Security Analytics with Mixed Event Sources and Ensembles

Simão Pedro Avelino Valente, 78089, MEEC
simaovalente13@gmail.com / valente.spa@mail.exercito.pt

***Abstract*—The present work shows the complexity associated with the creation of an Intrusion Detection System (IDS) considering host and netflow data, as well as the limitations of current detection systems, either host-based or network-based. It presents a new approach based on Windows OS logs and network flows. The main goal is to provide and test a process capable of detecting intrusions, the use of a high number of features and the use of different data sources by themselves and combined for intrusion detection. The process uses multiple unsupervised machine learning algorithms ensembled to detect outliers. To achieve the goal, several unsupervised machine learning algorithms were tested in performance either in time or outliers detection capability based on our classification criteria. The three algorithms that achieved better performance were considered. The process is composed of feature extraction, followed by normalization, clustering and clustering ensemble. After the clustering ensemble, the detected outliers are presented to an analyst, to mitigate them. Five different approaches were used for feature selection, three of them using only netflow data and one using Windows events. The last approach combined the best approach used on netflow with Windows events approach. To evaluate the developed system, a public artificial dataset was initially used. This dataset contains logs from machines with Windows OS and netflow information. Adittionally, a dataset with real logs and flows from an organization was also obtained and used for the same purpose. The present work was developed with the collaboration of Portuguese Army.**

Key-words: Cybersecurity, Machine Learning, Intrusion Detection System, Security Analytics, Logs, Netflow.

I. Introduction

Nowadays we observe an exponential growth in the use of technology and smart devices to connect people among themselves and the world. This makes daily tasks easier but also turns people more vulnerable to computer attacks. The number of attacks is increasing as well as attackers insight, leading to complex attack techniques that are more difficult to detect and, therefore, to mitigate or respond to. Since attacks are rapidly evolving, it is important to detect them and adapt to novel intrusion techniques. In fact, according to [14], in 2018 the mean time to detect an external attack was 184 days and an internal was 50.5 days. These numbers are high because not only attacks are difficult to detect but also many current detection techniques are still based on signatures. Signatures represent well known malicious code used to perform attacks. Basically, most of the IDS analyze code to detect these codes. Intrusion Detection Systems emerged from the need for reducing the number of attacks and their detection time. These IDSs can be classified in two different ways: according to the information given to the system, and based on the input data source. The first classification, according to the information is given to the system can be divided in anomaly and misuse detection. Anomaly detection focus on finding patterns that deviate from the normal behavior of an entity. Misuse detection tries to find a well known malicious behavior previously seen. Depending on the IDS input data source, they can be classified as: Host-based Intrusion Detection System (HIDS) or Network Intrusion Detection System (NIDS). According to [3, 6], HIDSs cover the events produced in the monitored host, aim for computer protection and prevent malicious code execution in the system. On the other side, NIDSs capture network traffic, covering all the machines/computers within the network. Currently, NIDSs are more used than HIDSs because they can analyze simultaneously multiple users and require less maintenance. A new approach in IDSs is to combine both HIDS and NIDS. It joins precious information from the host with relevant network traffic. It enables the detection of offline attacks and online attacks. Nevertheless, this new approach increases the amount of data provided to systems. This fact leads to the need for selecting the most relevant features for security analytics. Security enterprises have started to

use Machine Learning (ML) to improve their IDS. ML came not only to deal with big data but also because it is a powerful tool to analyze and interpret patterns and structures in data. Since ML algorithms can learn from data, it is very useful to security analytics. Nevertheless, the use of ML classifiers implies to know all attacks which represent a misuse approach with a set of well known features. This approach is vulnerable to new attacks. To use ML to detect novelty attacks a system should contain sets of dynamic features. This approach feeds the system with a set of relevant features used in a particular period.

II. Background

II-A. Machine Learning Methods

The significant increase in data production and complexity made human abilities insufficient to deal with such information. This is the main reason why Machine Learning is used. In 1959, Arthur Samuel, defined Machine Learning as *"the field of study that gives computers the ability to learn without being explicitly programmed"* [13]. Since then Machine Learning has been used in many applications such as prediction, medical image analysis, human activity recognition, machine translation, self-driving cars, security analytics and others. Nowadays we can find usage of Machine Learning almost everywhere. Machine learning has two major categories: supervised and unsupervised. In the present work, we used unsupervised ML.

