_2| \leq |R_1|$;
2. the R_2 rules intercept all traffic matched by the R_1 rules.

Table 3
BOUN and BUET main structural characteristics.

Dataset	No. samples	No. features	No. attack classes
BOUN	3 342 662	12	2
BUET	969 401	29	5

4.3. *Anomaly2Sign summary*

The entire process performed by Anomaly2Sign is summarized in Fig. 2. Given an appropriately processed dataset D, the DT is trained by employing an initial combination of hyperparameters. If the trained model shows a satisfactory D_{TR} value, it moves to the rule-building phase. The rule is built on the basis of the DT decision rules. During this phase, the system verifies that the generated ruleset is optimal according to Definition 4.2. If the conditions within such definitions are not met, Anomaly2Sign returns to the pre-training phase to update the model's hyperparameters, and the model will be again trained and tested. In general, the algorithm restarts from its pre-training phase whenever a hyperparameter update occurs. This condition can be met several times according to Ω size, which can still be redefined by successive attempts.

5. Experimental setup

In this section, the methods and materials used to analyze the performance of the proposed algorithm are described. First, the selected datasets are outlined with the following pre-processing strategies adopted to make the data suitable for Anomaly2Sign execution. In addition, the metrics used to assess the algorithm are outlined. Finally, the algorithms used for comparison are listed and discussed.

5.1. *Datasets used description*

Since the goal of this study is to provide an algorithm capable of automatically generating a set of rules suitable for intercepting DoS and DDoS anomalies, the BOUN [49] and BUET datasets [50] are used. The first has been selected since it represents a state-of-the-art dataset employed in several researches that aim to address the DoS/DDoS detection problem [51–54]. The second has been chosen because it shares some features contained in BOUN and extends the attack categories by adding three DDoS typologies. Table 3 reports the main characteristics of both datasets, which are further described in the following paragraphs, including the adopted pre-processing strategies.

5.1.1. *BOUN*

As reported in Table 3, the BOUN dataset consists of two distinct DDoS categories, i.e., TCP-SYN and UDP flood attacks, characterized by 12 different features. Furthermore, the dataset includes legitimate traffic, that is, common attack-free user traffic. Note that the BOUN dataset is an unlabeled dataset. However, a column identifying the traffic typology can be added according to the information provided in [49]. In fact, the column time gives a clear indication about the time interval in which both regular and attack traffic are implemented. A waiting period of 80 s followed by a 20 s attack period was practiced. To better highlight the network traffic characteristics found in the BOUN dataset, Table 4 provides a description of BOUN features. Fig. 3 shows the distribution of network traffic for each traffic topology in the BOUN dataset. According to this figure, anomalies within the dataset can be summarized as follows:

- SYN-flood [15]: This attack exploits the three-way handshake mechanism of the TCP protocol. In such a scenario, many connection requests are sent to the target server; this will cause the server to allocate resources for each of them until it reaches its resource exhaustion, since the target server has to keep these

Table 4
BOUN dataset feature list.

Feature name	Description	Type
Time	It represents the timestamp assigned to each network packet recording operation with a resolution of 10^{-6} s.	Numeric
Frame number	It represents a packet unit incremental counter.	Numeric
Frame length	It represents the packet size expressed in bytes.	Numeric
Source IP	It represents the network-level address of the network traffic source.	IP object
Destination IP	It represents the network-level address of the network traffic destination.	IP object
Source Port	It represents the transport layer port assigned to the network traffic source.	Numeric
Destination Port	It represents the transport layer port assigned to the network traffic destination.	Numeric
SYN	It defines whether the SYN flag contained in TCP packets is set or not.	Boolean
ACK	It defines whether the ACK flag contained in TCP packets is set or not.	Boolean
RST	It defines whether the RST flag contained in TCP packets is present.	Boolean
TTL	It represents the time-to-live field of the packer at the network-level.	Numeric
TCP Protocol	It indicates the transport layer packet protocol, i.e., TCP or UDP.	String

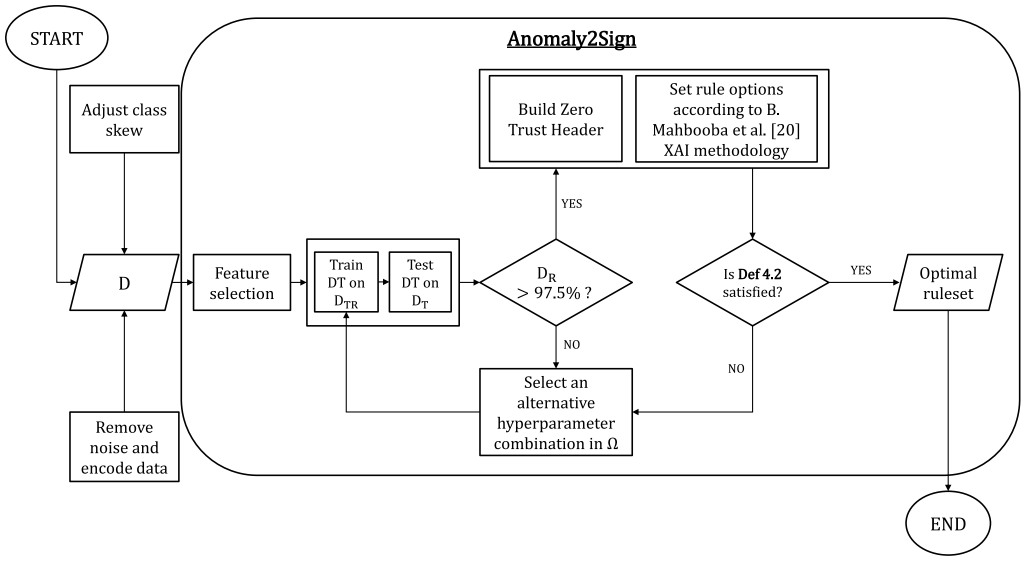


Fig. 2. Anomaly2Sign overview.

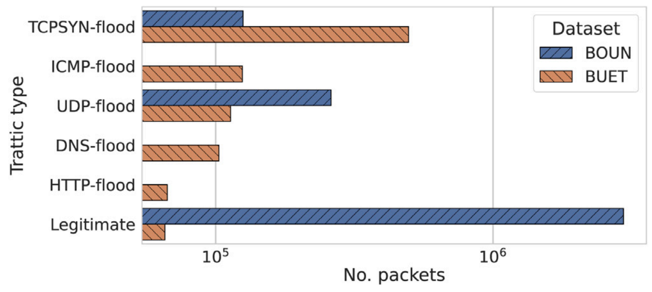


Fig. 3. Number of network packets per traffic type for both datasets.

connections open while waiting for the three-way handshake procedure to complete. This status is typically achieved using one of the following two strategies: (i) a connection to the target server is requested using a spoofed source IP (IP Spoofing) with the target server sending back a `SYN-ACK` packet and never receiving the last `ACK` packet from the real IP; (ii) the attacker sends a `SYN` request to the target server and receives the response containing `SYN-ACK`. In this situation, the attacker intentionally decides not to send the `ACK` response.

- UPD-flood [15]: In this scenario, the attacker sends several UDP packets with the same spoofed source IP and random destination ports. When receiving such packets, the destination server(s) checks whether any application listens on the specified port. Since this is maliciously crafted, this condition is not verified for most port numbers and results in `ICMP Destination Port Unreachable` messages sent back to the attack target.

5.1.2. BUET

The BUET dataset shares the same features listed in Table 4. Furthermore, according to Fig. 3, it extends the anomalies contained in BOUN to the other types of protocols widely exploited for flooding DDoS attacks [55], i.e.:

- ICMP-flood [15]: In this attack, the target is overloaded with `ICMP Echo Request` from multiple sources. For each of them, the victim uses its resources to generate a response for the sender. However, many packets could overwhelm the server resources, making it unavailable to legitimate users.
- DNS-flood [56]: This attack belongs to the class of amplification attacks. In such an attack typology, two main strategies are combined: reflection and amplification. Reflection is achieved via IP spoofing, whereas amplification takes advantage of UDP-based protocols. Unlike TCP, UDP-based protocols are connectionless, making spoofing trivial because the initiator cannot establish a handshake with them. Thus, the receiving server unknowingly reflects its responses toward the victim. For greater severity, the attacker exploits Internet protocols and services, such as DNS, resulting in a much larger response than the originating request.
- HTTP-flood [57]: These attacks are generally performed by compromised machines that belong to a botnet. These attacks can be classified according to the type of HTTP request, such as HTTP

Table 5
List of features (Λ) resulting from the feature selection.

Feature name	Suricata field(s)	Dataset(s)
Timestamp	threshold: count, seconds	BOUN/BUET
Protocol	proto	BOUN/BUET
TTL	ttl	BOUN/BUET
Length	dsize	BOUN/BUET
SYN	flags:S	BOUN/BUET
ACK	flags:A	BOUN/BUET
RST	flags:R	BOUN/BUET
PSH	flags:P	BUET
FIN	flags:F	BUET

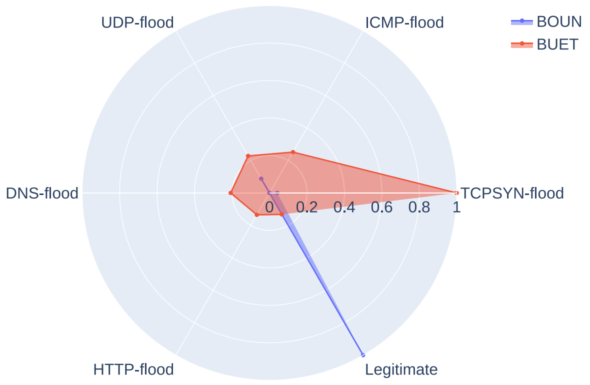


Fig. 4. Distribution of class skew on the used datasets.

GET flood when multiple sources send many requests for content, such as images, files, or any resource available through HTTP GET requests. It is very complicated to distinguish legitimate from malicious requests that occur simultaneously and ignore the latter, resulting in degraded server performance or service disruption.

The feature space of the BUET dataset was aligned with that of the BOUN dataset (see Table 4), adding information on the TCP flag values. In this case, legitimate traffic was collected considering the user attack-free traffic involved in the laboratory network used to dump all the traffic. As a final remark, since legitimate traffic involves the entire user traffic in both datasets, its features may be shared (e.g., protocol and related features) with the anomalous traffic.

