Hindawi
Security and Communication Networks
Volume 2017, Article ID 6047053, 10 pages
https://doi.org/10.1155/2017/6047053

*Research Article*

# Pattern Extraction Algorithm for NetFlow-Based Botnet Activities Detection

**Rafał Kozik and Michał Choraś**

*UTP University of Science and Technology in Bydgoszcz, Bydgoszcz, Poland*

Correspondence should be addressed to Rafał Kozik; rkozik@utp.edu.pl

Received 31 May 2017; Accepted 14 September 2017; Published 17 October 2017

Academic Editor: Pedro García-Teodoro



As computer and network technologies evolve, the complexity of cybersecurity has dramatically increased. Advanced cyber threats have led to current approaches to cyber-attack detection becoming ineffective. Many currently used computer systems and applications have never been deeply tested from a cybersecurity point of view and are an easy target for cyber criminals. The paradigm of security by design is still more of a wish than a reality, especially in the context of constantly evolving systems. On the other hand, protection technologies have also improved. Recently, Big Data technologies have given network administrators a wide spectrum of tools to combat cyber threats. In this paper, we present an innovative system for network traffic analysis and anomalies detection to utilise these tools. The systems architecture is based on a Big Data processing framework, data mining, and innovative machine learning techniques. So far, the proposed system implements pattern extraction strategies that leverage batch processing methods. As a use case we consider the problem of botnet detection by means of data in the form of NetFlows. Results are promising and show that the proposed system can be a useful tool to improve cybersecurity.

## 1. Introduction

The cyber ecosystem is constantly changing as new technology stacks are being created [1]. However, many existing solutions have vulnerabilities and are frequent targets of cyber criminals. Signature-based detection techniques are ineffective when faced with current cyber threats and the sophistication of precisely crafted malware software that constantly evolves and changes. Of course, protection technologies have also improved. For example, Big Data, distributed data mining, and machine learning are increasingly common candidate technologies to counter cyber attacks and cyber crime. Nowadays, one of the major cybersecurity challenges is to counter malicious software [2]. Usually, malware samples are carefully crafted pieces of computer programs that stay dormant while performing detailed surveillance of infected infrastructures and assets. Infected computers commonly connect via a telecommunication network and form a so-called botnet that can be easily centrally controlled by cyber-criminals for different malicious purposes such as DDoS attacks, SPAM distribution, ransomware, sensitive data thefts, and extortion attacks. Advancements in machine learning and data mining techniques in the area of Big Data introduce new possibilities to fight against the latest malware. In this paper, we propose a cybersecurity system to counter botnets, based on the Big Data concept and architecture. The main contribution of this work is the proposal of an innovative tool enhancing the cybersecurity of a local area network. The tool supports the network administrator in network traffic analysis by providing a distributed framework for visualisation, data mining, and feature extraction. Moreover, we propose scientific contributions in the form of an application of a distributed random forest classifier, and an algorithm for solving the problem of imbalanced data. The paper is structured as follows. First, we provide an overview of existing solutions and methods for botnet detection. Next, we describe the proposed system architecture as well as pattern extraction and classification methods for NetFlow analysis. Then, in the section devoted to experiments, we present the evaluation methodology and obtained results. The paper is concluded with final remarks and plans for future work.

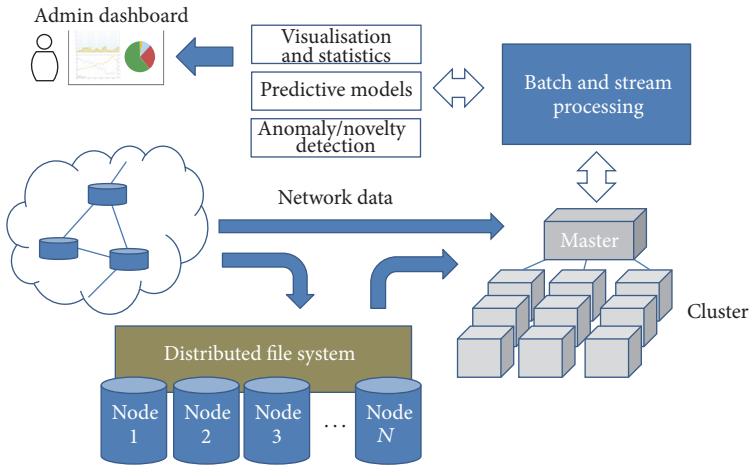


Figure 1: Overview of the proposed system for Big Data patterns extraction for cybersecurity.

