Stream Clustering Guided Supervised Learning for Classifying NIDS Alerts

Risto Vaarandi
Centre for Digital Forensics and Cyber Security
Tallinn University of Technology
Tallinn, Estonia
risto.vaarandi@taltech.ee

Alejandro Guerra-Manzanares
Center for Interacting Urban Networks
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
alejandro.guerra@nyu.edu

***Abstract*—A Network Intrusion Detection System (NIDS) is a network monitoring technology for identifying cyber attacks, botnet command and control traffic, and other unwanted network activity. Unfortunately, organizational NIDS solutions can often generate tens or hundreds of thousands of alerts on a daily basis, with a significant part of them having low importance or being false positives. Therefore, high priority alerts become hard to spot, which overloads security analysts and complicates their work. This paper addresses this problem and introduces a machine learning framework for classifying NIDS alerts with the help of stream clustering and supervised learning. We propose a stream-clustering-guided method for creating labeled NIDS alert data sets. The small data sets created using this method can be used for training high-performance supervised NIDS alert classifiers. This significantly reduces the human labeling effort and eases the application of supervised machine learning for NIDS alert classification. The proposed machine learning framework was evaluated on NIDS alerts collected over 2 months from the network of a large academic organization. The experimental results indicate that combining stream clustering and supervised learning into a NIDS alert classification framework significantly decreases the number of false positives, and thus reduces the workload of human security analysts. The framework also features low CPU time and memory consumption and can thus be run on commodity hardware. In conclusion, the proposed framework provides a cost-effective means of integrating machine learning into Security Operation Centers (SOCs). This enables the identification of critical NIDS alerts using high-performance classifiers, thereby assisting in the automation of alert handling tasks for SOC personnel.**

To address the lack of public data sets in the problem domain and foster further research, we publicly share the large labeled NIDS alert data set used in our experimental setup.

***Index Terms*—network security, intrusion detection, network intrusion detection system, IDS, NIDS, NIDS alert, high-priority NIDS alert, NIDS alert classification, supervised learning, small training data set, stream clustering, data set generation, data labeling, Security Operations Center, SOC, workload reduction**

I. Introduction

Currently, many organizations are using either open-source (e.g., Suricata [1] and Snort [2]) or commercial (e.g., Cisco Next-Generation Intrusion Prevention System (NGIPS) [3]) Network Intrusion Detection System (NIDS) platforms for identifying malicious traffic in organizational networks. Most widely used NIDS platforms utilize signatures written by human experts for recognizing malicious network traffic. However, a NIDS can generate a large number of alerts, with only a small fraction of them deserving closer attention from security analysts [4]–[7]. For example, in a recent paper we have described a NIDS that triggers more than 200,000 alerts per day, with over 95% of them representing threats of low importance [4]. Unfortunately, that makes high priority alerts difficult to spot for security analysts. To address this problem, several machine learning-based approaches have been suggested in the related literature [4]–[17]. These approaches can be divided into supervised machine learning, unsupervised machine learning, and semi-automated data mining methods.

For supervised machine learning, the human expert has to create a labeled data set which consists of data samples, so that each has a human-assigned label. The label indicates the class of the data point. In the simplest case of binary classification, two labels are used such as *irrelevant* and *important* to indicate the importance of a data point. The labeled data set is used to train a classifier which is employed to predict the class of unlabeled data samples in a production system. A significant drawback of supervised machine learning algorithms is the need for a labeled training data set. Unsupervised machine learning methods do not rely on labeled data sets but rather attempt to discover regularities and patterns among data points. Typically, the detected regularities and patterns help to identify different data point clusters or classes. However, according to [18], unsupervised methods tend to feature a higher number of false positives than supervised methods in the domain of cyber security (a more detailed discussion on this issue is provided in the end of Section II-B). Lastly, semi-automated data mining methods address the labeling issue by using unsupervised knowledge discovery algorithms (e.g., frequent itemset mining), enabling the human expert to interpret the results and develop data point classification rules manually.

Previously proposed machine learning-based NIDS alert classification methods suffer from significant drawbacks, which are described as follows.

First, existing approaches are costly either in terms of human labor or computational resources. The approaches that employ supervised machine learning methods [5], [9]–[12],

[17] require large training data sets, which are expensive to produce and update. For example, in [17], a training data set with 7 million data points was used. Besides, semi-automated data mining methods [14], [15] require the interpretation of detected knowledge and development of alert processing rules by human experts, which is a complex task. As for the unsupervised approaches, they tend to be computationally expensive [6], [7], [13].

Second, many existing approaches have assumed that the NIDS alert has the following attributes – *timestamp, transport protocol, source* and *destination Internet Protocol (IP) addresses* and *ports*, and *signature ID*. However, modern NIDS platforms, like Suricata [1], include many application layer specific attributes in the alerts. This additional information about malicious network traffic facilitates more efficient incident handling, but it is not used by most existing approaches.

Third, a number of recent papers have suggested methods for automated and semi-automated creation of labeled training data sets for supervised machine learning algorithms in the domain of network traffic analysis and security (e.g., see [19] for an overview). This greatly reduces the cost of creating labeled training data sets and thus addresses the main shortcoming of supervised machine learning. However, such methods have not been considered in past works for NIDS alert processing.

Finally, the NIDS alert data sets used in previous works have not been made public, which complicates comparative evaluations of NIDS alert processing methods. Some studies (e.g., [4], [7], [10], [12], [17]) have employed NIDS alerts from production systems that have not been released to public domain due to organization policies and privacy reasons. Other studies (e.g., [5], [8], [13]–[15]) have utilized publicly available network trace data sets that contain malicious packets. The network traces were replayed to NIDS so that its signatures would trigger alerts for malicious traffic. However, the number and the nature of resulting NIDS alerts depend on many factors that have not been fully clarified in past research papers. These include the NIDS platform and its version, all NIDS configuration settings (e.g., settings for packet capture, stream reassembly, and application layer protocol detection), and precise list of all NIDS signature definitions. Therefore, releasing the actual NIDS alert data set is essential to ensure reproducibility.

This paper addresses the limitations of previous related research and provides the following contributions:

- proposes a NIDS alert processing framework that combines a low-cost unsupervised stream clustering algorithm and supervised machine learning, as described in Sections III-A and III-C,
- proposes an efficient approach for the creation of labeled training data sets that is guided by stream clustering, so that supervised learning can be integrated into the NIDS alert processing framework with significantly less human effort (described in Section III-D),
- creates and makes publicly available a large labeled NIDS alert data set that aims to facilitate further research in the domain (described in Section III-B). To the best of our knowledge, none of the previous NIDS alert processing studies have published similar data sets.

The remainder of this paper is organized as follows. Section II provides an overview of related work and background information about the Customized Stream Clustering Algorithm for Suricata (CSCAS), a stream clustering solution for NIDS alerts. Section III introduces the proposed framework, Section IV describes the evaluation of the framework, Section V discusses the limitations of the framework, and Section VI concludes the paper.

II. Background Information

This section discusses the background information concerning NIDS alert classification, with Section II-A covering related works and Section II-B providing an overview of CSCAS, a stream clustering implementation for NIDS alerts.

A. Related Work

Table I provides an overview of related work on NIDS alert classification in chronological order. As can be seen in Table I, data mining-based approaches [7], [14], [15] were proposed in 2008–2010, whereas later studies have employed different approaches. While some earlier works [13] used neural networks (i.e., deep learning), their application has become more common in recent years [5], [12], [16]. Furthermore, most deep learning approaches in Table I have also utilized clustering algorithms [5], [13], [16]. As for studies that involve traditional supervised machine learning algorithms [9]–[12], [17], support vector machine and decision tree-based models have been the most frequently used algorithms. Finally, clustering algorithms have been used both in early and recent studies [4], [5], [7], [8], [13], [14], [16], [17].

As for the NIDS alert data sets used in previous studies, they are not publicly available (see Table I), although some studies [5], [8], [13]–[15] have utilized publicly available network traces for generating the NIDS alert data set. As explained in Section I, that involves replaying the network trace to a NIDS, making the resulting NIDS alert data set highly dependent on the NIDS platform and version, its numerous configuration settings, and signature definitions. For example, in [5] the authors generated a data set of 431,860 alerts from the CICIDS2017 network trace. In contrast, when replaying this network trace to the NIDS described in Section III-B of this paper, we observed less than 14,000 alerts. That finding illustrates the fact that sharing NIDS alert data sets is paramount to ensure reproducibility.

As Table I also reveals, past studies have measured different performance aspects of the proposed methods. While a number of studies have used traditional machine learning performance metrics such as F1-score (e.g., [5], [10], [16], [17]), some studies have focused on the reduction of false positive alerts and have thus reported detection rates for false positives [6], [13], [14]. Also, some studies have provided other performance details such as the alert processing throughput [8] and the alert compression rate [15].

TABLE I
RELATED WORK ON NIDS ALERT CLASSIFICATION

Paper	Year	Classificaton Method	Data Sets	Results
[17]	2023	Instance-Weighted Support Vector Machine (IWSVM) and clustering with cluster segmentation	Not public, 2 data sets with 674,239 samples and 7.3 million samples	F1-score $>$ 0.97
[16]	2022	Neural networks and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering	Not public, 10.5 million samples	F1-score $\approx$ 0.93
[4]	2021	Stream Clustering Algorithm for Suricata (SCAS)	Not public, 169,441 samples	F1-score $\approx$ 0.92
[10]	2021	k-Nearest Neighbors (*k*-NN), naive Bayes, AdaBoost (AB), linear discriminant analysis, decision tree, Support Vector Machine (SVM), Weighted Support Vector Machine (WSVM)	Not public, 674,239 samples	AB, SVM and WSVM reached the highest F1-score $>$ 0.98
[5]	2020	Long Short-Term Memory (LSTM) neural network and latent semantic analysis with DBSCAN clustering	Not public, 2 data sets with 5.5 million samples (generated from MCFP and private network traces) and 431,860 samples (generated from CICIDS2017 network trace)	The highest F1-score $\approx$ 0.79
[9]	2019	Gradient Boosting Decision Tree (GBDT) algorithms	Not public, 59,427 samples	AUC $\approx$ 0.94
[11]	2019	Binary SVM and one-class SVM	Not public, size unspecified	Performance data not provided
[12]	2017	Neural network, Random Forest (RF), SVM, Logistic Regression (LR)	Not public, size unspecified	RF reached the highest AUC $\approx$ 0.83
[6]	2014	Graph-based method	Not public, 3,226 samples	99.3% of false positive alerts were detected
[8]	2013	Clustering and visualization	Not public, size unspecified (generated from DARPA2000 and private network traces)	Alert processing rate 11,7000 alerts per second
[13]	2010	Self-Organizing Map (SOM) neural networks and *k*-means clustering	Not public, 2 data sets with 3,062 samples (generated from DARPA1999 network trace) and 2,556 samples (generated from private network trace)	87-90% of false positive alerts were detected
[7]	2010	Fully automated frequent itemset mining and clustering	Not public, 770,024 samples	F1-score $\approx$ 0.98
[14]	2009	Semi-automated clustering	Not public, 3 data sets with 302,473 samples (generated from private network trace), 295,484 samples (generated from DARPA1998 network trace), and 233,615 samples (generated from DARPA1999 network trace)	74% of false positive alerts were detected
[15]	2008	Semi-automated frequent itemset mining	Not public, size unspecified (generated from DARPA1999 and DARPA2000 network traces)	The number of alerts was reduced by $\approx$90%

The remainder of this section provides a more detailed description of the related work included in Table I.