II-A1 Unsupervised Machine Learning

Unsupervised ML goal is to find fundamental features on unlabeled data and form clusters of input patterns based on a specific cost function. This category of ML is very useful to detect unknown attacks because groups unlabeled data. In order to understand some definitions presented, it is important to know the clustering concept. It consists of assigning objects with similar features in groups (clusters) and their similarity is based on group proximity from other groups.

***k*-Means Clustering** [4] is the assignment of *X* data points into *k* clusters. It consists of finding the most similar centroid for each point. The data point distance to centroid indicates how similar they are. This algorithm can be performed into 4 steps: selection of initial cluster centroids (*k*); assign each data point to the cluster with the nearest centroid; recompute each cluster centroid based on the elements contained in it; repeat second and third steps until achieving convergence. To provide successful implementation of the *K*-Means method is important to carefully choose two aspects: the *k* value, which indicates the cluster number to be partitioned and the distance metric, where the most commonly used is the Euclidean distance. There are no guarantees that selected *k* is optimal, nevertheless, several approaches to evaluate the clustering performance have been done using stability, accuracy and other metrics.

Local Outlier Factor (LOF) [2] is a density-based method, which uses the nearest neighbor search. This method gives a score to each data point by determining the ratio of the average densities of neighbors points to the density of the point itself. The estimated density of a certain point *n* is the number of its neighbors divided by the sum of distances of the neighbor's points. For a better understanding consider $M(n)$ as a set of neighbors of point *n* and *k* the number of points in this set. The distance of a point *y* to point *n* is represented by $d(n, y)$and their estimated density is given by:

$$f(p) = \frac{k}{\sum_{y \in N(n)} d(n, y)} \tag{1}$$

Using previouly defined equation, its possible to define the LOF score as:

$$LOF(n) = \frac{\frac{1}{k} \sum_{y \in N(n)} f(y)}{f(n)} \tag{2}$$

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [5] is an algorithm designed to discover clusters and noise in a dataset. The ideal case is to know the appropriate parameters *Epsilon* and *MinPts* of each cluster. The *Epsilon* parameter defines the density of points needed in a point neighborhood to be considered as a member of a cluster. The *MinPts* represent the minimum number of points needed for a group to be considered a cluster. Without knowing these parameters *a priori*, this method starts with a random point *n* and determine all points density-reachable from *p* considering *Epsilon* and *MinPts*. If this point is a border point (point in the ends of a cluster) the algorithm moves to the next point in the dataset. Otherwise, if point *n* is a core point (point in the center region of a cluster), these procedures yield a cluster. This procedure is performed until all points of a dataset are covered. Once it started with parameter

global values, it may merge two clusters into one if their distance to each other are 'close'. If a point's distance to all clusters is too large, it is considered noise. Those noisy points are merged into a single cluster, normaly represented by -1.

Agglomerative [16, 18] consists of recursively merging pairs of clusters that minimally increase a given linkage distance. Similar to KMeans, Agglomerative can use a parameter *k* that defines the number of clusters to find. The linkage distance is used to determine the distance between sets of points and this algorithm main goal is to minimize this distance. This algorithm starts by considering all data points as an individual cluster. In each iteration merges similar clusters into each others. The stop criteria are when all data points are in the same cluster or when *k* clusters are formed. This algorithm can be defined in four steps:

1) Compute the proximity of individual points;
2) Consider each data point as a cluster;
3) Find the two closest clusters and merge them into a single cluster;
4) Recalculate clusters proximity;
5) Repeat steps 3-4, until achieve a single cluster or *k* clusters.

We applied these algorithms in our system to group entities based on their similarities behavior. These algorithms were implemented using Python and some of its libraries.

III. Related Work

III-A. Anomaly Detection

Anomaly detection consists of processing data and finding patterns that do not correspond to the expected behavior. This process is very useful for several cases, in this work it was useful for network monitoring and computer security. Nevertheless, there are other cases where anomaly detection could be useful, such as smart devices and IoT. The major anomaly detection algorithms need purely normal data to train the algorithm and consider any data never observed as anomalous. The outliers are defined based on measurements that normally correspond to the distance from data considered normal. In 2003, [8] made a comparative study in order to evaluate different anomaly detection methods. They used two data sets, 1998 DARPA Intrusion Detection Evaluation Data [9] and their real network data from the University of Minnesota. They studied four approaches:

- Local Outlier Factor (LOF), which consists in assigning to each data point a degree of being an outlier;
- Nearest Neighbor (NN), which is KNN with k=1;
- Mahalanobis-distance based outlier detection consists in determinate the mean μ of a particular point p and compute the distance as follows:

$$d_M = \sqrt{(p-\mu)^T \times \sum^{-1} \times (p-\mu)} \qquad (3)$$

- Unsupervised SVM, is trained with unlabeled data and consist in separate the entire set of training data from the origin. This means smaller regions with many data points will be considered a class and points in another region will be labeled as another class.

