Cluster Change-Based Intrusion Detection

Tiago Fernandes, 81299, MEEC
Instituto Superiro Técnico
Academia Militar
fernandes.tfc@exercito.pt

***Abstract*—The paper presents a network intrusion detection approach that flags malicious activity without previous knowledge about attacks or training data. The *Cluster Change-Based Intrusion Detection* approach (C2BID) detects intrusions by monitoring host behavior changes. For that purpose, C2BID defines and extracts features from network data, aggregates hosts with similar behavior using clustering, then analyses how hosts move between clusters along a period of time. This contrasts with previous work in the area that stops at the clustering step. We evaluated C2BID experimentally with an evaluation dataset and a real-world dataset, obtaining better F-Score than previous solutions.**

***Index Terms*—network intrusion detection, clustering, behavior change, security analytics**

I. Introduction

Since the last century, telecommunications, computing hardware, and software have been the cornerstone of our society, translating into a massive dependence on the Internet and the need to protect it. *Intrusion detection* has become an important research topic due to advances in networking technologies and the increasing number of network attacks [1]. Today, Intrusion Detection Systems (IDS) are widely deployed to identify threats and possible incidents [2]. Although these and other security measures allow detecting and blocking some attacks in real-time, other attacks are more elusive and may cause more serious damage. Specifically, Interestingly companies take many days to detect some attacks, e.g., roughly 58 days [3]. Therefore, *traditional security mechanisms do not provide enough protection and organizations should dig into traffic and logs to search for anomalous patterns in larger windows of time*.

There are two classical intrusion detection approaches: signature-based detection and anomaly-based detection. Signature-based (or misuse) detection identifies known attacks by matching patterns of attacks with observed behavior. It allows detecting known attacks without generating an overwhelming number of false alarms. Networks protected by misused detection systems may suffer from *long periods of vulnerability* between the appearance of a new vulnerability and the deployment of a signature to detect attacks that exploit it. Misuse detection is the most used approach [4]. Anomaly detection attempts to find patterns that do not conform to expected normal behavior [1]. This approach is appealing because of its ability to detect new attacks, but often leads to *high false alarm rates* because previously unseen (yet legitimate) behaviors may be flagged as anomalies [5]. *Both approaches require prior knowledge, respectively of signatures and of normal behavior, something that is inconvenient and that we circumvent in this work.*

Machine learning techniques are classically divided in two categories: supervised and unsupervised [5]. In *supervised* methods, there is a training dataset with labelled data, with the normal and anomalous (or malicious) classes in the case of IDSs. The typical approach in such cases is to build a predictive model. In the end, any unseen data is compared against the model to determinate which class it belongs to [1]. *Creating an IDS based on this approach is challenging* for two reasons: in training data, samples of the anomalous class are rarer than normal samples; obtaining accurate and representative labels, especially for the anomalous class, tends to be difficult [1].

Unsupervised learning – the set of techniques we use – is interesting for intrusion detection because by definition it does not require prior knowledge, unlike classical signature-based and anomaly-based detectors. In fact, unsupervised learning is concerned with finding patterns, structures, or knowledge in unlabelled data [5]. Unsupervised learning and specifically clustering have been receiving some attention in the context of intrusion detection. A clustering algorithm is applied to feature vectors, each vector representing an entity (e.g., a machine or a user), to group entities (machines, users) with similar behavior, i.e., with similar feature values. The resulting groups or clusters can be analysed and flagged as malicious or not using manual analysis, outlier detection, thresholds, or some heuristic like considering small clusters suspicious. This approach has been investigated by several authors [1], [5]–[14].

These works have several limitations. Many assume that only a small part of the traffic is malicious, e.g., [10], [11], [14]. This leads to attacks with several machines, e.g., a botnet, not being easily detected. Also, attackers can often circumvent machine learning-based attack detection by executing attacks at low pace or in multiple time windows, e.g., by doing a slow port scan [11]. Some models [12], [13] rely on clustering and threshold definition, which depends on a training period and can be avoided by attackers. Others are dependent on manual classification, at least in an initial phase [10], [15].

We propose a novel approach for network intrusion detection based on unsupervised learning: *Cluster Change-Based Intrusion Detection* (C2BID). Our approach still uses clustering, like the works above, but clustering is just a first step towards understanding if a host is malicious or not. The main idea of the C2BID approach is to detect intrusions by monitoring *host behavior changes*. For that purpose, C2BID defines and extracts features from network data – specifically network flow data [16]–[18] –, aggregates hosts with similar behavior using clustering, then analyses how hosts move between clusters along a period of time and detects outliers. It can detect attacks after a given time period, e.g., one day. This contrasts with previous work in the area that essentially stops at the clustering step.

C2BID analyses host movement between clusters, including the appearance of new clusters, through sequential series clustering in sequences of time windows. By studying the temporal behavior of clusters it is possible to identify anomalous behaviors and suspicious cluster formation, doing outlier detection that is another form of unsupervised learning. This results in higher precision than marking only one host cluster and faster analysis than using manual analysis. C2BID uses the idea of dynamic features – features defined in runtime based on observed TCP/UDP port activity – inspired in DynIDS [14]. This allows analysing the traffic in many ports while avoiding the curse of dimensionality [19], i.e., loosing the ability of detecting attacks due to an excessive number of features (e.g., more than 1000 features, when there are 2×2^{16} ports). C2BID improves previous works by correlating multiple time windows to detect attacks at different rates and dealing with fixed window limitations.

We implemented and compared C2BID with three very recent intrusion detection approaches based on clustering: FlowHacker [10], OutGene [11], and DynIDS [14]. We tested C2BID with an artificial dataset [20] and a real-world dataset from a military administrative network. Our evaluation shows that C2BID was able to detect not only the labeled attacks but also found unlabeled (unreported) attacks in both datasets, highlighting the advantages of its unsupervised approach. Moreover, C2BID obtained higher values for F-score and reduced the false positive rate.

The results of this work were partially published as: Tiago Fernandes, Luis Dias and Miguel Correia C2BID: Cluster Change-Based Intrusion Detection 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Guangzhou, China, December 29, 2020 - January 1, 2021 (Core A).

II. Background

This section provides some background on the use of clustering for intrusion detection and explains how we have chosen the outlier detection algorithms to apply.