Kidmose et al. [5] proposed a three-phase supervised method for NIDS alert classification. During the first phase, a mapping function was learned on training data for converting textual alerts into vectors, with the authors suggesting the use of an LSTM and latent semantic analysis for this purpose. In the second phase, alerts were converted into vectors with the mapping function. In the third phase, vectors were clustered with the DBSCAN algorithm, using cosine similarity as a distance function, and the most frequently occurring label in the cluster was assigned to the entire cluster. Finally, the detected clusters were used for classifying incoming alerts by converting alerts into vectors and calculating their distance to clusters' core points, as a measure of similarity.

Van Ede et al. [16] developed a semi-automated method to classify NIDS alerts and other security events. Initially, the method detected event sequences from the same device. Event sequences were analyzed with deep learning models (e.g., LSTM) to identify correlations between events. This information was used to cluster event sequences with the DBSCAN algorithm. The resulting clusters were labeled by a human analyst. The database of labeled clusters was then used for semi-automated classification of further event sequences. Event sequences without a cluster were manually reviewed by a human analyst, updating the database of labeled clusters accordingly.

Ban et al. [10] used a large NIDS data set for evaluating seven supervised machine learning methods: *k*-NN, naive Bayes, linear discriminant analysis, decision tree, AB, SVM, and WSVM. According to the authors, WSVM, SVM, and AB produced the best results, while two isolation forest-based unsupervised algorithms provided a lower precision than the evaluated supervised algorithms.

Shin et al. [11] presented an organizational platform applying machine learning for NIDS alert data that supports binary SVM and one-class SVM methods. Feng et al. [12] described another organizational implementation, which processed NIDS alerts and other security events for finding users at risk. According to their experiments with four supervised machine learning methods, RF and multi-layer neural network-based classifiers outperformed SVM and LR-based classifiers. Wang et al. [9] used a graph-based method to identify features to eliminate false alerts, and applied GBDT algorithms for alert classification.

In [17], a supervised NIDS alert classification method

was proposed for detecting critical alerts. For handling the imbalanced nature of NIDS alert data, the method relied on an IWSVM-based classifier. The classifier was trained by assigning higher weights to repeated identical data points and to the minority class of critical alerts. After critical alerts were identified, the alerts that represented the same incident were grouped together with the help of a clustering algorithm, where clusters were segmented by the source and destination IP address, service port, and alert occurrence time attributes.

Apart from supervised machine learning approaches, semi-automated data mining methods have been suggested for NIDS alert processing. Al-Mamory and Zhang [14] proposed a clustering algorithm that assigned NIDS alerts with the same root cause to the same cluster, so that each cluster was represented by a common pattern. The detected patterns were used by human experts to write filters for reducing the number of alerts. Ma et al. [15] developed a data mining method which first aggregated alerts into hyper-alerts, and then mined frequent sequential signature ID patterns from these hyper-alerts. The detected patterns represented frequent attack scenarios and were used to develop rules that recognized similar attacks in the future.

In several studies, unsupervised learning approaches have been proposed for handling NIDS alerts. The framework presented in [7] used frequent itemset mining and data clustering to identify frequent alert patterns on a daily basis. These patterns were employed to filter alerts of low importance. Spathoulas and Katsikas [8] suggested a method that first aggregated alerts related to the same incident and then clustered the aggregated alerts. As a final step, clusters were visualized, which eased the detection of security incidents for end users.

Tjhai et al. [13] proposed an unsupervised method that used SOM neural networks for generating a two-dimensional map from NIDS alerts. *K*-means algorithm was then applied to the map and the detected clusters were converted into data points. SOM neural networks and *k*-means clustering were then applied to data points to identify false positive alerts. Shittu et al. [6] suggested another unsupervised method which arranged incoming NIDS alerts into graphs and prioritized them. According to the authors, the method managed to reduce the number of false positives by assigning them low priority values, but during the experiments a significant part of true positives were incorrectly identified as low priority alerts, thus increasing the number of false negatives.

In [4], we presented *SCAS*, a lightweight stream clustering algorithm to detect NIDS alerts of high importance from an incoming alert stream. SCAS is used to build clusters for frequent alert patterns that correspond to frequently occurring low importance alerts, while outliers represent unusual alerts, which require closer inspection. Since the novel NIDS alert processing framework proposed in this paper relies on a Customized version of SCAS (i.e., CSCAS), the next section provides an overview of CSCAS and covers its differences with SCAS.

B. Overview of CSCAS

This section provides an overview of the CSCAS algorithm that is part of the NIDS alert processing framework proposed in this study. More precisely, CSCAS is a customized version of the original SCAS system, developed and implemented *ad-hoc* for the framework introduced in this study. CSCAS uses the same stream clustering method as SCAS [4], but unlike its predecessor, it has been augmented with a module that generates data points suitable for processing by supervised machine learning algorithms. CSCAS relies on special properties of NIDS alert data that have been identified in past research by us and other authors [4], [7], [20], such as:

- most signatures seldom generate any alerts,
- most alerts are triggered by a small fraction of signatures,
- the alerts that are generated by frequently matching signatures represent well-known threats of low importance (e.g., frequent scanning for old vulnerabilities by botnets[1]),
- an attack from the same external host can trigger a large number of alerts in a short time frame, and therefore it makes sense to consider these alerts together.

Based on these facts, provided that the alerts are close in time, CSCAS aggregates alerts generated by the same signature for the same external host into an *alert group*. The maximum time span the alert group can cover is *SessionLength* seconds, while the maximum time frame between two consecutive alerts is *SessionTimeout* seconds (i.e., *SessionLength* and *SessionTimeout* are hyperparameters of CSCAS). It should be noted that unlike CSCAS, SCAS creates alert groups on the basis of the external host only, possibly combining alerts by different signatures for the same external host into the same group [4]. However, since this does not allow for the creation of the data point associated with only one signature, we developed and used CSCAS for our experimentation. CSCAS requires all alerts in the same alert group to originate from the same signature.

To illustrate the process of alert group creation, an example is as follows. Consider the NIDS alerts in Table II, assuming that the *SessionTimeout* hyperparameter was set to 60 seconds. The first and the second NIDS alert from Table II would be assigned to the same alert group, since they share the same signature ID (23) and external host IP address (10.1.1.1), and are less than 60 seconds apart. However, the fifth alert would not be assigned to this alert group, because it occurs more than 60 seconds later than the last alert in the group (i.e., the second alert in Table II). Therefore, a different alert group will be created for the fifth alert. Since the third and the fourth alert in Table II do not share the same combination of signature ID and external host IP address, they would be assigned to different alert groups.

Note that *SessionTimeout* and *SessionLength* hyperparameters determine the reporting delay of alert groups after the corresponding NIDS alerts have been observed. For example,

[1] *Botnet* is a term used to characterize a network of compromised computers that are used for malicious purposes without the knowledge of their owners.

TABLE II
EXAMPLE NIDS ALERTS

Time	SignatureID	ExternalIP	InternalIP	Other Attributes
22:52:03	23	10.1.1.1	192.168.1.1	...
22:52:04	23	10.1.1.1	192.168.1.2	...
22:52:06	51	10.1.1.1	192.168.1.1	...
22:52:11	92	10.1.1.2	192.168.1.1	...
22:53:44	23	10.1.1.1	192.168.1.2	...

if *SessionTimeout* is set to 60 seconds, an alert group is reported if it has not been updated for more than 1 minute. Also, if *SessionLength* is set to 300 seconds, alert groups are allowed to be updated during at most 5 minutes, forcing their reporting after this time frame expires. Therefore, using small values for *SessionTimeout* and *SessionLength* hyperparameters allows for close-to-real-time generation of alert groups and timely detection of security incidents.

An alert group holds all attribute values from original alerts. For example, if the activity from external host 10.1.1.1 matched signature 23 that triggered two alerts for internal hosts (i.e., hosts in the home network) 192.168.1.1 and 192.168.1.2, the *intip* attribute of the alert group created for external host 10.1.1.1 and signature 23 holds two values for these IP addresses. Each alert group has the following attributes – *signature ID*, *transport protocol* (e.g., Transmission Control Protocol (TCP)), *external* and *internal IP addresses*, *external* and *internal port numbers* (for portless transport protocols like Internet Control Message Protocol (ICMP) port numbers are set to 0), and 28 application layer specific attributes (e.g., Hypertext Transfer Protocol (HTTP) user agent). If some application layer attribute is not relevant for the given signature (e.g., the signature for matching Secure Shell (SSH) traffic does not set HTTP user agent attribute), the attribute is left unset (i.e., it has the value N/A).

After creating a new alert group, CSCAS will either regard it as an *inlier* and assign it to an existing cluster, or regard the alert group as an *outlier*. Each detected *cluster* represents a frequently matching signature, and CSCAS maintains a *cluster centroid* data structure in memory for each cluster. The *cluster centroid* is updated after a new alert group has been assigned to the cluster. For outliers, which are alert groups generated by less frequently matching signatures, a similar *centroid of outliers* is maintained. This reflects potential commonalities among recently observed outliers. In addition, outlier alert groups trigger the creation and updating of *cluster candidates*, which allows CSCAS to adapt to environmental changes. For example, if a less verbose signature becomes a frequently matching signature over time, the cluster candidate representing this signature will be promoted to a cluster.

