Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

An empirical comparison of botnet detection methods

S. García [a,b,*], M. Grill [b], J. Stiborek [b], A. Zunino [a]

[a] ISISTAN Research Institute – CONICET, Faculty of Sciences, UNICEN University, Argentina
[b] Agents Technology Group, Department of Computer Science and Engineering, Czech Technical University in Prague, Czech Republic

ARTICLE INFO

Article history:
Received 21 October 2013
Received in revised form
29 April 2014
Accepted 27 May 2014
Available online 5 June 2014

Keywords:
Botnet detection
Malware detection
Methods comparison
Botnet dataset
Anomaly detection
Network traffic

ABSTRACT

The results of botnet detection methods are usually presented without any comparison. Although it is generally accepted that more comparisons with third-party methods may help to improve the area, few papers could do it. Among the factors that prevent a comparison are the difficulties to share a dataset, the lack of a good dataset, the absence of a proper description of the methods and the lack of a comparison methodology. This paper compares the output of three different botnet detection methods by executing them over a new, real, labeled and large botnet dataset. This dataset includes botnet, normal and background traffic. The results of our two methods (BClus and CAMNEP) and BotHunter were compared using a methodology and a novel error metric designed for botnet detections methods. We conclude that comparing methods indeed helps to better estimate how good the methods are, to improve the algorithms, to build better datasets and to build a comparison methodology.

1. Introduction

It is difficult to estimate how much a new botnet detection method improves the current results in the area. It may be done by comparing the new results with other methods, but this has already been proven hard to accomplish (Aviv and Haeberlen, 2011). Among the factors that prevent these comparisons are: the absence of proper documentation of the methods (Tavallaee et al., 2010), the lack of a common, labeled and good botnet dataset (Rossow et al., 2012), the lack of a comparison methodology (Aviv and Haeberlen, 2011) and the lack of a suitable error metric (Salgarelli et al., 2007).

Although the comparison of methods can greatly help to improve the botnet detection area, few proposals made such a comparison (García et al., 2013). As far as we know, only three papers (Wurzinger et al., 2010; Zhao et al., 2013; Li et al., 2010) made the effort so far.

Obtaining a good dataset for comparisons is difficult. Currently, most detection proposals tend to create their own botnet datasets to evaluate their methods. However, these datasets are difficult to create (Lu et al., 2009) and usually end up being suboptimal (Shiravi et al., 2012), i.e. they lack some important features, such as ground-truth labels, heterogeneity or real-world traffic. These custom datasets are often difficult to use for comparison with other methods. This is because each

* *Corresponding author*. ISISTAN Research Institute–CONICET, Faculty of Sciences, UNICEN University, Argentina.
E-mail addresses: sebastian.garcia@isistan.unicen.edu.ar, eldraco@gmail.com (S. García), grill@agents.fel.cvut.cz (M. Grill), jan.stiborek@agents.felk.cvut.cz (J. Stiborek), alejandro.zunino@isistan.unicen.edu.ar (A. Zunino).
http://dx.doi.org/10.1016/j.cose.2014.05.011
0167-4048/

method is usually focused on different properties of the dataset. The problem is to find a good, common and public dataset that can be read by all methods and satisfy all the constrains.

The difficultly to compare detection methods goes beyond the dataset. The lack of good descriptions of the methods and error metrics contribute to the problem. As stated by Rossow et al. (2012), the error metrics used on most papers are usually non-homogeneous. They tend to use different error metrics and different definitions of error. Moreover, the most common error metrics, e.g. FPR, seems to be not enough to compare botnet detection methods. The classic error metrics were defined from a statistical point of view and they fail to address the detection needs of a network administrator.

The goal of this paper is to compare three botnet detection methods using a simple and reproducible methodology, a good dataset and a new error metric. The contributions of our paper are:

- A deep comparison of three detection methods. Our own algorithms, CAMNEP and BClus, and the third-party algorithm BotHunter (Gu et al., 2007).
- A simple methodology for comparing botnet detection methods along with the corresponding public tool for reproducing the methodology.
- A new error metric designed for comparing botnet detection methods.
- A new, large, labeled and real botnet dataset that includes botnet, normal and background data.

We conclude that the comparison of different botnet detection methods with other proposals is highly beneficial for the botnet research community because it helps to objectively assess the methods and improve the techniques. Also, that the use of a good botnet dataset is paramount for the comparison.

The rest of the paper is organized as follows. Section 2 shows previous work in the area. Section 3 describes the CAMNEP detection method. Section 4 shows the BClus botnet detection method. Section 5 describes the BotHunter method. Section 6 describes the dataset and its features. Section 7 describes the comparison methodology, the public tool and the new error metric. Section 8 shows the results and compares the methods and Section 9 presents our conclusions.

2. Previous work

The comparison of detection methods is usually considered a difficult task. In the case of botnets it is also related to the creation of a new dataset. The next Subsections describe the previous work in the area of comparison of methods and the area of creation of datasets.

2.1. *Comparison of methods*

The comparison of a new detection method with a third-party method is difficult. In the survey presented by García et al. (2013), where there is a deep analysis of fourteen network-based botnet detection methods, the authors found only one paper that made such a comparison. The survey compared the motivations, datasets and results of the fourteen proposals. It concludes that it is difficult to compare the results with another proposal because the datasets tend to be private and the descriptions of the methods tend to be incomplete.

Another analysis of the difficulty of reproducing a method was described by Tavallaee et al. (2010), where they state that there is an absence of proper documentation of the methods and experiments in most detection proposals.

One of the detection proposals that actually made a comparison with a third-party method was presented by Wurzinger et al. (2010). The purpose of the paper is to identify single infected machines using previously generated detection models. It first extracts the characters strings from the network to find the commands sent by the C&C and then it finds the bot responses to those commands. The authors downloaded and executed the BotHunter program of Gu et al. (2007) on their dataset and made a comparison. However, the paper only compares the results of both proposals using the TPR error metric and the FP values.