A. Clustering

Clustering [5] is a set of techniques for finding patterns in high-dimensional unlabeled data. The general idea is that entities from the same cluster are more similar to each other than entities from different clusters [7]. In many works, the key idea is that big clusters represent normal behavior and the outliers (e.g., small clusters of entities or noise) can correspond to anomalous behavior. This is useful for creating a system to detect unknown attacks or anomalous behavior. However, different clustering algorithms have different forms of initialisation and produce different data partitions [21] according to the shape and structure of the data, so they may lead to different results. One option to overcome the limitations of using a single clustering technique is to use an ensemble of several algorithms [14], [22], [23].

K-Means [24] is a clustering algorithm that produces groups of data points where each point is more similar to its cluster centroid than to the other clusters' centroids. The algorithm works iteratively by assigning data points to clusters until convergence is achieved. It is known to produce good results in the context of intrusion detection [6], [10], [11], [14], as in many other areas [21], [25], and this is also our experience in previous works, so we use it for clustering. The algorithm used was provided by the Scikit-learn Python library [26]. K-Means supports different distance metrics but we use the classical Euclidean distance.

B. Outlier detection

Outliers are points in a dataset that are unlikely to occur in given a model of the data. When data is processed by a clustering algorithm like K-Means, outliers may be consider to be those entities in clusters with a single entity [1] or those that are farther from all the other data points than a given a threshold [12], [13]. In this work we use an *outlier detection algorithm* to find outliers, not clustering.

To decide which outlier detection algorithm to use, we tested several algorithms available in the literature and libraries: DBSCAN [7], Isolation Forest [27], Support Vector Machine [28], OPTICS [29], Local Outlier Factor [30], Elliptic Envolve [26], and Robust Random Cut Forest (RRCF) [31]. The criteria used to select the most appropriate algorithms were: the ability to identify an attacker in a (labelled) dataset as an outlier; minimum number of hyper-parameters and their simplicity of configuration (as complexity leads to errors); low number of false positives. The labelled dataset contains network traffic flows of 9 days [20]. Most of the algorithms considered were available in the Scikit-learn; the exception were the RRCF implementation [32]. The setting of the hyper-parameters was made using Scikit-learn functions or empirically. Dynamic parameter setting algorithms [33] were used for DBSCAN and OPTICS.

From the algorithms evaluated, we selected RRCF as it performed much better than the rest. RRCF considers a point as an anomaly if the complexity of the model increases substantially with the inclusion of the point [31]. The core data structure used to implement RRCF is the robust random cut tree [32]. A robust random cut tree is a binary search tree that can be used to detect outliers in a point set. Points located nearer to the root of the tree are more likely to be outliers. Given a point set, a robust random cut tree is constructed by recursively partitioning the points in the set until each point is isolated in its own bounding box. For each iteration of the tree construction routine, a random dimension is selected, with the probability of selecting a dimension being proportional to the difference between its minimum and maximum values. Next, a partition is selected between the minimum and maximum value of that dimension. If the partition isolates a point from the rest of the point set, a new leaf node for it is created, and the point is removed from the point set. The algorithm is then recursively applied to each subset of remaining points on either side of the partition. Given a new point, the algorithm follows the cuts and compute the average depth of the point across a collection of trees. The point is labelled an anomaly if the score fit in a predefined decision rule, e.g., the 5 higher values. Others authors argued that RRCF is an excellent algorithm to avoid false alarms [8] and used it in multiple scenarios [34], [35].

III. The C2BID Approach

The `C2BID` approach aims to automatically detect suspicious hosts by analysing how they change from cluster to cluster in a selected time period. Hosts that are outliers in terms of these changes, are flagged as anomalous. Host behavior is characterized using features extracted from network flow data, e.g., obtained in routers with the Netflow feature [16]–[18], so this is a network-based intrusion detection approach.

Figure 1 represents the approach that has four steps: feature extraction, clustering, history path creation, and outlier detection, each of them presented in-depth in the next sections.

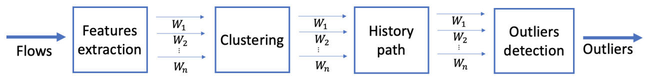

Fig. 1. Sequence of steps of the `C2BID` approach

A. Feature extraction

`C2BID` considers two time frames. First, detection is performed in a period of analysis $\mathcal{T}_a$, e.g., 1 day. Second, features are extracted from network flows in smaller time windows of several durations $\mathcal{W} = \{w_1, w_2, ..., w_n\}$. The size of the time windows w_n has to be at least four times smaller than $\mathcal{T}_a$, e.g., some minutes. The approach does clustering for every window of duration $w_i \in \mathcal{W}$ during $\mathcal{T}_a$ and analyses how each host changes of cluster.

For every time window of duration $w_i \in \mathcal{W}$, features are extracted from all flows that appear in that window. Flows are aggregated by host, identified by IP address, so that features are associated to a host and characterize that host. For simplicity we consider a bijective association between hosts and IP addresses. This is a simplification because hosts can have a few different IP addresses and IP addresses may change due to the use of DHCP. However, we conjecture that misbehavior can be observed by inspecting communications in one of the IP addresses of the host and assume that IP addresses change slowly enough in comparison to our time of analysis, so results are unaffected.

We consider two sets of features: fixed features and dynamic features. The *fixed features* are always the same. We considered 16 fixed features, which are commonly used in the literature. The first half of the 16 fixed features, with source IP as the aggregation key, are: number of different IPs contacted by the host, number of flows where the host is the source, number of different source ports used by the host, number of different destination ports contacted by the host, sum of total packet length received by the host, sum of total packet length sent by the host, average sent packet size and the ratio of the number of packets sent and their duration. These fixed features describe general network activity of a host. The other 8 are similar but for the destination IPs.

We also consider a set of *dynamic features*, or port-based features, following an idea recently proposed by Dias *et al.* [14]. The rationale is the following. Some attacks are targeted at specific TCP/UDP ports, e.g., SSH brute forcing at port 22, so it is important to have features for those ports. However, there are 2^{16} ports times 2 (TCP and UDP), whereas 0-1023 are System Ports and 1024-49151 are User Ports [36], so the number of features would be excessive. Therefore, for each period $\mathcal{T}_a$ the approach picks $\mathcal{N}_p$ ports according to some criteria and uses 4 features for each of these ports: number of packets sent from the port, number of packets sent to the port, number of packets received on the port and number of packets received from the port.