After creating a new alert group with the signature ID attribute having the value of *sid*, it is clustered according to the following scheme, creating a *data point* that represents this alert group:

1) If there is a cluster for *sid*, the alert group is classified as an *inlier*, otherwise it is classified as an *outlier*.
2) If the alert group is categorized as an *inlier*, it is processed as follows:
   a) a *data point* is calculated, which represents the alert group, using the data stored in the *cluster centroid*,
   b) the *cluster centroid* is updated with the alert group data.
3) If the alert group is categorized as an *outlier*, it is processed as follows:
   a) if there is no *cluster candidate* for *sid*, a *cluster candidate* is initialized,
   b) the *cluster candidate centroid* for *sid* is updated with the alert group data,
   c) a data point is calculated, which represents the alert group, using the data stored in the *centroid of outliers*,
   d) the *centroid of outliers* is updated with the alert group data.

After a period of time, CSCAS checks all clusters and cluster candidates, and drops a cluster or a candidate from memory if it has not been updated for more than *ClusterTimeout* or *CandTimeout* seconds, respectively (i.e., *ClusterTimeout* and *CandTimeout* are hyperparameters of CSCAS). Therefore, if a cluster of a formerly prolific signature is updated infrequently (i.e., the last update did not occur within *ClusterTimeout* seconds), the cluster will cease to exist. Also, if the cluster candidate has been in memory for a longer period (i.e., more than *MaxCandAge* seconds) due to frequent updates (i.e., the time frame between updates has not exceeded *CandTimeout* seconds), it will be promoted as a cluster (i.e., *MaxCandAge* is a hyperparameter). This enables CSCAS to adjust to environmental changes, with *clusters* representing frequently matching signatures at the current time and *outlier alert groups* representing unusual attacks of higher importance.

For calculating the data point representative for the alert group, the attribute tables of the corresponding centroid (cluster or outlier centroid) are used. The *attribute table* for attribute *A* is a hash table which contains recently seen values of *A* as keys, with each key pointing to a value μ such that $\mu \in \mathbb{R}$ and $0 \leq \mu \leq 1$. Each value μ is maintained as an Exponentially Weighted Moving Average (EWMA) according to the following scheme:

$$\begin{cases} \mu_1 = x_1 \\ \mu_i = \alpha * x_i + (1 - \alpha) * \mu_{i-1} \ , \ i > 1, \end{cases} \tag{1}$$

where x_i refers to an observation at time i, μ_i to the weighted average value at time i and α is the weighting factor that determines the importance of older data in the calculation of the EWMA. In this regard, a large value of α (closer to 1) gives more importance to recent data in the calculation of the EWMA and less importance to older data, whereas a small value (closer to 0) does the opposite, and gives more weight or importance to older data with respect to recent data. For time series $x_1, x_2, ...$, EWMA estimates the average of

last $(2/\alpha) - 1$ observations. The *attribute table* for attribute *A* is used to calculate the *similarity score for attribute A*. The calculated similarity scores become the *features* of the data point that represents the alert group.

To understand how attribute tables are employed to generate data features, consider a newly arrived alert group that has *intip* attribute set to values 192.168.1.1 and 192.168.1.2. Suppose that the attribute table for *intip* holds two key-value pairs *192.168.1.1 = 0.5* and *192.168.1.3 = 0.2* (note that key 192.168.1.2 is missing in the table, that is, it is an attribute value not seen recently). The similarity score for attribute *intip* is calculated as an average of values for keys 192.168.1.1 and 192.168.1.2: $(0.5 + 0)/2 = 0.25$ (the value of 0 is used for missing key 192.168.1.2).

When the centroid is updated with the above alert group data, the keys 192.168.1.1 and 192.168.1.3 in the attribute table of *intip* are updated according to Equation (1) with x_i set to 1 and 0, respectively, reflecting their set/unset status in the newly received data. Also, the key 192.168.1.2 is created with the value $1/((2/\alpha) - 1)$, indicating that the IP address 192.168.1.2 was seen for the first time for the last $(2/\alpha) - 1$ alert groups. Therefore, if the similarity score of attribute *A* is *s* for the alert group, the given values of *A* have been observed for an average of $s * ((2/\alpha) - 1)$ times for the last $(2/\alpha) - 1$ alert groups.

After calculating the similarity score for all attributes in the alert group, the *similarity score of the alert group* is calculated as the average of values of these attributes. Due to the nature of the similarity score of the alert group, this reflects how frequent the attribute values of the given alert group have been among other groups from recent past. Consequently, a high degree of commonality with other alert groups from the same cluster (or with other outliers if the alert group is an outlier) produces a high similarity score.

After their calculation, the similarity score of the alert group and the similarity scores of individual attributes will serve as features of the data point. If some attribute of the alert group is not set (for example, alert group was triggered by SSH related signature that does not create HTTP-specific attributes), the corresponding features in the data point are set to -1.

In addition to the numeric features that are based on attribute tables and range from 0 to 1 or are set to -1, some essential (mostly categorical) features are added to the data point – *numerical signature ID*, *numerical transport protocol ID* (e.g., 6 denotes TCP), *external* and *internal IP addresses* as integers (if the alert group contains several internal IP addresses, the value of -1 is used for the feature), *external* and *internal port numbers* (if the alert group contains several external or internal port numbers, the value of -1 is used for the feature), the *number of alerts in the alert group*, and the *average number of alerts per day for the signature that created the alert group*. These features include traditional NIDS alert properties commonly used in related literature (e.g., [6]–[8], [13]–[15]), and their presence allows the usage of these alert data in the classification process. Since a data point is an alert group's representation that supervised classifiers can process, we will use the terms *data point* and *alert group* interchangeably in the remainder of this article.

The described algorithm, CSCAS, was tested in a production setup for the period 2021–2022. During this time, CSCAS demonstrated a high recall – the vast majority of the high priority alert groups were reported as outliers. However, the precision metric of CSCAS tended to be lower with a high rate of false positives among outliers. For example, according to the data from a 60 days-long period that will be presented in Section III-B, CSCAS misclassified less than 1% of high priority alert groups as inliers, but of the reported outliers, more than 70% were false positives. This paper addresses this important issue of obtaining a high false positive rate (i.e., too many false alarms), and proposes a framework that combines CSCAS with supervised machine learning aiming to improve the precision of NIDS alert classification.

III. NIDS Alert Processing Framework

This section describes the NIDS alert processing framework that combines the CSCAS algorithm with supervised learning, and describes the main contributions of this research. More specifically, Section III-A provides an overview of the proposed framework, and Section III-B introduces the labeled data set that was used for all the evaluations in this work. Section III-C discusses the use of supervised learning with CSCAS and evaluates several supervised machine learning algorithms. Finally, Section III-D introduces a novel CSCAS-guided training data set generation methodology that greatly reduces the human labeling effort, thus enabling the cost-efficient integration of supervised machine learning into the NIDS alert processing framework.

A. Overview of the NIDS Alert Processing Framework

This section provides an overview of the proposed NIDS alert processing framework. It also summarizes the challenges for the implementation of the framework and proposes solutions to address these challenges.

Fig. 1 provides a graphical overview of the NIDS alert processing framework that consists of *real-time* and *offline* components.

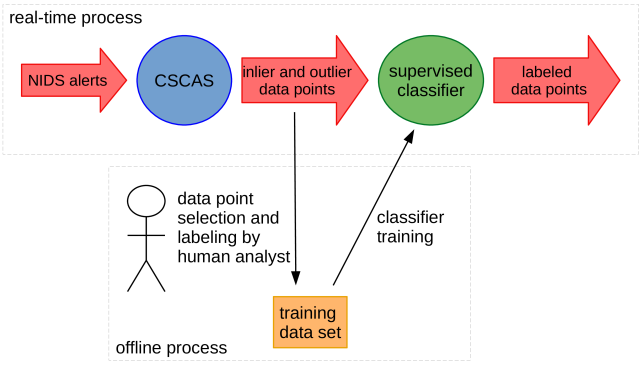

Fig. 1. NIDS alert processing framework.

The *real-time* component of the framework processes the stream of incoming NIDS alerts as follows:

- the stream of incoming NIDS alerts is aggregated into alert groups by CSCAS, so that each alert group represents one or more alerts triggered by the same signature for the same external IP address in a short time window,
- CSCAS identifies each alert group as an *inlier* or an *outlier*, and calculates a data point for the alert group,
- a previously trained classifier (i.e., a supervised machine learning classification model produced by the offline component) is applied in real time to the stream of data points created by CSCAS in order to improve the initial classification decisions made by CSCAS,
- the refined classification decisions relate to a binary classification problem. More precisely, if the data point represents an alert group that deserves closer attention from the human analyst, the classifier will label it as *important*, otherwise it will be labeled as *irrelevant*.

The *offline* component of the framework is responsible for the creation of a supervised classification model that will be used for improving the initial classification decisions made by CSCAS. The offline component involves a human analyst who needs to create a labeled data set for training a supervised classifier. During the data set labeling process, the labels *important* and *irrelevant* are used to denote alert groups of high and low importance, respectively.

The selection of the classification algorithm is one of the most important decisions when setting up the offline part of the framework. In this regard, Section III-C provides the results of the evaluation of several supervised machine learning algorithms on a large labeled data set described in Section III-B. The data set used was purposely created for this research and constitutes the second major contribution of this work, which aims to support similar future studies in the field.

The offline processing part of the framework also involves a substantial challenge – as discussed in Section I, the creation of labeled training data sets for supervised learning models is expensive and requires significant human effort and dedicated resources.

To address this challenge, several methods have been proposed for automated and semi-automated creation of labeled network security data sets in recent research (see [19] for an overview). As for automated methods, they assume a controlled network environment (e.g., a dedicated lab where known benign and malicious network traces are replayed), which eliminates the need for human labeling work [19]. However, since the NIDS alert processing framework is designed for production environments where the nature of network traffic and the importance of relevant NIDS alerts are not known in advance, such automated methods are not applicable in the context of this research.

As for semi-automated methods, they have been mostly based on *active learning* [19], [21]. In the active learning approach, the labeling process is an iterative process, where at each training step a trained classification model selects the most informative data points from an unlabeled pool of data samples for labeling by a human analyst. Once the selected samples are labeled, the data points are then used for updating the knowledge of the classifier. This aims to minimize the amount of labeling needed to achieve high performance models. Using this approach, Beaugnon et al. [22] employed an LR classifier for iterative labeling, while Torres et al. [23] proposed an RF classifier for that purpose.

However, active learning-based approaches have one subtle drawback – the labeling process is iterative and thus needs an initial training data set. However, the creation of this data set is not always straightforward [22], [29]. The use of unsupervised learning methods for assisting the human analyst during the labeling process may help to address this drawback. Despite that, the creation of network security training data sets with unsupervised methods has received little attention in the research literature.