5.1.3. *Strategies adopted for data pre-processing*

The analysis and pre-processing procedure starts with appropriate encoding of all categorical data into numeric attributes to make these data suitable for training an ML algorithm. Malformed samples are removed to reduce noise from the dataset. This operation does not affect the actual dimension of the dataset because it is sufficiently large. Furthermore, the consistency of the data is checked since it may contain format errors. According to the aforementioned described feature selection strategy, the list of features considered for both datasets is summarized in Table 5. In such a Table, for each future is assigned the corresponding field in the Suricata syntax and the dataset to which such a feature belongs.

Before starting the classifier training phase, the dataset was divided into training and test sets. In this study, 30% is considered training data, and the remaining 70% represents the test data as in [58]. In such a phase, it is paramount to check whether the classes within the training set are balanced; otherwise, the learning process will lead the classifier to perform better for the majority class due to such a class skew scenario. Both datasets suffer from class skew since in BOUN the dominant class is represented by the legitimate class, while in BUET the SYN-flood outnumbers all other classes as shown in Fig. 3. Furthermore, Fig. 4 points out the imbalanced ratio ρ computed for each traffic category with respect to the majority classes per dataset. Therefore, given $\mathrm{D_{TR_m}}$ and $\mathrm{D_{TR_M}}$, as the samples of the training set belonging to the minority and majority classes, respectively, the imbalance ratio ρ is calculated as follows [59]:

$$\rho = \frac{|\mathrm{D_{TR_m}}|}{|\mathrm{D_{TR_M}}|} \tag{5}$$

According to Eq. (5), the closer ρ to zero, the higher the class imbalance. Several strategies have been proposed in the current literature to address class skew in cyber security tasks. In general, these are divided into two main categories: cost-sensitive approaches and data sampling techniques [60]. The first typology aims at adjusting the learning process of an ML model to be unaffected by class skew, without changing the distribution of the original data [61–63]. The second strategy acts at the data-level by removing(adding) samples from(in) the dataset, i.e., acting as an under(over) sampler. In this study, we choose the second strategy, i.e., the following data-level sampling techniques have been selected:

- Undersampling:
  - Tomek-Links (T-Link) with Random UnderSampling (RUS) [64]: Given two samples x, y in $\mathrm{D_{TR_m}}$ and $\mathrm{D_{TR_M}}$, respectively, their Euclidean distance δ_{xy} is determined. This distance is considered a T-Link if one of the inequalities $\delta_{xy} < \delta_{xz}$ and $\delta_{xy} < \delta_{yz}$ is maintained for any sample z. Consequently, y is discarded. Finally, $|\mathrm{D_{TR_M}}| = |\mathrm{D_{TR_m}}|$ is obtained by randomly removing samples from $\mathrm{D_{TR_M}}$.
  - Cluster-Based Majority undersampling Prediction (CBMP) [65]: This technique takes advantage of the unsupervised learning algorithm K-Means so that, given $\mathrm{D_{TR_M}}$, it is divided into η clusters. For each ith cluster, with $1 \leq i \leq \eta$, a number of samples equal to $|\mathrm{D_{TR_M}}|_i$ corresponds. Then $r_i = \frac{|\mathrm{D_{TR_M}}|_i}{|\mathrm{D_{TR_M}}|}$ is computed to determine the number of samples to be selected from the majority class, that is, $s_i = r_i \times |\mathrm{D_{TR_m}}|$. Selection is performed using the following two strategies: (i) randomly, obtaining C_1; (ii) samples closest to the ith centroid in the ith cluster, obtaining C_2. Finally, the two obtained subsets are combined with $\mathrm{D_{TR_m}}$, resulting in the overall re-sampled training set.
- Oversampling:
  - Synthetic minority oversampling technique (SMOTE) [66]: As a first step, the K-nearest neighbors (K-NN) for each sample x within $\mathrm{D_{TR_M}}$ is identified. Subsequently, on the basis of the current value of ρ, a sampling rate γ is decided, and for each sample in $\mathrm{D_{TR_m}}$, γ samples are chosen from its K-NN to create $\mathrm{D_{TR_{m_{\gamma_1}}}}$. The data augmentation is finally realized for each sample $x_j \in \mathrm{D_{TR_{m_{\gamma_1}}}}$, with $j = 1, \dots, \gamma$, as $x_{NEW} = x + f_{RAND}(0,1) \times |x - x_j|$, where the function $f_{RAND}(0,1)$ randomly chooses a number in the interval $[0,1]$.
  - Less Important Components for Imbalanced multiclass Classification (LICIC) [67]: This algorithm creates new instances, balancing the minority classes and preserving non-linearity in the minor class patterns. It uses kernel principal component analysis (KPCA) and the permutation of the less important components to generate new instances as a linear combination of these components.

5.2. *Metrics used to evaluate model performance*

5.2.1. *Conventional classification metrics*

The adoption of data balancing strategies enables the use of conventional metrics to evaluate the classification performance achieved by the DT leveraged by the proposed algorithm. Anomalous(Legitimate) packets represent the positive(negative) class, so that correctly classified samples are denoted with true positive(negative) (TP(TN)), while misclassifications are indicated with false positive(negative) (FP(FN)). In particular, the metrics considered are [68]:

- Precision (PREC): The number of predicted packets that are actually identified as anomalous.
$$\mathrm{PREC} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{6}$$
- Recall or True Positive Rate (TPR): The number of anomalous packets identified as such from the total number of anomalous packets.
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{7}$$
Note that the TPR is equivalent to Eq. (4) when not only anomalous traffic is considered as done in [48] for the detection rate evaluation.
- F1 score: This represents the harmonic mean of the prior two metrics.
$$\text{F1 score} = 2 \times \frac{\mathrm{TPR} \times \mathrm{PREC}}{\mathrm{TPR} + \mathrm{PREC}} \tag{8}$$
- Area Under Receiver Operating Characteristic (AUC): The Receiver Operating Characteristic (ROC) curve relates Eq. (7) with the false positive rate (FPR), defined as $\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP+TN}}$. The area under the ROC is commonly used as a metric to assess the quality of a curve [69].

Since as an output of the data pre-processing strategy, a balanced data is obtained, we want an easily understandable metric for overall performance regardless of the class for in multi-class scenarios. As a consequence, the micro-averaged version of the above classification metrics is considered [68]. However, by evaluating micro-precision and micro-recall, it can be observed that the measurements obtained are equivalent [68]. Consequently, the micro-F1 score will also have the same value. Therefore, in our evaluation, we will only report the value of the micro-F1 score as representative of the three metrics. Note that in this scenario, the micro-average F1 score matches the classification accuracy [68].

5.2.2. Model complexity

To enforce the choice of the DT as an explainable model, the Akaike Information Criterion (AIC) metric is evaluated. Such a metric is typically employed in the fit analysis procedure, which evaluates the model not in the sense of hypothesis testing but for model selection. Therefore, the AIC provides feedback on the goodness of the model adopted in fitting the given data with respect to the model complexity. In particular, it is calculated as follows [70]:

$$\mathrm{AIC} = 2(\mathrm{k} - \ln(\mathcal{L})) \tag{9}$$

where k represents the number of model parameters, while $\mathcal{L}$ is the likelihood function produced by the model observing the training label data. As can be seen in Eq. (9), this metric grows with k, which is an indicator of the complexity of tuning a model. The greater the number of parameters, the larger the hypothesis space. In general, a higher AIC score corresponds to higher complexity of the model. However, k is not the unique factor to consider. In fact, such a metric is also a function of the likelihood natural logarithm. Given a supervised learning problem, $\mathcal{L}$ can be computed using the so-called mean squared error (MSE) on test data ($\mathrm{D_{TEST}}$) [71]:

$$\mathrm{MSE} = \frac{\sum_{i=1}^{|\mathrm{D_{TEST}}|} (y_i^* - y_i)^2}{|\mathrm{D_{TEST}}|} \tag{10}$$

By measuring the MSE, we can get a sense of how close the predictions (y_i^*) are to the actual values (y_i). Therefore, a low MSE score is desirable. For $0 < \mathrm{MSE} < 1$, $\ln(\mathcal{L}) \to -\infty$ is observed, therefore AIC > 0. Given two distinct models achieving similar performance, the model that results in the lowest AIC score must be preferred since it is less complex according to [19].

5.3. Decision Tree model employed

Based on one of the objectives of this study, in this section, the hyperparameter space Ω within which to vary the configuration of the model is defined. Note that this represents a simplified scenario where the only two hyperparameters recalled in Section 2.2.1 have been considered.

$$\Omega = \begin{cases} \mathrm{d_{MAX}} : [5, 7] \\ \mathcal{S}_c : [\mathit{entropy}, \mathit{gini}] \end{cases} \tag{11}$$

By varying the pair of $\mathcal{S}_c$ and $\mathrm{d_{MAX}}$ used, even if the detection performance remains acceptable, the variation in the number of rules produced (|R|) by each distinct DT model will be indicated for two cases, i.e., pairs (5, entropy) and (7, gini).

5.4. Algorithms selected for benchmark

This section describes the list of algorithms used for benchmarks divided into: (i) ML classifiers compared to the DT used by Anomaly2Sign; (ii) alternative methodologies for the automatic generation of Suricata rules.

