Cluster Change-Based Intrusion Detection

Tiago Francisco Capelo Fernandes

Thesis to obtain the Master of Science Degree in
Electrical and Computer Engineering

Supervisor: Professor Miguel Nuno Dias Alves Pupo Correia
Major Tm Luís Filipe Xavier Cavaco de Mendonça Dias

Examination Committee:
Chairperson: Professor Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Professor Miguel Nuno Dias Alves Pupo Correia
Members of the Committee: Professor Ibéria Vitória de Sousa Medeiros
Tenente-Coronel Tm Henrique Martins dos Santos Cunha

January 2021

Declaration

I declare that this dissertation is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgements

First of all I would like to thank Professor Miguel Correia and Major Tm Luís Dias, whose guidance, support and encouragement have been invaluable throughout this work.

I am grateful for the love and support given by my family for their continuous and unparalleled love, help and support. I am forever indebted to my parents for giving me the opportunities and experiences that have made me who I am. They selflessly encouraged me to explore new directions in life and seek my destiny. This journey would not have been possible if not for them.

To my girlfriend, Teresa Trindade, whose kindness was paramount.

Finally, my gratitude goes to comrades of the Military Academy that supported me all the time and gave me strength to make this work possible.

Abstract

This thesis presents a network intrusion detection approach that flags malicious activity without previous knowledge about attacks or training data. The *Cluster Change-Based Intrusion Detection (C2BID)* approach detects intrusions by monitoring host behaviour changes.

For that purpose, C2BID defines and extracts features from network data and aggregates hosts with similar behaviour using clustering. Afterwards it 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 artificial public dataset (evaluation dataset) that contains Netflow's information. Additionally, a dataset with real flows from an organisation was also used for the same purpose. C2BID obtained better F-Score than previous solutions.

Keywords: network intrusion detection, clustering, behaviour change, security analytics

Resumo

Nesta tese é apresentádo uma abordagem de deteção de intrusões.

O principal objectivo é apresentar uma abordagem de deteção de intrusões que consiga assinalar a atividade maliciosa sem informação prévia dos ataques ou sem dados de treino. A abordagem C2BID detecta intrusões através da monitorização de alterações no comportamento das máquinas.

Para esse efeito, o C2BID define e extrai características dos dados e agrega máquinas com comportamento semelhante utilizando o *clustering*. Depois analisa a forma como as máquinas se movem entre clusters ao longo de um período de tempo. Isto contrasta com trabalhos anteriores na área, que param na etapa de *clustering*.

C2BID foi avaliado experimentalmente com um conjunto artificial de dados públicos (conjunto de avaliação) que contém informação de Netflow. Além disso, foi também utilizado um conjunto de dados reais com fluxo de uma organização. C2BID obteve melhor F-Score do que as soluções anteriores.

Palavras-chave: detecção de intrusões, análise de agrupamento, mudança de comportamento, análise de segurança

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. 5, 9, 14, 20

APT Advanced Persistent Threats. 15, 20

C2BID Cluster Change-Based Intrusion Detection. iii, iv, vii, 2, 3, 19, 21, 23, 31–33, 35, 39–41, 47

DB Databases. 8, 10, 34, 49, 50

DBLP Digital Bibliography & Library Project. 17

DBSCAN Density-Based Spatial Clustering of Applications with Noise. 7, 8, 13, 20, 26, 40, 41

DHCP Dynamic Host Configuration Protocol. 12, 21

DNS Domain Name System. 28

DTW Dynamic Time Warping. 16

EM Expectation Maximum. 9, 12, 20

FN False Negatives. 32, 34

FP False Positives. 13, 15, 19, 26, 32–35, 39, 40, 47

FPR False Positive Rate. 3, 14, 20, 41

HMM Hidden Markov Model. 6, 18

IDS Intrusion Detection System. v, viii, 1, 2, 9–15, 19, 20, 32

IMDB International Movie Database. 16, 17

IP Internet Protocol. viii, 2, 21–23, 26, 28, 32–35, 39, 47, 49, 50

KNN K-Nearest Neighbor. 6

LANL Los Alamos National Laboratory. 13, 20

LOF Local Outlier Factor. 8, 13, 26

ML Machine Learning. 2–4, 6, 10–12, 14, 15, 20

NIDS Network Intrusion Detection System. 12

OPTICS Ordering Points to Identify the Clustering Structure. 8, 26

PCA Principal Component Analysis. 9, 12, 20

PREC Precision. vii, 26, 27, 32, 34–40

REC Recall. vii, 32, 34–38

RRCF Robust Random Cut Forest. v, 8, 9, 24, 26, 28, 31

SIEM Security Information and Event Management. 12

SOC Security Operation Center. 1, 35

SVM Support Vector Machine. 5, 26

TN True Negatives. 32

TP True Positives. 1, 32–34, 40, 50

UNIDS Unsupervised Network Intrusion Detection System. 11, 12, 20

VPN Virtual Private Network. 12

1. Introduction

This chapter defines the main motivation and the problem studied in the report.

1.1 Challenges and Motivation

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. According to the United States Homeland Security Council: 'Many of the nation's essential and emergency services, as well as our critical infrastructure, rely on the uninterrupted use of the Internet and communications systems, data, monitoring, and control systems that comprise our cyberinfrastructure. A cyber-attack could be debilitating to our highly interdependent Critical Infrastructure and Key Resources and ultimately to our economy and national security' [29]. The Portuguese Government [36] classified cyber-attacks as a growing threat to critical infrastructures, where potential attackers (terrorists, organized crime, states or isolated individuals) can collapse the technological structure of a modern organization.

Cybersecurity is a set of technologies and processes designed to protect computers, networks, programs and data from attack, unauthorised access, change or destruction [16]. An integral part of cybersecurity in organisations is the Security Operation Center (SOC), whose goal is to monitor security-related events from enterprise IT assets, including the IT network and perimeter defence systems [6]. According to Bhatt *et al.* [6], the main challenges of security incident and event management systems are:

- Identifying attacks from event streams;
- Analysis algorithm must learn and evolve continuously;
- The systems have to prioritise the True Positives (TP);
- More events might lead to statistically significant but ultimately meaningless correlations;
- Malicious attacker actions and benign user actions might generate the same event.

Intrusion detection has become a significant research topic due to advances in Internet technologies and the increasing number of network attacks [9]. To identify cyber threats and possible incidents, *Intrusion Detection System*s (IDS) are widely deployed in various computer networks [53]. It often takes several days to detect some cyber-attacks [32]. These shows that the large variety of traditional mechanisms deployed do not provide enough protection. Hence, organizations have to dig into traffic and logs to search for anomalous patterns in larger windows of time.

IDSs appear due to the notion that audit trails contained vital information that could be valuable in tracking misuse and understanding user behaviour [43]. The intrusion detection market rose in popularity around 1997. In that year, the security market leader developed a network IDS called RealSecure [43].

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 behaviour. 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 [18]. *Anomaly detection* attempts to find patterns that do not conform to expected normal behaviour [9]. 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) behaviours may be flagged as anomalies [16]. *Both approaches require prior knowledge, respectively of signatures and of normal behaviour, something that is inconvenient and that we circumvent in this work.*

Network flows, are built into network devices that collect properties measurements for each flow and exports them to another system for analysis [42]. These common properties may include packet header fields, such as source and destination IP addresses and port numbers, packet contents, and meta-information. By analysing flow data, a picture of network traffic flow and volume can be built. The log file is a file that records either events that occur in an operating system or other software [57]. Network flow (NetFlows) and log files are an information source for IDSs. IDSs can be classified as network-based (that inspect communication data, e.g., Netflows) or host-based (that inspect host activity, e.g., log files) [25].

Finally, IDSs can be classified as online (detect intrusions in runtime) and offline (detect intrusions later, when decided by someone) [25].

Unsupervised Machine Learning (ML) techniques are useful to correlate similar behaviours without any baseline. This can be used to detect malicious hosts in networks without relying on defined labels of any kind and baselines, solving traditional IDS dependence in labelled data. However, after getting well-defined learning models, they can be useful to rank future behaviours based on previous learning model using supervised ML methods.

Unsupervised ML 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 behaviour, 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 [67, 29, 9, 16, 39, 71, 9, 59, 25, 69, 70, 26].

These works have several limitations. Many assume that only a small part of the traffic is malicious, e.g., [25, 26, 59]. This leads to attacks with several machines, e.g., a botnet, not being easily detected. Besides that, 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 [25]. Some models [69, 70] 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 [59, 35].

1.2 Objectives and Key Results

We propose a novel approach for network intrusion detection based on unsupervised learning: *Cluster Change-Based Intrusion Detection* (C2BID). Our approach still uses clustering as the 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 behaviour changes*. For that purpose, C2BID defines and

extracts features from network data – specifically network flow data [20, 21, 65] –, aggregates hosts with similar behaviour using clustering, then analyses how hosts move between clusters along a period of time and detects outliers. This contrasts with previous work in the area that essentially stops at the clustering step. The approach aims to solve the mentioned problems. It uses Unsupervised ML methods applied to NetFlow data.

The objective of this work is to explore a novel approach for intrusion detection. *The main idea is to do clustering of entities (hosts or users) observed during sequential periods of time, then do detection by observing how entities move from one cluster to another between time periods.* We innovate concerning state of the art by using dynamic features, analysing multiple time windows simultaneously and correlating movements along time.

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 clusters' behaviour, it is possible to identify anomalous behaviours and suspicious cluster formation. 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 [26]. This allows analysing the traffic in many ports while avoiding the curse of dimensionality [64], i.e., losing 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 [59], OutGene [25], and DynIDS [26]. We tested C2BID with an artificial dataset [17] and a real-world dataset from a military administrative network. Our evaluation shows that C2BID was able to detect not only the labelled attacks but also found unlabelled (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 (FPR).

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).

1.3 Report Structure

This document is composed of six chapters: The second chapter describes concepts and components of the base framework that will be used. The third chapter introduces related work, developed by other authors. Chapter 4 presents a proposed solution and decisions made. In Chapter 5, we show the results obtained in all approaches made. Last chapter, 6, presents the conclusions and future work.

2. Background

This chapter introduces the concepts upon which this work was based on, from an academic literature standpoint.

2.1 Machine Learning

This section focuses on a literature survey for Machine Learning (ML) methods used in cyber analytics in support of intrusion detection. Short tutorial descriptions of the relevant ML methods are provided, either for *supervised* and *unsupervised* methods.

ML is defined by Arthur Samuel as a '*field of study that gives computers the ability to learn without being explicitly programmed*'. ML focuses on classification and prediction made by computers without being explicitly programmed [16].

During the definition of ML approach and according to the formulated problem, the following steps are often performed: identify class attributes and classes from training data; identify a subset of attributes necessary for classification; learn the model using training data and use the trained model to classify the unknown data. In theory, an ML approach has got two phases: training and testing, but in practice, it is possible to find three phases: training, validation and testing [16].

The performance of an ML approach has to be done empirically because it depends on the type of training experience. The learning machine has undergone, the performance evaluation metrics and the strength of problem definition [29].

The machine learning techniques are commonly divided into three categories [16]: unsupervised, semi-supervised and supervised. Next, it will be introduced supervised and unsupervised categories.

2.1.1 Supervised Machine Learning

Supervised methods are used to find a function or model that explains the data. They are characterized by having the training data completely labelled.

In supervised methods, we assume the existence of a training dataset, with labels instances for the normal as well as the anomaly class. 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 [10].

According to [10] we have two main issues in using supervised methods. First, in training data, anomalous classes are rarer than normal instances. Second, obtaining accurate and representative labels, especially for anomaly class is usually challenging.

The main supervised ML methods are [16, 29]:

- **Association Rule Classification**: It finds associated isomorphisms among multiple attributes. An association rule is considered strong if the support and confidence of a rule are greater than user-specified minimum support and minimum confidence thresholds.

  Association rules have advantages in elucidating interesting relationships, such as causality between the subsets of items (attributes) and class labels. Strong association rules can classify

frequent patterns of attribute-value pairs into various class labels. However, elucidation of all interesting relationships by rules can lead to computational complexity, even for moderately sized data sets.

- **Artificial Neural Network (ANN)**: An ANN is a machine-learning model that transforms inputs into outputs that match targets, through nonlinear information processing in a connected group of artificial neurons. The input data activates the neurons in the first layer of the network whose output is the input to the second layer of neurons in the network. Similarly, each layer passes its output to the next layer and the last layer outputs the result.

  ANN methods perform well for classifying or predicting latent variables that are difficult to measure and solving nonlinear classification problems, that are insensitive to outliers. ANN may generate classification results that are harder to interpret than those obtained from the classification methods that assume functional relationships between data points, such as using associate rules. However, ANN methods are data-dependent, such that the ANN performance can improve with increasing sample data size. They are commonly implemented on graphics processing units because the advanced versions of ANNs require more processing power.

- **Support Vector Machine (SVM)**: It is a classifier based on finding a separating hyper-plane in the feature space between two classes in such a way that the distance between the hyperplane and the closest data points of each class is maximized. SVM find linear, nonlinear, and complex classification boundaries accurately, even with a small training sample size.

  SVM can be addressed to outlier detection, [2]. In this case, it is assumed that the origin of a kernel-based transformed representation belongs to the outlier class. Once the model has been trained, any data point can be scored. From the perspective of the modelling assumptions of the support-vector machine, a negative value of the score indicates that the data point is an outlier, whereas a positive value indicates that the data point is a non-outlier.

  According to [29], this method is fast, but its running time quadruples when a sample data size doubles. Support Vector Machines root in binary classification, selecting kernel functions and fine-tuning the corresponding parameters is a trial-and-error procedure.

- **Decision Tree**: A decision tree is a tree-like structural model that has leaves, which represent classifications or decisions, and branches, which represent the conjunctions of features that lead to those classifications. The decision tree is built by maximizing the information gain at each variable split, resulting in a natural variable ranking or feature selection.

  The advantages of decision trees are intuitive knowledge expression, high classification accuracy, and simple implementation. They cannot guarantee the optimal accuracy that other machine-learning methods can and they are not a popular method of intrusion detection.

- **Bayesian Network**: it is a probabilistic graphical model that represents the variables and the relationships between them. The network is constructed with nodes as the discrete or continuous random variables and directed edges as the relationships between them, establishing a directed acyclic graph. The child nodes are dependent on their parents. Each node maintains the states of the random variable and the conditional probability form.

  Naïve Bayes is a simple Bayesian Network model that assumes all variables are independent. Naïve Bayes is efficient for inference tasks. However, Naïve Bayes is based on a strong independence assumption of the variables involved. Surprisingly, the method gives good results even if the independence assumption is violated.

- **Hidden Markov Model (HMM)**: An Hidden Markov Model is a statistical model where the system being modelled is assumed to be a Markov process with unknown parameters. HMM can be trained both in an unsupervised and a supervised fashion [13]. The main challenge is to determine the hidden parameters from the observable parameters.

  HMM is an elegant and sound method to classify or predict the hidden state of the observed sequences with a high degree of accuracy when data fit the Markov property. However, when the true relationship between hidden sequential states does not fit the proposed HMM structure, HMM will result in poor classification or prediction. Meanwhile, HMM suffers from large training data sets and complex computation, especially when sequences are long and have many labels.

- **Random Forest**: Random forest consists of many decision trees. The output of the random forest is decided by the votes given by all individual trees. Each decision tree is built by classifying the bootstrap samples of the input data using a tree algorithm. The accuracy of the random forest depends on the strength of the individual trees and a measure of the dependence between the trees.

- **K-Nearest Neighbor (KNN)**, [29, 6]: In KNN, each data point is assigned the label that has the highest confidence among the k data points nearest to the query point. The numbers of nearest neighbours (k) and the distance measure are key components for the KNN algorithm.