The result of the feature extraction step is one *vector of features* for each hosts. In the evaluation we consider only hosts that are internal to the organization we consider in each case, as the number of external hosts tends to be much larger and less interesting (only a small fraction of the traffic of those hosts is present in the flows).

B. Clustering

After features are extracted, the vectors of features are provided as input to the clustering algorithm. The idea is to group machines with similar behavior based on the fixed features and the $\mathcal{N}_p \times 4$ port-based features.

As mentioned above, the clustering algorithm selected was K-means. This algorithm has an hyperparameter: the number of clusters, k. To define the value of k for each time window of duration $w_i \in \mathcal{W}$ in $\mathcal{T}_a$, we use the elbow method [37]. The idea is to test various numbers of k to achieve the optimal number of clusters. The goal is to achieve the value of k where the middle distance from observation to the cluster centre has an accentuated decay. This value of k gives a balanced distribution of the entities without falling into overfitting. The Euclidean metric is used to determine the distance between two points.

Normalization is performed by scaling each feature to a range between zero (min) and one (max), using Equations (1) and (2). By normalizing all features extracted, we avoid discriminating features in relation to others.

$$X_{standard} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

$$X_{normalized} = X_{standard} \times (max - min) + min \tag{2}$$

The result of the clustering step is a set of clusters of hosts for each time window of duration $w_i \in \mathcal{W}$. Each host appears in cluster of each time window.

C. History path creation

Given an analysis period of duration $\mathcal{T}_a$ and windows of duration $w_i \in \mathcal{W}$ within the analysis period, we define the *history path* H of a host h as the sequence of clusters in which h appears. From the previous section it is clear that H has one cluster per period of duration w_i, but a clarification is needed: if h has no flows in a period, it is assigned to a special cluster that contains inactive hosts. For each host h and each $\mathcal{T}_a$ period, there is one history path H_i for each time window duration $w_i \in \mathcal{W}$. A history path represents the behavior of a host seen from the network during a period $\mathcal{T}_a$ looking at windows of size w_i; different attacks are easier to detect at windows of different durations, as shown in the experiments.

To construct the history path it is necessary to do cluster classification, i.e., to identify which clusters survive, disappear or emerge between two time windows. This is not trivial because clusters change, so the approach has to assess which clusters are similar in consecutive time intervals.

Cluster classification is done following the framework proposed by Landauer *et al.* [38]. Two clusters are considered similar if more than a certain overlap threshold o_t (e.g., 50%) of the elements contained in cluster C' would have been allocated to cluster C and if they had been used for the generation of cluster map C. The overlap is used to mathematically express cluster relations (Equation (3)). Dividing the union of these two intersected sets by the union of all sets means that the resulting value is in the interval $[0, 1]$, with 1 indicating a perfect match and 0 indicating a total mismatch. Two clusters are considered similar if the overlap is between $]0.5, 1]$. In Equation (3), R_{curr} are the hosts in C, R'_{prev} are the hosts in C that would also be in C', R'_{curr} are the hosts of C', and R_{next} the hosts of C' that would also be in C. Each cluster receives a unique ID that is used to represent it in all time windows where it appears.

$$overlap(C, C') = \frac{|\,(R_{curr} \bigcap R'_{prev}) \bigcup (R_{next} \bigcap R'_{curr})\,|}{|R'_{curr} \bigcup R'_{prev} \bigcup R_{next} \bigcup R_{curr}|} \quad (3)$$

To better illustrate the notion of history path, we present an example in Table I. The example considers a small dataset of 10 hosts, $\mathcal{T}_a = 1$ day and time windows of 120min, i.e., $\mathcal{W} = \{w_{120min}\}$. As the traffic starts after 8am and finishes before 8pm, each history path contains only 6 cluster identifiers (for 8am, 10am, etc.). There are 10 history paths, one per host. Needless to say, in a real case there would be a larger variety of clusters, i.e., more different identifiers, as well as many more hosts. The first column of the table shows network IPs; the first row shows the start time of each time window; each cell contains a number which is the cluster-ID where an IP was assigned in that time window. *nan* means that the IP is not active in that time window. For example, 172.31.69.8 has a suspicious behavior as it changes of cluster 2 times in ways that the others do not: $12pm \rightarrow 2pm$ and $2pm \rightarrow 4pm$. Although both clusters 3.0 and 2.0 contain more than one IP, the host 172.31.69.8 would be a candidate to be marked as an outlier by C2BID.

The result of the history path creation steps is a set of history paths. Next section explains the outlier detection.

TABLE I
EXAMPLES OF 10 HISTORY PATH FOR w_{120min}

Hosts	08:00	10:00	12:00	14:00	16:00	18:00
172.31.69.23	2.0	2.0	2.0	2.0	2.0	2.0
172.31.69.17	2.0	2.0	2.0	2.0	2.0	2.0
172.31.69.14	3.0	140.0	3.0	3.0	3.0	3.0
172.31.69.12	3.0	*nan*	3.0	3.0	3.0	3.0
172.31.69.10	3.0	140.0	3.0	3.0	3.0	3.0
172.31.69.8	*nan*	140.0	3.0	2.0	3.0	3.0
172.31.69.6	*nan*	*nan*	*nan*	3.0	3.0	*nan*
172.31.69.26	3.0	140.0	3.0	3.0	3.0	3.0
172.31.69.29	3.0	140.0	3.0	3.0	3.0	3.0
172.31.69.30	3.0	140.0	3.0	3.0	3.0	3.0

D. Outlier detection

Outlier detection is performed based on the kind of cluster changes that a host made between time windows in the analysed period. The relevant cluster changes are: from one cluster to another, to the same cluster, or from one cluster to inactivity.

Outlier detection is done in three steps (Figure 2):

- Calculating the likelihood of each host doing a certain set of changes, taking into account all the changes made in the analysed $\mathcal{T}_a$. This calculation is done in parallel for all different time windows $\mathcal{W} = \{w_1, w_2, ..., w_n\}$;
- Application of the RRCF algorithm, to detect probability values and outliers;
- Filtering of the results produced by RRCF according to certain criteria.

Each period of duration $\mathcal{T}_a$ is analysed individually. These periods can be subdivided into periods of time $\mathcal{T}_s$, in a way where the max W_n has at least one transaction in period $\mathcal{T}_s$, i.e., $2 \times W_n < \mathcal{T}_s < \mathcal{T}_a$. These divisions are analysed in parallel. Entities identified in at least one period as an outlier are considered a potential threat. These divisions aims to reduce the possibility of a host being completely excluded from the analysis because, as detailed later, a host inactive for more than 60% of the period $\mathcal{T}_s$ is not considered.