5.4.1. Machine learning classifiers

The same ML classifiers used in [20] are evaluated to compare both the timing and classification performance. Furthermore, to extend the evaluation, an artificial neural network was considered. In summary, the selected models can be described as follows:

- Logistic Regression (LR) [72]: This is a parametric ML model that addresses the problem of fitting training data by adjusting the parameters (θ) of the logistic function $h(\theta) = \frac{1}{1+e^{-f_\theta}}$, where $f_\theta = \theta_0 + \theta_1 x$ is a linear function. In particular, the parameter updates are performed using a gradient descent procedure that minimizes the error function $J(\theta) = -\frac{1}{|\mathrm{D_{TR}}|}\left[\sum_{i=1}^{|\mathrm{D_{TR}}|} y_i \log(h_\theta) + (1 - y_i)\log(1 - h_\theta)\right]$.
- Support Vector Machine (SVM) [72]: This algorithm searches for an optimal hyperplane capable of separating samples between classes. A good separation is achieved by the hyperplane with the greatest distance to the nearest training data point in any class, since the larger the margin, the lower the generalization error of the classifier. Given the hyperplane equation $\overrightarrow{\omega x} + b = 0$, for which $\frac{1}{2}\|\omega\|^2$ is minimum, is optimal. As the boundaries of the hyperplane represent the constraints, searching for the optimal hyperplane represents a constrained optimization problem that is addressed using the Lagrangian dual form obtaining $\overrightarrow{\omega x} + b = \sum_i \alpha_i y_i \overrightarrow{x_i} \cdot \vec{x} + b$, where the samples $\overrightarrow{x_i}$ considered are only the support vectors and α_i represents the ith Lagrangian multiplier.
- Multi-Layer Perceptron (MLP) [72]: This is a conventional feed-forward neural network composed of a single hidden layer consisting of $0.8 \times n_f$ neurons. This configuration was chosen according to the universal approximation theorem stated in [73]. Data are then forwarded to the classification layer activated by a rectified linear unit (RELU) function. In this case, $J(\Theta) = -\frac{1}{|D_{TR}|}\left[\sum_{i=1}^{|D_{TR}|}\sum_{k=1}^{n} y_{i_{(k)}} \log(h_\Theta(x_i))_{(k)} + (1 - y_{i_{(k)}})\log(1 - (h_\Theta(x_i))_{(k)})\right]$ is the loss function, with Θ the matrix of synaptic links between the layers of the network. The parameter update is performed by backpropagating the error computed by the network.

Because the addressed task is multi-class classification, each ML algorithm has been trained following a one-vs-rest strategy.

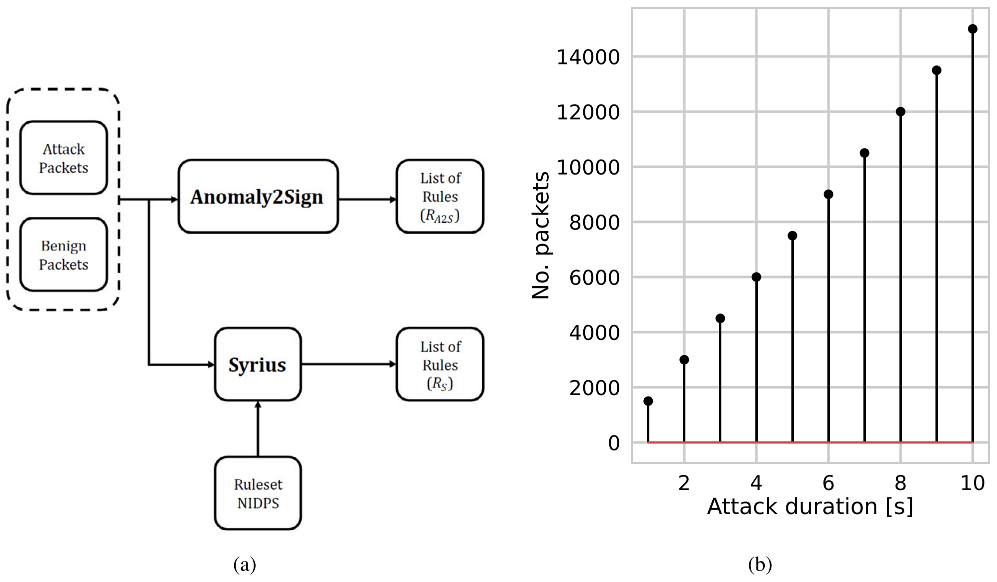


Fig. 5. (a) Setup of the comparison between Anomaly2Sign and Syrius. (b) Number of packets involved in each flood attack as a function of the attack timeframe..

5.4.2. *Methodology compared for automatic Suricata rules generation: Syrius*

This section describes Syrius [47], that is, the automatic NIDPS generation tool used to produce a confrontation case with the proposed contribution. In particular, the detection rate achieved by each produced rule and the execution time required by the generation process are evaluated. Syrius consists of a series of main modules that can be summarized as follows [47]:

- Reverse engineering: The first step aims to generate a rule, namely the seed rule, capable of intercepting the input of malicious traffic into the system. This is achieved by employing two different strategies to select the options that will be part of the rule. The first strategy (field strategy) is independent of the protocol, whereas the second strategy (payload strategy) acts only on HTTP traffic. Because only malicious traffic is considered in this step, the seed rule may be over-specified.
- Rules creation: This step aims to discard the rule options that would match the legitimate (negative) traffic due to the overfitting suffered from the seed rule. Using this technique, Syrius minimizes the rules generated in the previous step, guaranteeing that all malicious traffic will be captured. Here, an attack variation module is leveraged to ensure that the rule obtained continues to match the possible variant of the malicious traffic as input despite the removal of options implemented with the aim of minimizing the false negative rate.
- Ranking: This step ranks the rules using heuristic functions. The similarity between a new rule and rules from public set of rules is estimated using different criteria. In particular, the usefulness of rules is more likely to be reflected in rules that look similar to existing trusted rules. This ranking evaluation outputs the final rule, which is called golden rule.

Syrius has used thanks to the open source code released by the authors in [74]. To compare the performance of Anomaly2Sign and Syrius, the SYN-flood attack dataset released by the authors of Syrius is employed. A high-level overview of the test is shown in Fig. 5(a). The two algorithms produce two sets of rules R_{A2S} and R_S, respectively, which are empirically compared in terms of D_R and execution time (τ).

Since Anomaly2Sign focuses on DoS/DDoS attacks, it has been compared with Syrius using the `synflood`[1] and `positive-http`[2] datasets as anomalous and legitimate traffic, respectively.

Finally, to evaluate D_R (see Eq. (4)) achieved by R_{A2S} and R_S, some SYN-Flood attack scenarios have been implemented using `hping3`, i.e., a state-of-the-art tool used to generate DoS/DDoS anomalous traffic [75–77]. In particular, ten different attack scenarios are generated, each involving 1500 anomalous network packets per second for a time frame that increases by one second per attack, as shown in Fig. 5(b). The attack rate has been defined according to the same finding as in the baseline given as input to both generation processes.

Hardware

5.4.3. *Implementation details and hardware settings used*

Anomaly2Sign was implemented in Python. All ML algorithms (included the DT) were developed taking advantage of the `scikit-learn` framework [78]. The main package employed for the rule generation process was the so-called `suricata-parser` [79]. All tests were run on a Ubuntu-OS virtual machine from our laboratory with the following hardware settings: Intel Xeon(R) E5-2620 v3 CPU @ 2.40 GHz, 16 GB RAM.

6. Results and discussion

6.1. *Timing performance*

Since the process of automatically generating rules must be rapid, both training and testing times required by the ML model leveraged by Anomaly2Sign and the benchmark algorithms are pointed out.

6.1.1. *Training time*

Fig. 6 shows the training time required by each algorithm compared to different datasets. As a general rule, the higher the number of samples in D_{TR}, the longer the training time. Accordingly, given an algorithm, the strategy used to deal with class skew affects this

[1] https://github.com/STAR-RG/syrius/blob/master/syrius/Datasets/synflood.pcap [Accessed on 01/03/2023].

[2] https://github.com/STAR-RG/syrius/blob/master/syrius/Datasets/positive-http.pcap [Accessed on 01/03/2023].

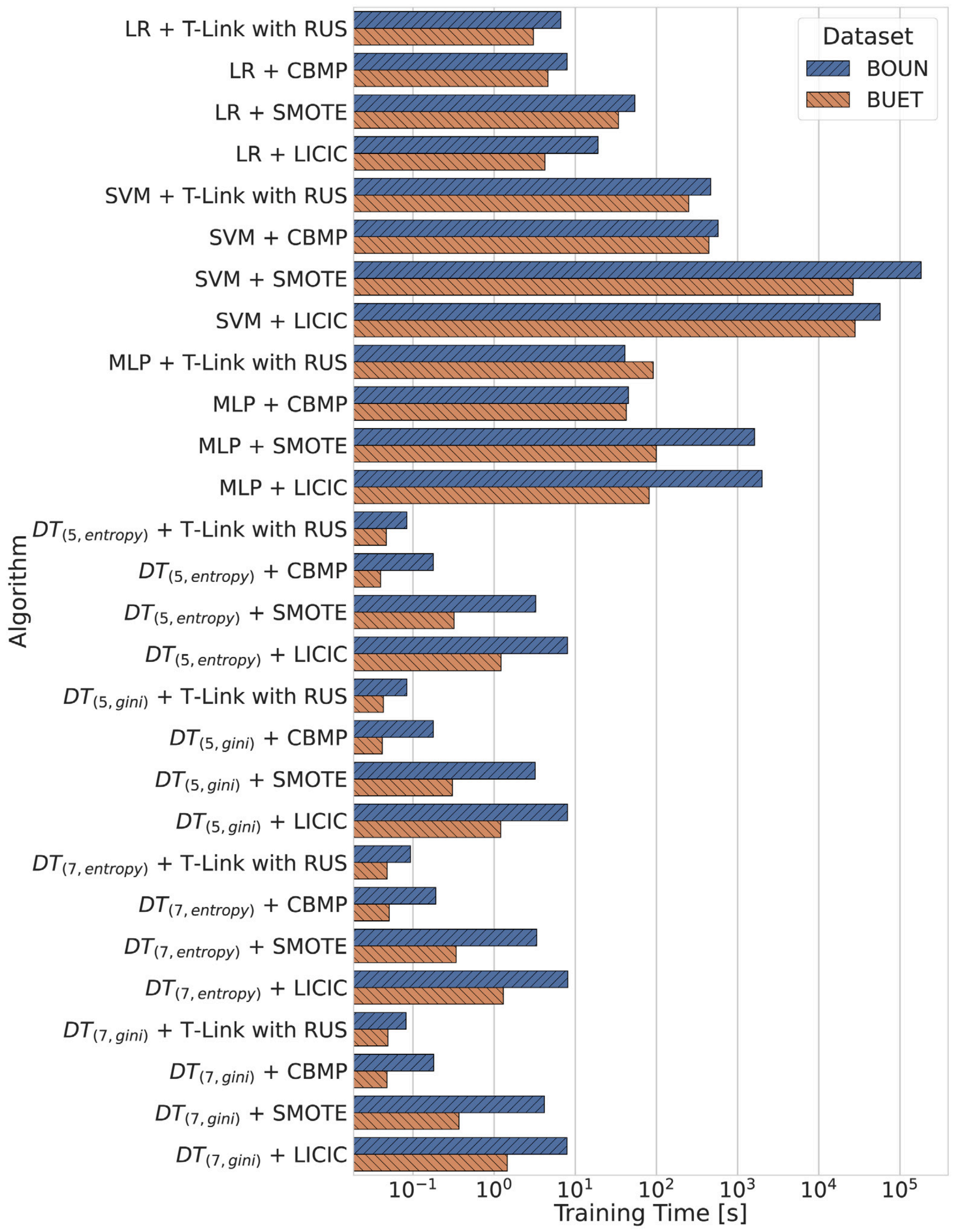


Fig. 6. Training time (in seconds) required by the algorithms compared for different tested datasets.