The other paper that made a comparison with a third-party method was presented by Zhao et al. (2013). This proposal selects a set of attributes from the network flows and then applies a Bayes Network algorithm and a Decision Tree algorithm to classify malicious and non-malicious traffic. The third-party method used for comparison was again BotHunter. There is a description of how BotHunter was executed, but unfortunately the only error metric reported was a zero False Positive. No other numerical values were presented.

The last proposal that also compared its results with a third-party method was made by Li et al. (2010). This paper analyzes the probable bias that the selection of ground-truth labels might have on the accuracy reported for malware clustering techniques. It states that common methods for determining the ground truth of labels may bias the dataset toward easy-to-cluster instances. This work is important because it successfully compared its results with the work of Bayer et al. (2009). The comparison was done with the help of Bayer et al., who run the algorithms described in Li et al. (2010) on their private dataset.

Regarding the creation of datasets for malware-related research, Rossow et al. (2012) presented a good paper about the prudent practices for designing malware experiments. They defined a prudent experiment as one being correct, realistic, transparent and that do not harm others. After analyzing 36 papers they conclude that most of them had shortcomings in one or more of these areas. Most importantly, they conclude that only a minority of papers included real-world traffic in their evaluations.

2.2. *Datasets available*

Regarding botnet datasets that are available for download, a deep study was presented in Shiravi et al. (2012) about the generation of datasets. It describes the properties that a dataset should have in order to be used for comparison purposes. The dataset used in the paper includes an IRC-based Botnet attack,[1] but the bot used for the attack was developed by the authors and therefore it may not represent a real botnet behavior. This dataset may be downloaded with authorization.

[1] http://www.iscx.ca/datasets.

Table 1 – Summary of available datasets.

Name	Available	Format	Background	Botnet	Normal	Labels
Shiravi et al. (2012)	?	?	√	√	–	?
PREDICT	√	CSV	–	√	–	–
CAIDA	√	CSV, pcap	–	√	–	–
Saad et al. (2011)	√	pcap	√	√	–	√
Szabó et al. (2008)	√	pcap	–	–	√	√
Contagio	√	pcap	–	√	–	√
NexGinRC (2013)	?	CSV	–	√	√	?
Cho et al. (2000)	√	pcap	√	–	–	–

The Protected Repository for the Defense of Infrastructure Against Cyber Threats (PREDICT) indexed three Botnet datasets[2] until May 16th, 2013. The first one is the *Kraken Botnet Sinkhole Connection Data* dataset, the second one is the *Flashback Botnet Sinkhole Connection Data* dataset and the third one is the *Conficker Botnet Sinkhole Connection Data* dataset. They were published as CSV text files, where each line is a one minute aggregation of the number of attempted connections of one IP address. Unfortunately, the aggregation method may not be suitable for comparisons with other proposals.

The CAIDA organization published a paper about the Sality botnet in Dainotti et al. (2012) along with its corresponding dataset.[3] Unfortunately, the CSV text format of the dataset may not be suitable for every detection algorithm because the content of the dataset only includes enough information to reproduce the techniques in the paper. CAIDA also published a dataset about the Witty Botnet in pcap format[4] and several datasets with responses to spoofed DoS traffic[5] and anomalous packets.[6] None of them are labeled.

A custom botnet dataset was created to verify five P2P botnet detection algorithms in Saad et al. (2011). Fortunately, this dataset was made public and can be downloaded.[7] The dataset is a mixture of two existing and publicly available malicious datasets and one non-malicious pcap dataset. They were merged to generate a new file. This was, at that time, the best dataset that can be downloaded for comparison purposes. Unfortunately, there is only one infected machine for each type of botnet, therefore no synchronization analysis can be done.

The Traffic Laboratory at Ericsson Research created a normal dataset that was used in Saad et al. (2011) and described in Szabó et al. (2008). This normal dataset is publicly available.[8] It is composed of pcap traffic files that were labeled by means of one IP header option field. This is the only normal dataset that is labeled inside the pcap file.

A considerable amount of malware traffic in pcap format was published in the Contagio blog.[9] It contains thirty one ATP pcap captures and sixty one crimenware pcaps. Each file contains the traffic of one malware without background traffic. Unfortunately, the captures are really short (mostly between 1 min and 1 h) and the traffic is not labeled. But since each scenario includes only one infected computer, it should be possible to label them.

Another dataset with malware logs and benign logs was collected in NexGinRC (2013). The malware logs are both real and simulated. The benign logs consist of 12 months of traffic. Unfortunately, the dataset is in CSV format, which may not be suitable for some detection algorithms because it does not have the same information as a NetFlow file or pcap file. Access to this dataset may be granted upon request[10].

The last dataset analyzed is currently created by the MAWI project described in Cho et al. (2000). It includes an ongoing effort to publish one of the most recent and updated background datasets to date. Its goal is to promote the research on traffic analysis and the creation of free analysis tools. However, the pcap files are not labeled, and therefore it is more difficult to use them for training or verification. There was an effort to label this dataset using anomaly detectors in Fontugne et al. (2010). The labels are not ground-truth, but may be useful to compare other methods.

A summary of the described datasets is presented in Table 1. This table shows that, so far, no dataset includes Background, Botnet and Normal labeled data.

3. The CAMNEP detection method

The Cooperative Adaptive Mechanism for NEtwork Protection (CAMNEP) (Rehak et al., 2009) is a Network Behavior Analysis system (Scarfone and Mell, 2007) that consists of various state-of-the-art anomaly detection methods. The system models the normal behavior of the network and/or individual users behaviors and labels deviations from normal behaviors as anomalous.

3.1. System architectures

CAMNEP processes NetFlow data provided by routers or other network equipment to identify anomalous traffic by means of several collaborative anomaly detection algorithms. It uses

[2] https://www.predict.org/Default.aspx?tabid=104.
[3] http://imdc.datcat.org/collection/1-06Y5-B=UCSD+Network+Telescope+Dataset+on+the+Sipscan.
[4] http://www.caida.org/data/passive/witty_worm_dataset.xml.
[5] http://www.caida.org/data/passive/backscatter_2008_dataset.xml.
[6] http://www.caida.org/data/passive/telescope-2days-2008_dataset.xml.
[7] http://www.isot.ece.uvic.ca/dataset/ISOT_Botnet_DataSet_2010.tar.gz.
[8] http://www.crysys.hu/~szabog/measurement.tar.
[9] http://contagiodump.blogspot.co.uk/2013/04/collection-of-pcap-files-from-malware.html.
[10] \textsc{http://nexginrc.org/Datasets/DatasetDetail.aspx?pageID=24}.

a multi-algorithm and multi-stage approach to reduce the amount of false positives generated by the individual anomaly detectors without compromising the performance of the system. The self-monitoring and self-adaptation techniques, described in Section 3.4, are very important to this purpose. They help improve the error rate of the system with a minimal and controllable impact on its efficiency.