  KNN suffers when the number of dimensions is high because the distance measures in high dimensions are not able to differentiate well between normal and anomalous instances. KNN is easy to implement and interpret because it does not need to train parameters for learning while it remains powerful for classification. However, KNN classification is time-consuming and storage intensive.

2.1.2 Unsupervised Machine Learning

Unsupervised techniques do not require training data [10]. In unsupervised learning problems, the main task is to find patterns, structures, or knowledge in unlabelled data [16].

The main unsupervised ML methods are:

- **Clustering**: Clustering [16] is a set of techniques for finding patterns in high-dimensional unlabelled data. Objects from the same cluster are more similar to each other than objects from different clusters [29]. Outliers are points in a dataset that are highly unlike to occur in a given model of the data. Usually, they belong to a cluster just with one entity [10].

  Some benefits of Clustering are:

  - For a partitioning approach, if k (number of clusters) can be provided accurately then the task is easy [10].
  - In case of large datasets to group it into a similar number of classes for detecting network anomalies, because it reduces the computational complexity during intrusion detection [10];
  - For intrusion detection it can learn from audit data without requiring the system administrator to provide explicit descriptions of various attack classes [16].

  The main drawbacks of clustering-based methods are the following, [10]:

  - Most techniques have been proposed to handle continuous attributes only;

– Use of an inappropriate proximity measure affects the detection rate negatively;
– Dynamic updating of profiles is time consuming;
– When using outlier-based algorithms, the anomalies must be most uncommon events in a network.

For most clustering algorithms, the number of clusters has to be defined before execution. This number can be chosen using the *Method Elbow* [7]. This method exists upon the idea that one should choose a number of clusters so that adding another cluster does not give better modelling of the data.

The idea of this method is to test various numbers of clusters in order to achieve the optimal number of clusters, i.e., to choose a number of clusters k such that adding another cluster does not improve much better the total within-clusters sum-of-squares.

Different clustering algorithms have different forms of initialisation and produce different data partitions [44] 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 [30, 68, 26].

Subspace Clustering, [29], it is a method that performs a localization search and focuses on only a subset of dimensions. It is used in high-dimensional data where objects are dispersed in space and distance, as the measure of 'sameness' becomes meaningless.

Density-based clustering algorithms provide better results in low-dimensional spaces, because high-dimensional spaces are usually sparse, making it difficult to distinguish between high and low-density regions [18].

Hierarchical clustering is a method of cluster analysis which builds a hierarchy of clusters. The hierarchy may start with each observation as a cluster, and pairs of clusters are merged as one moves up the hierarchy (bottom-up approach), or all observations start in one cluster, and splits are performed as one moves down the hierarchy (top-down approach) [49].

Some concrete clustering algorithms are:

– K-Means clustering [29] is a clustering technique where each data point is more similar to its cluster centroid than to the other cluster centroids. It generally consists of the steps described in [29]:
  1. Select the k initial cluster centroids;
  2. Assign each instance x in S to the cluster that has a centroid nearest to x;
  3. Recompute each cluster's centroid based on which elements are contained in it;
  4. Repeat 2 through 3 until convergence is achieved.
– Density-Based Spatial Clustering of Applications with Noise (DBSCAN), [62]: Discover clusters and noise. First is defined as the density of points needed in a point neighbourhood to be considered as a member of a cluster (ϵ) and the minimum number of points needed to a group be considered as a member of a cluster (min). Without knowing these parameters, *a prior*, this method starts with a random point, p, and determine all points density-reachable from p considering ϵ and min. If this point is a border[1] the algorithm moves to the next point in the dataset. Otherwise, if point n is a core point[2], this procedure yields a cluster. This procedure is performed until all points of a dataset are covered. If a

[1]Point at the end of a cluster
[2]Point in the centre region of a cluster

point distance to all clusters is too large, it is considered noise (outlier). The optimal ϵ can be calculated by getting the distance to the nearest n points for each point, sorting and plotting the results. Then where the change is most pronounced it corresponds to ideal ϵ [58].

- Ordering Points to Identify the Clustering Structure (OPTICS), [5, 14]: is an algorithm for the purpose of cluster analysis which does not produce a clustering of a data set explicitly; but instead it creates an augmented ordering of the Databases (DB) representing its density-based clustering structure. This cluster-ordering contains information which is equivalent to the density-based clustering corresponding to a broad range of parameter settings. OPTICS has the same principle as DBSCAN; the only difference is that it does not assign cluster memberships. For outlier detection it gives an outlier score to each point, that is a comparison to its closest neighbours rather than the entire set. Its advantages include finding varying densities, as well as very little parameter tuning.
- Agglomerative, [40]: This approach starts with each object forming a separate group. It successively merges the objects or groups close to one another, until all the groups are merged into one (the topmost level of the hierarchy), or a termination condition holds.

- **Anomaly Detection**: the process of identifying unexpected items or events in data sets, which differ from the norm. Anomaly detection has two basic assumptions: anomalies only occur very rarely in the data and their features differ from the normal instances significantly.

  There are some anomaly detection algorithms like:

  - Isolation Forest, [48, 2]: the principle is isolating anomalies, instead of the most common techniques of profiling normal points. An isolation forest is a combination of a set of isolation trees. In an isolation tree, the data is recursively partitioned with axis-parallel cuts at randomly chosen partition points in randomly selected attributes so as to isolate the instances into nodes with fewer instances until the points are isolated into one-leaf nodes containing one instance. In such cases, the tree branches containing outliers are noticeably less deep, because these data points are located in sparse regions. Therefore, the distance of the leaf to the root is used as the outlier score. The final combination step is performed by averaging the path lengths of the data points in the different trees of the isolation forest.
  - Local Outlier Factor (LOF), [22, 62]: This method gives a score to each data point by determining the ratio of average densities of neighbours points to the density of the point itself. The estimated density of a certain point is the number of its neighbours divided by the sum of distances of the neighbours points.
  - Elliptic Envelope, [34]: it fits a multivariate Gaussian distribution to the dataset. It will define the shape of the data by creating a frontier that delimits the contour. It assigns a pre-defined percentage of observations as outliers, the ones that worse fit in define shape.
  - Robust Random Cut Forest (RRCF), [38, 1]: considers a point as an anomaly if the complexity of the model increases substantially with the inclusion of the point. The core data structure used to implement RRCF is the robust random cut tree. RRCF 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.

    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 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. If including a new point significantly changes the model complexity, then that point is more likely to be an outlier. 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 [39] and used it in multiple scenarios [31, 61].

- **Principal Component Analysis (PCA)**, [29]: it represents the raw data in a lower-dimensional feature space to convey the maximum useful information. The extracted principal feature components are located in the dimensions that represent the data variability. PCA can extract uncorrelated features to describe the embedded statistical information of datasets.

- **Expectation Maximum (EM)**, [29]: This method is designed to search for the maximum likelihood estimates of the parameters in a probabilistic model. The EM methods assume that parametric statistical models, such as the Gaussian mixture model, can describe the distribution of a set of data points.

  The EM algorithm can result in a high degree of learning accuracy when given datasets have the same distribution as the assumption. Otherwise, the clustering accuracy is low because the model is biased.

- **SOM ANN**, [29]: SOM ANN forms a semantic map where similar samples are mapped close together and dissimilar samples are mapped further apart. It uses ANN in visualizing low-dimensional views of high-dimensional data by preserving neighbourhood properties of the input data.

2.2 Intrusion Detection Systems

The purpose of an IDS is to monitor assets to detect anomalous behaviour and misuse, being an important and integral component of cyber-security [43]. Cyber-security systems are composed of network security systems and computer security systems. IDSs helps discover, determine, and identify unauthorized use, duplication, alteration, and destruction of information systems [16].

IDSs can be classified as online (detect intrusions in runtime) and offline (detect intrusions later, when decided by someone) [26]. If an IDS monitors the characteristics of all local systems and the system events in a host for malicious activities, it is classified according to its location as host-based. If an IDS monitors network traffic and analyses its protocols for suspicious events, this is call network-based [53].

There are three mains types of cyber analytic in support of IDSs: misuse-based, anomaly-based and hybrid [16].

To improve the techniques of IDS, researchers have proposed hybrid detection techniques to combine anomaly and misuse detection techniques in IDS [29].

2.2.1 Misuse-based Techniques

Misuse-based techniques (signature-based) are designed to detect known attacks by using signatures of those attacks. They are effective for detecting known types of attacks without generating an overwhelming number of false alarms. They require frequent manual updates of DB with rules and signatures [16]. Networks protected by misused detection systems suffer from long periods of vulnerability between the diagnosis of a new attack and the construction of the new signature. Misuse detection is the most used approach in IDS [18]. In the case of misuse detection (signature-based), in the training phase, each misuse class is learned by using appropriate exemplars from the training set. In the testing phase, new data are run through the model and the exemplar is classified as to whether it belongs to one misuse class. If the exemplar does not belong to any of the misuse classes, it is classified as normal [16].

2.2.2 Anomaly Detection Techniques

Anomaly detection attempts to find patterns in data, which do not conform to expected normal behaviour [10]. They are appealing because of their ability to detect zero-day attacks. The main disadvantage of anomaly-based techniques is the potential for high false alarm rates because previously unseen (yet legitimate) system behaviours may be categorized as anomalies [16]. Anomaly detection requires training to construct profiles, which is time-consuming and depends on the availability of anomaly-free traffic instances [18]. In the case of anomaly detection, the normal traffic pattern is defined in the training phase. In the testing phase, the learned model is applied to new data, and every exemplar in the testing set is classified as either normal or anomalous [16]. Typically, the outputs produced by anomaly detection techniques are of two types [10]:

- Score a value that combine: distance or deviation with reference to a set of profiles or signatures, influence of majority in its neighbourhood and distinct dominance of relevant subspace.
- Label a value (normal or anomalous) given to each test

In recent decades, ML has started to play a significant role in anomaly detection. Many techniques work in specific domains, although others are more generic [10]. Although some algorithms are accepted to be better performing than others, the performance of a particular ML algorithm is application and implementation dependent [16]. In cyber security domain ML is trained daily, whenever the analyst requires, or each time a new intrusion is identified and its pattern becomes known, differently of most ML applications where a model is trained and is used for a long time [16].

In this chapter, the main Machine Learning methods were presented. It was possible to understand how major ML techniques work as well as their main advantages. In the last section was presented how ML insert in cyber-security. Finally, IDSs were reviewed about their variances and applicability.

3. Related Work

This section presents papers dealing with ML in cybersecurity with particular focus on clustering methods and time series analysis. It is divided into Sections 3.1, 3.2 and 3.3. Section 3.1 focus the work in the field of Clustering Based IDS, an overview of detecting malicious activity using unsupervised ML techniques studied in Chapter 2. The papers are presented chronologically to provide an overview of IDS's evolution, whether they are network-based or host-based. Section 3.2 explores approaches to detect malicious behaviours with unsupervised learning but not clustering. Section 3.3 will first provide an overview of some works related to sequential pattern mining and time-series clustering. Secondly, it will present a survey on data change exploration.

3.1 Clustering Based Intrusion Detection

Clustering-based IDS can extract information from flows, logs or both. Flows represent information extracted from network flows and are associated with network-based IDS. Logs are data produced by a wide variety of security products, e.g., firewalls, or by hosts in a network and are associated with host-based IDS. The information is later analysed with a clustering algorithm to find outliers, i.e., potentials attacks.

There are several surveys and books on ML (and data mining) for cyber-security and intrusion detection [27, 29, 9, 16, 39, 71, 9]. A seminal paper by Lee and Stolfo used ML for intrusion detection [46]. However, they use algorithms to mine association rules and frequent episodes, not clustering. There are many works on ML for intrusion detection, but we focus on those related to unsupervised methods using clustering and that do not need a training phase. An earlier study using clustering for network intrusion detection is due to Leung and Leckie [47]. They present their clustering algorithm, fpMAFIA, and test it with the old 1999 KDD Cup dataset [24]. The number of features used is unclear. BotMiner is focused on botnet detection without a priori knowledge, using clustering [37]. It uses a hierarchical 2-layer clustering scheme based on differentiating the C2 plane from the activity traffic plane. Yen *et al.* also detects botnets by clustering network traffic from Netflow and by considering that small clusters (hosts different from the majority) malicious [66]. NADO uses K-Means and the 1999 KDD Cup dataset [10]. It obtains a reference point from each cluster, builds a profile for each cluster, calculates a score for each data item, and raises an alert if it exceeds a threshold. The same authors presented another approach based on a tree-based subspace clustering technique, i.e., an hierarchical clustering technique [8]. The authors introduce a novel technique to label clusters (CLUSLab) that allows generating labelled datasets.

UNIDS is a IDS capable of detecting network attacks without relying on signatures, training or labelled traffic instances of any kind [18]. It can detect unknown attacks in a completely unsupervised fashion. UNIDS,[18], uses a novel unsupervised outliers detection approach based on Sub-space Clustering and Multiple evidence accumulation techniques to pin-point different kinds of network intrusions and attacks. This system first detects an anomalous time slot in which the clustering analysis will perform. Secondly, it takes as input all the flows in the time slots flagged as anomalous and identify outlying flows using a robust multicluster algorithm. Finally, in the third step, the top-

ranked outlying flows are flagged as anomalies, using a simple threshold detection approach. UNIDS proved its advantage in detecting new previous unseen attacks when compared with misused-based IDS. UNIDS differs from the other works because it first detects anomalous time slots, using time-series analysis and Afterwards it performs clustering in flagged time slots for outliers' search. The remaining works apply clustering to all time-slots and look for outliers in the results.

Beehive [67] uses logs generated by various network devices, e.g. DHCP servers and VPN servers. These logs are stored in a commercial Security Information and Event Management (SIEM) system. Beehive first looks to solve problems related to 'big data' on log analysis and detection of security incidents; it removes the noise and inconsistencies (data normalization). It extracts features from the logs and apply Principal Component Analysis (PCA) to remove dependencies between the features and reduce the dimensionality of features vectors. Then, they use a clustering algorithm to identify security incidents, since employees in the enterprise perform specific job functions on the corporate network, and there are multiple employees in most departments, so it is possible to observe groups of hosts (belonging to users with similar roles) exhibiting similar behaviours. In contrast, misbehaving hosts with unique behavioural patterns appear as *outliers*. For clustering analysis, they adapt the *K-means clustering algorithm*. In the case where extreme outliers bias the algorithm, they apply PCA and clustering algorithm again to the largest clusters. Beehive detected malware infections and policy violations that went otherwise unnoticed, over a two weeks evaluation in a large enterprise (EMC). Specifically, for log files collected, 25.24% of new detection incidents were confirmed to be malware-related or to warrant further investigation; 39.41% were policy violations and 35.33% were associated with unrecognized software or services.

Gonçalves *et al.* [35] propose an approach to detect misbehaviour in logs, that combines data mining and both supervised and unsupervised ML. This approach involves data mining a set of features from the logs, using clustering to create sets of entities with similar behaviours, using linear classification to detect misbehaving sets of entities. The approach can be divided into two main phases: first defining and configuring the detection mechanism and second executing the detection mechanism in runtime. The clustering algorithm used was EM, because it benefits of not requiring prior knowledge of the distribution of each feature and other parameters. The clustering was done using the numeric, normalized, values of features. To characterize a given cluster the features were labelled with tree tags: primary, secondary and non-relevant. A feature is tagged as primary for a cluster if it has a small deviation within that cluster. Non-relevant features are those that do not characterize the cluster, i.e., those in which the mean and the deviation are very similar in all clusters. Secondary features are those that are not tagged either as primary and non-relevant. To evaluate this approach, the authors used logs provided by Vodafone Portugal for 5 consecutive days. They were able to signalize thirteen suspicions classes.