performance. This can be seen in Fig. 6, as fixed the ML classifier, oversampling techniques result in a longer training time than undersampling techniques, regardless of the dataset considered. Regarding the performance achieved by each compared classifier, the main findings derived from Fig. 6 can be summarized as follows:

- The SVM model requires the longest training time among the compared classifiers. In the worst case, i.e., SVM trained on BOUN data sampled through the SMOTE algorithm, the training time required exceeds 10^5 s. In experiments involving the SVM classifier, the shortest training time is recorded when T-Link combined with RUS is used for reducing BUET. However, in this specific case, the training time is $\sim\frac{1}{2} \times 10^3$ s, which is very long.
- The MLP classifier is the second worst performer among the examined algorithms in terms of training time. When BOUN data are augmented using SMOTE or LICIC, this classifier requires a training time of $\sim 10^3$ s. On the other hand, when the BUET data are undersampled, the MLP reaches the shortest training time with respect to the eight experiments in which the MLP is tested, i.e., close to 10^2 s.
- The LR classifier reduces the training time overhead of the SVM and MLP. In fact, it learns on expanded BOUN data in less than 10^2 s for SMOTE and in $\sim$30 s for LICIC. Furthermore, in experiments involving LR, the shortest training time is achieved for BUET data undersampled with T-Link combined with RUS ($\sim$5 s.)
- The DT model outperforms all the compared classifiers because it requires the shortest training time in all cases. In particular, for BOUN data, the DT requires a training time close to 10^{-1} s when the data is reduced using T-Link combined with RUS or CBMP. On the other hand, when oversamplers are used, the training time remains short, reaching a maximum of 10 s. Finally, in the case

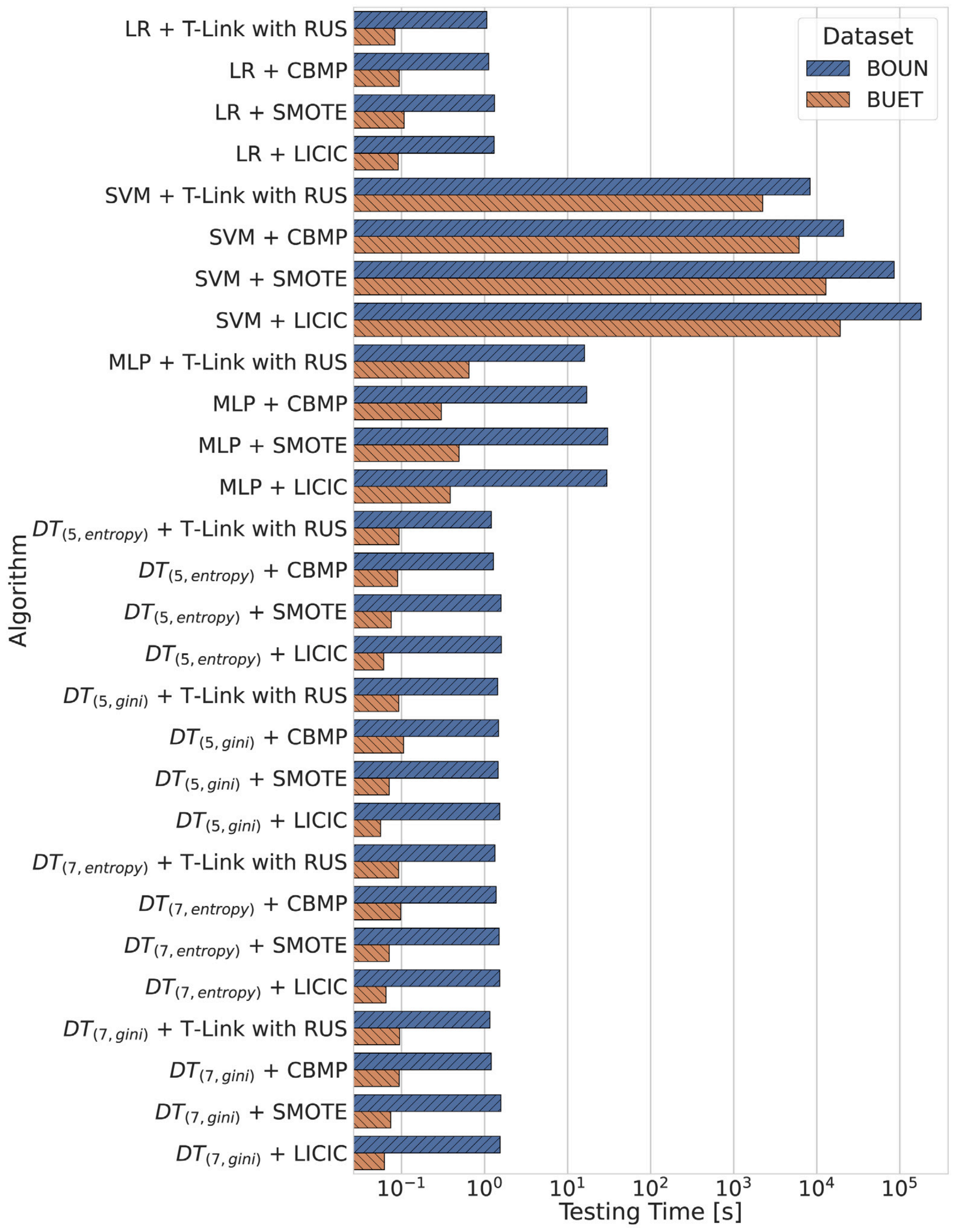


Fig. 7. Testing time (in seconds) required by the algorithms compared for different test cases.

of BUET data, the training time is less than 10^{-1} s in the case of undersampling, whereas it is between 0.5 and 1 s in the case of oversampling.

6.1.2. Testing time

Fig. 7 points out the testing time required by each algorithm combined with different data-level sampling techniques per different dataset evaluated. Note that the use of data sampling techniques affects only $|D_{TR}|$, which means that the number of data used in testing does not change. However, a classifier learns a policy that is influenced by its training dataset and thus also by how its class skew has been adjusted. Focusing on the four compared classifiers, the same trend already discussed in the training time analysis is confirmed:

- The SVM classifier requires the largest testing time, i.e., it is the worst performer among all algorithms compared. In particular, when this classifier is combined with LICIC, it takes more than 10^5 s to perform the predictions on BOUN data. On the other hand, among the four SVM cases, the shortest testing time is greater than 10^3 s when combined with T-Link with RUS in the case of BUET data.
- The MLP classifier needs 50 s to perform predictions on oversampled BOUN data. On the other hand, the testing time decreases to ∼20 s when the BOUN data are undersampled. In the case of BUET data, the MLP requires testing time between 0.5 and 1 s.
- The LR classifier outperforms SVM and MLP in terms of testing time. In particular, in the case of BOUN data, the LR requires ∼1 s, whereas for BUET data, the testing time required is 0.1 s.
- The DT requires a testing time of ∼1 s for BOUN data, regardless of the data-level sampling technique and hyperparameter combination used. Similarly, in the case of BUET data, the testing time required by the DT is less than 10^{-1} s.

Table 6
Classification metric scores achieved by comparing algorithms for the tested datasets.

Algorithm	Data sampling	BOUN		BUET	
		F1 score	AUC	F1 score	AUC
LR	T-Link with RUS	0.945	0.998	0.948	0.998
	CBMP	0.946	0.998	0.948	0.998
	SMOTE	0.946	0.998	0.948	0.998
	LICIC	0.713	0.770	0.173	0.500
SVM	T-Link with RUS	0.933	0.980	0.712	0.998
	CBMP	0.935	0.981	0.890	0.996
	SMOTE	0.960	0.988	0.986	0.998
	LICIC	0.755	0.789	0.666	0.667
MLP	T-Link with RUS	0.996	0.998	0.993	0.998
	CBMP	0.996	0.998	0.991	0.998
	SMOTE	0.996	0.998	0.990	0.998
	LICIC	0.713	0.706	0.470	0.688
DT (5, entropy)	T-Link with RUS	**0.997**	**0.999**	**0.997**	0.998
	CBMP	**0.997**	**0.999**	**0.997**	0.998
	SMOTE	**0.997**	**0.999**	**0.997**	0.998
	LICIC	0.871	0.500	0.072	0.500
DT (5, gini)	T-Link with RUS	**0.997**	**0.999**	0.994	0.998
	CBMP	**0.997**	**0.999**	0.994	0.998
	SMOTE	**0.997**	**0.999**	0.994	0.998
	LICIC	0.875	0.500	0.110	0.500
DT (7, entropy)	T-Link with RUS	**0.997**	**0.999**	**0.997**	**0.999**
	CBMP	**0.997**	**0.999**	**0.997**	**0.999**
	SMOTE	**0.997**	**0.999**	**0.997**	**0.999**
	LICIC	0.870	0.500	0.000	0.500
DT (7, gini)	T-Link with RUS	**0.997**	**0.999**	**0.997**	**0.999**
	CBMP	**0.997**	**0.999**	**0.997**	**0.999**
	SMOTE	**0.997**	**0.999**	**0.997**	**0.999**
	LICIC	0.870	0.500	0.012	0.500

The overall timing performance analysis indicates that the best performance for all the compared algorithms is achieved by the DT, i.e., the ML model leveraged by Anomaly2Sign, which ensures a rapid rule generation process.