This study fills this research gap by proposing the *CSCAS-guided labeling methodology* described in Section III-D, which is the third major contribution of this research. As shown in Section IV, the proposed labeling method requires much less human effort than traditional approaches for data set generation, and allows for the training of classifiers that feature high precision and recall metrics. In this context, *precision*, also called *positive predictive value*, is a measure of classification quality referring to the proportion of data points correctly predicted as important (*positive* class) among all instances labeled as important, while *recall*, also referred to as *sensitivity*, reports the number of important or positive class predictions among all important instances evaluated. In this regard, if *TP* denotes the number of true positives, *FP* the number of false positives, and *FN* the number of false negatives, then $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$. A related relevant performance metric is the *F1-score* or *F-Measure*, which reports in a single numeric value the harmonic mean of precision and recall on the test set (i.e., $F1\text{-}score = 2 \times \frac{precision * recall}{precision + recall}$).

B. The NIDS Alert Group Data Set

To collect data for the evaluation of different supervised machine learning algorithms, we set up CSCAS to process alerts from Suricata NIDS in the Tallinn University of Technology (TalTech) Security Operations Center (SOC). TalTech is the second largest university in Estonia with about 12,000 organizational users and a complex network infrastructure. A detailed description of TalTech SOC can be found in [24]. The data was collected during 60 days in January–March 2022 from the Suricata NIDS that was running on the external network perimeter of TalTech. During the data collection period, Suricata triggered alerts for malicious network traffic between 45,339 external hosts and 4,401 hosts of TalTech. This section describes the data collection process and provides insights into the properties of the collected data set. The labeled data set is made publicly available at *https://github.com/ristov/nids-alert-data*.

During the data collection period, CSCAS was running with the following settings – *SessionLength* = 300 (5 minutes),

SessionTimeout = 60 (1 minute), *ClusterTimeout* = 604,800 (1 week), *CandTimeout* = 36,000 (10 hours), *MaxCandAge* = 864,000 (10 days), and α = 0.01. These settings were found to be optimal in past experiments [4] and have been used for CSCAS in a production environment since 2021. As explained in Section II-B, setting *SessionTimeout* and *SessionLength* hyperparameters to 1 and 5 minutes allows for timely generation of alert groups. By setting *CandTimeout* and *MaxCandAge* parameters to 10 hours and 10 days, respectively, a cluster candidate must be updated at least once with a new alert group every 10 hours for 10 days for the candidate to become a cluster. In other words, clusters are created for signatures that are frequently matched in a 10-day period. Using a large value for *MaxCandAge* prevents CSCAS from creating clusters for intensive attack activities that correspond to infrequently matching signatures and could last for several days before attack mitigation measures take effect. In addition, setting *ClusterTimeout* to 1 week forces CSCAS to drop clusters for previously prolific signatures that have not matched any network traffic for an entire week. This setting of *ClusterTimeout* also prevents accidental dropping of clusters when the NIDS has temporarily not seen some network traffic due, for instance, network maintenance procedures (assuming they last less than a week). Finally, setting $\alpha = 0.01$ implies that numeric features of the alert group are calculated based on the previous $(2/\alpha) - 1 = 199$ alert groups from the same cluster. This allows adjusting to environment changes, while also considering a sufficient number of past alert groups when calculating the features. For instance, for a cluster that is updated about once in every hour, setting $\alpha = 0.01$ takes into account the alert groups for the last 199 hours (slightly more than a week).

Although Suricata NIDS had 50,000+ signatures deployed, only about 1% of signatures triggered alerts during the 2-month-long data collection time frame. More precisely, Suricata generated 11,598,649 alerts in 60 days, which were reduced to 1,395,324 alert groups by CSCAS, achieving a high alert compression rate of 8.3 alerts per group. CSCAS classified 72,672 alert groups (about 5.2% of the data) as outliers and the remaining 1,322,652 (about 94.8%) as inliers.

During the clustering process, the number of clusters ranged from 40 to 43, and the number of cluster candidates from 22 to 258. The outliers were generated by 547 signatures, whereas 44 signatures triggered inliers (note that since the set of signatures associated with clusters changed during the data collection period, some signatures generated both inliers and outliers). The vast majority of inlier alert groups corresponded to threats of low importance (e.g., scanning for well-known vulnerabilities) and frequently occurring false positives. In this regard, Table III presents the five most verbose signatures that generated 77.9% of inlier alert groups.

To establish the ground truth for the experiments described in this study, we reviewed the alert group data set and labeled each data point either as *irrelevant* or as *important*. The data set was also split into two parts – the first part (i.e., 10% of data points), which contains the data belonging the first 6 days of the 60 day time frame, was used as the training data set for the experiments, while the second part (i.e., 90% of data points), covering 54 days of data, was used as the testing set. This arrangement reflects the actual situation in production environments, where classifiers are trained on past data and knowledge about future events is not available at the training stage. Table IV provides more detailed information about the data set.

TABLE III
TOP 5 SIGNATURES FOR INLIER ALERT GROUPS

Signature text	Number of inlier alert groups
GPL SNMP public access udp	592,445 (44.8%)
GPL DNS named version attempt	135,622 (10.3%)
GPL RPC xdmcp info query	135,350 (10.2%)
ETPRO USER_AGENTS Observed Suspicious UA (Mozilla/5.0)	84,112 (6.4%)
ETPRO DOS Possible RPC Portmapper Scanning	82,503 (6.2%)

TABLE IV
ALERT GROUP DATA SET

	Full data set	Training data	Test data
Time coverage	Jan 20 00:00 – Mar 20 23:59	Jan 20 00:00 – Jan 26 06:23	Jan 26 06:23 – Mar 20 23:59
Number of data points	1,395,324	139,532	1,255,792
Number of irrelevant data points	1,374,372	137,767	1,236,605
Number of important data points	20,952	1,765	19,187
Number of inliers	1,322,652	133,614	1,189,038
Number of outliers	72,672	5,918	66,754
Number of inliers labeled as important	195	0	195
Number of outliers labeled as irrelevant	51,915	4,153	47,762

As can be seen in Table IV, the NIDS alert group data set is highly imbalanced, with 20,952 data points (about 1.5%) having the label *important* and 1,374,372 data points (about 98.5%) having the label *irrelevant*. Class imbalance is a commonly occurring scenario for NIDS and security alert data sets. For example, in [9] and [10], data sets with binary labels were described, with high priority alerts constituting less than 6% and 1% of total data points, respectively.

According to Table IV, a small fraction of *important* data points (i.e., 195 out of 20,952 or 0.9%) were regarded as *inliers* by CSCAS, and the vast majority of them (20,757 out of 20,952 or 99.1%) were identified as *outliers*. Therefore, CSCAS outlier detection mechanism yielded a *recall* or *true positive rate* of over 0.99 when identifying *important* data points. On the other hand, a large fraction of the data points classified by CSCAS as *outliers* (51,915 out of 72,672 or

71.4%) were actual *irrelevant* data points, and thus the *precision* of CSCAS outlier detection mechanism was below 0.3.

When investigating the reasons for the low *precision* of CSCAS, we discovered that it was mainly caused by aperiodic network scanning events for well-known vulnerabilities. Although this network activity created a CSCAS cluster candidate, the candidate was not updated frequently enough to be promoted to a cluster. An illustrative example of these events is the *scanning* activity detected by the *ET EXPLOIT Mikrotik Winbox RCE Attempt (CVE-2018-14847)* signature from the *Emerging Threats* rulebase. In this regard, although the signature could generate hundreds of alerts during intensive scans by the same attacker, there were typically at most a couple of such scans per week in our environment, and sometimes the signature did not trigger any alerts for several weeks.

C. Addressing CSCAS Limitations with Supervised Learning

To improve the precision of CSCAS, we experimented with several traditional supervised machine learning algorithms, and selected three decision tree-based classification methods. These were RF, AB with decision tree estimators, and GBDT. The rationale behind the selection of these algorithms were the following:

- these supervised learning algorithms are known to work well with data sets containing categorical features (i.e., not only numeric attributes),
- the training and inference of induced classifiers using these algorithms are not computationally demanding and can be easily accomplished on virtual machines or commodity hardware (e.g., on the security analyst laptop or a low-end server).

To compare the performance of these supervised learning algorithms with other traditional machine learning algorithms that require numeric features, we also evaluated LR, *k*-NN, and linear SVM algorithms, excluding categorical features from consideration for these methods.

When creating a labeled data set for training supervised learning algorithms, the traditional approach involves the labeling of all data points that are designated for training purpose. Since this often creates an imbalanced data set [9], [10], previous related works used techniques like *undersampling* [9] or machine learning methods that are not sensitive to class imbalance [10]. However, these approaches assume the prior labeling of the entire training data set, which involves a significant effort from the human expert. For example, as can be seen in the middle column in Table IV, 139,532 data points would need to be inspected and labeled in the case of our data set. In another example [17], a training data set with 7 million labeled data points was used, requiring a colossal effort from human analysts.

In our case, to train supervised machine learning algorithms on a data set created in a traditional way, we first used *random undersampling* (a technique commonly used to balance categories in imbalanced data sets) – after labeling all 139,532 training data points, we selected all 1,765 *important* data points from the training data (see the middle column in Table IV), and used random sampling to select 1,765 *irrelevant* data points from the training data. Given the stochastic nature of the sampling technique applied to the *irrelevant* class to balance the data, we created 5 different training data sets with 3,530 points each (i.e., all 1,765 *important* data points and 1,765 randomly selected *irrelevant* data points) and tested the performance of each trained classifier on the test data (see the right column in Table IV). Table V reports the average precision and recall for the induced classifiers (i.e., using the 5 different training sets per classification algorithm)[2].

TABLE V
PERFORMANCE COMPARISON OF SELECTED CLASSIFIERS

Classifier	Random undersampling			Guided by CSCAS		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
RF	0.669	0.963	0.789	0.868	0.952	0.908
AB	0.740	0.979	0.841	0.866	0.907	0.885
GBDT	0.727	0.956	0.824	0.858	0.909	0.883
LR	0.616	0.983	0.757	0.681	0.956	0.796
SVM	0.621	0.972	0.757	0.690	0.937	0.795
k-NN	0.558	0.988	0.713	0.606	0.979	0.748

According to Table V, although supervised classifiers were able to improve CSCAS precision (i.e., doubling it), it still remained relatively low, with the highest precision value being 0.74. Therefore, the usage of the random undersampling technique to balance the training data sets not only involves a significant effort for labeling the entire training data set, but it also offers a relatively modest precision improvement over CSCAS.