## 2. Related Work in Countering Botnets and Malware

There are two approaches for cyber-attack detection: signature-based and anomaly-based. The signatures (in the form of reactive rules) of an attack used by software like Snort [3] are provided by experts from the cyber community. Typically, for deterministic attacks, it is easy to develop patterns that will identify the specific attack. Then, those patterns (in the form of signatures) are added to the database, and whenever they match the examined traffic, the attack is recognised and the alarm is raised. However, the task of developing new signatures becomes more complicated when it comes to polymorphic worms or viruses. Such software commonly modifies and obfuscates its code (without changing the internal algorithms) to be less predictive and harder to detect. In such situations, signatures do not work as an efficient approach to attack detection. The development of an efficient and scalable method for malware detection is currently challenging also due to the general unavailability of raw network data. This is largely due to the trend of protecting users privacy for administrative and legal reasons (such as General Data Protection Regulation in Europe). Unfortunately, these important regulations create difficulties for research and development [4, 5]. A common alternative to solve the above-mentioned problem of the lack of the traffic data is called NetFlow [6] data, which is often captured by ISPs for auditing and performance monitoring purposes. NetFlow samples do not contain sensitive information and therefore are widely available to the research community. However, the disadvantage is that such samples do not contain the raw content of network packets. In the literature, there are different approaches focusing on the analysis of NetFlow data. In [7], the authors focused on host dependency modelling using NetFlow data analysis and malware detection. However, they focus only on peer-to-peer communication schemes. On the other hand, the BClus algorithm proposed in [8] uses a more generic approach based on behavioral analysis for botnet detection. In particular, the BClus method uses (similar to our proposed approach) low-level features to identify potentially malicious traffic. The algorithm aggregates NetFlows for specific IP addresses and clusters them according to statistical characteristics. The properties of the clusters are described and used for further botnet detection. However, the EM-based clustering procedure may lead to a situation where some significant information is missing or the Gaussian model is not accurate enough (due to the limited amount of the data in the cluster). The CCDetector method presented in [9] uses a state-based behavioral model of the known command and control channels. The author of this algorithm proposes to use a Markov Chain to model malware behavior and to detect similar traffic in unknown real networks. The key difference from both BClus and our own approach is that, instead of analysing the complete traffic of an infected computer as a whole, the authors separate each individual connection from each IP address and treat them as an independent connection. The results obtained with this method are very promising. However, one of the concerns is the complex and time-consuming learning phase. Another approach is used in the BotHunter [10] tool. It monitors the two-way communication flows between hosts within both an internal network and the Internet. BotHunter employs the Snort intrusion detection system. It models an infection sequence as a composition of participants and a loosely ordered sequence of network information exchanges. However, as shown in [8, 11], the effectiveness of this tool is unsatisfactory in some setups. In this paper, we propose another approach based on a Big Data architecture and distributed machine learning methodologies. The details of our contribution are given in the next section.

## 3. Architecture Overview

In Figure 1 the general overview of the system design is presented. The information flow follows the typical data processing pipeline used in Big Data processing. Hereby, we propose to deploy such an approach as a cybersecurity solution.

*3.1. Apache Spark.* The proposed system adapts the Big Data lambda architecture built on top of the scalable data processing framework named Apache Spark. It provides an engine that processes Big Data workloads. There are several key elements of the architecture that facilitate distributed computing. These terms will be further used in this paper; thus here we briefly explain the definitions:

(i) Node (or host) is a machine that belongs to the computing cluster.

(ii) Driver (an application) is a module that contains the application code and communicates with the cluster manager.

(iii) Master node (a cluster manager) is the element communicating with the driver and responsible for allocating resources across applications.

(iv) Worker node is a machine that can execute application code and holds an executor.

(v) Context (SparkContext) is the main entry for Spark functionalities, which provides API for data manipulations (e.g., variables broadcasting, data creations).

(vi) Executor is a process on a worker node that runs tasks.

(vii) Task is a unit of work sent to the executor.

Apache Spark uses the data abstraction called Resilient Distributed Dataset (RDD) to manage the data distributed in the cluster and to implement fault-tolerance.

*3.2. The Analysed Data.* The data acquired by the system have a form of NetFlows. NetFlow is a standardised format for describing bidirectional communication and contains information such as IP source and destination address, destination port, and amount of bytes exchanged.

The collected NetFlows are stored in the HDFS (Hadoop Distributed File System [12]) for further processing. In the current version of the proposed system, the data mining and feature extraction methods work in a batch processing mode. However, in the future, we plan to allow the system to analyse the streams of data containing the raw NetFlows directly as they are received.

It must also be noted that, in this paper, the problem of realistic testbed construction is not considered. This is because in order to evaluate the effectiveness of the proposed algorithms we have used the benchmark CTU-13 [8] dataset, and we followed the experimental setup of its authors in order to methodologically compare our results. The dataset contains different scenarios representing different infections and malware communication schemes with command and control centre. More details on the datasets are given in the following sections.

*3.3. Feature Extraction.* A single NetFlow usually does not provide enough evidence to decide if a particular machine is infected or if a particular request has malicious symptoms. Therefore, it is quite common [8, 9] that NetFlows are aggregated in so-called time windows, so that more contextual data can be extracted and malicious behavior recorded (e.g., port scanning, packet flooding effects).

In such approaches, various statistics are extracted for each time window. In the current version of the proposed system, the SparkSQL [13] language is used to extract such statistics. Following the information pipeline, this step is related to the "data processing" stage. The result of the SQL query is data abstraction called DataFrame, which can be further processed, stored, or converted to any other specific format that will satisfy the machine learning or pattern extraction algorithms.

In general, the proposed feature extraction method aggregates the NetFlows within each time window. For each time window, we group the NetFlows by the IP source address. For each group (containing NetFlows with the same time window and IP source address) we calculate the following statistics:

(i) Number of flows

(ii) Sum of transferred bytes

(iii) Average sum of bytes per NetFlow

(iv) Average communication time with each unique IP addresses

(v) Number of unique destination IP addresses

(vi) Number of unique destination ports

(vii) Most frequently used protocol (e.g., TCP, UDP).