6.2. *Classification performance*

Table 6 lists the classification performance achieved by all compared classifiers combined with different data-level strategies for BOUN and BUET datasets. The highest classification metric scores obtained per dataset were highlighted using bold type style. It is important to remark the objective of such an analysis, as the performance achieved by the classifier is reflected in the performance that would be achieved by the rules generated using the model itself. The main insights derived from Table 6 are listed as follows:

- The LR classifier achieves an F1 score of 94.5% and 94.8% for BOUN and BUET, respectively, using T-Link combined with RUS, CBMP, or SMOTE. Analogously, the AUC was 0.998 in the same cases examined. In contrast, a significant decrease in the same performance was obtained when class skew was addressed with LICIC.
- An F1 score of ∼93% is obtained by the SVM classifier using undersampling techniques to deal with BOUN imbalanced data. On the same dataset, such a metric is increased by three percentage points when class skew is tackled using SMOTE. The AUC score is in the range 0.98–0.99, except for LICIC, which again leads to poor performance in both BOUN and BUET datasets. With respect to the BUET data, the SVM performs well only when combined with SMOTE. In the case of reduced data, even with promising AUC values, the F1 score is at most equal to 0.89 for CBMP.
- The MLP classifier outperforms SVM and LR in terms of F1 score regardless of the data considered. In particular, this metric is equal to 99.6% and ∼99% for BOUN and BUET, respectively, when MLP is trained on data adjusted using T-Link with RUS, CBMP, or SMOTE. Once again, the LICIC oversampler results in the worst performance. Lastly, MLP achieves an AUC equal to 0.998 on both involved datasets.
- The ML classifier achieving the best classification performance among all compared classifiers is the DT. Furthermore, this result is obtained regardless of the hyperparameter pair selected from Ω. As shown in Table 6, all four DTs achieved an F1 score of 99.7% for BOUN and BUET data preprocessed using T-Link combined with RUS, CBMP, or SMOTE. The same trio of algorithms leads to an AUC of 0.999 for all four trees. In this case, the only exception is found when LICIC is used because an AUC value close to 0.5 denoting that the classifier cannot distinguish between malicious and legitimate traffic.

The classification performance analysis has shown that among the compared classifiers, the DT achieves the highest classification scores, which are stable in the range 99.7%–99.9%. Consequently, the classification rules derived from the DT decision rules can effectively detect the DoS/DDoS anomaly.

6.3. *Model complexity analysis*

According to the results obtained from the previous analyses, the complexity analysis was performed excluding from the evaluation performed so far those algorithms that met at least one of the following conditions:

- The worst ML algorithm (combined with all data-level sampling techniques) in terms of timing performance.
- Algorithms (the combination between data-level sampling and the ML classifier) that achieve an F1 score less than 97.5%, i.e., do not satisfy Eq. (4). In this regard, we remark that the micro F1 score, which is equal to the micro TPR, is equivalent to D_R when not only anomalous traffic is considered.

Fig. 8 reports the AIC scores computed according to Eq. (9) for all remaining algorithms. Regardless of the data-level sampling technique employed, the DT model outperforms the MLP in terms of the AIC score achieved. In particular, it results in AIC ∼31. In contrast, the MLP classifier leads to an AIC in the range 45–55. Because of the similar performance achieved in Table 6, such a result is mainly due to the difference between the models in terms of k (see Eq. (9)), in fact, an MLP model requires the configuration of more parameters than DT. Furthermore, in Fig. 8, it is possible to observe a decrease in the AIC value in the case of MLP as the sampling technique considered changes for BOUN data (the higher the number of training data, the lower the AIC). On the other hand, for BUET, the AIC follows a jagged trend. This trend is definitely attributable to the achieved MSE as k remains unchanged. Neither of these trends is found in the case of DT, regardless of the techniques used to deal with data class skew. Moreover, the change in hyperparameters does not change the AIC, which follows an almost constant linear trend. As a final point, since the lower the AIC, the lower the complexity of the model, this result enforces the advantage of choosing DT both as a classifier and as an explainer.

6.4. *Effectiveness of DT hyperparameter tuning to generate the optimal ruleset*

Deliberately, four DTs were compared as the chosen pair of hyperparameters varied starting from the hyperparameter space Ω defined in (11). The main objective of this comparison is to show how different DTs, in the sense of different hyperparameter settings, lead to the generation of different decision rules, i.e., different Suricata rules. Thus, let two distinct DTs be considered, with equivalent detection performance. The DT leading to the generation of the ruleset consisting of the smallest number of rules is chosen according to the stated

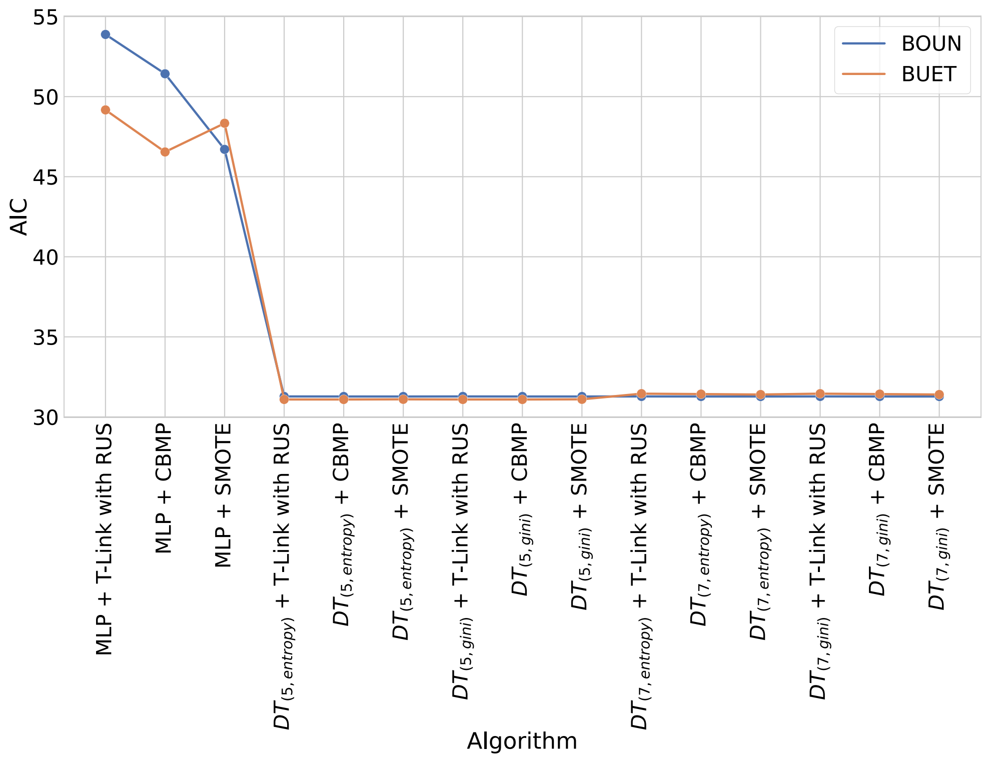


Fig. 8. AIC achieved by the algorithms compared for the tested datasets.

in 4.2. To point out such an analysis, let us consider the BUET dataset and the DTs tuned with pairs (entropy, 5) and (gini, 7) as S_C and d_{MAX}, respectively. First, both DTs share the detection performance and generate equivalent rules for SYN-Flood, UDP-Flood, ICMP-Flood, and DNS-Flood. The generated rulesets differ for the inferences produced on the HTTP-flood anomaly as follows:

- $R_{(entropy,\ 5)}$:

1.
```
drop http $ZERO_TRUST any ->
    $NET_TO_PROTECT $PORT_GROUP
    (msg:"HTTP_flood_attack";
    classtype:attempted-dos; sid:1;
    dsize:<595; flow:to_server;
    threshold:type both, track by_dst,
    count 2100, seconds 3;)
```

2.
```
drop http $ZERO_TRUST any ->
    $NET_TO_PROTECT $PORT_GROUP
    (msg:"HTTP_flood_attack";
    classtype:attempted-dos; sid:2;
    dsize:<372; flow:to_server;
    threshold:type both, track by_dst,
    count 2005, seconds 2;)
```

3.
```
drop http $ZERO_TRUST any ->
    $NET_TO_PROTECT $PORT_GROUP
    (msg:"HTTP_flood_attack";
    classtype:attempted-dos; sid:3;
    dsize:<1246; flow:to_server;
    threshold:type both, track by_dst,
    count 5496, seconds 2;)
```

4.
```
drop http $ZERO_TRUST any ->
    $NET_TO_PROTECT $PORT_GROUP
    (msg:"HTTP_flood_attack";
    classtype:attempted-dos; sid:4;
    dsize:<1400; flow:to_server;
    threshold:type both, track by_dst,
    count 13530, seconds 3;)
```

- $R_{(gini,\ 7)}$:

1.
```
drop http $ZERO_TRUST any ->
    $NET_TO_PROTECT $PORT_GROUP
    (msg:"HTTP_flood_attack";
    classtype:attempted-dos; sid:1;
    dsize:<683; flow:to_server;
    threshold:type both, track by_dst,
    count 4105, seconds 5;)
```

2.
```
drop http $ZERO_TRUST any ->
    $NET_TO_PROTECT $PORT_GROUP
    (msg:"HTTP_flood_attack";
    classtype:attempted-dos; sid:2;
    dsize:<1400; flow:to_server;
    threshold:type both, track by_dst,
    count 18976, seconds 5;)
```

According to Definition 4.2, the first check to be performed is on the dimension of the rulesets. In this case, $|R_{(gini,\ 7)}| < |R_{(entropy,\ 5)}|$. According to this result, $R_{(gini,\ 7)}$ is candidated as an optimal ruleset if no deterioration of detection performance is verified, i.e., all traffic matched by $R_{(entropy,\ 5)}$ is also intercepted by $R_{(gini,\ 7)}$. In this regard, the main difference between the two rulesets relies on the setting of `dsize` and `threshold` keywords. The first two rules in $R_{(entropy,\ 5)}$ matches traffic having a payload size at the transport level less than

595 and 372 bytes, respectively. On the other hand, the first rule in $R_{(\text{gini, }7)}$ is activated for a payload size smaller than 683 bytes at the transport level. Furthermore, the threshold values of this rule are the sum of those of the first two rules in $R_{(\text{entropy, }5)}$. Analogously, the second rule in $R_{(\text{gini, }7)}$ embeds the third and fourth rules in $R_{(\text{entropy, }5)}$ since it matches packets having a payload size less than 1400 bytes and for a threshold count and a time frame equal to the sum of the same values of both rules in $R_{(\text{entropy, }5)}$. Thus, $R_{(\text{gini, }7)}$ satisfies both the conditions in the optimality criterion. As a consequence, Anomaly2Sign would produce the rules generated by the DT tuned using (gini, 7) as $\mathcal{S}_c$ and d_{MAX}, respectively, reducing by 50% the HTTP-flood NIDPS rules with respect to signatures generated by the DT tuned using (entropy, 5).