CAMNEP consists of three principal layers that evaluate the traffic: anomaly detectors, trust models and anomaly aggregators.

The anomaly detectors layer (identified as Anomaly Detectors A and B in Fig. 1) analyze the NetFlows using various anomaly detection algorithms. We are currently using 8 different anomaly detection approaches. Each of them uses a different set of features, thus looking for an anomaly from slightly different perspectives. The output of these algorithms is aggregated into events using several statistical functions and the results are sent to the trust models. All the detection algorithms used in the system are described in detail in Section 3.2.

The trust models layer maps the NetFlows into traffic clusters. These clusters group together the NetFlows that have a similar behavioral pattern. They also contain the anomaly value of the type of the event that they represent. These clusters persist over time and the anomaly value is updated by the trust model. The updated anomaly value of a cluster is used to determine the anomaly of new NetFlows. Therefore, the trust models act as a persistent memory and reduce the amount of false positives by means of the spatial aggregation of the anomalies.

The aggregators layer creates one composite output that integrates the individual opinion of several anomaly detectors as they were provided by the trust models. The result of the aggregation is presented to the user of the system as the final anomaly score of the NetFlows. Each aggregator can use two different averaging operators: an order-weighted averaging

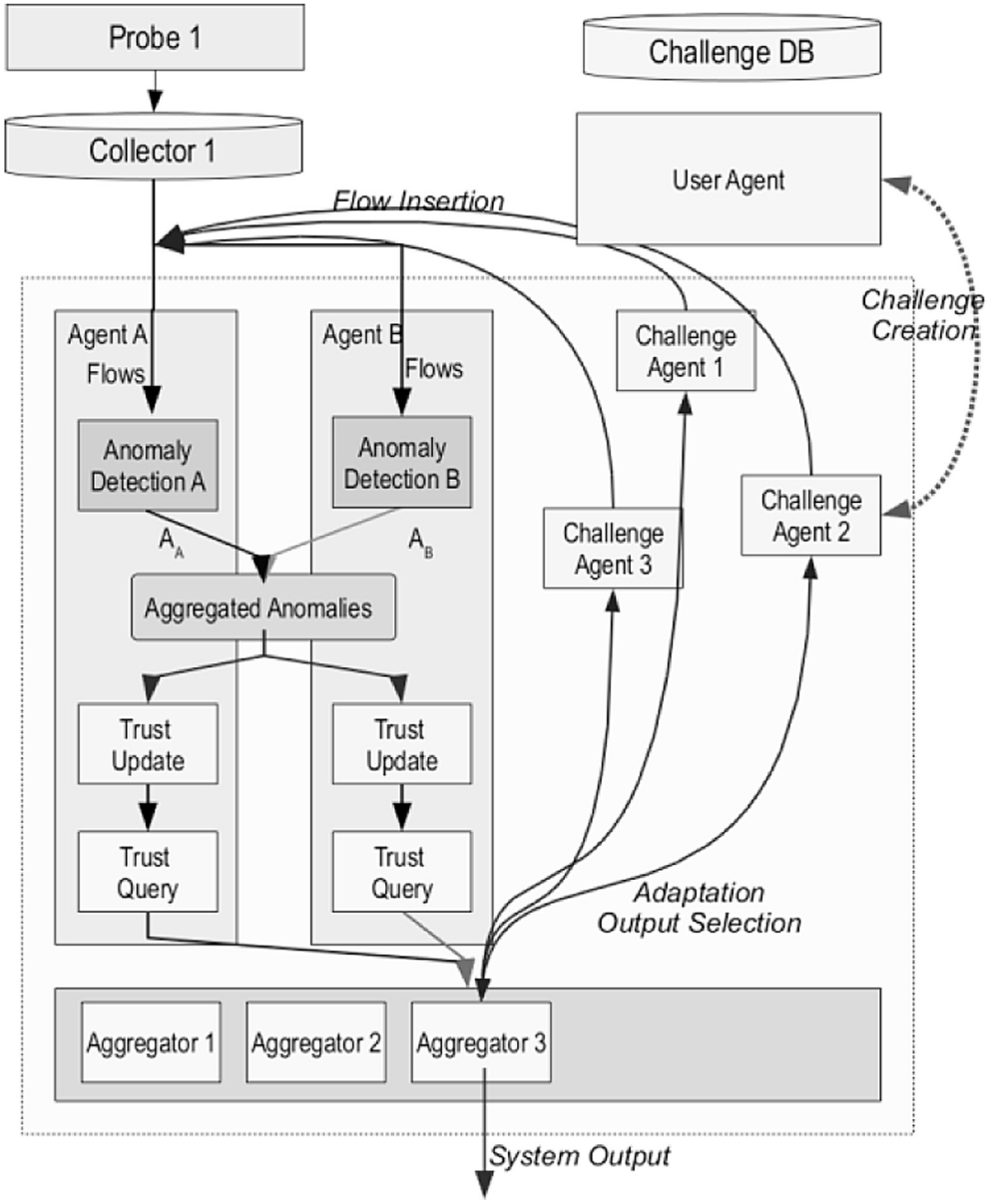

Fig. 1 – Adaptation process in the CAMNEP system.

(Yager, 1988) or simple weighted averaging. CAMNEP is using a simulation process to determine the best aggregation operator for the current type and state of network. This process is described in Section 3.4.