There are several anomaly detection systems already developed using both supervised and unsupervised ML, proving that this approach can achieve great results fighting against attacker's new techniques. NIDSs and HIDSs can both use the anomaly detection approach to find malicious behavior.

III-A1 Host Based Intrusion Detection

With the encrypted traffic increasing in networks, NIDS are losing strength against intrusions. Once several attacks are directed to hosts, it is useful to monitor the events produced on the local host (user's computer). This extend intrusions defense perimeter. [7] presents a system to detect host misbehavior in a large number of logs. Their approach consist in two phases. In the first phase, they define and configure the detection mechanisms by performing 6 steps: defining how logs data will be normalized; selecting features and defining the period for feature extraction, defining how features are extracted, extracting features and normalizing logs, performing clustering in group entities, defining classification. The output of this first phase was used in the next for classification. The second phase was the execution of the system at runtime. It was divided into 4 steps: extraction of features, clustering, clusters classification and if the classification were successful the system was updated, otherwise it has to be analyzed manually and updated the system. Their system had to deal with Big Data and for that MapReduce was used. To perform clustering they used the EM algorithm because it does not need prior knowledge of the

distribution of each feature. The number of clusters is given as input to the EM algorithm. In cluster analysis, they labeled cluster features as: primary, secondary, non-relevant. The primary features represent the best characterization of the cluster, based on their deviation from the cluster. The secondary features are similar to primary ones but for a different range. The non-relevant features are the ones that do not characterize the cluster. The author chose the Naive Bayes algorithm for classification since it converges faster than other models and needs a smaller dataset to obtain accurate results. Their system was not accurate enough to deal with real-time detection but provided analysts with precious security information from logs.

III-A2 Network Intrusion Detection

Nowadays the concept of network is used all over the world and the evolution of technology allowed people to use their phones and computers to access sensitive information from anywhere since an internet connection could be found. This gives people a great freedom and saves time, but also increases significantly the risk of having intercepted or compromised data. [12] presented a system combining supervised and unsupervised machine learning to detect network anomalies automatically. Their system was composed of five phases: features extraction, clustering to aggregate entities, genetic algorithm to select the most relevant features from clusters, classification, and output the misbehaving groups along with the relevant features. This system revealed to be significant in reducing the amount of entities to analyze, nevertheless, they had a lack of contextual information about the networks and it could be a great improvement in mitigating the abnormal activity.

III-A3 Hybrid Intrusion Detection

This chapter presents some literature about IDSs that combine both network and host-based intrusion detection. This combination seems to be recent, as few articles have been published. The most recent hybrid IDS is presented in [11]. Authors proposed a system entitled HOLMES with a main goal of detecting Advanced Persistent Threats. Their system starts by collecting audit data from the host to produce a detection signal. This signal will indicate the stages of the current APT campaign. At high level HOLMES techniques provide correlation between suspicious information flows. Those flows derived from events produced during the attacker campaign. HOLMES evaluation was performed on 9 real-life APT attack scenarios. It was running as real-time IDS and has a duration of two weeks of live experiment. The results show high precision and recall discerning attacks from benign scenarios.

IV. Datasets Description

The present work was developed using two datasets, an artificial dataset from CSE-CIC-IDS 2018 [15] and a real one from the Portuguese Army.

IV-A. CSE-CIC-IDS 2018

This dataset was developed towards network-based anomaly detector, with the goal of providing data to analyze, test and evaluate IDS. To generate such a dataset, its authors developed a systematic approach to produce a diverse and comprehensive benchmark dataset. In their approach, they created user profiles with abstract representations of activity seen on the network. This dataset contains information about ten days of normal activity and attacks performed. This victim's infrastructure is composed of 420 machines and 30 servers divided into 5 departments and an attacker infrastructure with 50 machines. It includes system logs and network traffic of each machine. The authors also provide a list of features, based on network traffic, that was used to perform clustering.