6.5. Automatic rule generation comparison

By running both tools, the following rules have been obtained:

- R_S:
  - Seed rule:

```
alert tcp any any -> any any (msg:
    "Testing rule 0"; flags:S;
    window:512; threshold:type both,
    track by_dst, count 5000, seconds
    5; sid:525;)
```

    First, Syrius identifies the involved anomaly by reading the name of the input file. According to this finding, some main characteristics of the rule to be generated are fixed, such as `flags:S`, `window:512`, and the `threshold` values in this particular case. Therefore, these rule values remain unchanged regardless of the effective features in the input dataset.
  - Golden rule:

```
alert tcp any any -> any any (msg:
    "Testing rule 0"; flags:S;
    threshold:type both, track by_dst,
    count 5000, seconds 5; sid:525;)
```

    Then, the seed rule is refined by removing the `window` Suricata keyword value according to the previously described ranking process.

- R_{A2S}:

```
alert tcp $ZERO_TRUST any -> $NET_TO_PROTECT
    $PORT_GROUP (msg: "Synflood";
    classtype:attempted-dos; sid:1; flags:S;
    flow:to_server, not-estabilished;
    threshold:type both, track by_dst, count
    1463, seconds 1;)
```

  As observed by analyzing R_{A2S}, the DT model identified the splitting of the SYN-flood anomaly on the SYN flag, which is a peculiar feature of the anomalous traffic considered when compared to the negative traffic given by HTTP packets for which such a flag is not set. Furthermore, the `threshold` values were set on the actual number of packets involved in the anomaly baseline and according to the actual attack time frame.
  Suppose to exclude the SYN feature from the pre-selection phase, making `flags:S` a fixed component of the generated rule, as done by Syrius. Consequently, Anomaly2Sign will generate the following rule:

```
alert tcp $ZERO_TRUST any -> $NET_TO_PROTECT
    $PORT_GROUP (msg: "Synflood";
    classtype:attempted-dos; sid:1; dsize:0;
    flags:S; flow:to_server, not-estabilished;
    threshold:type both, track by_dst, count
    1463, seconds 1;)
```

  Thus, in this scenario, the proposed method would define the split on the `frame size` feature that is transformed into an appropriate value for the `dsize` keyword, according to the reasoning provided in Section 4.1.1. Despite this rule being over-specified, i.e. less general than the previous one, it gives a clear explanation of the structure of the anomalous packets for which the absence of data within the packet payload can be inferred. Therefore, Anomaly2Sign is capable of accurately reflecting the main characteristics of the packets involved during an attack process implemented to realize the input baseline.

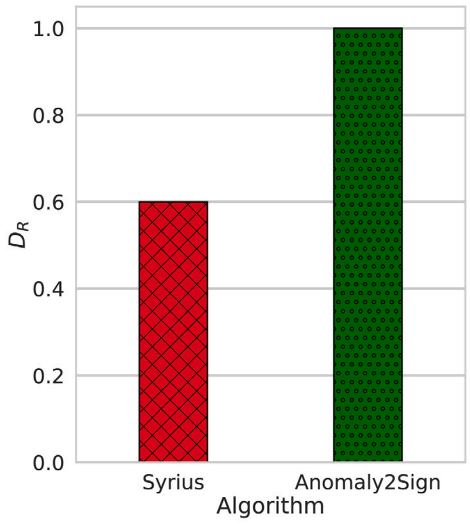
In Fig. 9, the detection rate D_R and the execution time τ (in seconds) required using the two methods to generate a single rule are shown. As a general rule, given two rules obtained from an automatic generation process and achieving good performance in terms of D_R, the process that requires less time in the overall generation procedure is more advantageous.

The performance achieved can be analyzed per algorithm as follows:

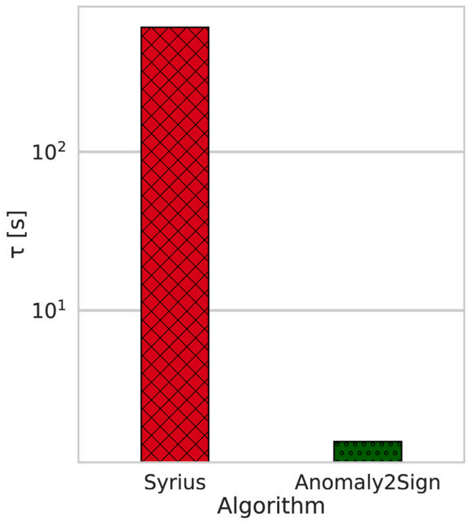
- In Fig. 9(a), the D_R achieved by Syrius is shown. This figure shows that the rule R_S can detect 6 out of 10 attack attempts in Fig. 5(b). This is due to the fact that the first four attempts last less than 5 s, i.e., the minimum time required to trigger the rule itself. Furthermore, Syrius requires an execution time equal to ~612 s, as shown in Fig. 9(b). A large portion of τ is spent to implement the attack variation. In particular, given the input anomaly, Syrius captures a series of related packets, by varying some feature evaluations. These are then used to refine the seed rule by reducing the false negative rate, ensuring that the detection rate of the rule remains unchanged. In this specific case, as previously described, the over-specified seed rule has been changed by removing only `window:512`. Such an operation cannot justify such high latency even in cases where the anomaly to be modeled leads to the generation of an all-too-simple rule.
- Fig. 9(a) highlights that Anomaly2Sign outperforms Syrius in the D_R achieved as it can detect all attack attempts ($A_D = T_A$ in Eq. (4)). The detection rate achieved by the rule R_{A2S} is obtained according to the capability of Anomaly2Sign in generating Suricata rules on the basis of actual baseline data. In fact, the threshold has been evaluated with ~1500 malicious samples per second because this setting exactly reflects the distribution of anomalous samples captured as input by both algorithms. Fig. 9(b) shows that Anomaly2Sign generates the rule in ~1.5 s, resulting in a marked improvement in execution time over Syrius. As expected, this result confirms the goodness of time performance discussed in Section 6.1. From this perspective, Anomaly2Sign is more advantageous than Syrius because in application contexts, it is important to make cyber threat response time efficient with tools that can provide very accurate feedback in a very short time.

7. Technique feasibility

This study focused on the application of Anomaly2Sign against flood attack types due to their widespread study in the current literature. Nevertheless, if the input baseline D is capable of accurately modeling other types of DoS/DDoS attacks, the proposed algorithm can be easily adapted, identifying the peculiar features of the generic anomaly to make them part of the feature selection phase implemented by Anomaly2Sign. This aims at converting such features into keywords of the generic Suricata rule. The relationships between features that determine the sample class will then be inferred from the DT which will be appropriately tuned at run-time to ensure the generation of a minimal ruleset but at the same time with a high detection rate. In general, the necessary condition for Anomaly2Sign to continue to work is that the Suricata syntax (or equivalent network-based IDPS systems) provides keywords that appropriately identify generic anomaly



(a) Detection rate.



(b) Time required for rule generation.

Fig. 9. Syrius and Anomaly2Sign performance comparison.

features. As a final point, the proposed contribution can effectively generate Suricata rules comparable in terms of D_R with other NIDPS, such as Snort, taking advantage of the high interpretability of the DT. Furthermore, the generation process is very fast because the ML model leveraged for classification purposes is more stable and less complex compared to other ML models belonging to other learning paradigms, such as deep learning.

8. Conclusions

The rapid growth of attack vectors such as DoS/DDoS has increased the demand for detection techniques that work effectively independently of the application context. In this paper, we proposed Anomaly2Sign, which addresses DoS/DDoS detection through the automatic generation of Suricata rules through an optimal decision tree in the sense of detection rate and minimal number of decision rules produced. The experiments carried out in our paper showed that among the various ML algorithms compared, the decision tree performs the best in terms of timing performance. Such a result is paramount for ensuring a rapid rule generation process. Furthermore, the DT leveraged by Anomaly2Sign outperformed the compared ML classifiers from the point of view of the classification scores achieved as both the F1 score and the AUC that remained in the range of 99.7%–99.9%. To definitively prove the advantage of selecting the DT model, the AIC was evaluated to point out its complexity, obtaining a very low value in all the cases examined. As a final experiment, we compared Anomaly2Sign with Syrius, i.e., a state-of-the-art automatic generator of NIDPS rules, obtaining a better detection rate for different DoS/DDoS attack scenarios and a significantly lower rule generation time. Among possible future directions of this research, the applicability of the proposed system will be evaluated in use cases where the anomalies to be detected are characterized by a large volume of traffic in a short time frame, such as brute-force SSH or FTP attacks, as well as other types of DoS/DDoS assaults.

CRediT authorship contribution statement

Antonio Coscia: Validation, Supervision, Conceptualization. **Vincenzo Dentamaro:** Validation, Supervision, Methodology, Formal analysis. **Stefano Galantucci:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis. **Antonio Maci:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization. **Giuseppe Pirlo:** Validation, Supervision, Project administration.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the Fondo Europeo di Sviluppo Regionale Puglia Programma Operativo Regionale (POR) Puglia 2014-2020-Axis I-Specific Objective 1a-Action 1.1 (Research and Development)-Project Title: CyberSecurity and Security Operation Center (SOC) Product Suite by BV TECH S.p.A., under Grant CUP/CIG B93G18000040007.