3.2. *Anomaly detectors*

The anomaly detectors in the CAMNEP system are based on already published anomaly detection methods. They work in two stages: (i) they extract meaningful features associated with each NetFlow (or group of NetFlows), and (ii) they use the values of these features to assign an anomaly score to each NetFlow. This anomaly score is a value in the [0,1] range. A value of 1 represents an anomaly and a value of 0 represents normal behavior. The anomaly detector's model is a fuzzy classifier that provides the anomaly value for each NetFlow. This value depends on the NetFlow itself, on other NetFlows in the current context, and on the internal traffic model, which is based on the past traffic observed on the network.

The following subsections describe each of the anomaly detectors used in the CAMNEP system.

3.2.1. *MINDS*

The MINDS algorithm (Ertoz et al., 2004) builds a context information for each evaluated NetFlow using the following features: the number of NetFlows from the same source IP address as the evaluated NetFlow, the number of NetFlows toward the same destination host, the number of NetFlows towards the same destination host from the same source port, and the number of NetFlows from the same source host towards the same destination port. This is a simplified version of the original MINDS system, which also uses a secondary window defined by the number of connections in order to address slow attacks. The anomaly value for a NetFlow is based on its distance to the normal sample. The metric defined in this four-dimensional context space uses a logarithmic scale on each context dimension, and these marginal distances are combined into the global distance as the sum of their squares. In the CAMNEP implementation of this algorithm, the variance-adjusted difference between the floating average of past values and the evaluated NetFlow on each of the four context dimensions is used to know if the evaluated NetFlow is anomalous. The original work is based on the combination of computationally-intensive clustering and human intervention.

3.2.2. *Xu*

In the algorithm proposed by Xu et al. (2005), the context of each NetFlow to be evaluated is created with all the NetFlows coming from the same source IP address. In the CAMNEP implementation, for each context group of NetFlows, a 3 dimensional model is built with the normalized entropy of the source ports, the normalized entropy of the destination ports, and the normalized entropy of the destination IP addresses. The anomalies are determined by some classification rules that divide the traffic into normal and anomalous. The distance between the contexts of two NetFlows is computed as the difference between the three normalized entropies, combined as the sum of their squares. Our implementation of the algorithm is close to the original publication, which was further expanded by Xu and Zhang (2005), except for the introduction of new rules that define a horizontal port scan as anomalous.

3.2.3. *Lakhina volume*

The volume prediction algorithm presented in Lakhina et al. (2004) uses the Principal Components Analysis (PCA) algorithm to build a model of traffic volumes from individual sources. The observed traffic for each source IP address with non-negligible volumes of traffic is defined as a three dimensional vector: the number of NetFlows, number of bytes and number of packets from the source IP address. The traffic model is defined as a dynamic and data-defined transformation matrix that is applied to the current traffic vector. The transformation splits the traffic into normal (i.e. modeled) and residual (i.e. anomalous). The transformation returns the residual amount of NetFlows, packets and bytes for each source IP address. These values define the context (identical for all the flows from the given source). An anomaly is determined by transforming the 3D context into a single value in the [0,1] interval.

Notice that the original work was designed to handle a different problem, that is, the detection of anomalies on a backbone. Also the original work modeled networks instead of source IP addresses. However, we modify it to obtain a classifier that can successfully contribute to the joint opinion when combined with others.

3.2.4. *Lakhina Entropy*

The entropy prediction algorithm presented by Lakhina et al. (2005) is based on the similar PCA-based traffic model than Section 3.2.3, but it uses different features. It aggregates the traffic from the individual source IP addresses, but instead of traffic volumes, it predicts the entropies of destination IP addresses, destination ports and source ports over the set of context NetFlows for each source. The context space is therefore three dimensional. An anomaly is determined as the normalized sum of residual entropy over all three dimensions. The metric is simple: a function measures the difference of residual entropies between the NetFlows and aggregates their squares. Also, the original anomaly detection method was significantly modified along the same lines as the volume prediction algorithm.

3.2.5. *TAPS*

The TAPS method (Sridharan et al., 2006) is different from the previous approaches because it targets horizontal and vertical port scans. The algorithm only considers the traffic sources that created at least one single-packet NetFlow during a particular observation period. These preselected sources are then classified using the following three features: number of destination IP addresses, number of destination ports and the entropy of the NetFlow size measured in number of packets. The anomaly value of the source IP address is based on the ratio between the number of unique destination IP addresses and destination ports. When this ratio exceeds a predetermined threshold the source IP address is considered as a scan origin. Using the original method, we have encountered an unusually high number of false positives. Therefore, we extended the method with the NetFlow size entropy to achieve better results.

3.2.6. KGB

The KGB anomaly detector presented by Pevný et al. (2012) is also based on Lakhina's work. It uses the same features as Lakhina Entropy detector described above. Similar to Lakhina's work, it performs a PCA analysis of the feature vectors for each source IP address in the dataset. The final anomaly is determined from the deviations of averaging the principal components.

There are two versions of KGB detector:

- KGBf – examines principal components with high variances
- KGBfog – examines principal components with low variances.

3.2.7. Flags

The Flags detector uses the same detection method as the KGB detector (Pevný et al., 2012). The only difference is in the input feature vector. The feature vector of the Flags detector is determined by the histogram of the TCP Flags of all the NetFlows with the same IP address. This detector is looking for a sequence or a combination of anomalous TCP flags.

3.3. Trust modeling

The trust models are specialized knowledge structures studied in multi-agent research (Ramchurn et al., 2004; Sabater and Sierra, 2005). The features of trust models include fast learning, robustness in response to false reputation information and robustness with respect to environmental noise.

Recent trust models, inspired by machine learning methods (Rettinger et al., 2007) and pattern recognition approaches (Rehak et al., 2007) make the trust reasoning more relevant for network security, as they are able to:

- include the context of the trusting situation into the reasoning, making the trust model situational;
- use the similarities between trustees to reason about short-lived or one shot trustees, e.g. NetFlows.

A *feature vector* includes the identity of a NetFlow and the context of the NetFlow by each trust model in the feature space. We use the term *centroid* to denote the permanent feature vectors that are positioned in the feature spaces of trust models. The centroids act as trustees of the model, and the trustfulness value of each centroid is updated. Each centroid is used to deduce the trustfulness of the feature vectors in its vicinity.