The advantages of such an approach are that it allows the network administrator to identify possibly infected IP addresses, and statistics are efficiently calculated with the Apache Spark SQL queries.

*3.4. Multiscale Analysis.* When it comes to malware behaviors recorded over a time period, it may be noticed that for different kinds of malware different time-scales may be observed. For example, some time-intensive malicious activities can be observed in a short time windows (e.g., spam sending or scanning) and these usually exhibit significant volume of data that is of a specific type (e.g., a number of opened connections, a number of distinct port numbers). On the other hand, more sophisticated attacks will be stretched over time and may exhibit relevant information in time windows of a varying length.

To address the above-mentioned issues, we have adopted a sort of multiscale analysis. This technique is often used in the image processing area to recognise an object at different scales. In this process, the original image is scaled by different magnification factors and an analysis is conducted for each of them. Moreover, the features calculated at specific scales can be used to build a more complex model (e.g., we can first detect wheels and a frame to help detect a bike). A similar approach can be used in our case, but instead of scaling the original signal, we scale the size of the time window where the feature vectors are calculated.

As it is shown in Figure 2, multidimensional NetFlows can be seen as a time series. Moreover, as it was described in the previous section, the NetFlows are grouped by time windows, and for each time window statistics are calculated.

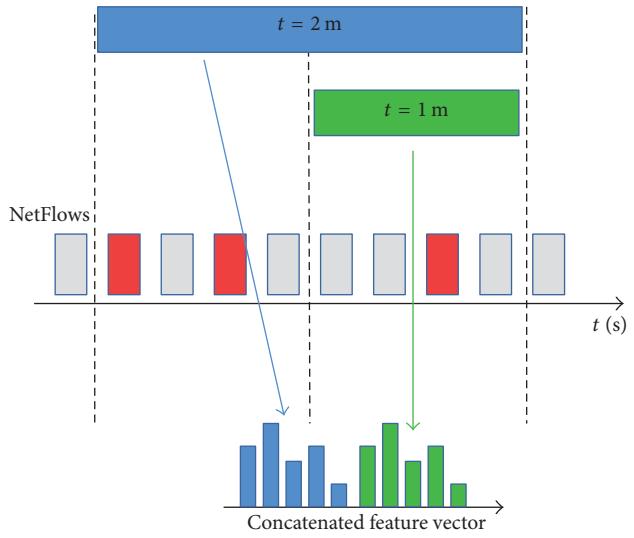


Figure 2: Overview of the technique used to multiscale NetFlows time series analysis.

For simplicity, in the presented example we are using two scales (one and two seconds wide time windows), but this approach can be easily extended to any number of scale factors. In the presented example, the final feature vector is a concatenation of vectors that are calculated for both of the time windows. From a technical point of view, the calculations for each of the time windows are happening in parallel on the Apache Spark cluster.

## 4. Distributed Machine Learning

*4.1. Decision Trees Bagging.* Decision trees have often been used in various machine learning tasks [14]. However, a common problem of this classifier is its accuracy. It is often related to the issue of the overfitting, as the structure of the tree may grow deep. Typically, the decision trees have low bias but high variance. One of the solutions to overcome this issue is to use the bagging algorithm, which allows for averaging multiple decision trees, each trained on different portions of the data.

In general, training the set of decision trees on the same data will produce highly correlated (or in many cases identical) trees. Therefore, the idea behind the bagging algorithm is to achieve an ensemble [15, 16] of decorrelated trees.

Let $L$ represent the learning dataset, which is a collection of $N$ pairs of input vectors and label values $(x_i, y_i), \ldots, (x_n, \ldots, y_n)$, where $x_i \in X$ and $y_i \in Y$. Following the bagging Algorithm 1, the learning dataset $L$ is sampled (with replacement) $B$ times. For each data sample, the decision tree is trained producing a classifier $D_b : X \to Y$. As we have $B$ classifiers, the average is computed using the following formula:

$$D(x) = \frac{1}{B}\sum_{b=1}^{B} D_b(x). \quad (1)$$

**Input:**
A training set $\{X, Y\} = \{x_i, y_i\}_{i=1}^{N}$
**Output:**
An ensemble of Decision Trees $D : X \to Y$
Training phase:
For $b = 1, \ldots, B$:
(1) Sample $B$ training samples from $X$ and $Y$
(2) Train the Decision Tree $D_b$ on the sampled data

Algorithm 1: Decision trees bagging.

*4.2. Classical Random Forest Classifier.* The random forest (RF) classifier adapts a modification of the bagging algorithm. The difference is in the process of growing the trees. The classical implementation of the RF classifier works according to the following procedure.

The final prediction obtained from the $B$ trained trees is calculated using the average in (1) or majority vote represented by (2), where $I$ stands for indicator function and $D_b$ indicates the $b$-th random tree in the ensemble.

$$D(x) = \arg\max_i \sum_{b=1}^{B} I(D_b = i). \quad (2)$$

*4.3. Distributed Random Forest Classifier.* The implementation of the random forest classifier in the MLlib Apache Spark library, in general, follows the classical algorithm presented in the previous section. However, it heavily leverages the distributed computing environment.