This dataset provides logs from machines (Windows XML Event Log (EVTX) files from Windows and text files from Linux). Our work were focus on Windows OS, so we only analysed those. Authors left default configurations on Windows machines, which means only system logs were saved and extracted, missing information about security and applications running on those machines. As this information comes in EVTX, readable in Windows Event Viewer, we needed to extract the information to a python readable file such as Comma Separated Values (CSV).

The first approach was to use a python script to read from EVTX[1] to JSON and then convert JSON to CSV, this was a slow process and missed some fields such as the time creation of an event. Once we needed that information from every event we had to change our approach. This second approach used a Powershell script to extract all information contained on EVTX files to

[1]Windows Event XML Log

a CSV. On a single step, we collect all the information needed, nevertheless, this was also a time-consuming process. Once we had our log information on a python readable file, we cloud extract our host-based features. The information from network traffic was provided as Package Capture (PCAP) files and also as CSV (processed with the 83 features proposed by authors). Our approach used IP addresses to identify each entity in a network and this information was not provided in processed files. This lead us to use their PCAP files and CICFlowMeter to extract the information we need. We used last release of CICFlowMeter (V4.0) which lead to a mistake in the number of packets sent and received in a biflow. This error was detected later with confirmation of our features and corrected by adding one packet to forward ones and subtracting one packet from backward ones.

IV-B. Portuguese Army Dataset

This dataset was collected from workstations in an administrative network medium-sized military infrastructure. It contains real logs from each workstation, as well as netflow information of five days. By default, the Windows OS does not enable the most interesting logs for security. In order to collect logs at security and applications level, it is necessary to enable/change (to success and failure state) some configurations, such as: force audit policy subcategory settings, logon, network policy server, other logon/logoff events, special logon, process creation, Remote Procedure Call (RPC) Events, security state change, security system extension, system integrity, file share. The amount of space used to save logs also need to be increased to prevent losing some of them. As performed in the previous dataset, we had to process EVTX files in the exact same way. Dataset flows were collected from a Netflow tool, which provides less information than CICFlowMeter. Nevertheless, the information provided was enough to apply our system for anomaly detection.

The produced dataset is more complete in terms of Windows logs, once it contains logs from Application, Security and System Levels, while the CSE-CIC 2018 dataset only contains logs from System level.

V. Clustering Algorithm Selection

This chapter describes tests made to choose the most appropriate clustering algorithms for anomaly detection. To evaluate different clustering approaches, we tested the most commonly used algorithms in each category (partition, hierarchical, neural models and density-based). The criteria used to choose the most appropriate clustering algorithms were: performance when applied to data with a big number of features (i.e., high dimensional data); efficiency to identify outliers (from a well-known dataset); time needed to perform clustering must be lower than the smallest window of time (10 minutes), in order to get results in real-time;

To apply these criteria we used a labeled dataset known to produce good results with KMeans [10] in a previous work [12]. The labeled dataset contains flow traffic from one day of the Portuguese Army dataset used by [12]. This day contains flows from 4616 entities, one of which is an attacker that performed a PortScan and a Dictionary attack. The algorithms tested were Kmeans, DBSCAN, Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN), AgglomerativeClustering, OPTICS, SimpSOM, SOMPY, and LOF. A benchmark from [1] was adapted to test our algorithms in terms of performance against large datasets. The data for tests was generated locally and data was increased until reach 30000 observations. Figure 1, shows the performance of each algorithm with the increase of data. Observing the results it is clear that SOM does not fulfill performance criteria for large datasets. The remaining algorithms fit the criteria once they could perform cluster to 20000 observations in less than 60 seconds.

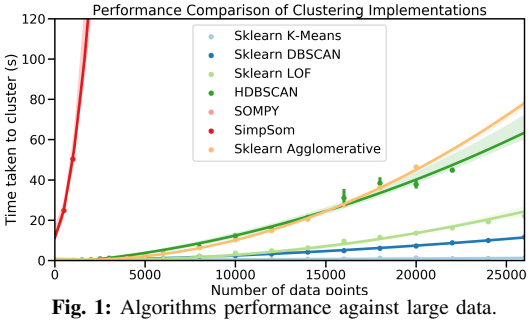

Fig. 1: Algorithms performance against large data.