The anomaly detectors integrate the anomaly values of individual NetFlows into their trust models. The reasoning about the trustfulness of each individual NetFlow is both computationally unfeasible and unpractical (the NetFlows are single shot events by definition), and thus the centroids of the clusters holds the trustfulness of significant NetFlow samples. The anomaly value of each NetFlow is used to update the trustfulness of centroids in its vicinity. The weight used for the update of the trustfulness of the centroids decreases with the distance. Therefore, as each model uses a distinct distance function, they all have a different insight into the problem.

Each trust model determines the *trustfulness* of each NetFlow by finding all the centroids in the NetFlows vicinity. It sets the trustfulness using the distance-based weighted average of the values preserved by the centroids. All the models provide their trustfulness assessment (conceptually a reputation opinion) to the anomaly aggregators.

3.4. Adaptation

The adaptation layer of CAMNEP identifies the optimal trustfulness aggregation function that achieves the best separation between the legitimate and malicious NetFlows. This layer is based on the insertion of challenges into the NetFlow data observed by the system. The challenges are NetFlows of past classified incidents. They are generated by short lived, challenge specific *challenge agents* and are mixed with the input traffic. They cannot be distinguished from the rest of the input traffic by the detectors/aggregators. They are processed and evaluated with the rest of the traffic. Also, they are used to update anomaly detection mechanisms and the trust models. Once the process is completed, the challenges are re-identified by their respective challenge agents and removed from the output. The anomaly value given to these NetFlows by the individual anomaly aggregators is used to evaluate those aggregations and to select the optimal output for the current network conditions.

There are two broad types of challenges: The *malicious challenges* correspond to known attacks, whereas the *legitimate challenges* represent known instances of legitimate events that tend to be misclassified as anomalous. Malicious challenges are further divided into broad attack classes, such as fingerprinting/vertical scan, horizontal scan, password brute forcing, etc. For each attack class, each aggregator has a probability distribution that is empirically estimated from the continuous anomaly values attributed to the challenges in that class. This characterization can be seen in Fig. 2. All the legitimate challenges are also defined by a distribution.

We assume that the anomaly values of both the legitimate and malicious challenges are defined by normal distributions.[11] The distance between the estimated mean normalized values of both distributions, represents the quality of the aggregator with respect to a given attack class. The *effectiveness* of the aggregator, defined as an ability to distinguish between the legitimate events and the attacks is defined as a weighted average of the effectiveness with respect to individual classes.

As the network traffic is highly dynamic, it is very difficult to predict which aggregation function will be chosen, especially given the fact that the challenges are selected from a challenge database using a stochastic process with a pseudo-random generator unknown to a potential attacker. The attacker therefore faces a dynamic detection system that unpredictably switches its detection profiles. Each profile has a utility value (i.e. detection performance) close to the optimum. This unpredictability, together with the additional

[11] Normality of both distributions is not difficult to achieve provided that the attack classes are properly defined and that the challenge samples in these classes are well selected, i.e. comparable in terms of size and other parameters.

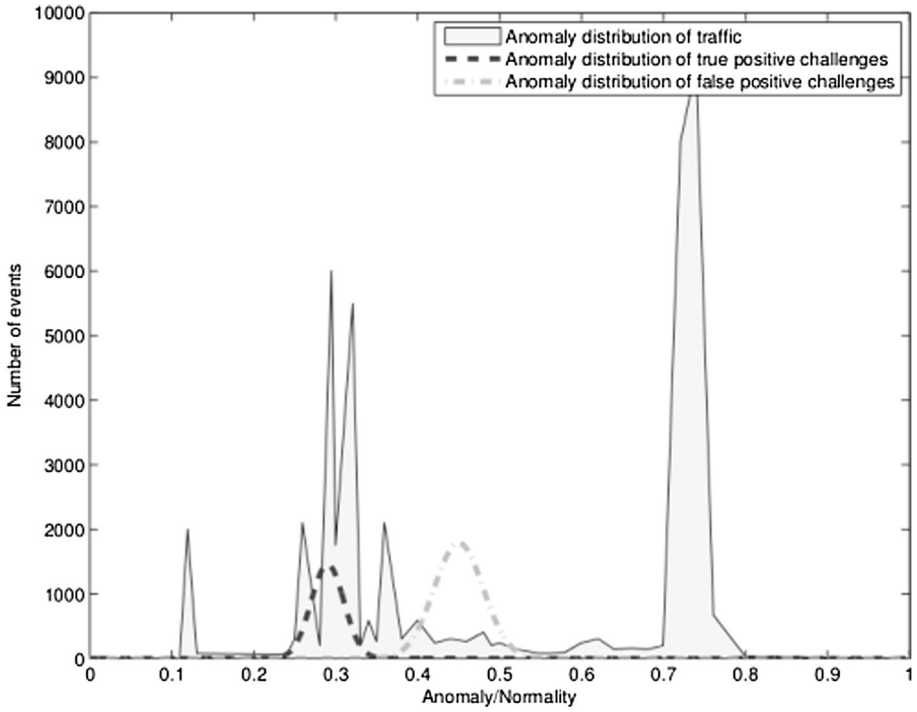

Fig. 2 – Distribution of challenges. The anomalies distribution of the malicious challenges (from one class) is on the left side of the graph, while the legitimate events are on the right.

robustness achieved by the use of multiple algorithms, makes the evasion attempt a much more difficult task than simply avoiding a single intrusion detection method (Rubinstein et al., 2009).

Furthermore, the system is able to find the optimal thresholds for the anomaly score when using the results from the adaptation process. The system is continuously modeling the normal distribution of malicious and legitimate challenges. The threshold is set to minimize the Bayes risk (posteriori expected loss) computed from the modeled legitimate and malicious behavior distributions. Thus, the final result of the system is a list of NetFlows with labels anomalous or normal.

3.5. *Training of the CAMNEP method*

Since the system needs the inner models of the anomaly detectors and trust models to have the optimal detection results, it is necessary to train them. Typically, the system needs 25 min of traffic to create its inner models and to adapt itself to the current type of network and its state. Therefore, the training data for the CAMNEP algorithm was created by trimming off some minutes at the start of each of the scenarios in the dataset described in Section 6.