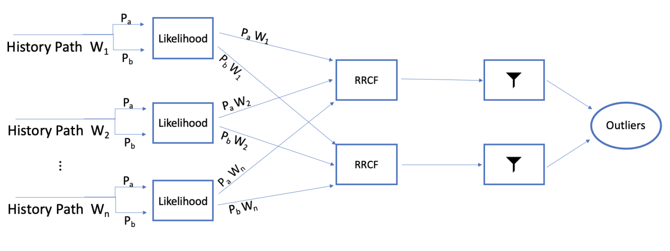

Fig. 2. Outlier detection Framework

1) Likelihood: Each host's likelihood is calculated in parallel by division $\mathcal{T}_s$ and by the time window ($\mathcal{W} = \{w_1, w_2, ..., w_n\}$). The final product consists of a numerical value that expresses the probability of an IP change between clusters in the period studied, taking into account all the transitions in that period $\mathcal{T}_s$. Hosts that do not have a considerable expression over a period $\mathcal{T}_s$ have to be removed. We consider a threshold of 60%, i.e., all entities hosts inactive for at least 60% of the time are removed. This exclusion can give rise to two cases:

- A host may not have enough activity in a shorter time window, e.g., w_{10min}, but the same may not happen with longer time windows, e.g., w_{120min}. If a host is not excluded from all studied time windows, the host probability in time windows in which it was excluded will be estimated based on the percentile of the windows in which it was not excluded;
- A host excluded from all time windows is analysed based on the method used by DynIDS [14]. In the active period, this means analysing if the host is in a unitary cluster; if it is, the host is considered an outlier.

From the transitions of all entities, a transition matrix is created, this matrix expresses the absolute frequency of each transition between two clusters. All transitions from a cluster to inactivity are excluded since these transitions do not provide useful information regarding intrusions in the network. After the removal, the sum of all rows and columns is calculated, and the matrix is normalised with the obtained value, the matrix became a relative frequencies matrix. For computing purposes, the transformation $f(x) = -\log(x)$ is applied to all matrix values. When applying this transformation, it becomes easier to calculate the product between two probabilities, since it is the sum. There is also an inversion of the relative order, i.e., the highest values became the lowest, and the lowest became the highest in order to avoid negative values.

The values in the transition matrix are analysed and weighted according to their relative value. The weights are assigned using a linear function (Equation (4)), where x' is the new likelihood value, x is the old likelihood value, $Q \in [0, 100]$ is x relative position in the matrix, and $\{m, b\}$ are linear function parameters. Ideally, when $Q = 50$ the value should remain the same, i.e., $mQ + b = 0$ and for $Q > 50 \rightarrow mQ + b > 0$, but it can be adjusted.

$$x' = x \times (1 + mQ + b) \tag{4}$$

In this process, only the different values are considered, that is, each different value is counted once, the values are all ascending sorted and its relative position, Q, is obtained by the position of x in the sorted vector.

To obtain the numerical value for the set of transitions made by a host, the likelihood corresponding to a host transition is added together. Any missing value, which does not exist in the transition matrix, is obtained through the equivalent percentile. For example, a host that has three transitions values: $[5, 10, x]$, can obtain the value of x through the median of the known values. That is, the value 5 belongs to the 20th percentile, and the value 10 belongs to the 20th percentile, so the value x will be the value of the 20th percentile ($median(20, 20)$) that corresponds to 7 (central value). Thus, the host has a probability associated with its path of $5 + 10 + 7 = 22$. Finally, the values of each probability of the different time windows are combined per host in a vector form, e.g., $P_{entity} = (P_{w_1}^{entity}, ..., P_{w_n}^{entity})$.

2) Robust Random Cut Forest: RRCF [31] is an unsupervised algorithm that detects outliers. This algorithm produces a metric for each entity, being able to handle: streaming data, irrelevant dimensions and duplicate values that can mask outliers. Recall that the distance used is the Euclidean distance, normalised using Equations (1) and (2).

Outliers are the entities with the highest metric value. For outlier detection, a decision rule is defined based on the lowest percentile from which a host is considered an outlier. RRCF uses randomly generated parameters, so two consecutive executions may not generate precisely the same result. This may mean that not all outliers are detected with a single run, but by running RRCF multiple times it is possible to mitigate this limitation.

3) Filtering: The RRCF algorithm does not distinguish between high and low likelihood values. Having in mind that the objective is to find hosts that take an unusual path (high likelihood), it is necessary to remove those that the RRFC has identified and do not have an unusual path (low likelihood). This process uses two mechanisms: a filter for lower values and a whitelist. The whitelist is defined based on network architecture. It may include DNS or web servers, that have traffic patterns that differ from other hosts. The lower values filter is defined based on a decision rule, k. The value of k is used in Equation (5) to calculate the percentile c for each division $\mathcal{T}_s$, n represents the number of different entities under analysis. Any value marked by the RRCF but with all probabilities below the filter ($P_1 < C_1 \wedge P_2 < C_2 ... \wedge P_n < C_n$) is no longer considered an outlier.

$$C = \frac{n - k}{n} \tag{5}$$

The result of this last step of the C2BID approach are outliers.

IV. Experimental evaluation

To develop and implement C2BID for evaluation, we used Python (v3) [39]. Additionally, we used popular libraries such as Pandas [40] for data manipulation, and Scikit-learn [26] for data processing and the clustering algorithms. All the experiments were done in commodity hardware (6-Core Intel Core i9 2.9GHz with 32GB RAM). The focus of the experiments is the comparison against different approaches and performance evaluation.

A. Metrics

We consider an outlier to be a host, identified by an IP address, flagged by the C2BID. In the following expressions, we consider True Positives (TP) to be hosts correctly classified as outliers, True Negatives (TN) hosts correctly classified as inliers, False Positives (FP) hosts wrongly classified as outliers and False Negatives (FN) hosts wrongly classified as inliers. The metrics used in the evaluation are:

- Precision (PREC) – the fraction of outliers that are real (i.e., true positives):

$$PREC = \frac{TP}{TP + FP} \tag{6}$$

- Recall (REC) – the fraction of outliers that are correctly classified as such by the detector:

$$REC = \frac{TP}{TP + FN} \tag{7}$$

- F-Score – a global detection score:

$$FScore = 2 \times \frac{PREC \times REC}{PREC + REC} \tag{8}$$

Another metric, accuracy, is frequently used in this context, but it is misleading with unbalanced datasets, which are essentially all realistic cases of intrusion detection. Therefore, we avoid using accuracy, and we privilege F-Score, which summarizes the overall performance.