In principle, the algorithm works in so-called batch mode, meaning that a large portion of the data needs to be available to begin learning the classifier. This is one of the drawbacks of the current implementation (v2.1.0), as nowadays significant value is put into the availability to perform online and stream classification. In fact, this is our plan for the further work.

In general, the dataset provided to Apache Spark is arranged into rows of training instances (feature vectors with labels) and distributed across a number of nodes and partitions. The training algorithm can benefit from the distributed environment, since the learning process for each decision tree can be performed in parallel.

When the random forest is trained in the Apache Spark environment, the algorithm samples the learning data and assigns it to the decision tree, which is trained on that portion of the data. To optimise the memory and space usage, the instances are not replicated explicitly but instead augmented with additional data that hold the probability that the given instance belongs to the specific data partition used for training.

The workload is distributed among a master node and workers. The main loop manages a queue of nodes and runs iteratively. In each iteration the algorithm searches for the best split for a node. In this procedure, the statistics are collected by a worker node.

**Input:**
$N$ training samples of the form $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ such that $x_i \in R^M$ is the feature vector and $y_i$ is its label
**Output:**
Random Forest Classifier $D : X \rightarrow Y$
The $N$ training samples (each with $M$ input variables) are sampled with replacement to produce $B$ partitions.
(1) Each of the $B$ partitions is used to train one of the trees.
(2) In the process of splitting, $m$ out of the $M$ variables are selected randomly.
(3) One variable out of $m$ is selected according to the impurity measure (e.g. Gini index or entropy [21]).
(4) The procedure is terminated either when the maximal depth of a tree is achieved or when there is no data to split.

Algorithm 2: Random forest training.

The feature used for splitting is chosen among the sampled list using the impurity criterion:

$$\sum_{i=1}^{C} f_i \left(1 - f_i\right), \tag{3}$$

where $C$ is the number of unique labels and $f_i$ is the frequency of label $i$. The algorithm terminates when the maximum height of the decision tree is reached, or whenever there is no data point that is misclassified.

Another option for this classification task is the entropy measure:

$$\sum_{i=1}^{C} -f_i \log\left(f_i\right). \tag{4}$$

The final output produced by the ensemble is the majority vote of the results produced by all decision trees.

Another drawback of the current version of the Apache Spark random forest classifier is that it does not handle cost-sensitive learning (e.g., assigning higher importance to data samples indicating an anomaly or cyber attack). As a result, it may be biased towards the majority class. Finding the right balance between detection effectiveness and the number false alarms is important from the perspective of anomaly detection or intrusion detection systems. In principle, such systems should have high recognition ratio but should not overwhelm the administrator with a large number of false alarms. Therefore, our proposals to handle this issue and improve the detection effectiveness are presented in the next section.

*4.4. Data Imbalance Problem and Cost-Sensitive Learning.* The problem of data imbalance has recently been deeply studied [17–19] in the areas of machine learning and data mining. In many cases, this problem negatively impacts the machine learning algorithms and deteriorates the effectiveness of the classifier. Typically, classifiers in such cases will achieve higher predictive accuracy for the majority class, but poorer predictive accuracy for the minority class.

This phenomenon is caused by the fact that the classifier will tend to bias towards the majority class. Therefore, the challenge here is to retain the classification effectiveness even if the proportion of class labels is not equal. The imbalance of labels in our case is significant, as we may expect that in common cases of cybersecurity incidents only a few machines in the network will be infected and produce malicious traffic, while the majority will behave normally. In other words, most data contains clean traffic, while only a few data samples indicate malware.

The solutions for solving such a problem can be categorised as data-related and algorithm-related. The methods belonging to the data-related category use data oversampling and undersampling techniques, while the algorithm-related ones introduce a modification to training procedures. This group can be further classified into categories using cost-sensitive classification (e.g., assigning a higher cost to majority class) or methods that use different performance metrics (e.g., Kappa metric).

Cost-sensitive learning is an effective solution for class-imbalance in large-scale settings. The procedure can be expressed with the following optimisation formula:

$$\min_{\beta} \quad \frac{1}{2}\left\|\beta\right\|^2 + \frac{1}{2}\sum_{i=1}^{N} W_i \left\|e_i\right\|^2, \tag{5}$$

where $\beta$ indicates classifier parameters, $e_i$ the error in classifier response for $i$-th (out of $N$) data samples, and $W_i$ the importance of the $i$-th data sample. In cost-sensitive learning, the idea is to give a higher importance to the minority class, so that the bias towards the majority class is reduced.

In general, our approach to cost-sensitive random forest classifier can be categorised as data-related. In our case, we have heavily imbalanced data. In some scenarios, we have more than 2.8M normal samples while having only 40k of samples related to malware activities. Therefore, we have adopted the undersampling technique within the process of learning the distributed random forest. As it is shown in Algorithm 2, the training procedure already adapts

the sampling technique. The Apache Spark implementation adapts Poisson sampling in order to produce subportions of the data for training the random tree. In our case, before introducing the data to the classifier, we undersample the majority class with an experimentally chosen probability.