References

[1] Arora Himanshu, Manglani Tanuj, Bakshi Geetanjli, Choudhary Shikha. Cyber security challenges and trends on recent technologies. In: 2022 6th international conference on computing methodologies and communication. IEEE; 2022, p. 115–8.
[2] Li Yuchong, Liu Qinghui. A comprehensive review study of cyber-attacks and cyber security; emerging trends and recent developments. Energy Rep 2021;7:8176–86.
[3] DDoS attack trends for 2023 Q1. https://blog.cloudflare.com/ddos-threat-report-2023-q1/.
[4] Rios Ana Laura Gonzalez, Li Zhida, Bekshentayeva Kamila, Trajković Ljiljana. Detection of denial of service attacks in communication networks. In: 2020 IEEE international symposium on circuits and systems. IEEE; 2020, p. 1–5.
[5] Chaudhari Rutika S, Talmale GR. A review on detection approaches for distributed denial of service attacks. In: 2019 international conference on intelligent sustainable systems. IEEE; 2019, p. 323–7.
[6] Azeez Nureni Ayofe, Bada Taiwo Mayowa, Misra Sanjay, Adewumi Adewole, der Vyver Charles Van, Ahuja Ravin. Intrusion detection and prevention systems: An updated review. In: Data management, analytics and innovation: proceedings of ICDMAI 2019, vol. 1, 2020, p. 685–96.
[7] Snortorg. Snort - network intrusion detection & prevention system. [Online] Available at https://www.snort.org/.
[8] Suricata. [Online] Available at https://suricata.io/.
[9] Waleed Abdul, Jamali Abdul Fareed, Masood Ammar. Which open-source IDs? snort, suricata or zeek. Comput Netw 2022;213:109116.

[10] Bada G, Nabare W, Quansah D. Comparative analysis of the performance of network intrusion detection systems: Snort suricata and bro intrusion detection systems in perspective. Int J Comput Appl 2020;176(40):39–44.

[11] Fadhilah Dede, Marzuki Marza Ihsan. Performance analysis of IDs snort and IDs suricata with many-core processor in virtual machines against DoS/DDoS attacks. In: 2020 2nd international conference on broadband communications, wireless sensors and powering. IEEE; 2020, p. 157–62.

[12] Sarker Iqbal H, Furhad Md Hasan, Nowrozy Raza. Ai-driven cybersecurity: An overview, security intelligence modeling and research directions. SN Comput Sci 2021;2:1–18.

[13] Shaukat Kamran, Luo Suhuai, Varadharajan Vijay, Hameed Ibrahim A, Xu Min. A survey on machine learning techniques for cyber security in the last decade. IEEE access 2020;8:222310–54.

[14] Nassif Ali Bou, Talib Manar Abu, Nasir Qassim, Dakalbab Fatima Mohamad. Machine learning for anomaly detection: A systematic review. Ieee Access 2021;9:78658–700.

[15] Aljuhani Ahamed. Machine learning approaches for combating distributed denial of service attacks in modern networking environments. IEEE Access 2021;9:42236–64.

[16] Khalaf Bashar Ahmed, Mostafa Salama A, Mustapha Aida, Mohammed Mazin Abed, Abduallah Wafaa Mustafa. Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods. IEEE Access 2019;7:51691–713.

[17] Gilpin Leilani H, Bau David, Yuan Ben Z, Bajwa Ayesha, Specter Michael, Kagal Lalana. Explaining explanations: An overview of interpretability of machine learning. In: 2018 IEEE 5th international conference on data science and advanced analytics. IEEE; 2018, p. 80–9.

[18] Zhang Yu, Tiňo Peter, Leonardis Aleš, Tang Ke. A survey on neural network interpretability. IEEE Trans Emerg Top Comput Intell 2021;5(5):726–42.

[19] Stoffi Falco J Bargagli, Cevolani Gustavo, Gnecco Giorgio. Simple models in complex worlds: Occam's razor and statistical learning theory. Minds Mach 2022;32(1):13–42.

[20] Mahbooba Basim, Timilsina Mohan, Sahal Radhya, Serrano Martin. Explainable artificial intelligence (xai) to enhance trust management in intrusion detection systems using decision tree model. Complexity 2021;2021:1–11.

[21] Zeek. The zeek network security monitor. [Online] Available at https://zeek.org/.

[22] Communityemergingthreatsnet. Emerging Threats - Ruleset. [Online] Available at https://community.emergingthreats.net/.

[23] Suricata Rules. Suricata Rules. [Online] Available at https://docs.suricata.io/en/latest/rules/index.html#suricata-rules.

[24] Russell Stuart J. Artificial intelligence a modern approach. Pearson Education, Inc; 2010.

[25] Costa Vinicius G, Pedreira Carlos E. Recent advances in decision trees: An updated survey. Artif Intell Rev 2023;56(5):4765–800.

[26] Papamartzivanos Dimitrios, Mármol Félix Gómez, Kambourakis Georgios. Dendron: Genetic trees driven rule induction for network intrusion detection systems. Future Gener Comput Syst 2018;79:558–74.

[27] Tangirala Suryakanthi. Evaluating the impact of gini index and information gain on classification using decision tree classifier algorithm. Int J Adv Comput Sci Appl 2020;11(2):612–9.

[28] Mantovani Rafael Gomes, Horváth Tomáš, Cerri Ricardo, Junior Sylvio Barbon, Vanschoren Joaquin, de Leon Ferreira de Carvalho André Carlos Ponce. An empirical study on hyperparameter tuning of decision trees. 2018, arXiv preprint arXiv:1812.02207.

[29] Gohil Maulik, Kumar Sathish. Evaluation of classification algorithms for distributed denial of service attack detection. In: 2020 IEEE third international conference on artificial intelligence and knowledge engineering. IEEE; 2020, p. 138–41.

[30] Ramadhan Ilham, Sukarno Parman, Nugroho Muhammad Arief. Comparative analysis of k-nearest neighbor and decision tree in detecting distributed denial of service. In: 2020 8th international conference on information and communication technology. IEEE; 2020, p. 1–4.

[31] Lucky Godswill, Jjunju Fred, Marshall Alan. A lightweight decision-tree algorithm for detecting DDoS flooding attacks. In: 2020 IEEE 20th international conference on software quality, reliability and security companion. IEEE; 2020, p. 382–9.

[32] Kareem Mohammed Ibrahim, Jasim Mahdi Nsaif. DDoS attack detection using lightweight partial decision tree algorithm. In: 2022 international conference on computer science and software engineering. IEEE; 2022, p. 362–7.

[33] Khare Mrunmayee, Oak Rajvardhan. Real-time distributed denial-of-service (DDoS) attack detection using decision trees for server performance maintenance. In: Performance management of integrated systems and its applications in software engineering. 2020, p. 1–9.

[34] Tinubu CO, Sodiya AS, Ojesanmi OA, Adeleke EO, Adebowale AO. Dt-model: A classification model for distributed denial of service attacks and flash events. Int J Inf Technol 2022;14(6):3077–87.

[35] Lakshminarasimman S, Ruswin S, Sundarakantham K. Detecting DDoS attacks using decision tree algorithm. In: 2017 fourth international conference on signal processing, communication and networking. IEEE; 2017, p. 1–6.

[36] Das Saikat, Agarwal Namita, Shiva Sajjan. DDoS explainer using interpretable machine learning. In: 2021 IEEE 12th annual information technology, electronics and mobile communication conference. IEEE; 2021, p. 0001–7.

[37] Ahmim Ahmed, Maglaras Leandros, Ferrag Mohamed Amine, Derdour Makhlouf, Janicke Helge. A novel hierarchical intrusion detection system based on decision tree and rules-based models. In: 2019 15th international conference on distributed computing in sensor systems. IEEE; 2019, p. 228–33.

[38] Mohammadi Sara, Mirvaziri Hamid, Ghazizadeh-Ahsaee Mostafa, Karimipour Hadis. Cyber intrusion detection by combined feature selection algorithm. J Inf Secur Appl 2019;44:80–8.

[39] Kousar Heena, Mulla Mohammed Moin, Shettar Pooja, Narayan DG. Detection of DDoS attacks in software defined network using decision tree. In: 2021 10th IEEE international conference on communication systems and network technologies. IEEE; 2021, p. 783–8.

[40] Chen Yixin, Pei Jianing, Li Defang. Detpro: A high-efficiency and low-latency system against DDoS attacks in sdn based on decision tree. In: ICC 2019-2019 IEEE international conference on communications. IEEE; 2019, p. 1–6.

[41] Sridaran R, et al. An sdn-based decision tree detection (DTD) model for detecting DDoS attacks in cloud environment. Int J Adv Comput Sci Appl 2022;13(7).

[42] Acosta Jaime C, Akbar Monika, Hossain M Shahriar, Rivas Veronica. Automatic data generation and rule creation for network scanning tools. In: Proceedings of the future technologies conference. Springer; 2023, p. 536–49.

[43] Vollmer Todd, Alves-Foss Jim, Manic Milos. Autonomous rule creation for intrusion detection. In: 2011 IEEE symposium on computational intelligence in cyber security. IEEE; 2011, p. 1–8.

[44] Guruprasad Sunitha, D'Souza Rio. Development of an evolutionary framework for autonomous rule creation for intrusion detection. In: 2016 IEEE 6th international conference on advanced computing. IEEE; 2016, p. 534–8.

[45] Kao Chia-Nan, Chang Yung-Cheng, Huang Nen-Fu, Liao I-Ju, Liu Rong-Tai, Hung Hsien-Wei, Lin Che-Wei. Automatic nids rule generating system for detecting http-like malware communication. In: 2015 international conference on intelligent information hiding and multimedia signal processing. IEEE; 2015, p. 199–202.

[46] Fallahi Naser, Sami Ashkan, Tajbakhsh Morteza. Automated flow-based rule generation for network intrusion detection systems. In: 2016 24th Iranian conference on electrical engineering. IEEE; 2016, p. 1948–53.

[47] Alcantara Lucas, Padilha Guilherme, Abreu Rui, d'Amorim Marcelo. Syrius: Synthesis of rules for intrusion detectors. IEEE Trans Reliab 2021;71(1):370–81.

[48] de Lima Filho Francisco Sales, Silveira Frederico AF, de Medeiros Brito Junior Agostinho, Vargas-Solar Genoveva, Silveira Luiz F. Smart detection: An online approach for DoS/DDoS attack detection using machine learning. Secur Commun Netw 2019;2019:1–15.

[49] Erhan Derya. Boğaziçi university DDoS dataset. 2019, http://dx.doi.org/10.21227/45m9-9p82.

[50] Hasan Md Mehedi. Buet-ddos. 2021, https://data.mendeley.com/datasets/bzgf9r36kp/2.

[51] Toldinas Jevgenijus, Venčkauskas Algimantas, Damaševičius Robertas, Grigaliūnas Šarūnas, Morkevičius Nerijus, Baranauskas Edgaras. A novel approach for network intrusion detection using multistage deep learning image recognition. Electronics 2021;10(15):1854.

[52] Erhan Derya, Özdel Süleyman, Anarim Emin. DDoS detection using statistical modelling. 2019.