When investigating the relatively low precision yielded by the random undersampling technique, we discovered that it was caused by the presence of too few training samples covering classification mistakes made by CSCAS. Since the precision of CSCAS was fairly low and recall was very high, most classification mistakes involved reporting *irrelevant* data points as outliers (see the last two rows in Table IV). However, since NIDS alert group data was highly imbalanced, the vast majority of data points were *irrelevant*, and *irrelevant* data points identified as outliers (i.e., most common CSCAS classification mistakes) formed only a small fraction of the training data. For example, according to Table IV there are 4,153 of such data points in the training data, which constitute about 3% of all 137,767 *irrelevant* data points. Therefore, when *irrelevant* data points are randomly selected for the training data set, the induced classifier will likely not have enough samples to learn from CSCAS classification mistakes.

To illustrate the importance of having more of these samples in the training data set, we developed and implemented a balanced training data set generation methodology named *Guided by CSCAS* that takes into account the classification

[2]In the rest of the article, all precision, recall and F1-score results are average values over 5 different training data sets generated using the same methodology. For the sake of brevity, we will not reiterate this fact for any further experiment.

information from CSCAS (i.e., whether a point is regarded as an inlier or as an outlier). Using this methodology, the training data set composition was constrained to respect the following proportions:

- 50% of *important* data points. In our implementation, all *important* data points from the training data were selected (1,765 points in total),
- 25% of *irrelevant* inlier data points. For our experiments, the *irrelevant* inlier data points from the training data were selected using random sampling (882 points in total),
- 25% *irrelevant* outlier data points. In our implementation, the *irrelevant* outlier data points from the training data set were selected by random sampling (883 points in total). This category of points represents CSCAS classification mistakes, ensuring the presence of these relevant data points in the training data set for model learning (note that this category lacks *important* inlier data points as these CSCAS classification errors were not present in the training data set, as reported in Table IV).

The experimental tests performed to evaluate the random undersampling were repeated (i.e., same classification algorithms) but using the data set constraints described above (i.e., *Guided by CSCAS* methodology).The results are provided in Table V (three rightmost columns). More precisely, the table reports the average precision, recall, and F1-score for all classifiers trained on above data sets with size 3,530. According to these results, the precision of all classifiers improved. The improvement was less significant for LR, *k*-NN, and SVM, with their precision remaining below 0.7 and F1-score below 0.8. The performance of decision tree-based methods was better with F1-scores exceeding 0.88. RF outperformed all the other algorithms evaluated, yielding the highest precision (0.868), recall (0.952), and F1-score (0.908) metrics. The experiment results indicate that increasing the number of training data points about CSCAS classification errors can significantly increase the performance of all methods and, particularly, for decision tree-based algorithms.

These results motivated the development of a CSCAS-guided training data set creation procedure, which is described in the next section. Furthermore, since according to Table V the RF algorithm provided the best performance among all tested classifiers, RF was used for all further experiments performed in this research, which are explained in Section IV (i.e., supervised NIDS alert group classification).

D. CSCAS-Guided Training Data Set Generation

The purpose of the training data set generation procedure presented in this section is to reduce the labeling workload of the human analyst and create a training data set of moderate size, so that classifiers trained on this data set would feature high performance.

Alert group stream G is defined as an infinite sequence $G = (g_1, g_2, ...)$, where the occurrence time of $g_i \in G$ is denoted by t_{g_i}. We assume that if $i < j$, then $t_{g_i} \leq t_{g_j}$ (ordered set). The training data set generation problem is defined as follows – at time t, the human analyst has access to a finite subset $S = (s_1, ..., s_n)$, $S \subset G$, $t_{s_n} \leq t$, to create the training data set $D = \{x_1, ..., x_k \mid x_i \in S\}$, so that $k << n$. In other words, the training data set D must have a moderate/small size and include a small fraction of data points from the alert group data set S available to the human analyst. The training data set will be used to create a classification model for predicting the category/label of alert groups $g_i \in G$, so that $t \leq t_{g_i}$. If x is a data point, let $h(x)$ denote the label or category assigned to x by the human analyst.

The training data set creation procedure, named as *SelectDataPoints*, is provided in pseudo-code in Algorithm 1. The procedure creates the training data set D as a union of the disjoint sets A, B, and C, where A contains *irrelevant* inlier points, B contains *important* outlier points, and C contains points that represent CSCAS classification errors (i.e., *irrelevant* outliers and *important* inliers).

Algorithm 1 *SelectDataPoints*

Input: k_1 (size of A), k_2 (size of B), k_3 (size of C)
Output: D (training data set)

Step 1:
$A \leftarrow \varnothing$, $B \leftarrow \varnothing$, $C \leftarrow \varnothing$
$I \leftarrow \{$ all inlier data points from S $\}$
$O \leftarrow \{$ all outlier data points from S $\}$
Step 2:
$x \leftarrow$ select randomly from I
$I \leftarrow I \setminus \{x\}$
$h(x) \leftarrow$ label x manually
if $h(x) =$ "irrelevant" **then** $A \leftarrow A \cup \{x\}$
if $h(x) =$ "important" $\wedge\, |C| < k_3$ **then** $C \leftarrow C \cup \{x\}$
if $|A| = k_1$ **then** GoTo Step 3 **else** Repeat Step 2
Step 3:
$y \leftarrow$ select randomly from O
$O \leftarrow O \setminus \{y\}$
$h(y) \leftarrow$ label y manually
if $h(y) =$ "important" **then** $B \leftarrow B \cup \{y\}$
if $h(y) =$ "irrelevant" $\wedge\, |C| < k_3$ **then** $C \leftarrow C \cup \{y\}$
if $|B| = k_2$ **then** GoTo Step 4 **else** Repeat Step 3
Step 4:
if $|C| = k_3$ **then** GoTo Step 5
$x \leftarrow$ select randomly from I
$I \leftarrow I \setminus \{x\}$
$y \leftarrow$ select randomly from O
$O \leftarrow O \setminus \{y\}$
$h(x), h(y) \leftarrow$ label x manually, label y manually
if $h(x) =$ "important" **then** $C \leftarrow C \cup \{x\}$
if $|C| = k_3$ **then** GoTo Step 5
if $h(y) =$ "irrelevant" **then** $C \leftarrow C \cup \{y\}$
Repeat Step 4
Step 5:
return $D \leftarrow A \cup B \cup C$

Step 1 of the procedure (lines 1–3) initializes A, B, and C as empty sets, and assigns all inliers and outliers in S to sets I and O, respectively.

In Step 2, the human analyst builds sets A and C by labeling randomly selected inliers from I (lines 4–6). If the selected inlier x is labeled as *irrelevant*, it is assigned to set A (line 7). If x is labeled as *important*, it is assigned to set C (line 8) because it represents a CSCAS classification error. Note that if the cardinality of C is equal to k_3, x will not be added to C. Step 2 repeats until the cardinality of A is k_1 (line 9).

In Step 3, the human analyst builds B and C by labeling randomly selected outliers from O (lines 10–12). If the selected outlier, y, is labeled as *important*, it is assigned to B (line 13). If y is labeled as *irrelevant*, it is assigned to C as it relates to a CSCAS classification error, provided that the cardinality of C is not equal to k_3 (line 14). Step 3 repeats until the cardinality of B is k_2 (line 15).

In the beginning of Step 4, the cardinality of C is checked, skipping the remainder of step 4 if C already has the desired size (line 16). If C contains fewer than k_3 data points, the human analyst augments C by labeling randomly selected inliers and outliers from I and O, respectively (lines 17–21). If the selected inlier x is labeled as *important* or the selected outlier y is labeled as *irrelevant*, it represents a CSCAS classification error and is assigned to C (lines 22 and 24). Step 4 repeats (line 25) until the cardinality of C is k_3 (lines 16 and 23).

Finally, Step 5 returns the training data set D as the union of A, B, and C, which have the expected sizes k_1, k_2, and k_3, respectively (line 26).

To quantify the workload needed to carry out the described procedure, the following paragraphs provide an analysis of the average amount of work a human analyst would need to accomplish in terms of the average number of data points that need inspection and labeling. If p is the probability that an inlier receives the label *important*, the human analyst would need to label $\frac{1}{1-p} \cdot k_1$ data points during Step 2 to build set A. Besides, $p \cdot (\frac{1}{1-p} \cdot k_1)$ of the labeled data points would be added to set C in Step 2.

If $prec$ is the precision of CSCAS, the human analyst would need to label $\frac{1}{prec} \cdot k_2$ data points during Step 3 to generate set B. Also, $(1 - prec) \cdot (\frac{1}{prec} \cdot k_2)$ from labeled data points would be added to set C in Step 3.

Finally, before Step 4, the set C would already contain $m = p \cdot (\frac{1}{1-p} \cdot k_1) + (1 - prec) \cdot (\frac{1}{prec} \cdot k_2)$ data points, thus needing additional $d = k_3 - m$ points. At each iteration of Step 4, $(1 - prec) + p$ points would be added to set C, so for adding d points, $\frac{d}{1-prec+p}$ iterations would be needed. Since each iteration involves the inspection of two data points, the human analyst would need to label $\frac{2 \cdot d}{1-prec+p}$ data points during Step 4.

To better illustrate the explanation described above and the computations performed for the estimation of workload needed, the following paragraphs elaborate further by providing a more specific example, first with parameters and later with real values.

Suppose that n is the desired training data set size and $k_1 = \frac{n}{4}$, $k_2 = \frac{n}{2}$, and $k_3 = \frac{n}{4}$ (i.e., the training data set has the same structure/proportions as the one generated using the *Guided by CSCAS* methodology in Section III-C). Furthermore, suppose that the precision of CSCAS is $prec = \frac{1}{3}$ and that the proportion of *important* data points among inliers is very small, so that we can assume $p = 0$. In this case, the human analyst would need to label $\frac{n}{4}$ data points to generate set A during Step 2, while no points would be added to set C in this step. During Step 3, $3 \cdot \frac{n}{2}$ data points would need labeling to build set B, and during this process $(1 - \frac{1}{3}) \cdot (3 \cdot \frac{n}{2}) = n$ data points would be found suitable for set C. However, since this would exceed the desired size of set C, only the first $\frac{n}{4}$ points would be added to C (as described in line 14 of the *SelectDataPoints* procedure in Algorithm 1). As the set C would already have the required size before Step 4, Step 4 would be skipped (as provided in line 16 of the *SelectDataPoints* procedure in Algorithm 1). As a result, the human analyst would need to label $\frac{n}{4} + 3 \cdot \frac{n}{2} = 1.75 \cdot n$ data points to build the training data set D of desired size n.