## 5. Experiments

*5.1. Evaluation Methodology.* To prove the effectiveness of the proposed method, the test methodology exploiting time-based metrics (defined in [8]) have been used. In order to keep this paper self-contained, the methodology is briefly introduced in this section. The authors of the methodology have created and published a tool called Botnet Detectors Comparer. It is publicly available for download [20]. We decided to follow their experiments and compare our results directly to their work. Their tool analyses the NetFlow file augmented with prediction labels produced by different methods and executes the following steps to produce the final effectiveness results [8]:

(1) NetFlows are separated into comparison time windows (we have used default time windows of 300 s length).

(2) The ground-truth NetFlow labels are compared to the predicted labels and the TP (true positive), TN (true negative), FP (false positive), and FN (false negative) values are accumulated.

(3) At the end of each comparison time window, the following performance indicators are calculated: FPR (false positive rate), TPR (true positive rate), TNR (true negatives rate), FNR (false negatives rate), precision, accuracy, error rate, and F-measure for that time window.

(4) When the whole file with NetFlows is processed the final error metrics are calculated and produced. As explained in [8], calculating the TP count at the NetFlow level does not make practical sense, because the administrator would be overwhelmed by the high amount of information (the number of NetFlows per second is high). Instead, IP-based performance metrics are used accordingly:

(i) TP: a true positive counter is incremented during the comparison time window whenever a Botnet IP address is detected as Botnet at least once.

(ii) TN: a true negative counter is incremented during the comparison time window whenever a Normal IP address is detected as nonbotnet during the entire time window.

(iii) FP: a false positive counter is incremented during the comparison time window whenever a Normal IP address is detected as botnet at least once.

(iv) FN: a false negative counter is incremented during the comparison time window whenever a Botnet IP address is detected as nonbotnet during the entire time window.

The authors of the tool have also incorporated time into the above-mentioned metrics to emphasise the fact that detecting a malicious IP earlier is better than later. The time-based component is called the correction function (cf) and is calculated using the following formula:

$$\mathrm{cf}(n) = 1 + e^{-\alpha n}, \tag{6}$$

where $n$ indicates a comparison time window and $\alpha$ is a constant value set to 0.01 (as suggested in [9]). Using the correction function the error metrics are calculated as follows:

$$\mathrm{tTP}(n) = \frac{\mathrm{TP} * \mathrm{cf}(n)}{N_{\mathrm{BOT}}}, \tag{7}$$

where tTP indicates time-based true positive number, cf the correction function, and $N_{\mathrm{BOT}}$ the number of unique botnet IP addresses present in the comparison time window. Similarly, we can define

$$\mathrm{tFN}(n) = \frac{\mathrm{FN} * \mathrm{cf}(n)}{N_{\mathrm{BOT}}}, \tag{8}$$

tFP and tTN do not depend on the time and are defined as follows:

$$\mathrm{tFP} = \frac{\mathrm{FP}}{N_{\mathrm{NORM}}}, \tag{9}$$

where $N_{\mathrm{NORM}}$ indicates the number of unique normal IP addresses present in the comparison time window. Similarly, the authors of the evaluation framework define

$$\mathrm{tTN} = \frac{\mathrm{TN}}{N_{\mathrm{NORM}}}. \tag{10}$$

Following the previous notations, the authors of [9] defined the final error metrics as follows.

(i) True positives rate is as follows:

$$\mathrm{TPR} = \frac{\mathrm{tTP}}{\mathrm{tFN} + \mathrm{tTP}}. \tag{11}$$

(ii) False positives rate is as follows:

$$\mathrm{FPR} = \frac{\mathrm{tFP}}{\mathrm{tTN} + \mathrm{tFP}}. \tag{12}$$

(iii) Precision is as follows:

$$P = \frac{\mathrm{tTP}}{\mathrm{tTP} + \mathrm{tFP}}. \tag{13}$$

(iv) F-measure is as follows:

$$\mathrm{FM} = 2 * \frac{P * \mathrm{TPR}}{P + \mathrm{TPR}}. \tag{14}$$

(v) Accuracy is as follows:

$$\mathrm{ACC} = \frac{\mathrm{tTP} + \mathrm{tTN}}{\mathrm{tTP} + \mathrm{tTN} + \mathrm{tFP} + \mathrm{tFN}}. \tag{15}$$

(vi) Error rate is as follows:

$$\mathrm{ERR} = \frac{\mathrm{tFP} + \mathrm{tFN}}{\mathrm{tTP} + \mathrm{tTN} + \mathrm{tFP} + \mathrm{tFN}}. \tag{16}$$

*5.2. Evaluation Dataset.* For the evaluation, we have used the CTU-13 dataset [8] and the same experimental setup as its authors. This dataset includes different scenarios which represent various types of attacks including several types of botnets. Each of these scenarios contains collected traffic in the form of NetFlows. The data was collected to create a realistic testbed. Each of the scenarios has been recorded in a separate file as a NetFlow using CSV notation. Each of the rows in a file has the following attributes (columns):

(i) StartTime: start time of the recorded NetFlow

(ii) Dur: duration

(iii) Proto: IP protocol (e.g., UTP, TCP)

(iv) SrcAddr: source address

(v) Sport: source port

(vi) Dir: direction of the recorded communication

(vii) DstAddr: destination address

(viii) Dport: destination port

(ix) State: protocol state

(x) sTos: source type of service

(xi) dTos: destination type of service

(xii) TotPkts: total number of packets that have been exchanged between source and destination

(xiii) TotBytes: total bytes exchanged

(xiv) SrcBytes: number of bytes sent by source

(xv) Label: label assigned to this NetFlow (e.g., background, normal, and botnet).