4. The BClus detection method

The BClus method is a behavioral-based botnet detection approach. It creates models of known botnet behavior and uses them to detect similar traffic on the network. It is not an anomaly detection method.

The purpose of the method is to cluster the traffic sent by each IP address and to recognize which clusters have a behavior similar to the botnet traffic. A basic schema of the BClus method is:

1. Separate the NetFlows in time windows.
2. Aggregate the NetFlows by source IP address.
3. Cluster the aggregated NetFlows.
4. Only Training: Assign ground-truth labels to the botnet clusters.
5. Only Training: Train a classification model on the botnet clusters.
6. Only Testing: Use the classification model to recognize the botnet clusters.

At the end the BClus method outputs a predicted label for each NetFlow analyzed.

The following Subsections describe each of these steps.

4.1. *Separate the NetFlows in time windows*

The main reason to separate the NetFlows in time windows is the huge amount of data that had to be processed. Some of our botnet scenarios produced up to 395,000 packets per minute. A short time window allow us to better process this information.

The second reason to use time windows is that botnets tend to have a temporal locality behavior (Hegna, 2010), meaning that most actions remain unchanged for several minutes. In our dataset the locality behavior ranges between 1

and 30 min. This temporal locality helps to capture all the important behaviors in the time windows.

The third reason for using time windows is the need to deliver a result to the network administrator in a timely manner. After each time window, the BClus method outputs some results and the administrator can have an input about the traffic.

An important decision on the time window separation criteria is the window width. A short time window does not contain enough NetFlows and therefore would not allow a correct analysis of the botnet behavior. In the other hand, a large time window would have a high computational cost. The time window used by the BClus method is of two minutes, since it is enough to capture all the botnet behaviors and it does not contain too much NetFlows.

The next step in the BClus method is to aggregates the NetFlows.

4.2. *Aggregate the NetFlows by source IP address*

The purpose of aggregating th NetFlows is to analyze the problem from a new high-level perspective. The aggregated data may show new patterns. We hypothesize that these new patterns could help recognize the behaviors of botnets. From the botnet detection perspective, the main motivations for aggregating NetFlows are the following:

- Each bot communicates with the C&C server periodically (AsSadhan et al., 2009).
- Several bots may communicate at the same time with the same C&C servers (Gu et al., 2008).
- Several bots attack at the same time the same target (Lee et al., 2008).

Inside each time window, the NetFlows are aggregated during one *aggregation window*. The width of the aggregation window should be less than the width of the time window, which was of two minutes. After some experimentation, a one minute aggregation window width was selected, which is enough to capture the botnet synchronization patterns and short enough not to capture too much traffic (García et al., 2012). Therefore, on each time window, two aggregation windows are used.

The NetFlows are aggregated by unique source IP address. The resulting features on each aggregation window are:

1. Source IP address
2. Amount of unique source ports used by this source IP address.
3. Amount of unique destination IP addresses contacted by this source IP address.
4. Amount of unique destination ports contacted by this source IP address.
5. Amount of NetFlows used by this source IP address.
6. Amount of bytes transferred by this source IP address.
7. Amount of packets transferred by this source IP address.

We call this group of seven aggregated features an *instance* to simplify the references. The aggregation step ends with a list of instances for each aggregation window. Next Subsection describes the clustering process of these instances.

4.3. *Cluster the aggregated NetFlows*

The continuous evolution of botnets suggests that a good detection method should be as independent of the network characteristics as possible. The BClus method, then, uses an unsupervised approach to cluster the instances described in the previous section. These natural groups of behaviors depend on the time window being analyzed and on the characteristics of the network where the algorithm is running.

The technique used for this task is WEKA's implementation of the Expectation-Maximization (EM) algorithm (Moon, 1996). EM is an iterative procedure that attempts to find the parameters of the model that maximize the probability of the observed data. Our dataset has many different network behaviors generated by normal, botnet and attack actions. We hypothesize that these behaviors are generated from different probabilistic models and that the parameters of these models can be found using the EM algorithm. The instances are assigned to the probability distribution that they most likely belong to, therefore building clusters.

After generating the clusters, the task of the BClus method is to find which of them belong to botnets. The features of a cluster are the average and standard deviation of the seven instances features described in Section 4.2. The following 15 cluster features are obtained for each cluster:

1. Total amount of instances in the cluster.
2. Total amount of NetFlows in the cluster.
3. Amount of source IP addresses.
4. Average amount of unique source ports.
5. Standard Deviation of the amount of unique source ports.
6. Average amount of unique destination IP addresses.
7. Standard Deviation of the amount of unique destination IP addresses.
8. Average amount of unique destination ports.
9. Standard Deviation of the amount of unique destination ports.
10. Average amount of NetFlows.
11. Standard Deviation of the amount of NetFlows.
12. Average amount of bytes transferred.
13. Standard Deviation of the amount of bytes transferred.
14. Average amount of packets transferred.
15. Standard Deviation of the amount of packets transferred.

Once the features are extracted, they are used in the next Subsection to assign the ground-truth labels to the clusters.

4.4. *Train a classification model on the botnet clusters*

The classification algorithm used to find the botnet clusters is JRIP. It is the WEKA's implementation of a "(…) a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), which was proposed by William W. Cohen as an optimized version of IREP" (Hall et al., 2009).

The JRIP algorithm receives a labeled group of clusters and output a group of rules to detect them. That means that the JRIP algorithm needs to be trained on how to recognize a botnet cluster. This training is done with the following leave-one-out algorithm:

1. Training phase.
   (a) Use a leave-one-out algorithm with the training and cross-validation datasets. For each round do:
      i. Separate the NetFlows in time windows (Section 4.1).
      ii. Aggregate the NetFlows by source IP address (Section 4.2).
      iii. Cluster the aggregated NetFlows (Section 4.3).
      iv. Assign ground-truth labels to the clusters based on the ground-truth labels of the NetFlows (Section 4.4.1).
      v. Train a JRIP classification model to recognize the botnet clusters.
      vi. Apply the JRIP model in the cross-validation dataset of this round.
      vii. Store the error metrics of this round.
2. Select the bests JRIP model based on the results of the leave-one-out.
3. Testing phase (Section 4.5).
   (a) Read the testing dataset.
   (b) Use the best JRIP classification model to recognize the botnet clusters.
   (c) Assign the labels to the NetFlows based on the labels of the clusters.