If we substitute the parameters for actual values, this shows that, for example, to create a training data set of size 3,530 for the above scenario from our training data (see the middle column in Table IV), it would require the labeling of only 1.75*3,530 $\approx$ 6,178 data points out of 139,532 points. In contrast, traditional data set creation procedures like random undersampling would require the labeling of the whole training data set, that is, 139,532 data points. Therefore, the proposed *SelectDataPoints* procedure would reduce the workload of the human analyst by more than 20 times for the above scenario. A more detailed analysis and discussion of the human labeling effort required for different training data set generation scenarios is provided in Section IV-B.

IV. Evaluation

This section describes the evaluation of different aspects of the proposed NIDS alert processing framework. More specifically, Section IV-A explores and evaluates different training data set generation strategies, assessing them based on the performance of RF classification models trained on respective data sets, while Section IV-B evaluates the human labeling effort that the best two strategies involve. Section IV-C analyzes the performance of classifiers for detecting critical alert groups that represent serious incidents (e.g., successful intrusions), employing RF classifiers trained on data sets created using the best strategy. Section IV-D investigates the interpretability of the RF models induced during the experiments and discusses the most important features of the models. Finally, Section IV-E describes the implementation of the NIDS alert processing framework, and evaluates CPU and memory requirements during real-time NIDS alert classification.

A. Evaluation of Strategies for Generating Training Data Sets

This section describes the evaluation of different training data set generation strategies using the proposed *SelectDataPoints* procedure. The generated data sets were used to train

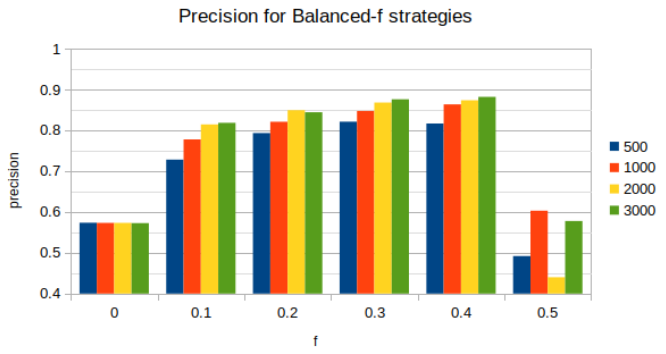

Fig. 2. Precision for Balanced-f strategies.

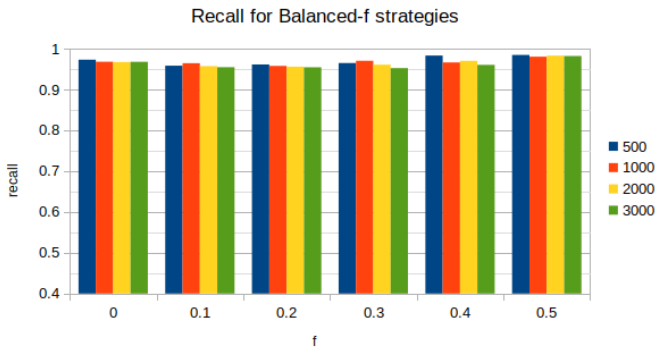

Fig. 3. Recall for Balanced-f strategies.

RF-based classifiers and assess their performance on the test data set (see the right column in Table IV). In the following paragraphs, n denotes the size of the training data set (i.e., $|D| = n$).

Initially, we evaluated training data set generation strategies that correspond to the following settings of the *SelectDataPoints* procedure: $k_1 = (0.5 - f) \cdot n, k_2 = 0.5 \cdot n, k_3 = f \cdot n$, where $0 \leq f \leq 0.5$. Specifically, f refers to the fraction of data points in the training data set that belong to set C (i.e., CSCAS classification errors). Since CSCAS has shown high recall and fairly low precision in a production environment, most points in the C set have the label *irrelevant* as a result. Also, since set A contains only *irrelevant* points and set B only *important* points, the data sets that are generated with the above settings of *SelectDataPoints* have a balanced (or almost balanced) nature. Therefore, we denote these data set generation strategies as *Balanced-f*.

According to recent malware detection research, the usage of imbalanced training data sets can yield classifiers that feature high-performance metrics [25]. This motivated us to evaluate *Imbalanced-f* strategies for generating the training data sets. The *Imbalanced-f* strategies correspond to the following settings of the *SelectDataPoints* procedure: $k_1 = k_2 = \frac{n-k_3}{2}, k_3 = f \cdot n$, where $0 \leq f \leq 1$. As in the *Balanced-f* strategy, f denotes the fraction of points in the training data set belonging to set C. However, in this case, the remaining data points are evenly divided between sets A and B, which leads to class imbalance (i.e., larger values of f increase the imbalance). For example, *Imbalanced-0.5* corresponds to the settings: $k_1 = 0.25 \cdot n, k_2 = 0.25 \cdot n, k_3 = 0.5 \cdot n$.

To evaluate the *Balanced-f* strategies, we set the values for f as follows: $f \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. For the *Imbalanced-f* strategies, the following values were tested: $f \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Note that we did not evaluate *Imbalanced-0* and *Imbalanced-1*, since *Imbalanced-0* is equivalent to *Balanced-0*, and *Imbalanced-1* would produce training data sets without *important* class samples using our training data (see the last two cells of the middle column in Table IV). All strategies were evaluated for small training data set sizes, $n \in \{500, 1000, 2000, 3000\}$, which would keep the workload

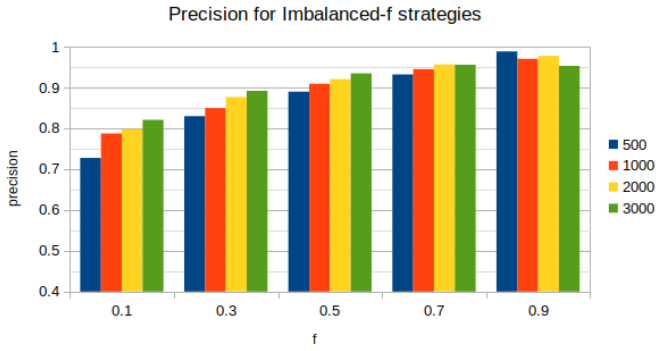

Fig. 4. Precision for Imbalanced-f strategies.

of the human analyst low in real-life environments. The bar charts in Fig. 2 and Fig. 3 report the precision and recall of RF classifiers on the test data set (see the right column of Table IV) for the *Balanced-f* strategies, while Fig. 4 and Fig. 5 provide the precision and recall metrics for the *Imbalanced-f* strategies.

As can been seen in Fig. 3, recall remained high for the *Balance-f* strategies for all values of f and n. However, as Fig. 2 reveals, the precision of the *Balanced-0* strategy was below 0.6, worse than the precision of the RF model using random undersampling, as reported in Table V. For $f = 0$, the training data contains no samples about CSCAS classification mistakes. The lack of these data points leads to a classifier that behaves similarly to CSCAS, featuring a low precision. From $f = 0.1$ until $f = 0.4$, precision gradually improved, with larger training data set sizes showing slightly better precision for the same value of f. The *Balanced-0.4* strategy yielded the best overall performance with the highest F1-scores for all training data set sizes, as shown in Fig. 6, while precision dropped suddenly for *Balanced-0.5*. In this case, for $f = 0.5$, the set A remains empty, and thus the training data contains no *irrelevant* data points correctly detected as inliers by CSCAS. This fact complicates the proper identification of such data points. In this relation, when investigating the classification mistakes of the classifier, it was discovered that a large number

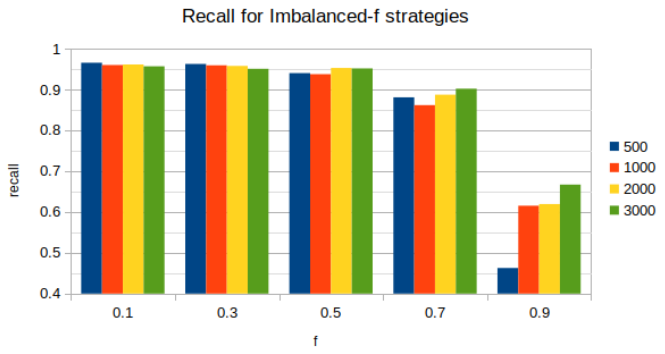

Fig. 5. Recall for Imbalanced-f strategies.

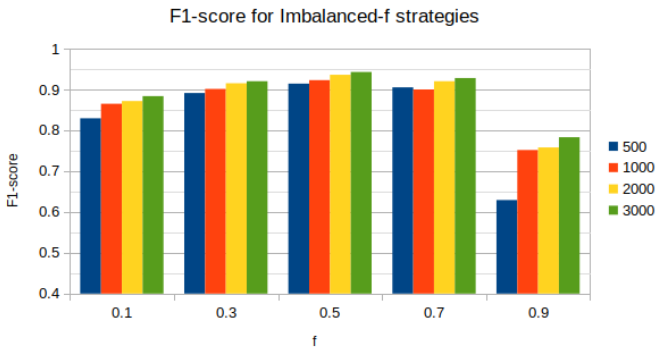

Fig. 7. F1-score for Imbalanced-f strategies.

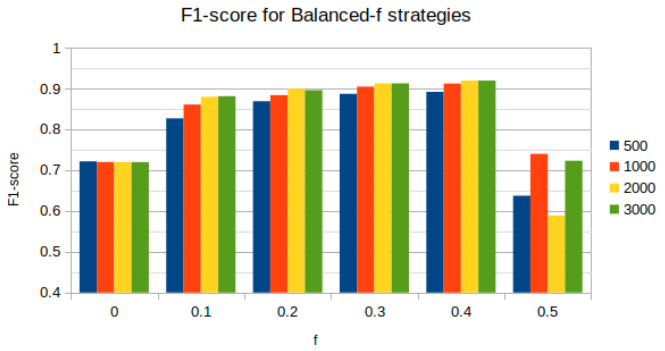

Fig. 6. F1-score for Balanced-f strategies.

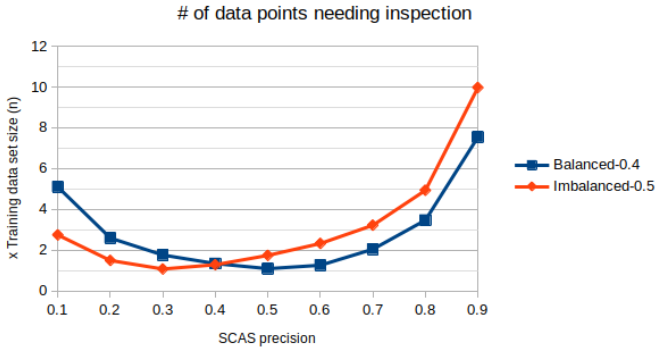

Fig. 8. Labeling effort for different training data set generation strategies.

of *irrelevant* inliers were mistakenly classified as *important*, thus leading to a sharp decline in precision.