B. Dataset characterization

We used two datasets. The first, *CIC-IDS2018* [20], that we designate *artificial dataset*, was created to test and evaluate network IDSs. 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. Benign behaviors were generated using B-profiles. Such a profile is designed to extract the abstract behavior of a group of human users, encapsulating the host behaviors of users using various machine learning and statistical analysis techniques. Malicious behavior is generated using M-Profiles. These profiles aim to describe an attack scenario unambiguously, in such a way that humans might interpret these profiles and subsequently carry their attacks. The network topology represents a typical medium company, with 6 subnets, deployed on the AWS cloud computing platform.

We consider 6 attacks scenarios: brute force attack, DoS attack, web attacks, infiltration attacks, DDoS and port scan (Table II). In all days except day 4, the attacks occurred in two distinct periods, one attack at a time. The rightmost column indicates the relation between the number of attackers and victims. The attacks were performed from one or more machines, using Kali Linux, in a specific network (within public IPs range) created only to attacker machines.

TABLE II
SUMMARY OF THE ATTACKS FOR THE CIC-IDS-2018 DATASET

Day	Attacks and Duration	Pattern
1	Brute Force to FTP and SSH (90min each)	1-to-1
2	DoS GoldenEye and Slowloris (40min each)	1-to-1
3	Brute Force to FTP and DoS Hulk (60min + 35min)	1-to-1
4	DoS LOIC-HTTP (60min)	n-to-1
5	DoS LOIC-UDP and HOIC (30min+60min)	n-to-1
6	Brute force Web/XSS and SQL inj. (60min+40min)	1-to-1
7	Brute force Web/XSS and SQL inj. (60min+70min)	1-to-1
8	Infiltration and port scan (70min+60min)	1-to-1
9	Infiltration and port scan (60min+90min)	1-to-1

The second dataset, military network dataset or *real-world dataset*, was obtained from the Security Information and Event Management (SIEM) system in production in that network, which collects Netflow events from internal routers [41]. Collecting these flows can give us insights of misbehavior of internal hosts, undetected by deployed security systems. The dataset corresponds to a full month, with approximately 5,500 computers and 160 GB of data. The attacks were stealth dictionary attacks (against SSH and RDP) preceded by a port scan at a slow pace (5-second interval). The main reasons for choosing these attacks were: (1) to have attacks that go unnoticed by traditional protection systems; (2) to capture internal reconnaissance activities (e.g., port scans) and slow dictionary attacks used by attackers with privileged information.

C. Results with the artificial dataset

The counting of positives was done considering all IPs flagged by C2BID in each $\mathcal{T}_a$. This counting method is susceptible to FPs since each attack counts only as one TP. We use the parameters in Table III. Parameters definition depends on network architecture and use, as well as the sensitivity desired by operators. For example, for $\mathcal{N}_p$ definition, it was considered the one with the best results in DynIDS [14]. Only the internal IPs were analysed since C2BID needs a continuous source of information which is hard to get with external IPs. In the particular case of this dataset, all attackers have an external IP, and all victims have an internal IP. For each day, there is one victim. The whitelist is all internal IPs contacted by more than 90 of the internal IPs, taking into account every day in the dataset.

TABLE III
SUMMARY OF PARAMETERS USED WITH ARTIFICIAL DATASET

Parameter	Value	Description
$\mathcal{N}_p \times 4$	400	Number of port-based features
$\mathcal{W}$=$\{w_1, ...w_n\}$	10,30,120min	Time window durations
$\mathcal{T}_a$	1 day	Analysis period of time
$\mathcal{T}_s$	12 hours	$\mathcal{T}_a$ sub-divisions
o_t	50%	Overlap threshold
m	0.004	Slope of Equation (4)
b	-0.2	Constant of Equation (4)
k	2	Filter parameter
-	99.6	RRCF sensibility

C2BID flagged some IPs that were not listed as malicious by the dataset authors.[1]. Initially we considered them FPs, but further inspection has shown that they were TPs. The manual analysis was performed by inspecting the flows involving FPs. For each FP (potential victim), the most contacted IPs were identified (potential attackers). Then, we observed that indeed their behavior was suspicious and we searched for the latter in public databases of malicious IPs and found them. This confirmed that these were TPs, not FPs, so we started counting them as such.

We were able to identify all victims all days. Table IV shows a summary of the results for the 9 days. If we (wrongly) considered as TPs just the IPs marked by the authors of the datasets and the others as FPs, we would get a PREC of 0.473, REC of 1, and F-score of 0.643.

[1]day 1: 172.31.66.82 and 172.31.67.109; day 2: 172.31.66.112; day 4: 172.31.65.56; day 5: 172.31.69.19; day 9: 172.31.66.100

TABLE IV
SUMMARY OF THE RESULTS IN ARTIFICIAL DATASET

$\mathcal{W}$	PREC	REC	F-score
$10, 30, 120min$	0.789	1	0.882

D. Results with the real-world dataset

In this dataset, we applied the values in Table III to all parameters presented in Section III, except for $\mathcal{T}_s$ which was set to 4 hours. $\mathcal{T}_s$ was reduced from 12 hours to 4 hours because the real-world dataset has almost a continuous flow of data all day (24 hours), whereas the artificial dataset data only between 8am and 8pm. The dataset has four servers included in the whitelist. Only the internal IPs were analysed.

The identified attacks were: (1) slow port scan with 5s pace (attacker and victim); (2) stealth dictionary attack RDP (attacker and victim); and (3) stealth dictionary attack SSH (attacker). We also identified some hosts not involved in emulated attacks but marked as outliers due to a misconfiguration confirmed by the Security Operations Center.

In summary, the alerts raised by C2BID corresponded to real threats or anomaly. We were able to identify attackers and victims in a universe of 5000 hosts. All in all, C2BID proved to be useful in a practical setting without significant effort to deploy since it just needs to be fed with NetFlow events.