We applied our algorithms using their features in order to understand algorithms ability to detect outliers. Based on experiments we decided to use LOF on clusters with suspicious entities. These entities were known *a priori*. Analysing Table I it is possible to determine which cluster performed better detecting the

TABLE I: Algorithms capacity to detect outliers.

Algorithm	Attacker Cluster	Cluster Size	Attacker LOF Score (attacker_score/higest_score)
Agglomerative	0	4	Not applicable
OPTICS	-1	2491	4/601
KMeans	13	1	Not applicable
HDBSCAN	-1	2245	4/436
DBSCAN	-1	29	2/2

attacker. The first column identifies the algorithm and the second one identify the cluster where the attacker was grouped. The third column specifies attacker cluster size and last column represent the relation between the attacker's LOF score and the highest score of his cluster. In the cases where the attacker cluster size is smaller than the number of neighbors, LOF cannot be applied. From better to worst clustering algorithm we have KMeans, which isolated attacker, Agglomerative, grouped attacker with other 3 machines, DBSCAN, found 29 outliers nevertheless attacker LOF score was the higher in outliers cluster, HDBSCAN, found 2244 outliers and attacker LOF score was far from good, OPTICS detected 2491 outliers and the scores was also far from good.

According to results obtained, we performed clustering using KMeans, Agglomerative and DBSCAN, applied LOF to outlier cluster from last one.

V-A. Selecting Algorithm Parameters

An important decision when using clustering algorithms is to correctly select the parameters. In our case, we had to choose 5 parameters: *n_clusters* for Agglomerative, *k* for KMeans, *eps* for DBSCAN and *n_neighbors* and *contamination* LOF. To chose parameters for KMeans, Agglomerative and DBSCAN we decided to use *Elbow Method*. The idea of this method is to test various numbers of clusters in order to achieve the optimal number of clusters. Increasing the number of clusters in KMeans reduce the distance between cluster, nevertheless, the distance between observations intra-clusters will increase. Since each time-window of data processed can have different entities and features, our data could vary significantly. This means that fixing *k* and *epsilon* would not be a good choice since it could be unfitted to that specific data. To reduce this problem, we applied elbow method to each time-window, for example, to 5 time-windows of 10min we could have 5 different values of parameters.

VI. Proposed Solution

VI-A. Processing Data

The present work combines network traffic and logs from hosts in the network. The use of both implies being able to manage data from at least two different sources. To our process, we handled data separately and merged the network and host data after feature extraction. To perform features extraction and based on results achieved by [12] we decided to use different time-windows. The time-windows used were 10 min, 30 min, 1 hour, 2 hours and 4 hours. With these time-windows, we avoid the attenuation of attacks behavior.

This process was performed in three phases. The first one consisted in processing only events from Windows OS, the next one only flow data and finally the combination both (events and flows).

VI-A1 Feature Engineering

To choose features we analyzed each source (host and network) independently based on the information provided by the CSE-CIC-IDS 2018. The approach on host features was to count all Windows OS events individually and their level (Critical, Error, Warning, Informational, Verbose), as well as, the total number of events. For example a machine in time-window with only 30 events of eventID 7024, of which 10 in Warning level and 20 in Informational level, will have 4 features: eventID 7024 with value 30, Warning with value 10, Informational with value 20 and TotalEvents with value 30.

Concerning network features, we started by setting 10 fixed features, which were considered in all the next approaches. These fixed features describe the general network activity of an entity. The first approach considered the 33 ports most used in the CIC-CSE-IDS dataset and set dynamically other 77 ports. The 33 ports were selected based on the most used ports in all days of the CIC-CSE-IDS dataset. We selected 33 ports because was the number of ports (mostly well-known and registry ports) with a significant use compared to all other ports. The dynamic features emerge as an attempt to detect attacks without using

only well-known ports but using the most used ports in a processed time-window. This approach selected the 77 ports most used by all dataset entities in a processed time-window. We choose 77 ports dynamically to make up a total of 100 ports as features. We wanted to test the impact of a huge amount of features in intrusion detection. In fact, we started by not defining a limit for those dynamic ports, however, the feature extraction time was high and did not fit our criteria. The second approach was to use only the 33 fixed ports and the final approach was to use [12] port features (ports 80, 194, 25 and 22) plus the ports used by the service attacked (e.g. for FTP attack we added port 21). Each port taken in consideration produced 4 features. For example, for port 22 we had 22SrcSent, number of packets sent from port 22 in source view, 22SrcRcvd, number of packets received from port 22 in source view, 22DstSent, number of packets sent from port 22 in destination view and 22DstRcvd, number of packets received in port 22 from destination view. For each time-window, features were extracted individually to avoid overlay off fixed features.