Comparing Beehive [67] and Gonçalves *et al.* [35] we can see that they are very close, but Beehive does not do the supervised classification of the clusters, he considers much fewer features and detects malicious users instead of hosts.

FlowHacker [59] is an approach were malicious traffic and malicious hosts are identified by flows inspection, without requiring either previous knowledge about attacks or traffic without attacks. FlowHacker NIDS (Figure 1) involves splitting traffic (flows) into clusters with the larger corresponding to normal traffic, and the smaller are signalized as threats. Later, those threats are classified as malicious or benign using unsupervised ML (*K-means clustering*). This classification allows reducing the amount of time spent by Humans in flow analyses. The approach works in a loop, iteratively, and continuously detecting network attacks and malicious hosts. FlowHacker was evaluated with a synthetic dataset (ISCX) and a real dataset provided by Vodafone Portugal. It was possible to conclude that the system was able to detect and classify attacks under traffic analysis. However, it

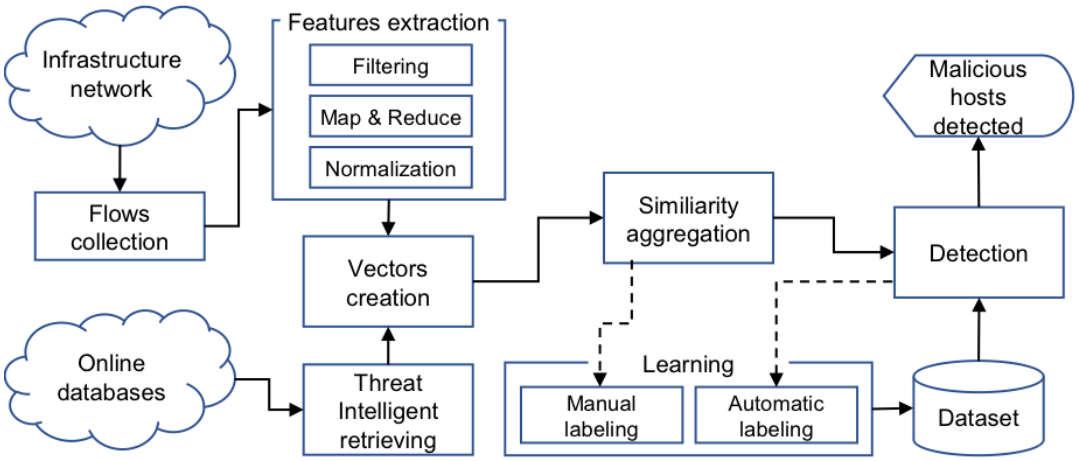


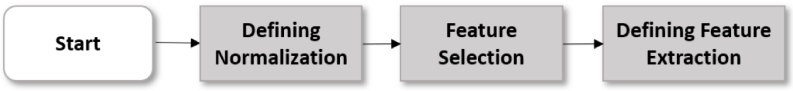
Figure 1: FlowHacker architecture [59]

also generated some FP and missed some malicious flows.

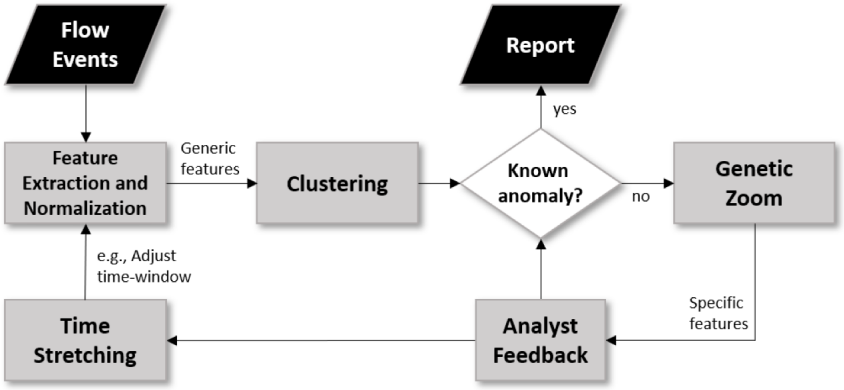
OutGene [25] is an approach to assist human analysts in pin-pointing malicious clusters, introducing the notion of *genetic zoom*, that consists of using a genetic algorithm to identify the features that are more relevant to characterize a cluster. This approach is an online network-based IDS based on network flows. OutGene analyses data on different time-windows. The approach is divided into two phases (Figure 2): pre-runtime is where the user does generic features definition, extraction and normalization and runtime where detection occurs, the process starts by generic features extraction and normalization, the extracted features are given as input to the *K-means clustering algorithm*. If clustering produces outliers observed before, it is reported. If it is a new outlier manual intervention by a human analyst is required (using genetic zoom). Experimental evaluation was made with datasets from LANL and from an extensive military infrastructure. During the evaluation, *genetic zoom* proved to be useful to understand which subset of features were relevant to differentiate the cluster and *time-stretching* allowed to detect attacks independently of their pace.

FlowHacker [59] and OutGene [25] aim to detect attacks with network flows. Gonçalves *et al.* [35] and FlowHacker [59] have the same features extraction framework, both with semi-automatic cluster labelling. OutGene distinguishes from the others because processing is not done on a daily basis and it has *genetic zoom* that helps in outlier explanation.

Valente proposes a new hybrid system [62]. It collects information from Windows logs and network flows. The approach uses an ensemble of clustering algorithms (K-Means, Agglomerative, LOF, and DBSCAN) to detect malicious behaviour from entities in a specific time-window. The work combines the results of four different clustering algorithms to improve accuracy and precision. LOF was applied to DBSCAN to attribute a score to all entities in outliers clusters. Clustering parameters for all algorithms used were chosen using *Elbow Method*. To perform clustering ensemble, the authors used clusters intersection, i.e. consider outliers classified by all algorithms. The criteria to identify an outlier was to consider all entities isolated in a cluster by *K-Means clustering* and Agglomerative and the entities detect as outliers by DBSCAN. Experimental evaluation was made with an artificial dataset from CSE-CIC-IDS 2018 and a real dataset obtained from the Portuguese Army. For network flows results, the best performance was an accuracy of 77%, with five missed attacks. The results based on Windows logs achieved an accuracy of 96%. However, attacks with more than one machine (e.g. Bot Attack) had a very low probability of being detected. When combining both events and flows, he was unable to detect a single anomaly. DynIDS [26] presented framework with dynamic port selection based on most used, less used and uncommon ports. They also applied three clustering



(a) Pre-runtime Phase



(b) Runtime phase

Figure 2: Fluxograms of the two main phases of OutGene [25]

algorithms (K-Means, DBSCAN and Agglomerative) and got a reduction of FPR. DynIDS was an extension of the work made in [62] and got promising results on detecting attacks. They studied multiple approaches regarding dynamic port features. They achieved better results with DYN3_100, where 33 ports were the most used ports, 33 the less used and 33 the more uncommon ports.

Valente [62] diverge from previous works by analysing both logs and network flows. It also uses more than one clustering algorithm for outlier detection with dynamic features selection, while other works are based on just one clustering algorithm with static features. DynIDS [26] is the first work to use dynamic features. In all previous work detection of attacks made by more than one machine was very unlikely. Neither of these works focus entities cluster changes, i.e., changes from one cluster to another in two followed time-windows.

3.2 Unsupervised Intrusion Detection Systems

The previous section presents several works that do intrusion detection based on clustering, which is a form of unsupervised learning. This section presents a few others that also use unsupervised learning but not clustering.

AI^2 [63] is an analyst-in-the-loop security system, where the system has four key features: a big data behaviour analytics platform, an ensemble of outlier detection methods, a mechanism to obtain feedback from security analysts and a supervised learning module. They develop an *Active Model Synthesis* approach where density-based, matrix decomposition-based, and replicator ANN are used to modelling the join behaviours of different entities within a big raw dataset. The analysis is done with log generated from web-scale platforms. The system has four main components: extensive data processing system (quantify the behaviours of different entities, and compute raw data), outlier detection (learns a descriptive model from the data *via* unsupervised ML), a feedback mechanism (incorporates analyst input through a user), and supervised ML module (learns a model that predicts whether a new incoming event is *normal* or *malicious*). Outlier score interpretation is made by projecting all scores into the same space, interpretable as probabilities. AI^2 evaluation was made with real-world dataset consisting of 3.6 billion log lines. The result shows that the system learns

to defend against unseen attacks: as an increasing trend, improving by 3.41 times concerning a state-of-art unsupervised anomaly detector, and reducing FP by more than five times.

Marchetti *et al.* [50] propose an approach for automatic defence aimed at early detection of Advanced Persistent Threats (APT)[1] in large network systems. The goal of the framework is to detect the few hosts that show suspicious activities, allowing security analysts to be more effective because they can focus their competence and attention on a limited number of hosts. They focus on traffic information that can be easily collected and to deal with high volumes of traffic efficiently, they extract and analyse *flow records*. The framework focuses on individual host comparative statistics concerning their past behaviour and with respect to towards internal hosts of the same organization. The main steps in the framework are flow collection and storage, features extraction, features normalization, computation of suspiciousness scores and ranking. In the computation of suspiciousness scores the approach considers three points in the features space: the values of the feature of the host at the present time (t_i), the centroid of the values of the features of the same host in a *historical window* of size W, that is, between t_{i-1-W} and t_{t-1}, and the centroid of the values of the features of all host at time t_i. To perform the comparison of the movements and positions of the internal host they use as measures: normalized distance of an internal host to the centroid of the features space, percentage increment the magnitude of movement, and unlikelihood the direction of the movement (respecting the movements of all internal host). The effectiveness of the proposed solution has been proved by implementing a prototype that is deployed in a real large network environment. The proposed approach can analyse about 140 millions of flows related to approximately 10,000 internal hosts in about 2 minutes. Experimental results demonstrate the ability of the framework to identify burst and low-and-slow exfiltration. Marchetti *et al.* [50] introduce the concept of Historic window, where is presented the past behaviour of the host in the network.

AI^2 [63] and Marchetti *et al.* [50] are IDSs that do not use clustering for outlier detection. They apply different unsupervised ML approaches to identify threats in logs and net flows. According to the authors, AI^2 [63] describes good results in a real-world dataset, however, the authors do not provide numeric values about the results, presenting only qualitative measures. AI^2 [63] needs external intervention in Supervised ML module. Marchetti *et al.* [50] only uses three features for APT identification and do not identify the route cause. This approach may raise some false alarms due to new software applications.

There are a few other works that fall in this category of unsupervised learning but not clustering: Cinque *et al.* used entropy to infer deviations from a baseline [19]; DeepLog was inspired in natural language processing and interprets logs as elements of a sequence that follows grammar rules [28]; Kitsune uses an ensemble of neural networks called autoencoders to differentiate between normal and abnormal traffic patterns collectively [54].

3.3 Change Detection and Series Analysis

Data changes over time. These changes may reveal patterns, groups or similar values, properties, and entities. Answering to: 'Why does the data change this way?', 'Are there regular patterns?' and 'Have any events caused large scale changes?', it could be useful to detect attacks made by more than one machine or where new clusters are created in two followed time-windows.

[1] APT's are human-driven infiltration, perpetrated over long periods of time, customized for the targeted organization after some intelligence analyses, possibly on open sources, and even leverage unknown exploits to infiltrate vulnerable systems [50].

3.3.1 Sequential Pattern and Time-Series Analysis

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. The following approaches present solutions to detect data changes in time.

He *et al.* [41] present a method for abrupt dynamic change detection of correlated time series. It is possible to detect abrupt dynamic change without being affected by noise and time-windows size. They based their approach on detrended fluctuation analysis by computing the scaling exponent in multiple data segments. Finally, they calculate the variance of the scaling exponents and got the time-instant of abrupt change. This method is suitable to be in various fields where we have complex behaviours characterized by long-range power-law correlations.

In a survey about time-series clustering [3], time series are described as dynamic data because the values of its features change as a function of time. Time-series data are often high dimensional which makes handing these data difficult for many clustering algorithms. The authors classified time-series clustering into three categories according to analysed data. Whole time-series clustering is performed in discrete objects, where objects are time-series. Subsequent clustering is considering segments from a single long time-series. Timepoint clustering is a clustering of time points based on their proximity, not all points are assigned to clusters. Time-series can be clustered based on their shape (shapes are matched as well as possible), features (time-series converted in a vector of lower dimensions) or model (time-series is transformed into model parameters and then a suitable model distance). According to the authors, time-series clustering main components are representation methods (dimension reduction), similarity and dissimilarity, clustering prototypes and clustering algorithms. Time-series representation is focused on speed up the process and reduce execution time or on the accuracy of the representation method. Similarity and dissimilarity have to deal with problems such as noise, amplitude scaling, offset translation, longitudinal scaling, linear drift, and temporal drift. The most popular similarity measures are Euclidean distance and DTW. Clustering prototypes consist of minimizing the distance between all time-series in the cluster and its prototype. Efforts have been taken on presenting a distance measure based on raw time-series. Authors conclude saying that the unique characteristics of time-series data are barriers that fail most conventional clustering algorithms.

Bornemann *et al.* [11] approach help to detect abnormal changes using time series clustering. Clustering is used because it does not require domain knowledge or ground truth, both of which are usually not available in an exploration scenario. The *change cube* is an attempt to create a general data model that can represent both changes in data as well as changes in the schema. The change-cube is a representation that stores changes as quadruples of the following format: (t,id,p,v), which describes that at a certain point of time t an entity with id id was changed in the property p which from them has the new value v. The framework (Figure 3) groups change cubes by a user-supplied criterion and subsequently aggregates changes in the data to numerical time series by distributing change records into buckets and counting the number of changes records in each bucket. The time series are then subsequently transformed, and clustered using user-defined transformation and clustering algorithms. Using the framework, Bornemann *et al.* were able to discover previously unknown patterns in the voting behaviour of the users of IMDB. In the dataset consisting of changes made to Wikipedia infoboxes, the framework enabled to discover events that caused automated changes for several templates.

Bornemann *et al.* [12] have an application of [11] where clustering approaches were applied to discover groups of similar changes. To enable exploration of data, they define a set of exploration primitives and then implement a change exploration system described in [12]. They defined as

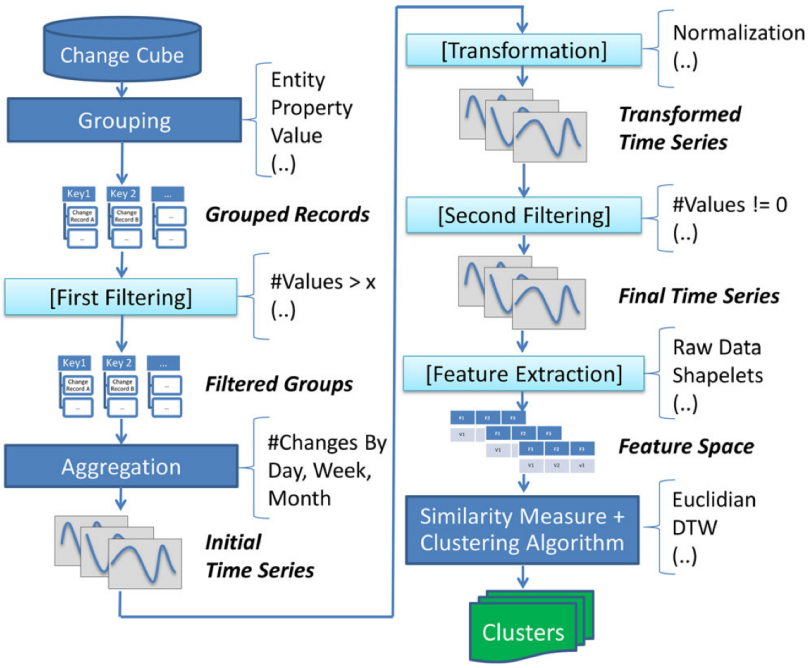


Figure 3: Transforming a change cube into clustering based on temporal change [11]

exploration operators on sets of *change cube*: sort according to specific criteria, slice to reduce *change cube* dimensions, split to create a set of *change cubes*, union of two *change cubes*, and rank/prune to focus on *change cubes* of interest. The framework was tested in public datasets with access to their history, such as Wikipedia, DBLP, IMDB, and Music Brainz.