The rest of the Subsections describe each of the steps.

4.4.1. *Assign ground-truth labels to the botnet clusters*

Ground-truth labels should be assigned to the clusters because we need to train the JRIP classification algorithm with them. Once the JRIP algorithm knows which are the botnet clusters, it can create a model to recognize them.

To assign a ground-truth label to a cluster, we should first assign a ground-truth label to all of its instances (aggregated NetFlows). However, to assign a ground-truth label to an instance, we should first assign a ground-truth label to all of its NetFlows.

The ground-truth label of each NetFlow is known from the original NetFlow files that are part of the dataset. Therefore, ground-truth label of each instance is known, since an instance is composed of all the NetFlows from the same IP address. However, a cluster is composed of different instances coming from different IP addresses, and then it is not straightforward to know which ground-truth label should be assigned to a cluster. This Subsection describes how we assigned a ground-truth label to each cluster.

To help us decide which label should be assigned to each cluster, a new feature was computed for each cluster: The percentage of botnet NetFlows on the cluster. This value is expected to be bigger on botnet clusters and smaller on background clusters and it was used to select the ground-truth label for the cluster. Notice that this feature is only used to assign the ground-truth label in the training phase and is not stored nor used in the testing phase.

As this new feature is a percentage, a correct threshold had to be found. This threshold decision is very important, because different percentages correspond to different botnet behaviors. If it is above 0%, it means that every cluster with at least one botnet NetFlow is considered a representative of a botnet behavior. If it is above 1%, it means that only clusters with more that 1% of botnet NetFlows are considered a representative of a botnet behavior. A manual analysis of the dataset determined that most of the real botnet clusters had between 0% and 1% of botnet NetFlows. To find out which threshold between 0% and 1% was the best, we implemented the leave-one-out algorithm described in Section 4.4 to try the following ten candidates thresholds: 0.1%, 0.2%, 0.3%, 0.4%, 0.5%, 0.6%, 0.7%, 0.8%, 0.9% and 1%.

After running the leave-one-out technique we found that the group of clusters that has the best error metrics for the BClus algorithm was generated with a threshold of 0.4%. The set of JRIP rules generated by the 0.4% percentage became the best detection model applied in next Subsection.

4.5. *Testing phase. Use the classification model to recognize the botnet clusters*

Once the best detection model was found in previous Subsection, we applied it in the testing dataset to knew the real performance of the BClus algorithm.

The testing dataset was processed in the same way that the training dataset. That is, it was separated in two minutes time windows, the NetFlows in each time windows were aggregated by its source IP address every one minute and those aggregated instances were clustered. Then, the best JRIP model was applied to detect the botnet clusters.

If a cluster was classified as *botnet*, then all of its instances were labeled as *botnet*, and in turn all of the NetFlows in those instances were labeled as *botnet*. Finally, the BClus method outputted a list of NetFlows with the predicted label assigned.

This list of labeled NetFlows for each testing scenario is the output that will be compared to the CAMNEP and BotHunter methods in Section 8.

5. The BotHunter Method

The BotHunter method was proposed by Gu et al. (2007) to detect the infection and coordination dialog of botnets by matching a state-based infection sequence model. It consists of a correlation engine that aims at detecting specific stages of the malware infection process, such as inbound scanning, exploit usage, egg downloading, outbound bot coordination dialog and outbound attack propagation.

It uses an adapted version of the Snort IDS[12] with two proprietary plugin-ins, called *Statistical Scan Anomaly Detection Engine* (SCADE) and *Statistical Payload Anomaly Detection Engine* (SLADE). SLADE implements a lossy n-gram payload analysis of incoming traffic flows to detect divergences in some protocols. SCADE performs port scan analyzes.

An infection is reported when one of two conditions is satisfied: first, when an evidence of local host infection is

[12] http://www.snort.org.

found and evidence of outward bot coordination or attack propagation is found, and second, when at least two distinct signs of outward bot coordination or attack propagation are found. The BotHunter warnings are tracked over a temporal window and contribute to the infection score of each host.

The BotHunter proposal is compared to the BClus and CAMNEP methods to have the reference of an accepted detection method in the community. The version of BotHunter used in the comparison is 1.7.2.

Section 8.1 describes how the results of the BotHunter proposal were adapted and incorporated into the comparison.

6. Creation of the dataset

In order to compare the methods, a good dataset is needed. According to (Sperotto et al., 2009; Shiravi et al., 2012), a good dataset should be representative of the network were the algorithms are going to be used. This means that it should have botnet, normal and background labeled data, that the balance of the dataset should be like in a real network (usually the percentage of botnet data is small), and that it should be representative of the type of behaviors seen on the network. The difficulties of obtaining such a dataset are discussed in Shiravi et al. (2012) and the importance of these characteristics are discussed in Rossow et al. (2012).

Due to the absence of a public botnet dataset with the characteristics needed, we created a new public dataset that complies with the following design goals:

- Must have real botnets attacks and not simulations.
- Must have unknown traffic from a large network.
- Must have ground-truth labels for training and evaluating the methods.
- Must include different types of botnets.
- Must have several bots infected at the same time to capture synchronization patterns.
- Must have NetFlow files to protect the privacy of the users.

The topology used to create the dataset consisted in a set of virtualized computers running the Microsoft Windows XP SP2 operating system on top of a Linux Debian host. At the time of designing the topology, the Windows XP SP2 was the most used operating system by the malware. Each virtual machine was being bridged into the University network. Fig. 3 shows a diagram of the testbed. The traffic was captured both on the Linux host and on one of the University routers. The traffic from the Linux host was exclusively composed of botnet traffic and was used for labeling purposes. The traffic from the University router was used to create the final dataset. The tool used to capture the traffic was tcpdump (Jacobson et al., 1997).