E. Comparison with previous approaches

This section compares C2BID with three previous works in the area: OutGene [11], DynIDS [14], and FlowHacker [10]. We selected these three because they are recent. We do not compare with more solutions as they would be older, no implementations are available, and/or do not follow the same assumptions regarding no need of training data.

All of the three approaches have two phases: feature extraction and clustering. They differ from each other in terms of features and clustering algorithms used. An IP is considered an outlier if it is in a cluster with just one element. They only consider as outlier IPs that are isolated in a specific time window, not analysing cluster changes along time. The features were extracted by time windows which, for comparison purposes, are the same as those shown in Table III and the $\mathcal{T}_a$ value (1 day).

For positive counting we used the same method as before: number of different IPs marked as outliers in each $\mathcal{T}_a$ (1 day). This method of counting differs from those used by the authors of the three papers, which leads to different results from those provided in the original works. This change is due to C2BID not being adequate for detection in small windows of time, e.g., of minutes, as these works do; C2BID monitors cluster changes along several of these time windows. We used both datasets.

In all cases, it was possible to obtain better values of PREC and F-score with C2BID, with both datasets. For the real-world dataset, REC was the same in almost all algorithms. The main difference between these approaches and C2BID is that they produced much more FPs.

Figures 3, 4 and 5 show the results regarding OutGene, FlowHacker and DynIDS for the artificial dataset. The dotted black line in each graph represents the values obtained for C2BID (cf. Table IV).

1) Outgene: Outgene uses only fixed features: number of different IPs contacted by an entity, number of flows were the host is the source, number of different source ports used by an entity, number of different destination ports contacted by an entity, sum of total packets length received by an entity, sum of total packets length sent by an entity. The other 8 are similar but for the destination IPs. It also counts packages send/received by a few well-known ports such as 80, 194, 25 and 22. For clustering, it used K-means. We used the elbow method to get the optimal number of clusters. The original work defines the number of clusters based on empirical experiences. In Figure 3 we can observe small values for PREC and, consequently, small values for F-score for Outgene.

2) FlowHacker: FlowHacker builds two feature vectors (statistic and count-based) using IP addresses as a source or destination aggregation key and processes both keys independently. It also counts packages send/received by well-known port such as 80, 194, 25, 22 and 6667. This approach has different features comparing with Outgene, DynIDS and C2BID, such as the ratio of ICMP packets and ratio of packets with SYN flag. We only present results with destination as aggregation key because they were better than the others. In the real-world dataset, due to a lack of information, the feature about the SYN flag rate was not considered. For clustering, it used K-means with the elbow method to get the optimal number of clusters. FlowHacker initially depends on manual classification for outlier detection, and we shortcut it by considering as outliers one-host clusters. Figures 3, 4, and 5 show the results with the artificial dataset. It it is possible to observe worse results when comparing to C2BID.

3) DynIDS: DynIDS introduces the idea of dynamic features. The paper presents a few variants of the scheme, but we consider the one the authors consider to be DynIDS, which is the one with better results: DYN3_100 [14]. The extraction of features in DynIDS is very similar to ours. The features are the same except for average sent/received packet size and the ratio of the number of packets sent/received and its duration. DynIDS uses three clustering algorithms, K-Means, Agglomerative and DBSCAN, and an outlier is flagged only when returned simultaneously by the three. In Figures 3, 4 and 5 we can see that DynIDS performed worse than C2BID for all metrics, due to a higher number of FPs. Despite similar results for PREC, DynIDS was not able to detect all victims on 120min and 1day time windows.

V. Related Work

There are several surveys and books on intrusion detection based on machine learning [1], [5], [7]–[9], [41]. An earlier study using clustering for network intrusion detection is due

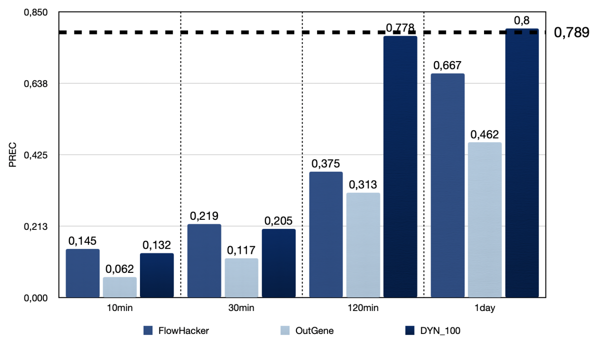

Fig. 3. PREC comparison in OutGene, FlowHacker and DynIDS for the artificial dataset. The dotted black line represents the value obtained for `C2BID`.

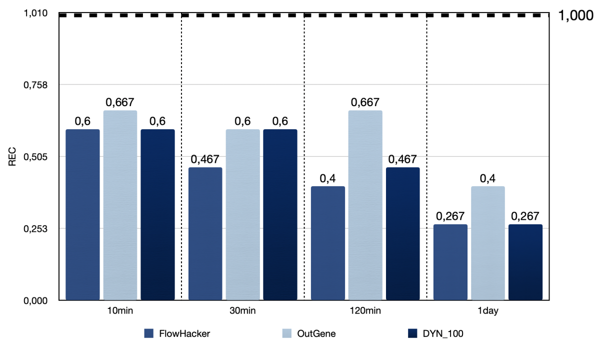

Fig. 4. REC comparison in OutGene, FlowHacker and DynIDS for the artificial dataset. The dotted black line represents the value obtained for `C2BID`.