VI-A2 Feature Extraction and Normalization

After performing the feature engineering we need to understand how each feature will be extracted. In the CIC-CSE-IDS dataset, flow features were extracted using CICFlowMeter and logs were extracted from Windows, Linux and Mac OSs. In the Portuguese Army dataset, we extracted host features using a script to collect the events from all machines in a specific subnetwork. The flow features were extracted from a netflow tool used by the Portuguese Army to control and analyze the network traffic. Normalization is the process of finding a scale to fit all data without loosing critical information. To perform normalization we used a library from sklearn.preprocessing [17] named *MinMaxScaler*. This library performs normalization over each feature in a range. We defined our range between [0,1]. Normalization is performed by the *fit()* method.

VI-B. Clustering

One of the main goals of this work was to combine the results of clustering algorithms in order to improve accurarcy and precision.

VI-B1 Applying Clustering Algorithms

As mentioned before, we intended to use three algorithms to perform clustering, KMeans, Agglomerative, DBSCAN. To apply these algorithms we divided each dataset into two subsets, external entities, and internal entities. This was performed as an attempt to reduce the differences between entities. Internal entities will have a similar behaviour with themselves than with external entities. By performing this division we joined internal entities resulting in a more uniform subset. This will reduce the probability of having entities misclassified.

VI-B2 Defining Classification

The entities' classification is the process of labeling an entity as benign or malicious. This is a crucial and maybe the hardest phase of intrusion detection. This phase consist in defining the criteria of classifying an entity as benign or malicious. For example, if we define the criteria of being benign as the clusters with more than 5 entities, every cluster which does not fulfill the criteria is considered malicious. Considering this example, if we had only one victim[2] our accuracy might be lower than expected, nevertheless if we had more than five victims the probability of missing some of them is higher. In the process of classification, we defined inliers and outliers. An inlier is an entity considered with 'normal' behavior and outlier an entity with 'abnormal' behaviour. The behaviour of each entity is determined by the values of its features. Our criteria to identify an outlier was to consider all entities isolated in a cluster (cluster with a single entity) by KMeans and Agglomerative and the entities detected as outliers by DBSCAN (assigned to cluster -1).

VI-B3 Ensemble Method

Our version consists of intersecting small clusters from algorithms not based in density with entities in outliers clusters from density based algorithms. The method could be defined in four steps: select all KMeans clusters with only one entity, select all Agglomerative clusters with only one entity, select DBSCAN outlier cluster (-1), intersect all clusters and drop the entities outside the intersection and the entities remaining correspond to outliers. This ensemble method was chosen based on the good results obtained from our classification criteria using Agglomerative and KMeans. Nevertheless, our DBSCAN results were not so good and we wanted to improve them. After analyzing the entities in the DBSCAN outlier cluster and the one entity clusters in Agglomerative and KMeans, we found that all entities in small clusters were con-

[2]Represent an entity attacked by other machine(s).

sidered outliers in DBSCAN. By intersecting those clusters we were able to reduce the number of entities in DBSCAN outlier cluster. This ensemble method is also able to improve the accuracy of Agglomerative or KMeans. Considering an example where we have one single victim and KMeans found two clusters with a single entity. However Agglomerative found one single cluster. Applying this ensemble method will filter KMeans clusters and select the one containing the entity in common.

VI-C. Evaluation Metrics

To evaluate our system we defined: true positives (TP) as entities correctly classified as outliers, false positives (FP) as entities wrongly classified as outliers, true negatives (TN) as entities correctly classified as inliers, false negatives (FN) as entities wrongly classified as inliers, *accuracy* as the degree to which the result of clustering conforms to the correct value of an entity, *precision* the fraction of relevant results, *recall* the fraction of total relevant results correctly classified by the algorithm and *F1* the relation between *precision* and *recall*. Each of these metrics was applied individually to each algorithm in the first phase of our work.

VII. Experimental Evaluation

This chapter presents the results obtained with the artificial dataset and with the combination of clustering algorithms. It also exhibitions the analysis of network and host features individually, as well as when combined. Our experiments were divided into two phases, the first one we analysed algorithm results individually and the second one we used a clustering ensemble method in order to combine output from the different algorithms.