As Fig. 4 indicates, precision for the *Imbalanced-f* strategies increased with the value of f, and in most cases larger training data set sizes provided a slightly better precision. Although recall was 0.94–0.97 for $f \leq 0.5$ (see Fig. 5), it dropped to 0.86–0.90 for $f = 0.7$, and decreased sharply for $f = 0.9$, when the training data was highly imbalanced. As can be seen in Fig. 7, the *Imbalanced-0.5* strategy offered the best performance, showcasing the highest F1-score value for all training data set sizes.

When the two best strategies are compared, *Balanced-0.4* and *Imbalanced-0.5*, the former reached F1-score metrics of 0.89–0.92 (see Fig. 6), while the latter showed higher F1 values of 0.91–0.94 (see Fig. 7). The lower F1-score values of the *Balanced-0.4* strategy were due to its lower precision, ranging from 0.82 to 0.88 (see Fig. 2), while for the *Imbalanced-0.5* strategy, the precision ranged between 0.89 and 0.93 (see Fig. 4). As these results indicate, RF-based classifiers achieve the best performance metrics when trained on moderately imbalanced data sets, where class balance is sacrificed for the sake of proper representation of CSCAS classification errors.

B. Analysis of Human Labeling Effort

The best two overall strategies, *Balanced-0.4* and *Imbalanced-0.5*, can also be compared in terms of the human labeling effort required. In this regard, Fig. 8 reports the average number of data points a human expert would have to inspect for both strategies. The results reported in Fig. 8 are based on the analysis performed in Section III-D for different CSCAS precision values, estimating $p = 0$.

As can be observed in Fig. 8, when the precision of CSCAS is 0.3 (the closest value to CSCAS precision in our data set, as reported in Table IV), the human analyst would have to inspect $1.77 \cdot n$ data points to create the training data set of size n with the *Balanced-0.4* strategy. In contrast, for the *Imbalanced-0.5* strategy $1.08 \cdot n$ points would need inspection (i.e., the labeling overhead is only 8%). However, in the case of higher CSCAS precision values starting from 0.5, the *Balanced-0.4* strategy involves less amount of human labor. Despite that, for very high precision values, starting from 0.9, the training data sets are more expensive to create with both strategies. For example, to produce a data set with 1,000 data points when CSCAS precision is 0.9, the human analyst would need to inspect 7,500–10,000 data points.

Since the *Imbalanced-0.5* strategy was the best in terms of both performance and human labeling effort, we investigated the impact of the training data set size on the performance of

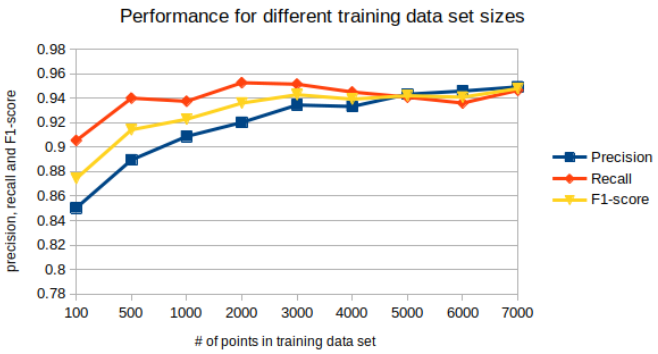

Fig. 9. The impact of training data set size on performance.

the classification model, and evaluated 9 data set sizes, ranging from 100 to 7,000 data points. The results are provided in Fig. 9. As described in Section III-C, each value in Fig. 9 represents the average value for 5 different data sets of the same size. Consequently, the data reported in Fig. 9 represents the performance of 45 RF-based classifiers. As the line chart indicates, increasing the data set size from 100 points to 3,000 led to the greatest improvements in performance, while increasing the data set size further kept the F1-score at the same level. Therefore, the RF classifier can feature a high performance even when the training data set is small and involves less human labeling effort than the creation of larger data sets.

C. Detection of Critical Alert Groups

In SOC environments, the identification of alerts that correspond to serious incidents such as successful intrusions is particularly important so that these alerts can be treated with the highest priority [26].

To assess the precision and recall of classifiers on handling novel attacks of critical importance, we set up a dedicated honeypot server that ran for 3 non-consecutive days on the external network perimeter of TalTech, monitoring the attacks against the server with Suricata NIDS. More specifically, the honeypot server ran the *T-Pot* honeypot distribution [27], which contains more than 20 honeypots for different application layer protocols and services such as HTTP, SSH, and Teletype Network (TELNET). This allowed us to lure attackers into conducting a wide variety of attacks, which were often successful. Since several honeypots ran services not present in the organizational outer network perimeter, many attacks were unique to the honeypot server. We measured the performance of classifiers (i.e., induced using the same combinations as in Fig. 9) on alert groups generated by CSCAS for the honeypots.

During the 3 days of the experiment, 542 alert groups were created by 52 signatures for the honeypots. From these, CSCAS detected 82 alert groups as inliers and 460 as outliers, with 123 of them being actual *irrelevant* and 419 actual *important* alert groups. As a result, CSCAS yielded a precision of 0.91 and a recall of 1.0 (similar precision and recall were provided by SCAS detecting unusual alert groups triggered by pen-testing activity in [4]). Regarding the *irrelevant* alert groups, they were produced by 16 signatures, with the first, second, and fifth most prolific signatures from Table III being among them. The *important* alert groups were produced by 36 signatures, with 28 signatures triggering alerts only for honeypots. As a result, 405 out of 419 *important* alert groups were produced by these 28 signatures. Therefore, most of *important* alert groups reflected novel attacks.

The classification models trained on data sets of size 100 demonstrated lower performance than the classifiers built with other data set sizes. More precisely, the models built with data sets of size 100 provided an F1-score value ranging from 0.928 to 0.987. The classifiers trained on data set sizes 500–7,000 did not generate any false positives and thus reported a precision of 1.0. The number of false negatives ranged from 0 to 4, yielding F1-scores ranging from 0.995 to 1.0 (3 classifiers achieved an F1-score of 1.0). These results illustrate that RF-based classifiers can provide a very high performance for detecting critical alert groups, even when training data sets of small size are used (e.g., 500 points). The performance starts to degrade only for very small training data sets (e.g., 100 points).

D. Analysis of Most Important Features

To understand what features were the most important for the classification models discussed in the previous section, we investigated the 45 RF models reported in Fig. 9. For this purpose, *feature* or *variable importance*, a built-in *importance* metric of RF models was used [28]. All models shared the same top 3 features, in the exact same order but with different *importance* scores. Table VI details the most important features for the models with their minimum and maximum importance values. The following paragraphs discuss each feature in detail, thus increasing the interpretability of the classification models used in this study.

TABLE VI
MOST IMPORTANT FEATURES FOR TRAINED RF MODELS

Ranking	Feature	Min importance	Max importance
1	SignatureMatchesPerDay	0.191	0.326
2	SignatureIDSimilarity	0.106	0.178
3	Similarity	0.056	0.118

The *SignatureMatchesPerDay* feature reflects the average number of alerts per day triggered by the signature that also generated the current alert group. As discussed in Section II-B, alerts that are generated by frequently matching signatures represent well-known threats of low importance, and therefore, large values of *SignatureMatchesPerDay* are common for *irrelevant* alert groups. In contrast, *important* alert groups are often triggered by infrequently matching signatures, and the value of *SignatureMatchesPerDay* is usually much smaller in these cases.

The *SignatureIDSimilarity* feature is the similarity score of the signature ID attribute (calculated as described in Section

II-B), which reflects how often the signature ID of the current alert group has been seen among recent alert groups that belong to the same cluster (or among recent outliers if the current alert group is an outlier). Since all alert groups from the same cluster share the same signature ID, the value of *SignatureIDSimilarity* quickly converges from the initial value to 1 for an alert group that belongs to a cluster (see Equation (1) in Section II-B). If the alert group is an outlier, the value of *SignatureIDSimilarity* is usually very small, since outliers are generated by a wide variety of signatures. In contrast, larger values of *SignatureIDSimilarity* indicate that the current signature has triggered many outliers in the recent past. This usually happens when a new cluster is about to form for a signature that has become prolific, or when a signature is matching aperiodic network scanning that triggers a large number of alerts in a short time frame. Analyzing our data set (described in Table IV), we found that for *important* outliers the average value of *SignatureIDSimilarity* was 0.05, while for *irrelevant* outliers it was 0.71. Therefore, a small value of this feature tends to indicate that the alert group is *important*.

The *Similarity* feature is the similarity score of the alert group (as detailed in Section II-B), which reflects the average similarity of an inlier alert group with other recently seen groups from the same cluster, or the average similarity of an outlier alert group with other recently seen outliers. As with the *SignatureIDSimilarity* feature, higher values of *Similarity* are common for *irrelevant* alert groups, while *important* alert groups have usually smaller values – the average values in our data set were of 0.86 and 0.21, respectively. Furthermore, *irrelevant* outliers tend to have a higher *Similarity* value than *important* ones, with the respective averages in our data set being 0.45 and 0.20.

In conclusion, the features from Table VI were used by the RF-based classifiers induced in this research to distinguish effectively *important* alert groups from *irrelevant* ones. In addition, if there is no opportunity to train a supervised classifier or implement the proposed framework, these features allow the human analysts to improve the low precision of CSCAS by creating filtering rules to exclude as many *irrelevant* outliers as possible from further consideration.

E. Framework Implementation and Performance

We have created an implementation of the NIDS alert processing framework described in Section III-A that is publicly available at *https://github.com/ristov/nids-alert-proc*. The framework is designed for Suricata NIDS. CSCAS is implemented in Perl and the supervised learning module in Python using *scikit-learn* library.

To measure the resource consumption of the framework proposed in this study, the framework was implemented and evaluated for 1 month on a Linux server (Intel Xeon Silver 4214 CPU and 64 GB of memory) processing alerts from Suricata NIDS at the external network perimeter of TalTech. During the testing period, about 200,000 NIDS alerts per day were processed by the framework in real time. When processing the stream of incoming NIDS alerts, the maximum memory footprint of CSCAS reached 445.0 MB and its average CPU utilization was 0.9% (per one CPU core, i.e., 100% would mean that only one CPU core was fully utilized). When the RF-based classifier was processing the alert group data points created by CSCAS, its average CPU utilization was 0.4% and the maximum memory footprint reached 108.4 MB. Consequently, as these figures indicate, the overall CPU and memory consumption of the framework are very low, making the framework suitable to be employed on commodity hardware, virtual machines, or directly on NIDS appliances.