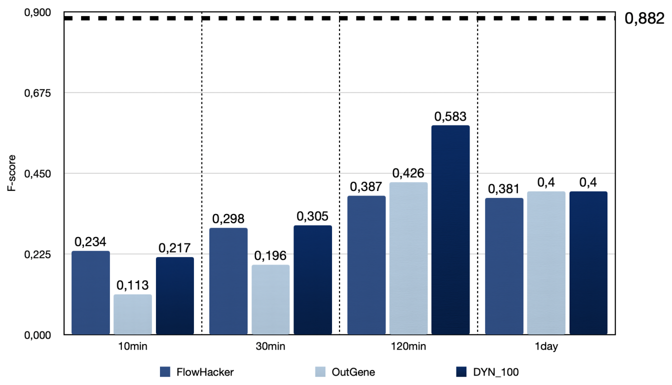

Fig. 5. F-score comparison in OutGene, FlowHacker and DynIDS for the artificial dataset. The dotted black line represents the value obtained for `C2BID`.

to Leung and Leckie [42]. They present their own clustering algorithm (fpMAFIA) and test it with an old dataset (KDD 99). Several other approaches appeared afterwards [43]–[46], always focusing on attack identification without prior knowledge. UNIDS outperforms traditional approaches by applying an unsupervised outlier detection based on sub-space clustering and multiple evidence accumulation [4]. By using PCA and clustering in a substantial volume of logs, Beehive was able to detect malware infections and policy violations that went otherwise unnoticed [6]. Gonçalves *et al.* used logs analysis to detect misbehavior [15]. They combined supervised and unsupervised (clustering with expectation–maximisation) machine learning. The output of the process was not accurate enough to take automatic actions. It extracted relevant information from logs that otherwise were not directly observable. Malicious traffic and malicious hosts are identified by flows inspection in FlowHacker, without requiring either previous knowledge about attacks or traffic without attacks [10]. Outliers were identified using K-means. The system was able to detect and classify attacks through traffic analysis. It generated some false positives and missed some malicious flows. OutGene also applied K-Means to network flows [11]. It used a genetic algorithm to identify the features that were more relevant to characterise a cluster. DynIDS is a framework with dynamic port selection based on most used ports, less used and more uncommon ports [14]. They also applied three clustering algorithms (K-Means, DBSCAN and Agglomerative) and got a reduction of FP rate.

Some recent works do do not use clustering and rely on knowledge of what good behavior is. Cinque *et al.* used entropy to infer deviations from a baseline [47]. DeepLog was inspired in natural language processing and interprets logs as elements of a sequence that follows grammar rules [48]. Kitsune uses an ensemble of neural networks called autoencoders to differentiate between normal and abnormal traffic patterns collectively [49]. AI^2 was a log-based system where density-based, matrix decomposition, and replicator ANN were used to modelling the joining behavior of different hosts within a big raw dataset [50]. This system was able to learn and defend against unseen attacks. Marchetti *et al.* developed an automatic and early detection APT system [51]. It extracted and analysed flow records to build a comparative analysis of past behavior of each host by the computation of suspiciousness scores. The framework could identify burst and low-and-slow exfiltration.

Data change can be seen as a time series if we refer to an ordered list of numbers, or as a sequential pattern if it is a list of nominal values, e.g., symbols. There are several approaches to detect data changes in time [52]–[55]. Bornemann *et al.* applied the concept of time series clustering to detect abnormal changes [56], [57]. Although this kind of analysis is useful when the subject is a list of numbers, it can not be used on nominal values. Han *et al.* proposed that collective outliers in temporal sequences can be detected by learning a Markov model from the sequences [58]. A subsequence can then be declared as a collective outlier if it significantly deviates from the model.

Cluster monitoring allows us to correlate changes in clusters of two consecutive time windows. Over time clusters may suffer some changes. These changes are detectable by associating clusters over multiple time windows. MClusT is a framework that uses bipartite graphs and conditional probabilities to monitor transitions defined in the defined taxonomy [59]. Landauer *et al.* based their approach on cluster evolution techniques where the same elements are observed and clustered over time [38]. They use cluster evolution in order to process log data in a streaming manner rather than being limited to fixed-size data sets. To detect anomalies, the authors checked whether a future value lies within the prediction interval, using the last recorded time step and thus create a forecast for upcoming values. Authors identify a high amount of FPs in the evaluation results of their framework. C2BID apply a classification system similar to the one presented in [38] to construct the history path. We use the binomial cluster and entity instead of log and entity. C2BID decrease the FPs because it uses a different outliers detection method and study NetFlows instead of logs. On the contrary of MClusT, their proposal does not rely on clustering association on conditional probability but on direct association between elements in different time windows.

Clustering methods in cybersecurity have only been applied to specific time windows, not to correlate results from sequences of time windows. Time series works do not apply directly to cybersecurity and are sometimes dependent on supervised learning methods. The existing cluster classification methods produce a high amount of FPs. We advance previous work by considering multiple time windows and understanding host movement between clusters through sequential series clustering techniques in sequences of time windows.

VI. Conclusion

We present C2BID, an approach for network intrusion detection, based on unsupervised learning, that detects undefined attacks without signatures and clean training data. Our approach is not focused on real-time intrusion detection as we need a considerable period of flows to get results. The approach is based on clustering, i.e., on aggregating hosts with similar traffic patterns; on time analysis, i.e., on the behavior of a host in a time period; and on cluster monitoring, i.e., analyzing cluster changes to detect outliers. In the experiments, almost all flagged outliers were either attacks or misconfigured hosts. By correlating more than one time window it was possible to detect attacks occurring in different rates. C2BID was able to reduce the number o FPs even in big datasets comparing with other approaches as OutGene, FlowHacker, and DynIDS. When compared with previous works, C2BID achieved better results both with an artificial dataset and in a real-world scenario.

References

[1] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–336, 2013.

[2] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han, "When intrusion detection meets blockchain technology: A review," *IEEE Access*, vol. 6, pp. 10 179–10 188, 2018.

[3] Fireye and Mandiant, "Special report," M-TRENDS, Report, 2020.

[4] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[5] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[6] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 199–208.

[7] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*. CRC press, 2016.

[8] R. A. A. Habeeb, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A survey," *International Journal of Information Management*, vol. 45, pp. 289–307, 2019.

[9] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: A survey," *Journal of Big Data*, vol. 2, no. 1, p. 3, 2015.

[10] L. Sacramento, I. Medeiros, J. Bota, and M. Correia, "Flowhacker: Detecting unknown network attacks in big traffic data using network flows," in *17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2018, pp. 567–572.

[11] L. Dias, H. Reia, R. Neves, and M. Correia, "OutGene: Detecting undefined network attacks with time stretching and genetic zoom," vol. 2019, no. 830892, pp. 1–20,

[12] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Evaluation of anomaly detection algorithms made easy with RELOAD," in *Proceedings of the International Symposium on Software Reliability Engineering*, 2019, pp. 446–455.

[13] T. Zoppi, A. Ceccarelli, L. Salani, and A. Bondavalli, "On the educated selection of unsupervised algorithms via attacks and anomaly classes," *Journal of Information Security and Applications*, vol. 52, 2020.

[14] L. Dias, M. Correia, and S. Valente, "Go with the flow: Clustering dynamically-defined netflow features for network intrusion detection with DynIDS," in submission, 2020.