It must be noted that the "Label" field is an additional attribute provided by the authors of the dataset. Normally, the NetFlow will have 14 attributes and the "Label" will be assigned by the classifier. The dataset consists of 13 scenarios. In order to obtain comparable results presented in [8], the evaluation procedure must be preserved. Therefore, the scenarios available in the CTU-13 dataset have been used in the same manner as proposed by its author, in which the system is trained on one set of scenarios and evaluated on different ones.

## 6. Results

*6.1. Comparison with Other Methods.* During the experiments, different configurations of the proposed pattern extraction methods have been analysed. The extracted feature vectors have been used to feed the random forest machine learning algorithms, which were also evaluated for different configurations. The following notation is used in Table 1 to describe a different configuration of the proposed method:

$$\mathrm{RF}\{N\},\{W\},\{S\}, \tag{17}$$

where $N$ indicates the number of trees used by the random forest classifier, $W$ indicates the width of disjoint time windows, and $S$ represents the friction of vectors undersampled from the majority class (in order to balance the data before learning).

**Input:**
$N$ training samples of the form $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ such that $x_i \in R^M$ is a feature vector and $y_i \in L$ is its class label.
$P = \{p_i \mid i \in L\}$ sampling rates of labels $L$, where $p_i$ indicates the sampling probability of the $i$-th label
**Output:**
Random Forest Classifier $D : X \to Y$
For $t = 1, \ldots, T$:
(1) Sample $\{X, Y\}$ using $P$ sampling rates
(2) Train Random Forest Classifier $D_t$
(3) Calculate the effectiveness $E_t$ of the $D_t$ classifier on the remaining training data
Return the $D_{t^*}$ classifier with the highest accuracy, where $t^* = \arg\max_t E_t$

Algorithm 3: Cost-sensitive random forest training.

Table 1 also includes other algorithms that have been mentioned in the related work section, namely, BClus [8], CCD [9], and BotHunter [10]. The effectiveness of these methods (for the CTU-13 dataset) has already been presented in [8] and [9], respectively. We are able to also compare our approach with these tools, since we have used the same experimental setup. In Table 1 we report the performance metrics defined in the previous section, such as precision, accuracy, and F-measure.

Table 1 presents results obtained for five scenarios recorded in the testing dataset containing botnet activities. Scenario 1 corresponds to an IRC-based botnet that sends spam. In this scenario, CCD outperforms our proposed approach in terms of TPR, but it suffers a higher FPR rate. Moreover, in this scenario the proposed algorithm for cost-sensitive random forest classifier learning allows us to find an improved balance between the TP and FP rates. For all experiments we report the results for the same sampling ratios $P$ (Algorithm 3).

In Scenario 2, the same IRC-based botnet as Scenario 1 sends spam, but for a shorter timespan. Similarly with the previous scenario, the cost-sensitive learning allows us to decrease the number of false positives in contrast to the original classifier. Moreover, for the same number of false positives (2%), our approach allowed us to achieve higher values of TPR.

In Scenario 6, the botnet scans the SMPT mail servers for several hours and connects to several remote desktop services. However, it does not send any spam. The proposed approach performs very well for this type of attack when compared to the other methods.

In Scenario 8, the botnet contacts different C&C hosts and receives some encrypted data. All of the methods achieve a relatively low value of TPR. However, the cost-sensitive random forest gives a higher detection rate in contrast to CCD.

Table 1: Effectiveness of the proposed cost-sensitive distributed random forest classifiers compared to the other benchmark methods reported by the authors of the benchmark dataset in [8, 9].