Bornemann *et al.* in [11] and [12] apply some concepts and ideas for time-series clustering explained in [3], in order to detected similar and abnormal changes. Although this kind of analysis is useful when the subject is a list of numbers, it cannot be used on nominal values, such as symbols.

Sequential Pattern Mining is studied in [33]. Pattern mining is defined as discovering interesting, useful, and unexpected patterns in databases. Although it was originally designed to be applied to sequences (list of nominal values), it can also be applied to time series (ordered list of numbers) after converting time-series to sequences using discretization techniques. In sequential pattern mining, there is always a single correct answer, usually the task is enumerating all patterns that have support no less than the minimum support threshold set by the user. All sequential pattern mining algorithms explore the search space of sequential patterns by performing two basic operations called *s-extensions* and *i-extensions*. For example s_a is an *i-extensions* of s_b if $s_a = \langle\{a\}\rangle$ and $s_b = \langle\{a, b\}\rangle$, and s_c is an *s-extensions* of s_d if $s_c = \langle\{a, b\}\rangle$ and $s_d = \langle\{a, b\}, \{c\}\rangle$. A Breath-first search algorithm, such as Generalized Sequential Pattern, first scans the database to find frequent 1-sequences[2], then it generates 2-sequences by performing *i-extensions* and *s-extensions* of 1-sequences, then 3-sequences and so on until no sequences can be generated. The depth-first search algorithm, such as Spade or PredixSpan, starts from the sequences containing single items and then recursively performs *i-extensions* and *s-extensions* with one of these sequences to generate larger sequences. Then, when a pattern can no longer be extended, the algorithm backtracks to generate other patterns using other sequences. The previously described algorithms may generate patterns not appearing in the database. To address this problem pattern-growth algorithms, such as FPGrowth or PrefixSpan, only consider the patterns appearing in the database, performing database scans. In general, sequential pattern mining algorithms differ in whether they use a depth-first or breadth-first search, the type of database representation that they use internally or externally, how they generate or determine the next patterns

[2]Sequential patterns containing a single item

to be explored in the search space, and how they count the support of patterns to determine if they satisfy the minimum support constraint. Pattern mining algorithms assume that sequence databases are static if the database is updated, algorithms need to be run again from scratch to obtain the updated patterns. A limitation of traditional sequential pattern mining studies is that they solely focus on the discovery of items that are positively correlated in sequence databases, mining negative sequential patterns is more difficult than mining only positive patterns as the search space becomes larger.

Han *et al.* [40] proposed that to detect collective outliers in temporal sequences, one method is to learn a Markov model from the sequences. A subsequence can then be declared as a collective outlier if it significantly deviates from the model.

Allahdadi *et al.* [4] propose to explore the temporal usage behaviour of the network applying various types of HMM. Anomalous trends were signalized by their likelihood[3]. In the likelihood series the least frequent transitions are more likely to be among the anomalous instances, hence they can be indicated as outliers. On the other hand, HMM indicators monitor the characteristic of each data instance concerning the model parameters which determine whether there is any sign of abnormality associated with data points and assert the potential source of the problem from the model parameters' viewpoint.

3.3.2 Cluster Monitoring

Clustering 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. One basic approach is to apply a clustering algorithm in all centroids of clusters formed over multiple time-windows. Centroids of two different time-window in the same cluster would be considered as being the same cluster. This approach has some limitations to detect more complex 'clusters movements', e.g., a cluster splitting in two. The following approaches differ from previous (Section 3.3.1) by presenting a closer look in algorithms to compute clusters correlation in multiple time-windows.

MClusT [55] is a framework that uses bipartite graphs and conditional probabilities to monitor transitions according to the defined taxonomy. The framework can detect birth, death, merge and survival of clusters. The key concept is mapping, i.e., the process of discovering the exact matches between clusters. In this framework, the mapping process explores the concept of conditional probability. These probabilities are computed for every pair of possible connections between clusters obtained at different timepoints and they represent the edge's weights in a bipartite graph. The evaluation, performed in datasets from *Banco de Portugal* and *Instituto Nacional de Estatistica*. They obtained results that vary according to the select clustering algorithm. The framework proved to be efficient and able to provide effective diagnosis of the transitions experienced by clusters.

Landauer *et al.* [45] based their approach on cluster evolution techniques where the same elements are observed and clustered over time. They use cluster evolution in order to process log data in a streaming manner rather than being limited to fixed-size data sets. Two clusters were considered similar if the majority of the elements contained in C' would have been allocated to cluster C if they had been used for the generation of cluster map C. Thereby, occurrences are not only used for creating the cluster map of that time step, but are also allocated to the clusters from the cluster maps preceding and succeeding that map. According to [45], between two time-windows may occur: survival, split, absorption, disappearance, and emergence. Survival is when one cluster survives and transforms into one similar cluster. Split corresponds to a cluster division into parts, and they share

[3]The probability of observing a sequence, assuming that it is generated by an HMM

a minimum amount of similarity with the original cluster. Absorption is when two or more clusters merge into one cluster, and individual parts share a minimum amount of similarity with the resulting cluster. A cluster disappearance is when it does not exist in the next time-window. Emergence is when the cluster does not exist in the previous time window. Some measures were applied to take advantage of considered clustering model: growth rate (absolute difference between the member sizes of two consecutive time steps), change rate (relative difference between the cluster allocations of the lines from the former time step with respect to the total number of lines that were processed in the corresponding time window), stability rate (fraction of appeared, disappeared, merged and split members between two consecutive time steps), novelty rate (fraction of newly appeared members), and split rate (fraction of members that were split). 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. Their approach was evaluated in a semi-synthetically generated log file where anomaly detection showed promising performances when applied to the evolution of individual clusters. The evaluation performed in real log file revealed anomalies corresponding to system behaviour changes. Authors identify a high amount of false-positives in evaluation results of their framework. Our work, C2BID applies a classification system similar to the one presented in [45] to construct the history path. We used the binomial cluster and entity instead of log and entity. C2BID decreases the FP because it uses a different outliers detection method and studies NetFlows instead of logs.

Landauer *et al.* [45] proposal, contrarily to MClusT [55], 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 correlating 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.

A summary of studied articles related to IDS is present in Table 1

Table 1: Essential literature summary about IDS

Paper	Year	Type	Approach	Dataset	Conclusion
UNIDS [18]	2012	Network-based	It uses an unsupervised outliers detection approach based on Subspace Clustering and Multiple evidence accumulation.	KDD99 and Real-world dataset	It was showed that it outperforms traditional approaches for outliers detection.
Beehive [67]	2013	Host-based	Analyses a very large volume of log. Clustering and PCA used to detect outliers.	EMC	Detection of malware infections and policy violations that went otherwise unnoticed
Gonçalves *et al.* [35]	2015	Host-based	Logs analysis to detect misbehaviour. Combine supervised and unsupervised (clustering with EM) ML techniques.	Vodafone Portugal	The output of the process was not accurate enough to take automatic actions. It extracted relevant information from the logs that otherwise were not directly observable.
AI^2 [63]	2016	Host-based	Log-based system where density-based, matrix decomposition-based and replicator ANN are used to modelling the join behaviours of different entities within a raw big data set.	Real-world dataset	System is able to learn and defend against unseen attacks.
Marchetti *et al.* [50]	2016	Network-based	Automatic and early detection of APT. Extract and analysis flow records. Comparative analysis of past individual behaviour of each host and other hosts. Computation of suspiciousness scores are performed using distances in features space.	Real-world dataset	Framework can identify burst and low-and-slow exfiltration.
FlowHacker [59]	2018	Network-based	Malicious traffic and malicious hosts are identified by flows inspection, without requiring either previous knowledge about attacks or traffic without attacks. Outiliers are identified using K-Means clustering.	Vodafone Portugal and ISCX	The system was able to detect and classify attacks under traffic analysis. It generated some false positives and missed some malicious flows.
OutGene [25]	2018	Network-based	K-means applied to network flows analysis. Used a genetic algorithm to identify the features that are more relevant to characterize a cluster	LANL and Portuguese Army	*Genetic zoom* proved to be useful to understand which subset of features were relevant to differentiate the cluster.
DynIDS [26]	2020	Network-based	It collects information from network flows. Applies three clustering algorithms: K-Means, Agglomerative, DBSCAN. Several approaches were tested for dynamic features regarding the Ports.	CSE-CIC-IDS 2018 and Portuguese Army	Best results with DYN3_100. FPR reduction due to the combination of different clustering algorithms and dynamic features.

4. Proposed Solution

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 behaviour is characterized using features extracted from network flow data, e.g., obtained in routers with the NetFlow feature [20, 21, 65], so this is a network-based intrusion detection approach.

Features are extracted from network flows in multiple time-windows, $W = \{w_1, w_2, ..., w_n\}$. Then, the features are applied to a clustering algorithm (K-means) to group all IP according to their characteristics. Afterwards, it is created a relation between different time windows by correlating their clusters. Finally, the clusters' changes are studied to detect uncommon occurrences, which may point to an outlier.

Figure 4 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.

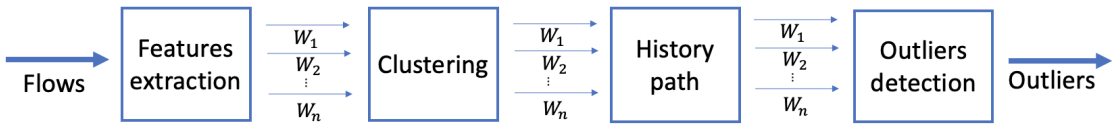


Figure 4: Sequence of steps in C2BID

4.1 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 misbehaviour 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 (Table 2), 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.

Table 2: Fixed features, with source IP as the aggregation key

Feature	Description
IPContacted	Number of IPs contacted by the host
IPFlows	Number of flows where the host is the source
PortUsed	Number of different source ports used by the host
PortContacted	Number of different destination ports contacted by the host
TotLenRcv	Sum of total packet length received by the host
TotLenSent	Sum of total packet length sent by the host
AvgSent	Average sent packet size
SentRatio	Ratio of the number of packets sent and their duration

We also consider a set of *dynamic features*, or port-based features, following an idea recently proposed by Dias *et al.* [26]. 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 [23], so the number of features would be excessive. Therefore, for each period $\mathcal{T}_a$ the approach picks $\mathcal{N}_p$ ports 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 number of ports, $\mathcal{N}_p$, is split in: the first third is ports that appear in more flows, the second third is ports that appear in fewer flows, and the last third is ports used by fewer machines. Having this in mind, the limit of search space for the most used ports and uncommon ports is the range of System and User Ports (0-49151). The search space for less frequently used ports were limited to System Ports (0-1023), only the range more prone to probes and scans. For each IP in $\mathcal{T}_a$ is extracted the port-based features regarding all ports consider in $\mathcal{N}_p$ and used by the host.

The result of the feature extraction step is one *vector of features* for each host. In the evaluation we consider only hosts that are internal to the organization. We consider in each case, that 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).

4.2 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 behaviour 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 [7]. 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 distance is used to determine the distance between two points.

K-means is known to produce good results in the context of intrusion detection [25, 59, 67, 26], as in many other areas [44, 60], and this is also our experience in previous works, so we used it for clustering.

Normalization is performed by scaling each feature to a range between zero (min) and one (max), using Equation 4.1. By normalizing all features extracted, we avoid discriminating features in relation to others.

$$X_{standard} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.1}$$

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 a cluster of each time window.

4.3 History Path

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 w_i, 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 behaviour of a host active in 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.* [45]. Two clusters are considered similar if more than a certain overlap threshold o_t (e.g., 50%) of the elements contained in 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 4.2). 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 4.2, 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.

$$overlap(C, C') = \frac{|\left(R_{curr} \bigcap R'_{prev}\right) \bigcup \left(R_{next} \bigcap R'_{curr}\right)|}{|R'_{curr} \bigcup R'_{prev} \bigcup R_{next} \bigcup R_{curr}|} \tag{4.2}$$

To better illustrate the notion of history path, we present an example in Table 3. The example considers a small dataset of ten hosts, $\mathcal{T}_a = 1$ day and time-windows of 120 min, i.e., $\mathcal{W} = \{w_{120min}\}$. As the traffic starts after 8 a.m. and finishes before 8 p.m., each history path contains only 6 cluster identifiers (for 8 a.m., 10 a.m., etc.). There are 10 history paths, one per host. Needless to say, in a real case there would be 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 behaviour as it changes of cluster two times in ways that the others do not: 12 p.m.$\rightarrow$ 2 p.m. and 2 p.m. $\rightarrow 4pm$. 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 3: 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*	140.0	3.0	3.0	3.0	3.0
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

4.4 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 [9] or those that are farther from all the other data points than a given a threshold [69, 70].

In this work we use an *outlier detection algorithm* to find outliers, not clustering. 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 5):

- Calculating the likelihood of each host doing a certain set of changes, considering 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 aim 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.

4.4.1 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, bearing in mind all the transitions in that period

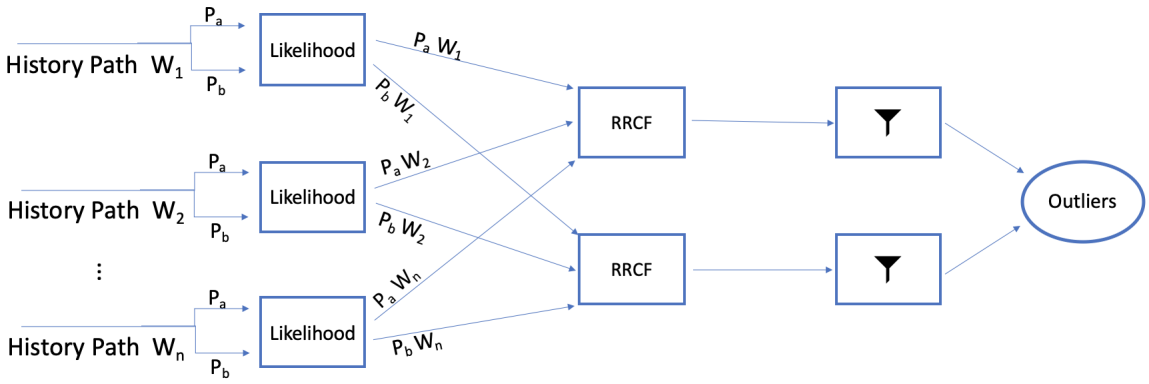


Figure 5: Outlier detection framework

$\mathcal{T}_s$. 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 host 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 [26]. 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, with that the matrix became a relative frequency 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 4.3), where x' is the new likelihood value, x is the old likelihood value, $Q \in [0, 1]$ is x relative position in the matrix, and $\{m, b\}$ are linear function parameters. Ideally, when $Q = 0.5$ the value should remain the same, i.e., $mQ + b = 0$ and for $Q > 0.5 \rightarrow mQ + b > 0$, but it can be adjusted.

$$x' = x \times (1 + mQ + b) \tag{4.3}$$

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., $P_{entity} = (P_{w_1}^{entity}, ..., P_{w_n}^{entity})$.

4.4.2 Robust Random Cut Forest

RRCF [38] 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 metric used is the Euclidean distance, the data are normalised by 4.1 as in K-means application.

Outliers are the entities with the highest metric value. For outliers' detection, a decision rule is defined based on the lowest percentile from which an entity (IP) 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 when running multiple times, it is possible to mitigate this limitation.