[15] D. Gonçalves, J. Bota, and M. Correia, "Big data analytics for detecting host misbehavior in large logs," in *14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2015, pp. 238–245.

[16] B. Claise, *Cisco systems netflow services export version 9*, IETF Request For Comments 3954, 2004.

[17] B. Claise, B. Trammell, and P. Aitken, *Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information*, IETF Request For Comments 7011, Sep. 2013.

[18] M. Wang, B. Li, and Z. Li, "sFlow: Towards resource-efficient and agile service federation in service overlay networks," in *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems*, 2004, pp. 628–635.

[19] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *International Work-Conference on Artificial Neural Networks*, 2005, pp. 758–770.

[20] Canadian Institute for Cybersecurity and University of New Brunswick, *CSE-CIC-IDS2018*, 2018. [Online]. Available: https : / / registry . opendata.aws/cse-cic-ids2018/ (visited on 11/18/2019).

[21] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[22] V. Engen, "Machine learning for network based intrusion detection: An investigation into discrepancies in findings with the KDD Cup'99 data set and multi-objective evolution of neural network classifier ensembles from imbalanced data," Ph.D. dissertation, Bournemouth University, Jun. 2010.

[23] C.-X. Zhang, J.-S. Zhang, N.-N. Ji, and G. Guo, "Learning ensemble classifiers via restricted Boltzmann machines," *Pattern Recognition Letters*, vol. 36, pp. 161–170, 2014.

[24] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–297.

[25] D. Steinley, "K-means clustering: A half-century synthesis," *British Journal of Mathematical and Statistical Psychology*, vol. 59, no. 1, pp. 1–34, 2006.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in Python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[29] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM SIGMOD Record*, vol. 28, no. 2, pp. 49–60, 1999.

[30] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000, pp. 93–104.

[31] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, "Robust random cut forest based anomaly detection on streams," *33rd International Conference on Machine Learning (ICML)*, vol. 6, pp. 3987–3999, 2016.

[32] M. Bartos, A. Mullapudi, and S. Troutman, "Implementation of the robust random cut forest algorithm for anomaly detection on streams," *Journal of Open Source Software*, vol. 4, no. 35, p. 1336, 2019.

[33] N. Rahmah and I. S. Sitanggang, "Determination of optimal epsilon (eps) value on dbscan algorithm to clustering data on peatland hotspots in sumatra," in *IOP Conference Series: Earth and Environmental Science*, IOP Publishing, vol. 31, 2016, p. 012 012.

[34] D. Eswaran, C. Faloutsos, S. Guha, and N. Mishra, "Spotlight: Detecting anomalies in streaming graphs," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1378–1386.

[35] N. Tatbul, T. J. Lee, S. Zdonik, M. Alam, and J. Gottschlich, "Precision and recall for time series," in *Advances in Neural Information Processing Systems*, 2018, pp. 1920–1930.

[36] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry.," *RFC*, vol. 6335, pp. 1–33, 2011.

[37] P. Bholowalia, "EBK-Means: A clustering technique based on elbow method and K-Means in WSN," *International Journal of Computer Applications*, 2014.

[38] M. Landauer, M. Wurzenberger, F. Skopik, G. Settanni, and P. Filzmoser, "Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection," *Computers and Security*, vol. 79, pp. 94–116, 2018.

[39] F. Menczer, S. Fortunato, and C. A. Davis, "Python Tutorial," in *A First Course in Network Science*, 2020.

[40] W. McKinney, "Data Structures for Statistical Computing in Python," *Proceedings of the 9th Python in Science Conference*, 2010.

[41] L. F. Dias and M. Correia, "Big data analytics for intrusion detection: An overview," in *Handbook of Research on Machine and Deep Learning Applications for Cyber Security*, IGI Global, 2020, pp. 292–316.

[42] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the 28th Australasian Conference on Computer Science*, 2005, pp. 333–342.

[43] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *17th USENIX Security Symposium*, 2008, pp. 139–154.

[44] T.-F. Yen, "Detecting stealthy malware using behavioral features in network traffic," Ph.D. dissertation, Carnegie Mellon University Department of Electrical and Computer Engineering, 2011.

[45] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita, "NADO: Network anomaly detection using outlier approach," in *Proceedings of the 2011 International Conference on Communication, Computing & Security*, 2011, pp. 531–536.

[46] ——, "An effective unsupervised network anomaly detection method," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, 2012, pp. 533–539.

[47] M. Cinque, R. Della Corte, and A. Pecchia, "Entropy-based security analytics: Measurements from a critical information system," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2017, pp. 379–390.

[48] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[49] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[50] K. Veeramachaneni, I. Arnaldo, A. Cuesta-Infante, V. Korrapati, C. Bassias, and K. Li, "AI^2: Training a big data machine to defend," in *Proceedings of the 2nd IEEE International Conference on Big Data Security on Cloud*, 2016, pp. 49–54.

[51] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido, "Analysis of high volumes of network traffic for advanced persistent threat detection," *Computer Networks*, vol. 109, pp. 127–141, 2016.

[52] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

[53] A. Allahdadi, R. Morla, and J. S. Cardoso, "Outlier detection in 802.11 wireless access points using hidden markov models," in *7th IFIP Wireless and Mobile Networking Conference*, 2014, pp. 1–8.

[54] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.

[55] W. He, G. Feng, Q. Wu, T. He, S. Wan, and J. Chou, "A new method for abrupt dynamic change detection of correlated time series," *International Journal of Climatology*, vol. 32, no. 10, pp. 1604–1614, 2012.

[56] T. Bleifuß, L. Bornemann, T. Johnson, D. V. Kalashnikov, F. Naumann, and D. Srivastava, "Exploring change: A new dimension of data analytics," *Proceedings of the VLDB Endowment*, vol. 12, no. 2, pp. 85–98, 2018.

[57] L. Bornemann, T. Bleifuß, D. Kalashnikov, F. Naumann, and D. Srivastava, "Data change exploration using time series clustering," *Datenbank-Spektrum*, vol. 18, no. 2, pp. 79–87, 2018.

[58] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2012.

[59] M. Oliveira and J. Gama, "Bipartite graphs for monitoring clusters transitions," in *International Symposium on Intelligent Data Analysis*, Springer, 2010, pp. 114–124.