| Algorithm | Scenario 1 | | | | | |
|---|---|---|---|---|---|---|
| | TPR | FPR | Precision | Accuracy | Error rate | *F*-measure |
| RF30, 1 m, 0.7 | 0.81 | 0.01 | 0.96 | 0.95 | 0.06 | 0.88 |
| RF30, 1 m, 0.01 | 0,90 | 0,14 | 0,68 | 0,87 | 0,13 | 0,77 |
| *CCD* | 1,00 | 0,05 | 0,86 | 0,96 | 0,03 | 0,92 |
| BClus | 0,40 | 0,40 | 0,50 | 0,50 | 0,40 | 0,48 |
| BH | 0,01 | 0,00 | 0,80 | 0,40 | 0,50 | 0,02 |
| Algorithm | Scenario 2 | | | | | |
| | TPR | FPR | Precision | Accuracy | Error rate | *F*-measure |
| *RF30, 1 m, 0.7* | 1.00 | 0.02 | 0.97 | 0.99 | 0.01 | 0.99 |
| RF30, 1 m, 0.01 | 0.95 | 0.25 | 0.73 | 0.83 | 0.17 | 0.82 |
| CCD | 0.74 | 0,02 | 0,96 | 0,88 | 0,11 | 0,92 |
| BClus | 0,30 | 0,20 | 0,60 | 0,50 | 0,40 | 0,41 |
| BH | 0,02 | 0,00 | 0,90 | 0,30 | 0,60 | 0,04 |
| Algorithm | Scenario 6 | | | | | |
| | TPR | FPR | Precision | Accuracy | Error rate | *F*-measure |
| *RF30, 1 m, 0.7* | 1.00 | 0.00 | 1.00 | 1.00 | 0.00 | 1.00 |
| RF30, 1 m, 0.01 | 0.96 | 0.19 | 0.74 | 0.86 | 0.14 | 0.83 |
| CCD | 0.00 | 0,00 | — | 0,64 | 0,35 | 0,00 |
| BClus | 0,00 | 0,00 | 0,40 | 0,40 | 0,50 | 0,04 |
| BH | 0,06 | 0,00 | 0,98 | 0,38 | 0,61 | 0,11 |
| Algorithm | Scenario 8 | | | | | |
| | TPR | FPR | Precision | Accuracy | Error rate | *F*-measure |
| *RF30, 1 m, 0.7* | 0.20 | 0.04 | 0.94 | 0.81 | 0.19 | 0.33 |
| RF30, 1 m, 0.01 | 0.25 | 0.13 | 0.37 | 0.73 | 0.27 | 0.30 |
| CCD | 0.00 | 0,00 | — | 0,64 | 0,35 | 0,00 |
| BClus | 0,00 | 0,04 | 0,00 | 0,66 | 0,33 | — |
| BH | 0,00 | 0,00 | 0,00 | 0,42 | 0,57 | — |
| Algorithm | Scenario 9 | | | | | |
| | TPR | FPR | Precision | Accuracy | Error rate | *F*-measure |
| *RF30, 1 m, 0.7* | 0.92 | 0.01 | 0.99 | 0.95 | 0.05 | 0.96 |
| RF30, 1 m, 0.01 | 0.94 | 0.09 | 0.99 | 0.94 | 0.06 | 0.95 |
| CCD | 0,38 | 0,04 | 0,93 | 0,59 | 0,40 | 0,54 |
| BClus | 0,10 | 0,20 | 0,40 | 0,40 | 0,50 | 0,25 |
| BH | 0,02 | 0,00 | 0,90 | 0,40 | 0,50 | 0,03 |

In Scenario 9, some hosts are infected with the Neris malware, which actively starts sending spam e-mails. As in the previous scenarios, our approach also allows us to achieve higher effectiveness than the other methods.

In summary, the results of our system are good in comparison to the state of the art. It is worth noting that, even if TPR is not higher, usually the FPR is lower which satisfies the requirements of security administrators.

The first aspect that should be commented on here is the poor performance of BClus and CCD in Scenarios 6 and 8. Upon consulting the original authors regarding this, it turned out that their results reported in [8, 9] are achieved for unidirectional NetFlows, while the published dataset contains only bidirectional NetFlows. This may cause deterioration of the results. It may also be caused by the fact that (e.g., in the case of Scenario 6) the malware has used only a few and quite specific communication channels to communicate with C&C, which has not been reflected in the training data. On the other hand, the poor performance of BClus could also be related to the fact that this method builds a statistical model that utilises GMM clustering in NetFlow parameters feature space, and probably the dataset (at least for some scenarios) cannot be modelled well by a Gaussian distribution.

It must be noted that high error rates are also reported for BotHunter. The BotHunter (BH) is a popular tool that uses a mix of anomaly- and signature-based techniques. As was noted in [22], the reason for such poor performance could be (a) the fact that BH is not able to detect machines that have already been infected before the BH deployment and (b)

Table 2: Comparison of TP and FP obtained for different time window scaling factors.

| Feature Vectors | Error Metrics | Scenarios 1 | 2 | 6 | 8 | 9 |
|---|---|---|---|---|---|---|
| 1 m | TPR | 0.81 | 1.00 | 1.00 | 0.20 | 0.92 |
| | FPR | 0.01 | 0.02 | 0.00 | 0.04 | 0.01 |
| 1 m + 4 m | TPR | 0.75 | 0.90 | 1.00 | 0.20 | 0.82 |
| | FPR | 0.01 | 0.07 | 0.00 | 0.01 | 0.01 |
| 1 m + 3 m | TPR | **0.86** | 0.92 | 1.00 | 0.20 | **0.94** |
| | FPR | 0.02 | 0.05 | 0.01 | **0.01** | 0.01 |
| 1 m + 2 m | TPR | 0.84 | 0.95 | 1.00 | 0.20 | 0.92 |
| | FPR | 0.01 | 0.07 | 0.01 | 0.01 | 0.01 |

the fact that the data from the external bots are not directly observed and are not sufficient for dialog analysis.

The second aspect is the statistical significance of the results. The reason why we did not include the confidence levels for the presented results is due to the procedure proposed by the authors of the CTU benchmark dataset in [9]. The dataset consists of 13 cybersecurity related scenarios, which have been divided into two parts: one for training and one for validation. Therefore, we can train the system on one part of the data and present the results (generated with the dedicated software tool) for another part. In a similar way, other authors present their results in [8, 22]. In such experimental protocol, it is not relevant to provide confidence levels.

*6.2. Evaluation of Multiscale Feature Extraction Technique.* In this section, we present results for our feature extraction algorithm, which uses a multiscale analysis technique. In particular, we have investigated under which circumstances the concatenation of feature vectors obtained for several time windows of different lengths can further improve the performance of our approach.