V. LIMITATIONS

Despite the high-performance metrics reported by the proposed framework in our experimental setup, the system is not free of limitations. The following paragraphs expose the limitations of the framework.

One of the limitations of the NIDS alert processing framework is its inability to detect an unexpected significant increase in the number of *irrelevant* alert groups that might manifest a DDoS attack[3], intensive reconnaissance activity from botnets, and other abnormal network activity that deserves closer attention. For example, although the *GPL SNMP public access udp* signature in Table III might usually trigger alert groups representing Simple Network Management Protocol (SNMP) scanning activity of low importance, a sudden 100-fold increase in the hourly rate of such alert groups might be a symptom of a wider SNMP-based DDoS attack. In this regard, the framework proposed in this paper does not track such unexpected fluctuations in the number of alert groups. To address this problem, the framework needs to be augmented with other methods like time series analysis (e.g., Viinikka et al. [20] used time series modeling for this purpose).

As discussed in Section IV-C, the detection of critical alert groups that represent successful intrusions is essential in SOC environments. Although the framework achieves a high F1-score on detecting such intrusions against honeypots (see section IV-C), verifying that the attack reported by the NIDS was successful requires a thorough inspection of the victim host (e.g., inspection of the system log files, local file system and process table, etc.). As per current framework design, the data points do not include any features that are derived from the system data of the victim host, which can be regarded as another limitation of the framework. However, including such features is not an easy task since it requires not only creating a system data collection scheme for all hosts in the organizational network, but also non-trivial feature engineering to convert the collected data into meaningful metrics.

This task is further complicated by the fact that different types of victim nodes might need particular features and classifier types. For example, Alzubi et al. [30] tackled the problem of detecting infected mobile devices, employing an optimized SVM classifier.

[3]The term *DDoS (Distributed Denial of Service) attack* refers to an attempt to overwhelm the victim computer or network with a flood of network traffic originating from a large number of hosts controlled by attackers.

Finally, the current version of the framework implements binary classification. However, the usage of an additional class/category (e.g., *critical*) for alert groups that represent successful intrusions would be valuable in SOC environments. Despite that, addressing this limitation would involve the same issues discussed in the previous paragraph, and we have thus left this research question for future work.

VI. Conclusion and Future Work

The experimental results described in this research indicate that using the proposed framework for NIDS alert classification decreases significantly the number of false positives and thus reduces the workload of human analysts. Furthermore, employing the information provided by CSCAS allows the creation of training data sets for supervised classifiers with much less labeling effort from human experts than required by traditional methods. Finally, the implementation of the proposed framework requires a small amount of computational resources, which greatly eases its deployment, making it suitable for resource-constrained environments (i.e., there is no need for a dedicated computing platform).

In future work, we plan on augmenting the alert model with additional features that capture the state of victim hosts. This might allow for better detection of alert groups that represent successful intrusions, using a separate class for such alert groups. Besides, the extension of the framework with time series analysis capabilities to detect the unexpected increase in the arrival rate of alert groups from specific signatures is another relevant future enhancement. Finally, we also plan to study the concept drift phenomenon for NIDS alert data and investigate measures to address it.

Acknowledgment

This research was funded by the EU Horizon2020 project *MariCybERA* (agreement No 952360). The authors thank Dr. Adrian Venables for comments and suggestions that helped to improve the quality of this paper.

References

[1] The Open Information Security Foundation, Suricata, https://suricata.io, 2023 (accessed 15 August 2023)

[2] Cisco, Snort, https://www.snort.org, 2023 (accessed 15 August 2023)

[3] Cisco, Secure IPS (NGIPS), https://www.cisco.com/c/en/us/products/security/ngips/index.html, 2023 (accessed 15 August 2023)

[4] R. Vaarandi, A Stream Clustering Algorithm for Classifying Network IDS Alerts, in: IEEE International Conference on Cyber Security and Resilience, IEEE, 2021, pp. 14–19, https://doi.org/10.1109/CSR51186.2021.9527926.

[5] E. Kidmose, M. Stevanovic, S. Brandbyge, J. M. Pedersen, Featureless Discovery of Correlated and False Intrusion Alerts, IEEE Access (8) (2020) 108748–108765, https://doi.org/10.1109/ACCESS.2020.3001374.

[6] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, R. Muttukrishnan, OutMet: A New Metric for Prioritising Intrusion Alerts using Correlation and Outlier Analysis, in: IEEE Conference on Local Computer Networks, IEEE, 2014, pp. 322–330, https://doi.org/10.1109/LCN.2014.6925787.

[7] R. Vaarandi, K. Podiņš, Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering, in: IEEE International Conference on Network and Service Management, IEEE, 2010, pp. 451–456, https://doi.org/10.1109/CNSM.2010.5691262.

[8] G. P. Spathoulas, S. K. Katsikas, Enhancing IDS performance through comprehensive alert post-processing, Computers and Security (37) (2013) 176–196, https://doi.org/10.1016/j.cose.2013.03.005.

[9] T. Wang, C. Zhang, Z. Lu, D. Du, Y. Han, Identifying Truly Suspicious Events and False Alarms Based on Alert Graph, in: IEEE International Conference on Big Data, IEEE, 2019, pp. 5929–5936, https://doi.org/10.1109/BigData47090.2019.9006555.

[10] T. Ban, S. Ndichu, T. Takahashi, D. Inoue, Combat Security Alert Fatigue with AI-Assisted Techniques, in: Cyber Security Experimentation and Test Workshop, ACM, 2021, pp. 9–16, https://doi.org/10.1145/3474718.3474723.

[11] I. Shin, Y. Choi, T. Kwon, H. Lee, J. Song, Platform Design and Implementation for Flexible Data Processing and Building ML models of IDS Alerts, in: Asia Joint Conference on Information Security, IEEE, 2019, pp. 64–71, https://doi.org/10.1109/AsiaJCIS.2019.000-4.

[12] C. Feng, S. Wu, N. Liu, A User-Centric Machine Learning Framework for Cyber Security Operations Center, in: IEEE International Conference on Intelligence and Security Informatics, IEEE, 2017, pp. 173–175, https://doi.org/10.1109/ISI.2017.8004902.

[13] G. C. Tjhai, S. M. Furnell, M. Papadaki, N. L. Clarke, A preliminary two-stage alarm correlation and filtering system using SOM neural network and K-means algorithm, Computers and Security (29) (2010) 712–723, https://doi.org/10.1016/j.cose.2010.02.001.

[14] S. Al-Mamory, H. Zhang, Intrusion detection alarms reduction using root cause analysis and clustering, Computer Communications (32) (2009) 419–430, https://doi.org/10.1016/j.comcom.2008.11.012.

[15] J. Ma, Z-t. Li, W-m. Li, Real-Time Alert Stream Clustering and Correlation for Discovering Attack Strategies, in: International Conference on Fuzzy Systems and Knowledge Discovery, IEEE, 2008, pp. 379–384, https://doi.org/10.1109/FSKD.2008.522.

[16] T. van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, G. Vigna, DEEPCASE: Semi-Supervised Contextual Analysis of Security Events, in: IEEE Symposium on Security and Privacy, IEEE, 2022, pp. 522–539, https://doi.org/10.1109/SP46214.2022.9833671.

[17] T. Ban, T. Takahashi, S. Ndichu, D. Inoue, Breaking Alert Fatigue: AI-Assisted SIEM Framework for Effective Incident Response, Applied Sciences, (13-11) (2023), 6610, https://doi.org/10.3390/app13116610.

[18] G. Apruzzese, P. Laskov, E. Montes de Oca, W. Mallouli, L. B. Rapa, A. V. Grammatopoulos, F. Di Franco, The Role of Machine Learning in Cybersecurity, Digital Threats: Research and Practice, (4-1) (2023) 1–38, https://doi.org/10.1145/3545574.

[19] J. L. Guerra, C. Catania, E. Veas, Datasets are not enough: Challenges in labeling network traffic, Computers and Security (120) (2022) 1–17, https://doi.org/10.1016/j.cose.2022.102810.

[20] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, M. Tarvainen, Processing intrusion detection alert aggregates with time series modeling, Information Fusion Journal (10-4) (2009) 312–324, https://doi.org/10.1016/j.inffus.2009.01.003.

[21] B. Settles, Active Learning Literature Survey, Computer Sciences Technical Report 1648, University of Wisconsin-Madison, 2009, http://digital.library.wisc.edu/1793/60660.

[22] A. Beaugnon, P. Chifflier, F. Bach, ILAB: An Interactive Labelling Strategy for Intrusion Detection, in: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer, Cham, 2017, pp. 120-140, https://doi.org/10.1007/978-3-319-66332-6_6.

[23] J. L. Guerra Torres, C. A. Catania, E. Veas, Active learning approach to label network traffic datasets, Journal of Information Security and Applications (49) (2019) 1–13, https://doi.org/10.1016/j.jisa.2019.102388.

[24] R. Vaarandi, S. Mäses, How to Build a SOC on a Budget, in: IEEE International Conference on Cyber Security and Resilience, IEEE, 2022, pp. 171–177, https://doi.org/10.1109/CSR54599.2022.9850281.

[25] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, L. Cavallaro, TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time, in: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, 2019, pp. 729–746, https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury.

[26] M. Vielberth, F. Böhm, I. Fichtinger, G. Pernul, Security Operations Center: A Systematic Study and Open Challenges, IEEE Access (8) (2020) 227756-–227779, https://doi.org/10.1109/ACCESS.2020.3045514.

[27] Deutsche Telekom Security, T-Pot - The All In One Multi Honeypot Platform, https://github.com/telekom-security/tpotce, 2022 (accessed 7 July 2023)
[28] L. Breiman, Random Forests, Machine Learning 45 (2001) 5–32, https://doi.org/10.1023/A:1010933404324.
[29] A. Guerra-Manzanares, H. Bahsi, On the application of active learning for efficient and effective IoT botnet detection, Future Generation Computer Systems (141) (2023), https://doi.org/10.1016/j.future.2022.10.024.
[30] O. A. Alzubi, J. A. Alzubi, A. M. Al-Zoubi, M. A. Hassonah, U. Kose, An efficient malware detection approach with feature weighting based on Harris Hawks optimization, Cluster Computing Journal (25) (2022) 2369–2387, https://doi.org/10.1007/s10586-021-03459-1.