4.4.2.1 RRCF choice

To decide which outlier detection algorithm to use, we tested several algorithms available in the literature and libraries: DBSCAN [29], Isolation Forest [48], SVM [22, 2], OPTICS [5], LOF [15], Elliptic Envelope [56, 34], and RRCF [38]. The criteria used to select the most appropriate algorithms were:

1. the ability to identify an attacker in a (labelled) dataset as an outlier;
2. minimum number of hyper-parameters and their simplicity of configuration (as complexity leads to errors);
3. low number of false positives.

The labelled dataset contains network traffic flows of nine days [17]. Most of the algorithms considered were available in the Scikit-learn; the exception was the RRCF implementation [1]. The setting of the hyper-parameters was made using Scikit-learn functions or empirically. Dynamic parameter setting algorithms [58] were used for DBSCAN and OPTICS.

To illustrate our choice was calculated the Precision (PREC), Figure 6, as well as F-score, Figure 7. It is possible to see in both cases that RRCF is the one that produces less FP as well as has better success in detecting all attacks.

Regarding the hyperparameters choice, both DBSCAN and OPTICS revealed an high sensitivity to them, so these options were excluded. In RRCF, the default parameters of the library were used (*number of trees* = 128 and *tree size* = 100). However, we tested variances in them and got no significant changes in the results. From the algorithms evaluated, we selected RRCF as it performed much better than the rest.

4.4.3 Filtering

The RRCF algorithm does not distinguish between high and low likelihood values. Having in mind that the objective is to find IPs that take an unusual path (high likelihood), it is necessary to remove those that the RRCF has identified and do not have an unusual path (low likelihood). This

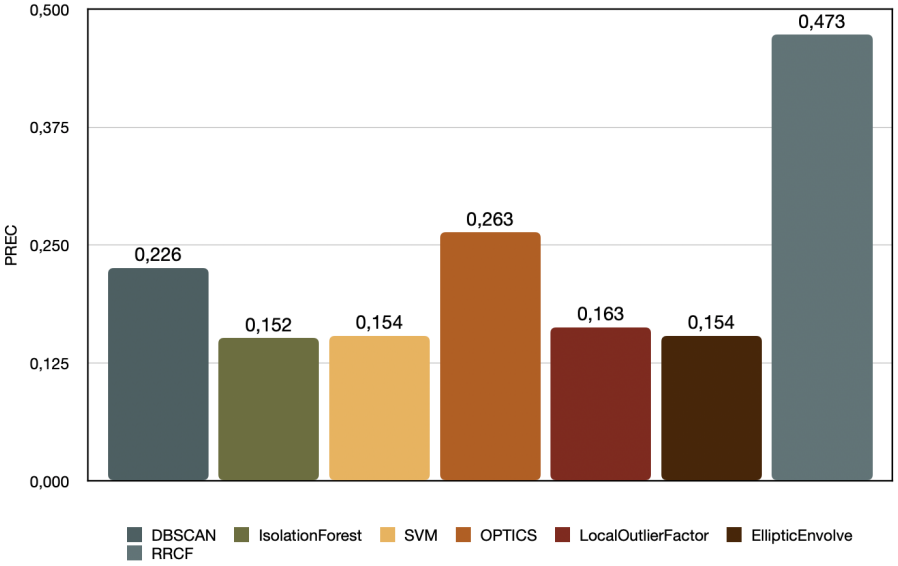


Figure 6: PREC comparison in tested algorithms

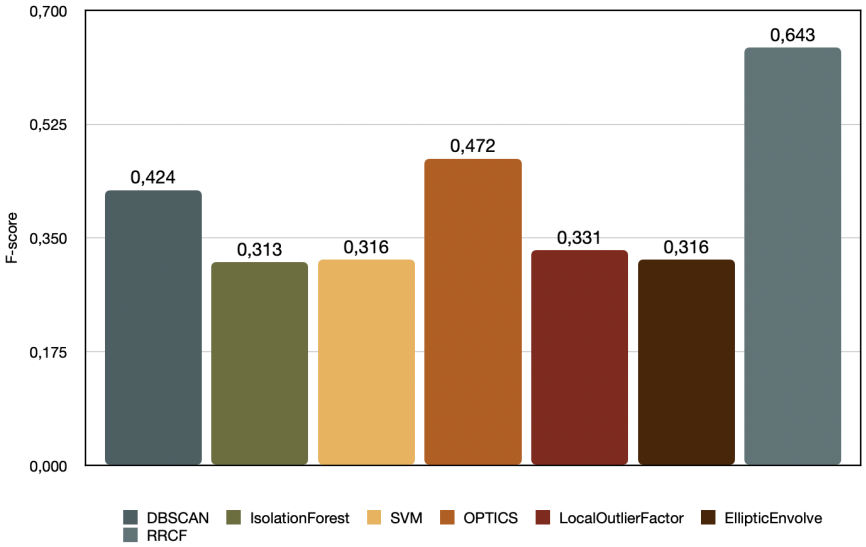


Figure 7: F-score comparison in tested algorithms

process uses two mechanisms: a filter for lower values and a whitelist. The whitelist is according to network architecture. It may include DNS, web servers, or others, which typically behaves differently from the majority of hosts. The lower values filter is defined based on a decision rule, k. The value of k is used in 4.4 to calculate the percentile c for each time-window, 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.

$$C = \frac{n - k}{n} \tag{4.4}$$

4.4.4 Detection Example

In this section, we present an example to illustrate better how Likelihood is computed.

We consider time windows $\mathcal{W}$=$\{w_{120}\}$ of 120min, an analysed period of time $\mathcal{T}_a$ with 10 hours and sub-divisions $\mathcal{T}_s$ with 10 hours, i.e., we do not apply sub-divisions in $\mathcal{T}_a$ to keep the example simple. In relation to equation 4.3 we choose a slope, m, of 0.004 and a constant, b, of -0.2. Figure 8 plots the line $y = 0.004 \times x - 0.2$ to better understand the weights added by equation 4.3.

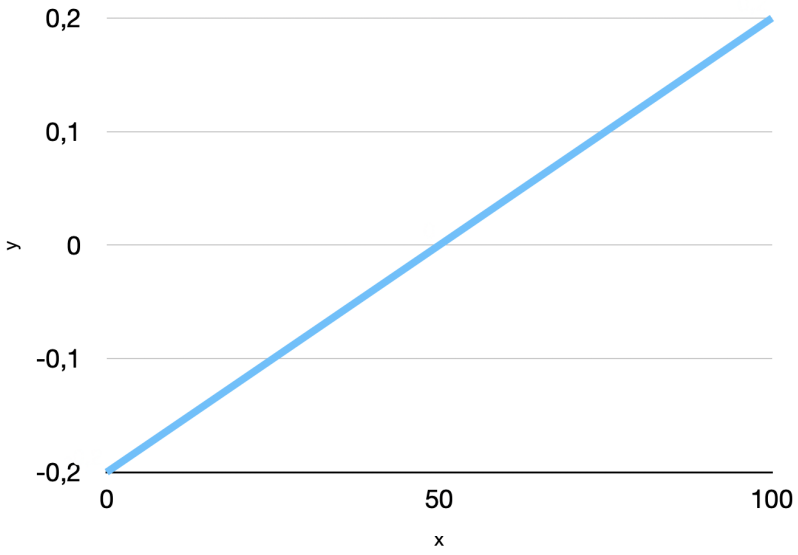


Figure 8: Plot of $y = 0.004 \times x - 0.2$

Next, we explain step-by-step all computations done in likelihood module:

- As explained in Section 4.3 a history path for each host is produced (Table 4). The example considers 10 history paths, one per host. The first column of the table shows network IPs; the first row shows the start time of each time window; each cell contains the cluster-ID where an IP was assigned in that time window $(2.0, 3.0, 140.0)$. Clusters-IDs are random generated. *nan* means that the IP is not active in that time window.

- Compute inactive time for each host. Inactive time is the number of *nan* against the total number of widows, as in expression 4.5. Table 5 has the values of inactive time refereeing each host.

$$\frac{\Sigma_{\sharp nan}}{\Sigma_{\sharp w_{120}}} \times 100 \tag{4.5}$$

- All hosts with an inactive time higher than 60% are removed from the analysis. In this particular case, 172.31.69.6 is removed. Since we only consider one-time window, we cannot estimate the

Table 4: Set of history path

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

Table 5: Inactive time for each host

Hosts	Inactive time (%)
172.31.69.23	0
172.31.69.17	0
172.31.69.14	0
172.31.69.12	16.67
172.31.69.10	0
172.31.69.8	16.67
172.31.69.6	66.67
172.31.69.26	0
172.31.69.29	0
172.31.69.30	0

likelihood from another time window. The excluded host is analysed, in the active period, separately from others. If this host is in a unitary cluster, it will be considered an outlier. The set of history paths is updated as in Table 6.

Table 6: Actualized set of history path

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
172.31.69.26	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

- From Table 6 a transition matrix is created with the absolute frequency of each transition. Transitions involving *nan* are not considered. Table 7 is the transition matrix for this example.

Table 7: Transition matrix with absolute frequency

Clusters	2	3	140
2	10	1	0
3	1	18	5
140	0	6	0

- The sum of all columns and rows is calculated and the matrix (Table 7) is normalized, with this it becomes a relative frequency matrix (Table 8).

Table 8: Transition matrix with relative frequency

Clusters	2	3	140
2	0.24	0.024	0
3	0.024	0.44	0.12
140	0	0.14	0

- The transformation $f(x) = -log(x)$ is applied. This transformation has only computing purpose, since in a real-life scenario we have at least hundreds of different clusters which means small values for their relative frequency causing underflow in future operations. Table 9 represents transition matrix after $f(x) = -log(x)$ transformation.

Table 9: Transition matrix after $f(x) = -log(x)$ transformation

Clusters	2	3	140
2	0.61	1.61	0
3	1.61	0.35	0.92
140	0	0.85	0

- Values in Table 9 are weighted according to their relative position. We have six possible values: $[0, 0.35, 0.61, 0.85, 0.92, 1.61]$, the relative position of each value is given by the reason of its position in the vector and the total number of values, e.g., relative position of 0.61 is 50%, since it is the third number in six. For all values: $[0, 0.35, 0.61, 0.85, 0.92, 1.61]$ we have repetitively the following relative positions (value of Q in equation 4.3): $[16\%, 33\%, 50\%, 66\%, 83\%, 100\%]$. Equation 4.3 is applied to Table 9 resulting in Table 10.

Table 10: Weight Transition matrix

Clusters	2	3	140
2	0.610	1.932	0
3	1.932	0.326	1.040
140	0	0.900	0

- Missing values, i.e., transitions between a cluster and *nan* in Table 6, are obtained based on equivalent percentile. First, all values of transition matrix (Table 10) are sorted and duplicates values are removed, e.g., $[0, 0.326, 0.610, 0.900, 1.040, 1.932]$. Then, the value of each percentile is computed. In this example we will only consider quartile instead of per-

centile due to the small number of clusters used. Considering this, there are four quartiles: $[0, 0, 2445, 0.755, 1.263, 1.932]$.

In this example there are three missing values (Table 6):

- 172.31.69.12 between 08:00 and 10:00;
- 172.31.69.8 between 08:00 and 10:00;
- 172.31.69.12 between 10:00 and 08:00.

For 172.31.69.12 we have: three transitions $3 \rightarrow 3$ and two missing values, transition $3 \rightarrow nan$ and $nan \rightarrow 3$. Transitions $3 \rightarrow 3$ has value 0.326 (see Table 10), this corresponds to the second quartile. The median of all defined transitions is the quartile three, so the missing values will inherit the middle value of third quartile, 0.4995. Regarding 172.31.69.8 there are the following transitions: $nan \rightarrow 140$, $140 \rightarrow 3$, $3 \rightarrow 2$, $2 \rightarrow 3$, and $3 \rightarrow 3$. According to Table 10, the last four transitions have, respectively the following values: 0.900 (third quartile); 1.932 (fourth quartile); 1.932 (fourth quartile) and 0.326 (second quartile). The median of all quartiles is 3.5, that is rounded to quartile 4. This way the missing value ($nan \rightarrow 140$) will be 1.486 (middle value of fourth quartile).

- Finally, for each host all transitions values are added, as in Table 11.

Table 11: Final Likelihood values

Hosts		Likelihood
172.31.69.23	$5 \times 0.610 =$	3.05
172.31.69.17	$5 \times 0.610 =$	3.05
172.31.69.14	$1.040 + 0.900 + 3 \times 0.326 =$	2.918
172.31.69.12	$0.4995 + 0.4995 + 3 \times 0.326 =$	1.977
172.31.69.10	$1.040 + 0.900 + 3 \times 0.326 =$	2.918
172.31.69.8	$1.486 + 0.900 + 1.932 + 1.932 + 0.326 =$	6.576
172.31.69.26	$1.040 + 0.900 + 3 \times 0.326 =$	2.918
172.31.69.29	$1.040 + 0.900 + 3 \times 0.326 =$	2.918
172.31.69.30	$1.040 + 0.900 + 3 \times 0.326 =$	2.918

Afterwards, Table 11 would be analysed through RRCF (Section 4.4.2) to find abnormal values. By visual inspection, it is possible to find twos hosts (172.31.69.8 and 172.31.69.12) susceptible to being marked with as an outlier by the RRCF. 172.31.69.8 with a much higher value than the others and 172.31.69.12 with a much smaller value. A filter (Section 4.4.3) would be applied in these hosts, and the one with a small likelihood value would be excluded (172.31.69.12). 172.31.69.8 would be a candidate to be marked as an outlier by C2BID.

This chapter explains the C2BID approach. An implementation of the approach relies on four modules: *Feature extraction* is responsible for generating the features used to characterize hosts; *Clustering* is used to group hosts with similar behaviour; *History Path* traces hosts behaviour along time; *Outlier Detection* works in three modules: first it is computed a metric for each host based in its History Path, second RRCF is applied to detect outliers, third outliers are filtered according to user-defined rules.

5. Evaluation

This chapter presents the evaluation of C2BID. First, we define our evaluation metrics and make a dataset characterisation. Then, the results regarding C2BID and other frameworks are compared.

To develop and implement C2BID for evaluation, we used Python (v3) [52]. Additionally, we used popular libraries such as Pandas [51] for data manipulation, and Scikit-learn [56] 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.

5.1 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) as hosts correctly classified as inliers; False Positives (FP) as hosts wrongly classified as outliers and False Negatives (FN) as 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):

$$PREC = \frac{TP}{TP + FP} \tag{5.1}$$

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

$$REC = \frac{TP}{TP + FN} \tag{5.2}$$

- F-Score – a global detection score:

$$FScore = 2 \times \frac{PREC \times REC}{PREC + REC} \tag{5.3}$$

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 privileged F-Score, which summarizes the overall performance.

5.2 Dataset Characterization

We used two datasets. The first, *CIC-IDS2018* [17], 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 behaviours were generated using B-profiles. Such a profile is designed to extract the abstract behaviour of a group of human users, encapsulating the host behaviours of users using various machine learning and statistical analysis

techniques. Malicious behaviour 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 six 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 12). 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 12: 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 [27]. Managing these flows can give us insights of misbehaviour 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.

5.3 Results With the Artificial Dataset

The counting of positives was done considering all IPs flagged by C2BID in each $\mathcal{T}_a$. The number of positives is calculated as the number of IPs flagged by C2BID in each time interval $\mathcal{T}_a$. This tends to increase the proportion of false positives in comparison to systems that count the positives for each time interval in $\mathcal{W}$. We use the parameters in Table 13. Only the internal IPs were analysed since those are the ones we want to protect and C2BID needs continuous communications 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, i.e., servers, taking into account every day in the dataset.