Results are presented in Table 2. In all experiments, we trained the proposed cost-sensitive random forest algorithm using feature vectors created from two time windows. The first row in the table presents the basic (using only one time window) algorithm indicated in Table 1 as RF30, 1 m, 0.7. It can be noted that the proposed multiscale feature extraction technique improves results in 3 of the 5 analysed scenarios. Concatenating the feature vectors calculated for 1-minute time windows with feature vectors calculated for 3-minute time windows (1 m + 3 m) allows us to improve the TPR in Scenarios 1 and 9 by 5% and 2%, respectively. In both of these scenarios, the algorithm was able to achieve similar values of FPR. In the case of Scenario 8, the proposed method was also able to decrease the FPR. However, we observed a deterioration of results for Scenario 2.

When concatenating 1-minute time windows with 2-minute time windows, an improvement in results can be observed; however, it is less significant.

## 7. Conclusions

In this paper, the results of a proposed malware detection distributed system based on a Big Data analytics concept and architecture have been presented. The proposed approach analyses the malware activity that is captured by means of NetFlows. This paper presented the architecture of the proposed system, the implementation details, and an analysis of promising experimental results. Moreover, we contributed a method utilising a distributed random forest classifier to address the problem of data imbalance (typical in cybersecurity). Future work is dedicated to further improvements towards online machine learning concept (e.g., to analyse online streams).

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## References

[1] R. Kozik, M. Choraś, A. Flizikowski, M. Theocharidou, V. Rosato, and E. Rome, "Advanced services for critical infrastructures protection," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 6, pp. 783–795, 2015.

[2] M. Choraś, R. Kozik, R. Renk, and W. Hołubowicz, "A Practical Framework and Guidelines to Enhance Cyber Security and Privacy," in *International Joint Conference*, vol. 369 of *Advances in Intelligent Systems and Computing*, pp. 485–495, Springer International Publishing, Cham, 2015.

[3] SNORT. Project homepage. http://www.snort.org/.

[4] T. Andrysiak, Ł. Saganowski, M. Choraś, and R. Kozik, "Network Traffic Prediction and Anomaly Detection Based on ARFIMA Model," in *International Joint Conference SOCO'14-CISIS'14-ICEUTE'14*, vol. 299 of *Advances in Intelligent Systems and Computing*, pp. 545–554, Springer International Publishing, 2014.

[5] M. Choraś, R. Kozik, D. Puchalski, and W. Hołubowicz, "Correlation approach for SQL injection attacks detection," *Advances in Intelligent Systems and Computing*, vol. 189, pp. 177–185, 2013.

[6] B. Claise, "Cisco Systems NetFlow Services Export Version 9," Tech. Rep. RFC3954, 2004.

[7] J. Francois, S. Wang, and R. State, "BotTrack: tracking botnets using NetFlow and PageRank," in *Proceedings of the IFIP/TC6 Networking*, pp. 1–14, Springer.

[8] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Journal of Computers and Security*, vol. 45, pp. 100–123, 2014.

[9] S. Garcia, *Identifying, Modeling and Detecting Botnet Behaviors in the Network [Ph.D. thesis]*, 2014.

[10] BotHunter homepage, url: http://www.bothunter.net/about.html.

[11] F. Haddadi, D. Le Cong, L. Porter, and A. N. Zincir-Heywood, "On the effectiveness of different botnet detection approaches," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 9065, pp. 121–135, 2015.

[12] HDFS. Hadoop Distirbuted File System –Architecture Guide, url: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[13] SparkSQL. Apache Spark SQL query language –overview, url: http://spark.apache.org/docs/latest/sql-programming-guide.html#sql.

[14] S. K. Jayanthi and S. Sasikala, "Reptree classifier for identifying link spam in web search engines," *ICTACT Journal on Soft Computing*, vol. 03, no. 02, pp. 498–505, 2013.

[15] G. Folino and F. S. Pisani, "Combining ensemble of classifiers by using genetic programming for cyber security applications," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 9028, pp. 54–66, 2015.

[16] R. Kozik and M. Choras, "Adapting an Ensemble of One-Class Classifiers for a Web-Layer Anomaly Detection System," in *Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015*, pp. 724–729, pol, November 2015.

[17] M. Woźniak, "Data and Knowledge Hybridization," in *Hybrid Classifiers*, vol. 519 of *Studies in Computational Intelligence*, pp. 59–93, Springer Berlin Heidelberg, Berlin, Germany, 2013.

[18] R. Kozik and M. Choraś, "Solution to Data Imbalance Problem in Application Layer Anomaly Detection Systems," in *Hybrid Artificial Intelligent Systems*, vol. 9648 of *Lecture Notes in Computer Science*, pp. 441–450, Springer International Publishing, 2016.

[19] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[20] Botnet Detectors Comparer, The download page of the tool. url:http://downloads.sourceforge.net/project/botnetdetectorscomparer/BotnetDetectorsComparer-0.9.

[21] Y. Wang and S. Xia, "Unifying attribute splitting criteria of decision trees by Tsallis entropy," in *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2507–2511, New Orleans, LA, March 2017.

[22] J. Wang and I. C. Paschalidis, "Botnet detection based on anomaly and community detection," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 392–404, 2017.