[53] Ali Mohammed Hasan, Jaber Mustafa Musa, Abd Sura Khalil, Rehman Amjad, Awan Mazhar Javed, Damaševičius Robertas, et al. Threat analysis and distributed denial of service (DDoS) attack recognition in the Internet of Things (IoT). Electronics 2022;11(3):494.

[54] Kalkan Kubra, Gur Gurkan, Alagoz Fatih. Defense mechanisms against DDoS attacks in sdn environment. IEEE Commun Mag 2017;55(9):175–9.

[55] Gupta Brij B, Dahiya Amrita. Distributed denial of service (DDoS) attacks: classification, attacks, challenges and countermeasures. CRC Press; 2021.

[56] Anagnostopoulos Marios. Amplification DoS attacks. In: Encyclopedia of cryptography, security and privacy. Springer; 2020, p. 1–3.

[57] Vishnu NS, Batth RanbirSingh, Singh Gursharan. Denial of service: types, techniques, defence mechanisms and safe guards. In: 2019 international conference on computational intelligence and knowledge economy. IEEE; 2019, p. 695–700.

[58] Erhan Derya, Anarim Emın. Hybrid DDoS detection framework using matching pursuit algorithm. IEEE Access 2020;8:118912–23.

[59] Thabtah Fadi, Hammoud Suhel, Kamalov Firuz, Gonsalves Amanda. Data imbalance in classification: Experimental evaluation. Inform Sci 2020;513:429–41.

[60] Wheelus Charles, Bou-Harb Elias, Zhu Xingquan. Tackling class imbalance in cyber security datasets. In: 2018 IEEE international conference on information reuse and integration. IEEE; 2018, p. 229–32.

[61] Maci Antonio, Santorsola Alessandro, Coscia Antonio, Iannacone Andrea. Unbalanced web phishing classification through deep reinforcement learning. Computers 2023;12(6):118.

[62] Gupta Neha, Jindal Vinita, Bedi Punam. Cse-ids: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems. Comput Secur 2022;112:102499.

[63] Telikani Akbar, Gandomi Amir H, Choo Kim-Kwang Raymond, Shen Jun. A cost-sensitive deep learning-based approach for network traffic classification. IEEE Trans Netw Serv Manag 2021;19(1):661–70.
[64] Elhassan T, Aljurf M. Classification of imbalance data using tomek link (t-link) combined with random under-sampling (RUS) as a data reduction method. Global J Technol Optim S 2016;1:2016.
[65] Zhang Yan-Ping, Zhang Li-Na, Wang Yong-Cheng. Cluster-based majority under-sampling approaches for class imbalance learning. In: 2010 2nd IEEE international conference on information and financial engineering. IEEE; 2010, p. 400–4.
[66] Chawla Nitesh V, Bowyer Kevin W, Hall Lawrence O, Philip Kegelmeyer W. Smote: Synthetic minority over-sampling technique. J Artif Intell Res 2002;16:321–57.
[67] Dentamaro Vincenzo, Impedovo Donato, Pirlo Giuseppe. Licic: Less important components for imbalanced multiclass classification. Information 2018;9(12):317.
[68] Grandini Margherita, Bagli Enrico, Visani Giorgio. Metrics for multi-class classification: An overview. 2020, arXiv preprint arXiv:2008.05756.
[69] Narkhede Sarang. Understanding AUC-ROC curve. Towards Data Sci. 2018;26(1):220–7.
[70] Cavanaugh Joseph E, Neath Andrew A. The akaike information criterion: Background, derivation, properties, application, interpretation, and refinements. Wiley Interdiscip Rev: Comput Stat 2019;11(3):e1460.
[71] Qi Jun, Du Jun, Siniscalchi Sabato Marco, Ma Xiaoli, Lee Chin-Hui. On mean absolute error for deep neural network based vector-to-vector regression. IEEE Signal Process Lett 2020;27:1485–9.
[72] Murphy Kevin P. Machine learning: A probabilistic perspective. MIT Press; 2012.
[73] Guliyev Namig J, Ismailov Vugar E. On the approximation by single hidden layer feedforward neural networks with fixed weights. Neural Netw 2018;98:296–304.
[74] Syrius. 2022. https://github.com/STAR-RG/syrius. [Available Online and Accessed on 01 March 2023].
[75] Singh Jagdeep, Behal Sunny. Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions. Comput Sci Rev 2020;37:100279.
[76] Ahda Aidil, Wulandari Citra, Husellvi Hanifa Putri, Alhuda Marcello Yasta, Reda Muhammad, Zahwa Putri, Ananda Sherly. Information security implementation of DDoS attack using hping3 tools. JComce-J Comput Sci 2023;1(4).
[77] Tampati Ihsan Fadli, Setyawan Faizal Gani, Sejati Wiyar Wilujengning, Kardian Aqwam Rosadi. Comparative analysis of CPU performance on freebsd 64-bit and redhat 64-bit operating system against denial of service (DoS) using hping3. CESS (J Comput Eng Syst Sci) 8(1):209–19.
[78] Pedregosa Fabian, Varoquaux Gaël, Gramfort Alexandre, Michel Vincent, Thirion Bertrand, Grisel Olivier, et al. Scikit-learn: Machine learning in python. the J Mach Learn Res 2011;12:2825–30.
[79] Pure python parser for snort/suricata rules. 2022, https://github.com/m-chrome/py-suricataparser. [Available Online and Accessed on 18 January 2022].

Antonio Coscia received the degree in Computer Science from the University of Bari, Italy, in 2004. During his professional career he worked in telecommunications and defense industries as a Software Engineer. He currently works as a Team Leader and Cyber Security Software Engineer for the R&D Cyber Lab of BV Tech S.p.A, in Grottaglie, Italy. His main research interests include Network Intrusion Detection Prevention Systems, Application Layer security, Evolutionary computations.

Vincenzo Dentamaro is an assistant professor at the University of Bari. He received the degree in computer science from the Department of Computer Science, University of Bari, Italy, and the M.Sc. degree in machine learning from the Georgia Institute of Technology, Atlanta, GA, USA. He received a Ph.D. in computer science from the University of Bari with a scholarship offered by InnovaPuglia S.p.A.. He was a Software Engineer at Johnson Controls Inc.; an Intern at IBM Rome; and the CEO and CTO Nextome S.R.L. He is currently publishing in various pattern recognition journals and conferences. He is also a Reviewer of IEEE Access, Elsevier Pattern Recognition Journal, MDPI Sensor, MDPI Information, and so on. He has previously published about indoor positioning and localization techniques on Microsoft Research Journal and holds two international patents on localization technologies. His awards and honors include the 1st prize Busan Metropolitan City, South Korea, in the Seal of Excellence European Commission; IBM's Global Mobile Innovator Tournament Award at the Mobile World Congress; and the MIT Technology Review award.

Stefano Galantucci received the M.Sc. degree (Hons.) in cybersecurity, defending a thesis on the generation of multiple cryptographic keys equivalent to each other. He is currently pursuing the Ph.D. degree in computer science with the University of Bari. He is involved in research in the areas of cybersecurity, cryptography, and biometrics. He is an adjunct lecturer teaching Cryptography at the Master's Degree in Computer Security at the University of Bari. He is also a Reviewer for some Journals, including IEEE Transactions on Information Forensics and Security, Computers & Security and many others.

Antonio Maci received the M.Sc. degree (Hons.) in Automation Engineering from the Polytechnic University of Bari, Bari, Italy, in 2022. He is currently pursuing the second level Master in Artificial Intelligence and Data Science with the University of Calabria, Rende, Italy. During his studies, he developed transversal skills and knowledge on Cyber Security, working firstly as a SOC Analyst and currently as a Cyber Security Software Specialist for the R&D Cyber Laboratory of BV TECH S.p.A, Grottaglie, Italy. His main research interests include Malware Analysis, Artificial Intelligence algorithms for Network Security and Cyber Physical Systems Safety.

Giuseppe Pirlo received the degree (cum laude) in computer science from the Department of Computer Science, University of Bari, Italy, in 1986. Since 1986, he has been carrying out research in the field of computer science and neuroscience, signal processing, handwriting processing, automatic signature verification, biometrics, pattern recognition, and statistical data processing. Since 1991, he has been an Assistant Professor with the Department of Computer Science, University of Bari, where he is currently a Full Professor. He developed several scientific projects and authored more than 250 articles on international journals, scientific books, and proceedings. He is a member of the Governing Board of Consorzio Interuniversitario Nazionale per l'Informatica (CINI), the Governing Board of the Societa Italiana di e-Learning, the e-learning Committee of the University of Bari, the Gruppo Italiano Ricercatori in Pattern Recognition, the International Association of Pattern Recognition, the Stati Generali dell'Innovazione, and the Gruppo Ingegneria Informatica. He was the General Chair of the International Workshop on Emerging Aspects in Handwriting Signature Processing, Naples, in 2013, and the International Workshop on Image-Based Smart City Applications, Genoa, in 2015; and the General Co-Chair of the International Conference on Frontiers in Handwriting Recognition, Bari, in 2012. He was an Editor of the Special Issue Handwriting Recognition and Other PR Applications of the Pattern Recognition journal in 2014 and the Special Issue Handwriting Biometrics of the IET Biometrics journal in 2014. He was a Guest Editor of the Special Issue of Journal of e-Learning and Knowledge Society: Steps Toward the Digital Agenda: Open Data to Open Knowledge (JeLKS) in 2014. He is also an Associate Editor of the IEEE Transactions on Human–Machine Systems. He also serves as a Reviewer for many international journals, including the IEEE Transactions on Pattern Analysis and Machine Intelligence, the IEEE Transactions on Fuzzy Systems, IEEE Transactions on Systems, Man, and Cybernetics: Systems, the IEEE Transactions on Evolutionary Computation, the IEEE Transactions on Image Processing, the IEEE Transactions on Information Forensics and Security, the Pattern Recognition, the International Journal on Document Analysis and Recognition, and the Information Processing Letters. He is also a Guest Co-Editor of the Special Issue of the IEEE Transactions on Human–Machine Systems on Drawing and Handwriting Processing for User-Centered Systems. He is also an editor of several books. He was a Reviewer in the scientific committee and program committee of many international conferences in the field of computer science, pattern recognition, and signal processing, such as ICPR, ICDAR, ICFHR, IWFHR, ICIAP, VECIMS, and CISMA.