C2BID flagged some IPs that were not listed as malicious by the dataset authors (Table 14). 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

Table 13: 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 (4.3)
b	-0.2	Constant of Equation (4.3)
k	2	Filter parameter
-	99.6	RRCF sensibility

Table 14: Summary of the attacks (not listed in [17]) for the CIC-IDS-2018 dataset

Day	Victim	Attack
1	172.31.66.82	Brute Force
1	172.31.67.109	Brute Force
2	172.31.66.112	Brute Force
4	172.31.65.56	Brute Force
5	172.31.69.19	Brute Force
9	172.31.66.100	Brute Force

behaviour was suspicious, and we searched for the latter in public DBs of malicious IPs and found them. This confirmed that these were TPs, not FPs, so we started counting them as such. This process is illustrated in Appendix A.

We were able to identify all victims all days. Table 15 shows a summary of the results for the nine 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.

Table 15: Summary of the results in artificial dataset

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

5.4 Results With the Real-World Dataset

In this dataset, we applied the values in Table 13 to all parameters presented in Section 4, 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 had only between 8 am and 8 pm. The dataset has four servers included in the whitelist. Only the internal IPs were analysed.

The framework detects IPs that, during a time period, shows a notably different behaviour concerning the other analysed IPs. In this dataset, several types of attacks are co-occurring and involving the same IPs. In the case of one marked IP being in two attacks at the same time, its detection only counts as one TP. The same occurs with FP and FN.

The identified attacks were:

- slow port scan with 5s pace (attacker and victim);
- stealth dictionary attack RDP (attacker and victim);
- 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 SOC.

In summary, the alerts raised by C2BID corresponded to real threats or anomalies. 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.

5.5 Comparison With Previous Approaches

This section compares C2BID with three previous works in the area: OutGene [25]; DynIDS [26] and FlowHacker [59]. We selected these ones 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 for training data.

5.5.1 Comparison

All 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 13 and the $\mathcal{T}_a$ value (1 day).

For positive counting, we used the same method as before: the number of different IPs marked as outliers in each $\mathcal{T}_a$ (1 day). This counting method 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 with 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 9, 10 and 11 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 15). Figures 12, 13 and 14 represents the same for the real-world dataset.

We also evaluate OutGene, FlowHacker and DynIDS for the artificial dataset, only considering the attacks with labels given by the dataset authors (Table 12), i.e., all attacks identified in Table 14 were excluded. This change allowed OutGene, FlowHacker and DynIDS to achieved better performances in the artificial dataset. Figures 15, 16 and 17 show the results obtained. This way, it is possible to show that most attacks of Table 12 pass unnoticed to all compared approaches. Even not considering attacks from Table 14, C2BID had the best F-score, meaning a better relation between REC and PREC.

Next we will discuss each of the related approaches.

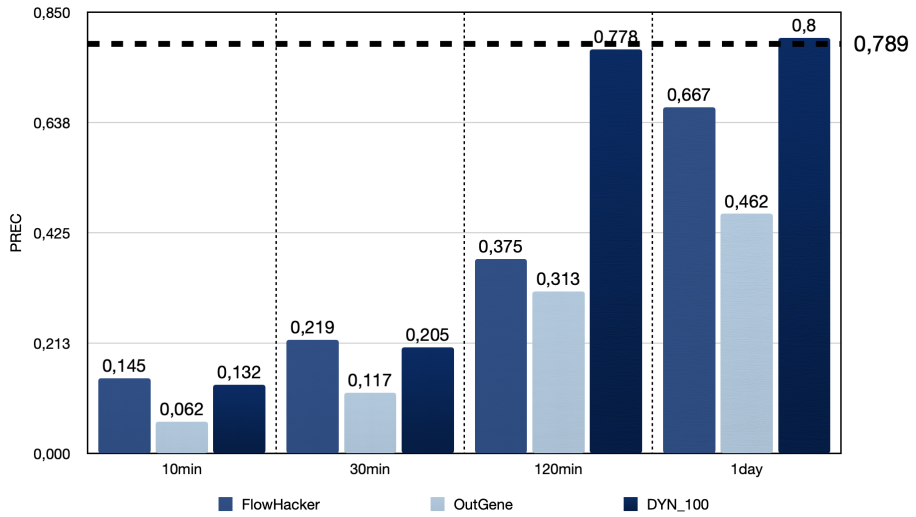


Figure 9: PREC comparison of OutGene, FlowHacker and DynIDS for the artificial dataset

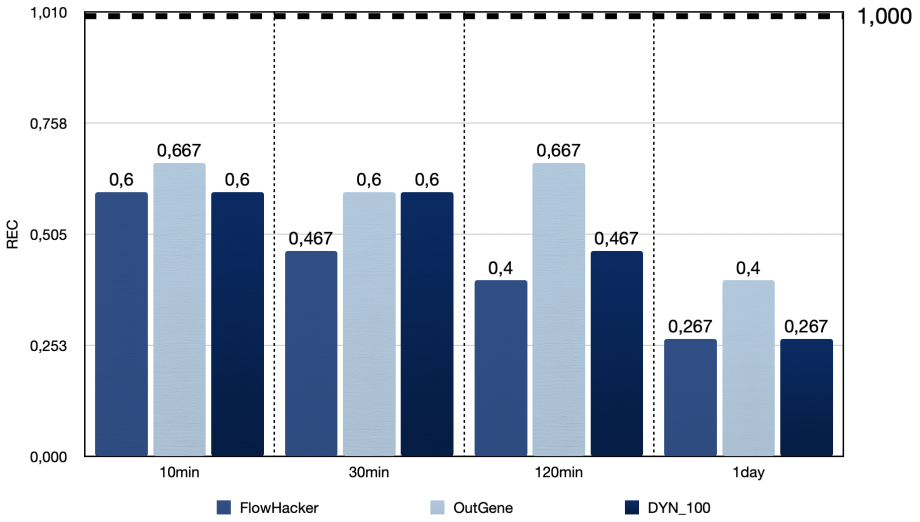


Figure 10: REC comparison in OutGene, FlowHacker and DynIDS for the artificial dataset

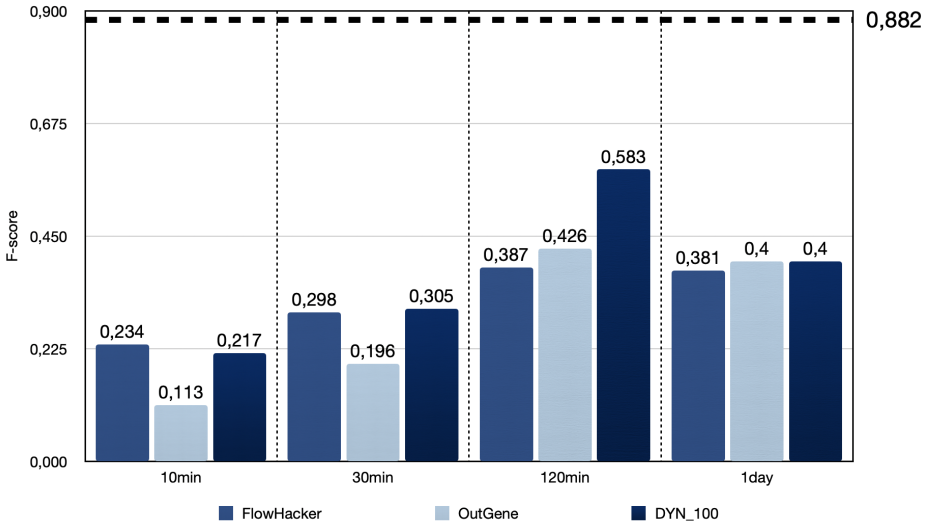


Figure 11: F-score comparison of OutGene, FlowHacker and DynIDS for the artificial dataset

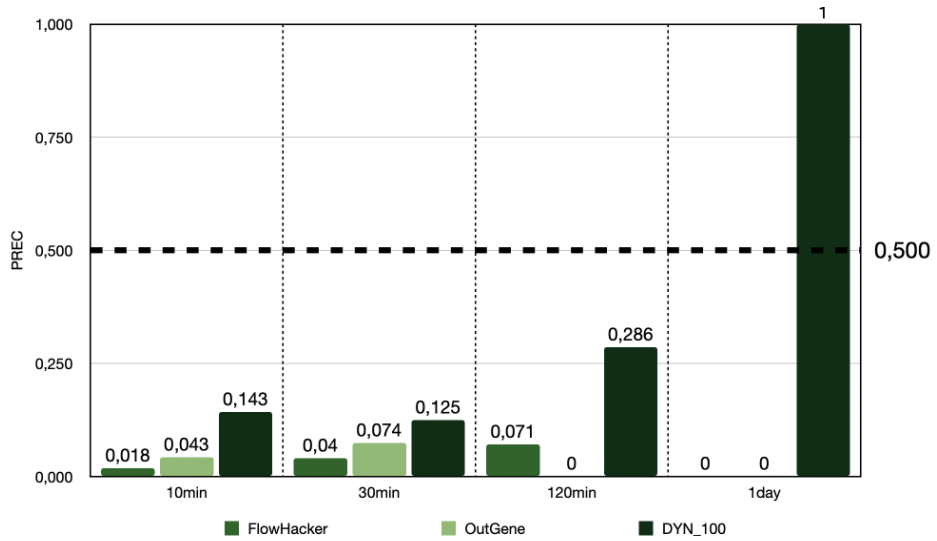


Figure 12: PREC comparison of OutGene, FlowHacker and DynIDS for the real-world dataset

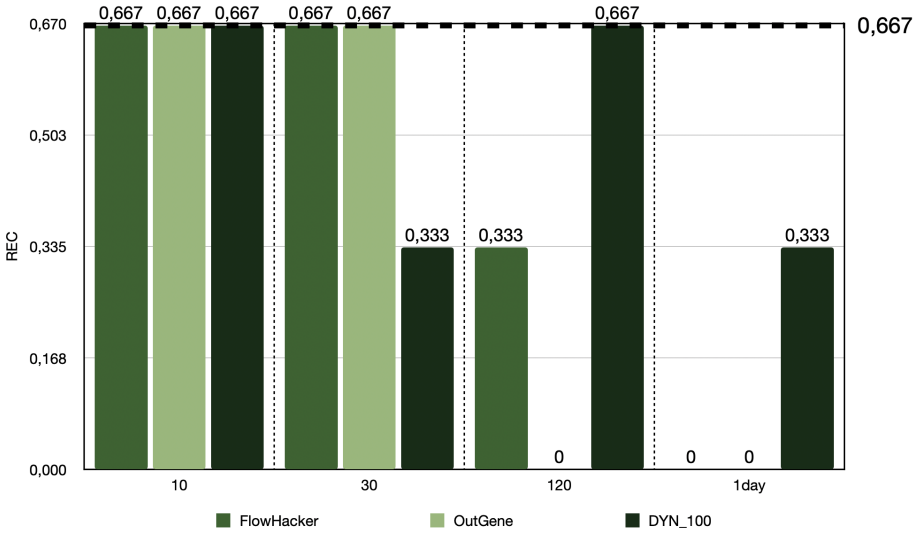


Figure 13: REC comparison of OutGene, FlowHacker and DynIDS for the real-world dataset

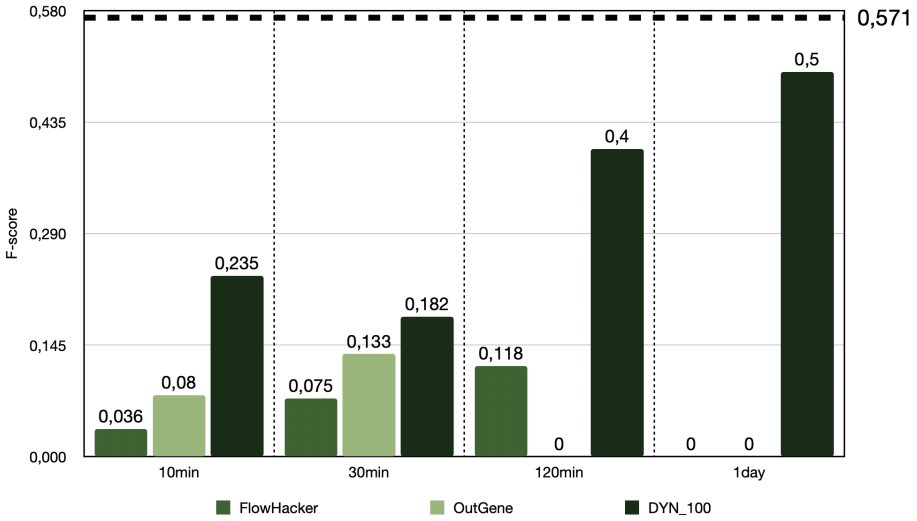


Figure 14: F-score comparison of OutGene, FlowHacker and DynIDS for the real-world dataset

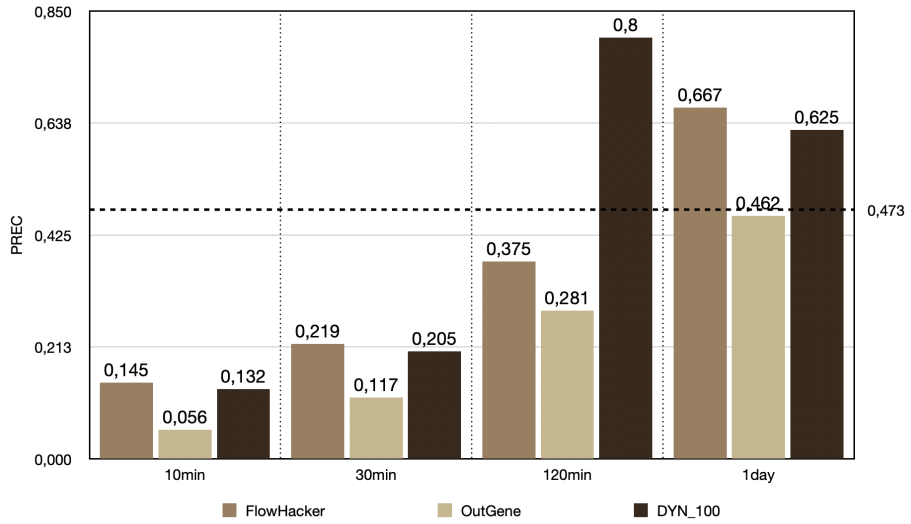


Figure 15: PREC comparison of OutGene, FlowHacker and DynIDS for the artificial dataset, without attacks in Table 14

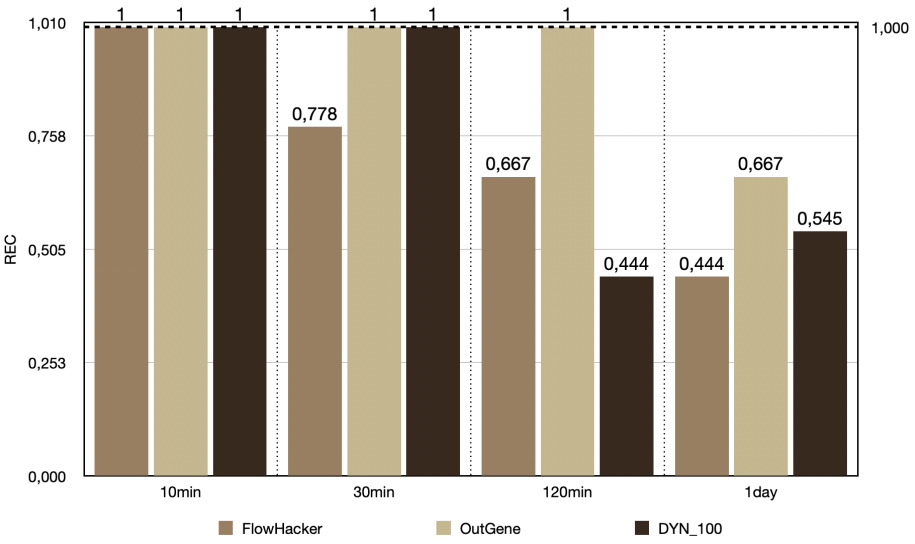


Figure 16: REC comparison of OutGene, FlowHacker and DynIDS for the artificial dataset, without attacks in Table 14

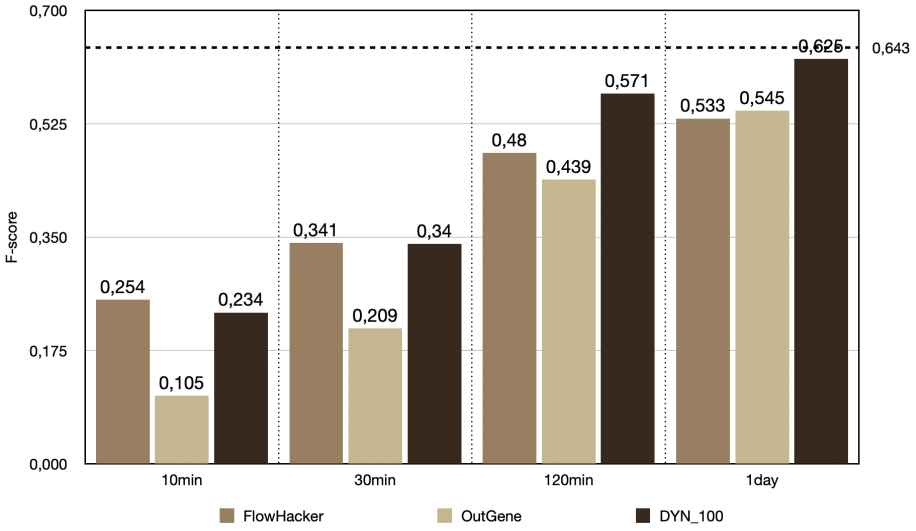


Figure 17: F-score comparison of OutGene, FlowHacker and DynIDS for the artificial dataset, without attacks in Table 14

5.5.2 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 eight 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 uses 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 9 we can observe small values for PREC and, consequently, small values for F-score for OutGene. The same can be seen in Figure 13 where Outgene was able to identify as many attacks as our framework but due to a huge number of FP the PREC and F-score values are significantly smaller. For one day, time window Outgene did not identify any host as a potential threat.