VII-A. Combining Algorithms in Netflow Sources

We found 17 attacks simulated in an artificial dataset, described in Chapter 4. From those approaches, we conclude that using less features improves the results since the third approach was the one which detected more attacks. The 1^{st} approach did not detect any of the attacks and the second approach detected 11 attacks. From all the 17 attacks only the LOIC UDP from day 20/02/18 was not detected. The second approach achieved a mean accuracy of 0.97, a mean precision of 0.64, a mean recall of 1 and a mean F1 of 0.73. The third approach achieved a mean accuracy of 0.77, mean precision of 0.44, mean recall of 1 and mean F1 of 0.52. These metrics proved that the second approach performed better in detecting victims/attackers, however, this approach missed five attacks. The best time-window in the second and third approaches was the 10 minutes window.

VII-B. Combining Algorithms in Windows Event Sources

From the three days of attacks to Windows machines, we were able to detect the attacks performed in two of them. On the last day, the Bot attack was executed to more than one machine. It means with our classification criteria the probability of detecting all the victims was low, either because the victims have the same behavior and were grouped in a single cluster or they all had different behaviors and were isolated in 10 different clusters. The results in this source of data achieved a mean accuracy of 0.96, a mean precision of 0.27, a mean recall of 1 and a mean F1 of 0.42.

VII-C. Merging Events and Flows

In the days which was possible to combine both events and flows, we were unable to detect a single anomaly. These results show that event logs from the system and flows approaches are incompatible, either because a big amount of features or attack footprints in event logs are unrelated to flows. Nevertheless, with a restricted choice of features, it should be possible to combine both.

VIII. Conclusions

These work contributions are: increasing the number of features will reduce the capability of detecting outliers; LOF performs well in scoring outliers; a new ensemble method which improves results; considering entity's behavior in both points of view (source and destination) allows the detection of more attacks and a new dataset with real logs and flows from a medium-size organization.

References

[1] *Benchmarking Performance and Scaling of Python Clustering Algorithms*. URL: https://hdbscan.readthedocs.io/en/latest/performance_and_scalability.html. accessed: 05.06.2019.

[2] Markus M Breunig et al. “LOF: identifying density-based local outliers”. In: *ACM sigmod record*. Vol. 29. 2. ACM. 2000, pp. 93–104.

[3] G. Creech and J. Hu. “A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns”. In: *IEEE Transactions on Computers* 63.4 (Apr. 2014), p. 808.

[4] S. Dua and X. Du. *Data Mining and Machine Learning in Cybersecurity. Data Mining, Inference and Prediction*. Auerbach Publications, 2011.

[5] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: *KDD*. Vol. 96. 34. 1996, pp. 226–231.

[6] G.Vigna and C.Kruegel. “Host-Based Intrusion Detection”. In: (June 2005), p. 1.

[7] D. Gonçalves, J. Bota, and M. Correia. “Big Data Analytics for Detecting Host Misbehavior in Large Logs”. In: *Proceedings of IEEE Trustcom*. 2015.

[8] Aleksandar Lazarevic et al. “A comparative study of anomaly detection schemes in network intrusion detection”. In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM. 2003, pp. 25–36.

[9] R. P. Lippmann et al. “Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation”. In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00*. Vol. 2. Jan. 2000, 12–26 vol.2. DOI: 10.1109/DISCEX.2000.821506.

[10] James MacQueen and others. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[11] S. Momeni Milajerdi et al. “HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. Vol. 00, pp. 430–445. DOI: 10.1109/SP.2019.00026.

[12] H. Reia. “Online Security Analytics”. MA thesis. Instituto Superior Técnico, Oct. 2018.

[13] A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM Journal of Research and Development* 3.3 (June 1959), pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[14] Fireeye Mandiant Services. *M-TRENDS 2019*. 2019.

[15] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”. In: (Jan. 2018), pp. 108–116. DOI: 10.5220/0006639801080116.

[16] *sklearn.cluster.AgglomerativeClustering*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html. accessed: 10.09.2019.

[17] *sklearn.preprocessing.MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler. accessed: 05.09.2019.

[18] *Understanding the concept of Hierarchical clustering Technique*. URL: https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec. accessed: 10.09.2019.