5.5.3 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 the 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 use 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 9, 10, and 11 show the results with the artificial dataset. It it is possible to observe worse results when comparing to C2BID. For the real-world dataset, Figures 12,13 and 14, FlowHacker identified the same attacks but also produced more FP witch results in worst PREC and F-score then our framework. For 1 day time-window FlowHacker just marked FPs as a potential threat.

5.5.4 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 [26]. 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 9, 10 and 11 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. Figures 12,13 and 14 have the results for real-world dataset. In this dataset, DynIDS produces a bigger number of FP than our framework. We were able to achieve better precision and F-score, mainly because of a small FP number, exception for one day time-window where DynIDS marked just one host (TP) as potential threat.

6. 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 system is not focused on real-time intrusion detection as we need a considerable period of time (e.g., one day) 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 behaviour of a host in a time period; and on cluster monitoring, i.e., analysing 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 at different pace.

C2BID was able to reduce the FPR 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.

Future work would pass for a more detailed study in clustering algorithm (section 4.2), by trying to apply different algorithms, e.g., Agglomerative or DBSCAN in search for better group hosts with similar behaviour.

Another improvement would be analysing other sources such as firewall and host-based events. Correlating NetFlow logs with these events should provide more information about every entity, increasing the complexity of how data is processed. Different features should be also analysed to try to detect more attacks and better characterise the data.

The development of a graphical interface would greatly simplified the use of C2BID, opening the possibility of applying other analysis tools, e.g., genetic zoom [25].

References

[1] Implementation of the robust random cut forest algorithm for anomaly detection on streams. *Journal of Open Source Software, The Open Journal*, 4(35):1336, 2019.

[2] Charu C Aggarwal. Outlier analysis. In *Data mining*, pages 237–263. Springer, 2015.

[3] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information Systems*, 53:16–38, 2015.

[4] Anisa Allahdadi, Ricardo Morla, and Jaime S Cardoso. Outlier detection in 802.11 wireless access points using hidden markov models. In *7th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 1–8, 2014.

[5] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[6] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. The operational role of security information and event management systems. *IEEE Security and Privacy*, 12(5):35–41, 2014.

[7] Purnima Bholowalia. EBK-Means: A Clustering Technique based on Elbow Method and K-Means in WSN. *International Journal of Computer Applications*, 105(9), 2014.

[8] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. An effective unsupervised network anomaly detection method. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, 2012.

[9] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. Network anomaly detection: methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16(1):303–336, 2013.

[10] Monowar H Bhuyan, DK Bhattacharyya, and Jugal K Kalita. NADO: network anomaly detection using outlier approach. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, pages 531–536, 2011.

[11] Tobias Bleifuß, Leon Bornemann, Theodore Johnson, Dmitri V Kalashnikov, Felix Naumann, and Divesh Srivastava. Exploring change: a new dimension of data analytics. *Proceedings of the VLDB Endowment*, 12(2):85–98, 2018.

[12] Leon Bornemann, Tobias Bleifuß, Dmitri Kalashnikov, Felix Naumann, and Divesh Srivastava. Data change exploration using time series clustering. *Datenbank-Spektrum*, 18(2):79–87, 2018.

[13] Antal van den Bosch. *Hidden Markov Models*, pages 493–495. Springer US, Boston, MA, 2010.

[14] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Optics-of: Identifying local outliers. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 262–270. Springer, 1999.

[15] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[16] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2015.

[17] Canadian Institute for Cybersecurity and University of New Brunswick. CSE-CIC-IDS2018, 2018.

[18] Pedro Casas, Johan Mazel, and Philippe Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772–783, 2012.

[19] Marcello Cinque, Raffaele Della Corte, and Antonio Pecchia. Entropy-based security analytics: Measurements from a critical information system. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 379–390, 2017.

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

[21] 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, 9 2013.

[22] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[23] Michelle Cotton, Lars Eggert, Joe Touch, Magnus Westerlund, and Stuart Cheshire. Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry., 2011.

[24] KDD Cup. Dataset. *available at the following website http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html*, 72:15, 1999.

[25] Luís Dias, Hélder Reia, Rui Neves, and Miguel Correia. Outgene: Detecting undefined network attacks with time stretching and genetic zooms. In *International Conference on Network and System Security*, pages 199–220. Springer, 2019.

[26] Luis Dias, Simão Valente, and Miguel Correia. Go with the flow: Clustering dynamically-defined netflow features for network intrusion detection with DynIDS. In *19th IEEE International Symposium on Network Computing and Applications (NCA)*. IEEE, 2020.

[27] Luis Filipe Dias and Miguel Correia. Big data analytics for intrusion detection: an overview. In *Handbook of Research on Machine and Deep Learning Applications for Cyber Security*, pages 292–316. IGI Global, 2020.

[28] Min Du, Feifei Li, Guineng Zheng, and Vivek 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*, pages 1285–1298, 2017.

[29] Sumeet Dua and Xian Du. *Data mining and machine learning in cybersecurity*. CRC press, 2016.

[30] Vegard 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*. PhD thesis, Bournemouth University, 6 2010.

[31] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1378–1386, 2018.

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

[33] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[34] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[35] Daniel Gonçalves, João Bota, and Miguel Correia. Big data analytics for detecting host misbehavior in large logs. In *2015 IEEE Trustcom/BigDataSE/ISPA*, pages 238–245. IEEE, 2015.

[36] Governo de Portugal. Conceito Estratégico de Defesa Nacional, 2013.

[37] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. 2008.

[38] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. *33rd International Conference on Machine Learning, ICML 2016*, 6:3987–3999, 2016.

[39] Riyaz Ahamed Ariyaluran Habeeb, Fariza Nasaruddin, Abdullah Gani, Ibrahim Abaker Targio Hashem, Ejaz Ahmed, and Muhammad Imran. Real-time big data processing for anomaly detection: A survey. *International Journal of Information Management*, 45:289–307, 2019.

[40] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: Data mining concepts and techniques*. Morgan Kaufmann Publishers, Waltham, third edition, 2012.

[41] Wenping He, Guolin Feng, Qiong Wu, Tao He, Shiquan Wan, and Jifan Chou. A new method for abrupt dynamic change detection of correlated time series. *International Journal of climatology*, 32(10):1604–1614, 2012.

[42] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with netflow and IPFIX. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064, 2014.

[43] Paul Innella. The evolution of intrusion detection systems. *Tetrad Digital Integrity, LLC*, 2001.

[44] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[45] Max Landauer, Markus Wurzenberger, Florian Skopik, Giuseppe Settanni, and Peter Filzmoser. Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection. *Computers and Security*, 79:94–116, 2018.

[46] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, January 1998.

[47] Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the 28th Australasian Conference on Computer Science-Volume 38*, pages 333–342, 2005.

[48] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *8th IEEE International Conference on Data Mining*, pages 413–422, 2008.

[49] Oded Maimon and Lior Rokach. Data mining and knowledge discovery handbook. 2005.

[50] Mirco Marchetti, Fabio Pierazzi, Michele Colajanni, and Alessandro Guido. Analysis of high volumes of network traffic for advanced persistent threat detection. *Computer Networks*, 109:127–141, 2016.

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

[52] Filippo Menczer, Santo Fortunato, and Clayton A. Davis. Python Tutorial. In *A First Course in Network Science*. 2020. https://cambridgeuniversitypress.github.io/FirstCourseNetworkScience/.

[53] Weizhi Meng, Elmar Wolfgang Tischhauser, Qingju Wang, Yu Wang, and Jinguang Han. When intrusion detection meets blockchain technology: a review. *IEEEAccess*, 6:10179–10188, 2018.

[54] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.

[55] Márcia Oliveira and João Gama. Bipartite graphs for monitoring clusters transitions. In *International Symposium on Intelligent Data Analysis*, pages 114–124. Springer, 2010.

[56] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[57] Thomas A Peters. The history and development of transaction log analysis. *Library hi tech*, 1993.

[58] Nadia Rahmah and Imas Sukaesih 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*, volume 31, 2016.

[59] Luis Sacramento, Ibéria Medeiros, João Bota, and Miguel Correia. Flowhacker: detecting unknown network attacks in big traffic data using network flows. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 567–572. IEEE, 2018.

[60] Douglas Steinley. K-means clustering: a half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1):1–34, 2006.

[61] Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. Precision and recall for time series. In *Advances in Neural Information Processing Systems*, pages 1920–1930, 2018.

[62] Simão Valente. Security Analytics with Mixed Event Sources and Ensembles. Master's thesis, Universidade de Lisboa, 2019.

[63] Kalyan Veeramachaneni, Ignacio Arnaldo, Vamsi Korrapati, Constantinos Bassias, and Ke Li. Ai2: training a big data machine to defend. In *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 49–54. IEEE, 2016.

[64] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *International Work-Conference on Artificial Neural Networks*, pages 758–770, 2005.

[65] Mea Wang, Baochun Li, and Zongpeng 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*, pages 628–635, 2004.

[66] Ting-Fang Yen et al. Detecting stealthy malware using behavioral features in network traffic. *Carnegie Mellon University Department of Electrical and Computer Engineering*, 2011.

[67] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 199–208, 2013.

[68] Chun-Xia Zhang, Jiang-She Zhang, Nan-Nan Ji, and Gao Guo. Learning ensemble classifiers via restricted Boltzmann machines. *Pattern Recognition Letters*, 36:161–170, 2014.

[69] Tommaso Zoppi, Andrea Ceccarelli, and Andrea Bondavalli. Evaluation of Anomaly Detection Algorithms Made Easy with RELOAD. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2019-October:446–455, 2019.

[70] Tommaso Zoppi, Andrea Ceccarelli, Lorenzo Salani, and Andrea Bondavalli. On the educated selection of unsupervised algorithms via attacks and anomaly classes. *Journal of Information Security and Applications*, 52, 2020.

[71] Richard Zuech, Taghi M Khoshgoftaar, and Randall Wald. Intrusion detection and big heterogeneous data: a survey. *Journal of Big Data*, 2(1):3, 2015.

A. False Negatives in the CIC-IDS-2018 dataset

As explained in Section 5.3, we found suspect behaviours in the CIC-IDS-2018 dataset that were not identified as attacks in that dataset. Therefore, we did manual analysis to understand if they were attacks from the standpoint of our algorithm. The manual analysis was performed by inspecting the flows involving FPs flagged by C2BID. The following example shows how this analysis occurred. We randomly chose one IP (172.31.66.82) from Table 14. The analysis of the remainder is similar to the one presented below and it was suppressed because of its considerable extension.

For each potential attacker, the flows were detected and chronologically ordered, and these were inspected for irregularities. Traffic irregularity corresponds to an unbalanced sequence of flows $IP_A \rightarrow IP_B$ and $IP_B \rightarrow IP_A$, e.g., a high amount of traffic in only one direction (Attacker$\rightarrow$ victim) with a specific and well-known port, but a random source port.

The studied IP has an unbalanced sequence of flows, i.e., a high amount of traffic in only one direction (212.92.116.6 $\rightarrow$ 172.31.66.82) with a specific and well-known port (3389).

Next, it is presented a sequence of all flows involving 212.92.116.6 and 172.31.66.82 in CIC-IDS-2018 dataset (day 1). The following flows are organized with attacker IP in red, victim IP in green and target Port in blue. The data presented is divided by commas, each value representing in order, the following:

1. Flow ID;
2. Source IP;
3. Source Port ;
4. Destination IP;
5. Destination Port ;
6. Timestamp: time when Flow was registered, this time is four hours ahead of real time;
7. Number of packages sent;
8. Number of packages received;
9. Size of packages sent;
10. Size of packages received.

```
553086,212.92.116.6,58588,172.31.66.82,3389,2018-02-14 12:31:47,9,12,1396.0,1699.0
552501,212.92.116.6,53103,172.31.66.82,3389,2018-02-14 12:31:59,9,12,1434.0,3047.0
549581,212.92.116.6,64716,172.31.66.82,3389,2018-02-14 12:32:33,11,15,1461.0,3066.0
550952,212.92.116.6,53597,172.31.66.82,3389,2018-02-14 12:33:10,8,10,1128.0,1618.0
554899,212.92.116.6,54395,172.31.66.82,3389,2018-02-14 12:34:14,8,10,1171.0,1600.0
```

552462,212.92.116.6,56520,172.31.66.82,3389,2018-02-14 12:35:03,7,8,1128.0,1581.0
547902,212.92.116.6,65269,172.31.66.82,3389,2018-02-14 12:35:20,7,8,1128.0,1581.0
553991,212.92.116.6,57187,172.31.66.82,3389,2018-02-14 12:56:40,9,11,1262.0,1640.0
554390,212.92.116.6,52990,172.31.66.82,3389,2018-02-14 13:08:56,7,8,1128.0,1581.0
548810,212.92.116.6,55376,172.31.66.82,3389,2018-02-14 13:09:05,9,12,1347.0,1933.0
554812,212.92.116.6,58635,172.31.66.82,3389,2018-02-14 13:09:34,8,10,1171.0,1600.0
549204,212.92.116.6,53098,172.31.66.82,3389,2018-02-14 13:10:14,8,9,1789.0,1581.0
550480,212.92.116.6,54041,172.31.66.82,3389,2018-02-14 13:11:10,8,10,1821.0,1581.0
551266,212.92.116.6,62090,172.31.66.82,3389,2018-02-14 13:11:50,10,12,2450.0,1618.0
553569,212.92.116.6,53127,172.31.66.82,3389,2018-02-14 13:12:00,7,8,1128.0,1581.0
547834,212.92.116.6,63874,172.31.66.82,3389,2018-02-14 13:37:34,8,11,1262.0,1933.0
550486,212.92.116.6,53201,172.31.66.82,3389,2018-02-14 13:45:42,9,12,1412.0,1699.0
553971,212.92.116.6,49519,172.31.66.82,3389,2018-02-14 13:45:47,9,13,1347.0,1970.0
550254,212.92.116.6,61011,172.31.66.82,3389,2018-02-14 13:45:59,10,12,1187.0,1600.0
549543,212.92.116.6,49518,172.31.66.82,3389,2018-02-14 13:46:44,7,8,1128.0,1581.0
550780,212.92.116.6,50693,172.31.66.82,3389,2018-02-14 13:47:39,9,11,1821.0,1874.0
550990,212.92.116.6,59052,172.31.66.82,3389,2018-02-14 13:48:17,10,14,1299.0,1912.0
548071,212.92.116.6,52842,172.31.66.82,3389,2018-02-14 13:48:22,7,10,1128.0,1655.0
552190,212.92.116.6,63892,172.31.66.82,3389,2018-02-14 14:18:11,9,12,1262.0,1677.0
550691,212.92.116.6,54927,172.31.66.82,3389,2018-02-14 14:21:51,8,11,1278.0,1677.0
555143,212.92.116.6,56766,172.31.66.82,3389,2018-02-14 14:22:06,10,13,1171.0,1020.0
549698,212.92.116.6,58913,172.31.66.82,3389,2018-02-14 14:22:06,10,13,1333.0,4500.0
553653,172.31.66.82,0,212.92.116.6,0,2018-02-14 14:22:10,1,1,0.0,0.0
554707,212.92.116.6,57006,172.31.66.82,3389,2018-02-14 14:22:49,8,10,1349.0,2754.0
550647,212.92.116.6,54509,172.31.66.82,3389,2018-02-14 14:23:44,8,10,1128.0,1618.0
549761,212.92.116.6,56758,172.31.66.82,3389,2018-02-14 14:24:21,9,12,1554.0,3927.0
554518,212.92.116.6,60765,172.31.66.82,3389,2018-02-14 14:24:22,8,10,1128.0,1618.0
552482,212.92.116.6,52790,172.31.66.82,3389,2018-02-14 14:58:09,8,10,1213.0,1874.0
549199,212.92.116.6,61787,172.31.66.82,3389,2018-02-14 14:58:19,7,9,1144.0,1618.0
551856,212.92.116.6,50045,172.31.66.82,3389,2018-02-14 14:58:29,7,9,1144.0,1618.0
550212,212.92.116.6,61861,172.31.66.82,3389,2018-02-14 14:59:03,9,13,1230.0,1619.0
552209,212.92.116.6,60487,172.31.66.82,3389,2018-02-14 14:59:16,7,9,1128.0,1618.0
555723,212.92.116.6,50749,172.31.66.82,3389,2018-02-14 15:00:09,8,11,1213.0,1911.0
552455,212.92.116.6,54877,172.31.66.82,3389,2018-02-14 15:00:24,8,9,1128.0,1581.0
549408,212.92.116.6,51380,172.31.66.82,3389,2018-02-14 15:00:30,7,9,1144.0,1618.0
552713,212.92.116.6,57660,172.31.66.82,3389,2018-02-14 15:34:16,7,8,1128.0,1581.0
551689,212.92.116.6,64809,172.31.66.82,3389,2018-02-14 15:34:19,7,9,1128.0,1618.0
550286,212.92.116.6,65264,172.31.66.82,3389,2018-02-14 15:34:41,7,9,1144.0,1874.0
551812,212.92.116.6,53286,172.31.66.82,3389,2018-02-14 15:35:04,8,11,1213.0,2167.0
550617,212.92.116.6,50778,172.31.66.82,3389,2018-02-14 15:36:12,7,8,1128.0,1581.0
548774,212.92.116.6,56756,172.31.66.82,3389,2018-02-14 15:36:14,8,10,1262.0,1640.0
552327,212.92.116.6,53581,172.31.66.82,3389,2018-02-14 15:36:27,9,13,1321.0,1696.0
553075,212.92.116.6,49966,172.31.66.82,3389,2018-02-14 15:39:59,8,10,1187.0,1600.0
555058,212.92.116.6,62812,172.31.66.82,3389,2018-02-14 16:10:15,10,13,1966.0,1659.0
551844,212.92.116.6,56830,172.31.66.82,3389,2018-02-14 16:10:20,7,8,1128.0,1581.0
553289,212.92.116.6,54730,172.31.66.82,3389,2018-02-14 16:10:48,7,8,1144.0,1581.0
551189,212.92.116.6,65300,172.31.66.82,3389,2018-02-14 16:10:59,9,11,1171.0,1600.0
552027,212.92.116.6,57089,172.31.66.82,3389,2018-02-14 16:11:38,8,10,1789.0,1874.0
544523,212.92.116.6,51943,172.31.66.82,3389,2018-02-14 16:12:10,8,12,1213.0,2204.0
551963,212.92.116.6,50713,172.31.66.82,3389,2018-02-14 16:12:23,7,9,1128.0,1618.0
547876,212.92.116.6,65136,172.31.66.82,3389,2018-02-14 16:20:59,7,9,1128.0,1874.0

554922,212.92.116.6,58744,172.31.66.82,3389,2018-02-14 16:46:08,8,10,1171.0,1600.0
551131,212.92.116.6,51220,172.31.66.82,3389,2018-02-14 16:46:27,8,9,1789.0,1581.0
553018,212.92.116.6,49726,172.31.66.82,3389,2018-02-14 16:47:01,10,15,1376.0,2847.0
548349,212.92.116.6,56159,172.31.66.82,3389,2018-02-14 16:47:04,9,11,1923.0,1640.0
554550,212.92.116.6,65008,172.31.66.82,3389,2018-02-14 16:47:15,8,10,1333.0,2754.0
550444,212.92.116.6,49908,172.31.66.82,3389,2018-02-14 16:48:12,7,8,1128.0,1581.0
550809,212.92.116.6,51152,172.31.66.82,3389,2018-02-14 16:48:20,9,13,1305.0,1696.0
548120,212.92.116.6,51943,172.31.66.82,3389,2018-02-14 17:01:39,9,13,1994.0,3964.0
549872,212.92.116.6,65232,172.31.66.82,3389,2018-02-14 17:21:55,7,8,1144.0,1581.0
551584,212.92.116.6,61777,172.31.66.82,3389,2018-02-14 17:22:08,9,12,1305.0,1659.0
552086,212.92.116.6,52383,172.31.66.82,3389,2018-02-14 17:22:32,9,13,1347.0,1992.0
548624,212.92.116.6,63039,172.31.66.82,3389,2018-02-14 17:23:11,9,12,1256.0,1893.0
553516,212.92.116.6,52419,172.31.66.82,3389,2018-02-14 17:23:14,10,13,2160.0,2813.0
551867,212.92.116.6,62275,172.31.66.82,3389,2018-02-14 17:24:00,7,8,1128.0,1581.0
548833,212.92.116.6,50470,172.31.66.82,3389,2018-02-14 17:24:09,8,10,1171.0,1600.0
548700,212.92.116.6,58110,172.31.66.82,3389,2018-02-14 17:42:47,8,9,1128.0,1581.0
550013,212.92.116.6,50098,172.31.66.82,3389,2018-02-14 17:57:53,8,9,1789.0,1581.0
550144,212.92.116.6,61109,172.31.66.82,3389,2018-02-14 17:57:57,8,11,1333.0,3927.0
552022,212.92.116.6,52098,172.31.66.82,3389,2018-02-14 17:58:01,7,8,1128.0,1581.0
552996,212.92.116.6,65214,172.31.66.82,3389,2018-02-14 17:59:04,9,11,2450.0,1618.0
555076,212.92.116.6,64012,172.31.66.82,3389,2018-02-14 17:59:17,7,8,1128.0,1581.0
552759,212.92.116.6,60092,172.31.66.82,3389,2018-02-14 17:59:51,8,10,1789.0,1618.0
551258,212.92.116.6,62246,172.31.66.82,3389,2018-02-14 17:59:52,8,9,1789.0,1581.0
548564,212.92.116.6,65327,172.31.66.82,3389,2018-02-14 18:23:33,8,9,1789.0,1581.0
555349,212.92.116.6,51099,172.31.66.82,3389,2018-02-14 18:33:07,8,10,1333.0,2754.0
549355,212.92.116.6,51941,172.31.66.82,3389,2018-02-14 18:33:36,2,5,0.0,0.0
549943,212.92.116.6,63273,172.31.66.82,3389,2018-02-14 18:33:41,10,13,2037.0,2773.0
550747,212.92.116.6,59863,172.31.66.82,3389,2018-02-14 18:34:43,8,10,1333.0,2754.0
555256,212.92.116.6,58362,172.31.66.82,3389,2018-02-14 18:35:10,10,13,1363.0,1933.0
548837,212.92.116.6,55314,172.31.66.82,3389,2018-02-14 18:35:22,7,11,1144.0,1692.0
551044,212.92.116.6,49396,172.31.66.82,3389,2018-02-14 18:35:27,8,9,1821.0,1581.0
551778,212.92.116.6,54448,172.31.66.82,3389,2018-02-14 19:04:07,8,10,1278.0,1640.0
554230,212.92.116.6,61460,172.31.66.82,3389,2018-02-14 19:08:16,8,10,1333.0,2754.0
554275,212.92.116.6,53065,172.31.66.82,3389,2018-02-14 19:09:09,9,11,1923.0,1640.0
550971,212.92.116.6,62486,172.31.66.82,3389,2018-02-14 19:09:13,7,9,1128.0,1618.0
555182,212.92.116.6,49379,172.31.66.82,3389,2018-02-14 19:10:04,8,10,1333.0,2754.0
553449,212.92.116.6,52107,172.31.66.82,3389,2018-02-14 19:10:41,8,10,1333.0,2754.0
550508,212.92.116.6,52051,172.31.66.82,3389,2018-02-14 19:10:55,9,12,1298.0,2167.0
548565,212.92.116.6,54726,172.31.66.82,3389,2018-02-14 19:10:56,9,11,1171.0,1600.0
554247,212.92.116.6,58386,172.31.66.82,3389,2018-02-14 19:43:07,2,5,0.0,0.0
554527,212.92.116.6,58443,172.31.66.82,3389,2018-02-14 19:44:24,10,14,1461.0,3066.0
555719,212.92.116.6,58241,172.31.66.82,3389,2018-02-14 19:44:31,7,8,1144.0,1581.0
550585,212.92.116.6,61321,172.31.66.82,3389,2018-02-14 19:44:47,8,9,1821.0,1581.0
552147,212.92.116.6,53283,172.31.66.82,3389,2018-02-14 19:45:19,9,12,2026.0,2791.0
553244,212.92.116.6,64064,172.31.66.82,3389,2018-02-14 19:45:44,7,10,1128.0,1911.0
548214,212.92.116.6,61033,172.31.66.82,3389,2018-02-14 19:46:11,8,11,1262.0,1699.0
555308,212.92.116.6,53241,172.31.66.82,3389,2018-02-14 19:46:22,7,8,1128.0,1581.0

Finally, the attacking IPs were found in an online DB[1]. This DB consists of a centralised repository of storage locations, system logs and other components that can report IP groups associated with

[1]https://www.abuseipdb.com

malicious activities. In Figure 18 it is possible to observe an example of the analysed IP. Here we could find the number of reports as well as the nature of each one [2].

From the 9 IP marked as TP, the identified attacks were mainly Brute Force to port 3389 (Windows Remote Desktop) and reconnaissance activities. Three of the eight different attackers were found in the public database mentioned above.

[2]This DB is updated frequently, and some IPs might no longer be marked as dangerous, but they were on Dataset creation date (February 2018).

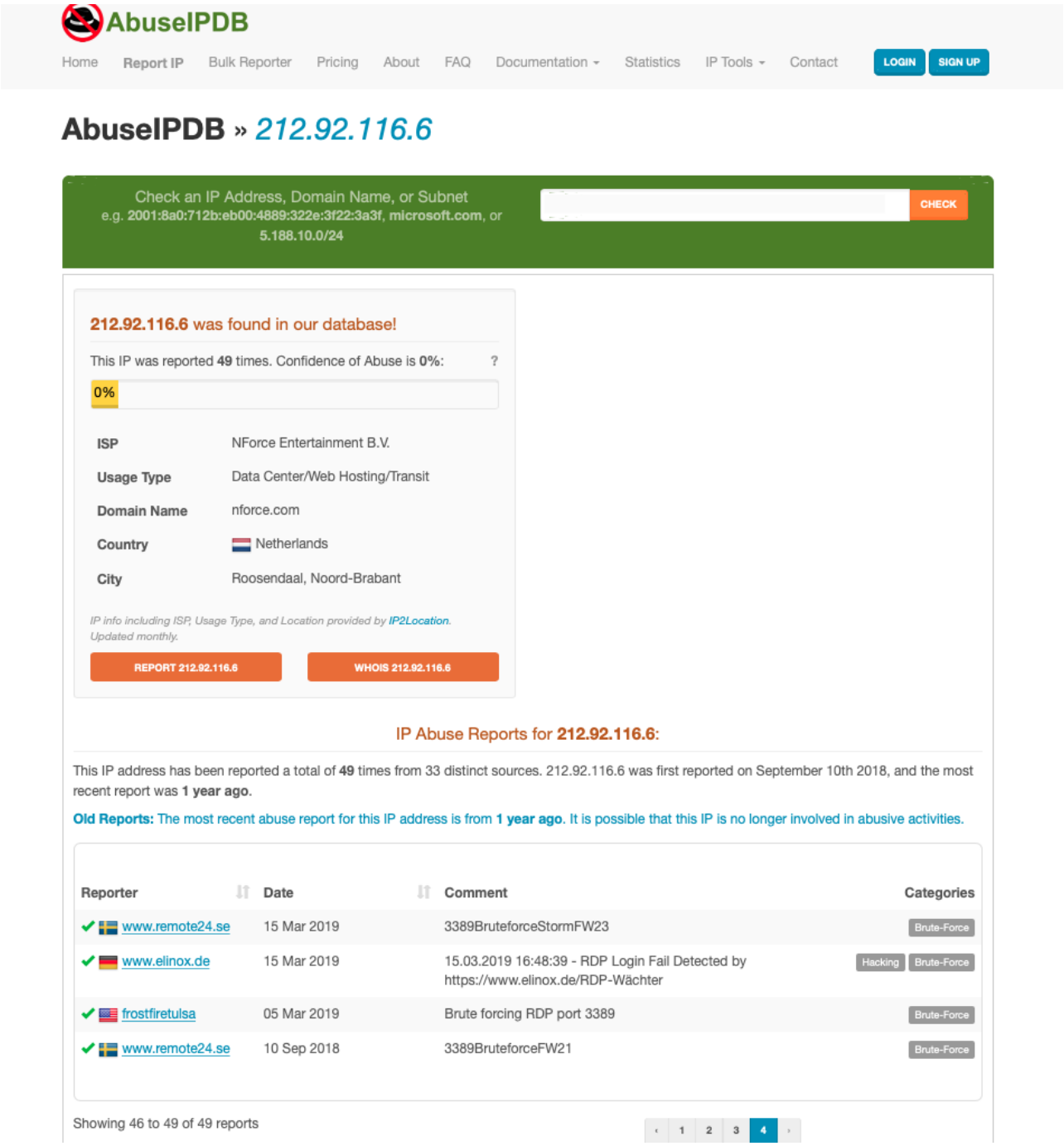


Figure 18: 212.92.116.6 search result in www.abuseipdb.com (the screenshot was taken on 5th October 2020)