The next Subsections describe each of the captures, its design principles, the preprocessing of the dataset, the assignment of labels, the separation in training and testing and the publication of the dataset.

6.1. *Design of the botnet scenarios*

A botnet scenario, in the context of this paper, is a particular infection of the virtual machines using a specific malware. Thirteen of these scenarios were created, and each of them was designed to be representative of some malware behavior.

The main characteristics of the scenarios and their behaviors are shown in Table 2. It describes if they used IRC, P2P or HTTP protocols, if they sent SPAM, did Click-Fraud, port scanned, did DDoS attacks, used Fast-Flux techniques or if they were custom compiled.

The features related with the network traffic of each scenario are shown in Table 3. It presents the size, duration, number of packets, number of flows, number of bots and bot family.

The network topology used to make the captures had a bandwidth control mechanism. However, the traffic going out to the Internet was not filtered. This decision may seem controversial (Rossow et al., 2012), but it was taken with the explicit determination of capturing real attacks. We believe

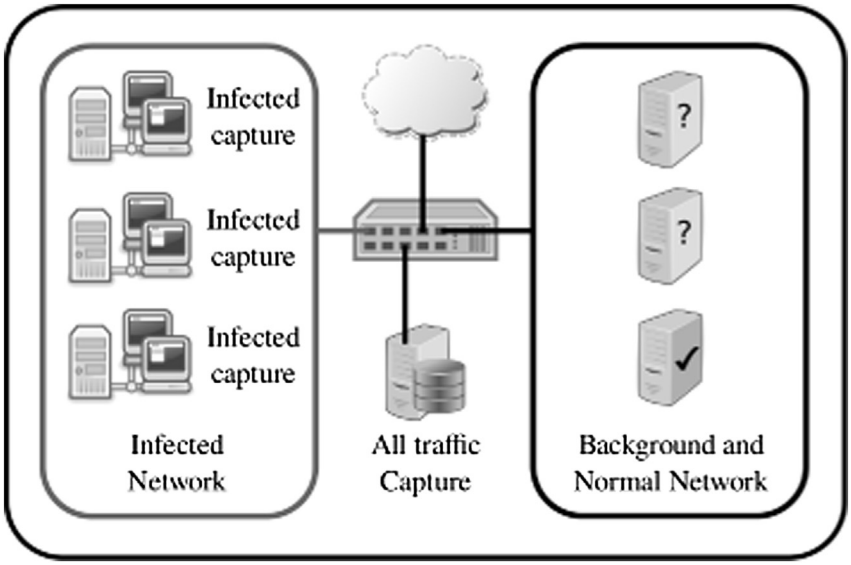

Fig. 3 – Testbed network topology.

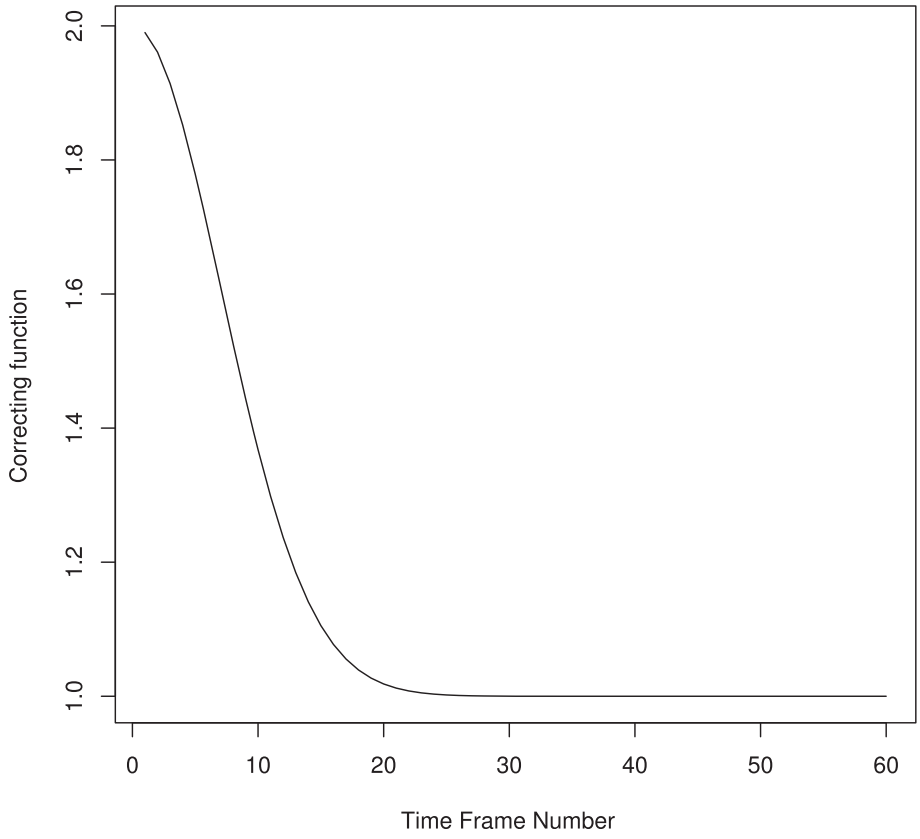

Fig. 4 – Correcting Function applied to 60 time windows.

that the best way to study and model an attack is to capture real attacks.

The next Subsection describes how these scenarios were preprocessed to obtain a more usable dataset.

6.2. *Dataset preprocessing*

After capturing the packets, the dataset was preprocessed and converted to a common format for the detection methods. The format selected was the NetFlow file standard (Clais, 2008), which is considered the ad-hoc standard for network data representation. The conversion from pcap files to NetFlow files was done in two steps using the Argus software suite (Argus, 2013). First, the *argus* tool was used to convert each pcap file into a bidirectional Argus binary storage file. The exact configuration of argus is published with each scenario. Second, the *ra* Argus client tool was used to convert each Argus binary storage file into a NetFlow file. This can be done by specifying in the ra configuration the output fields. The ra configuration is also published with each scenario. These final NetFlow files were composed of the following fields: *Start Time*, *End Time*, *Duration*, *Source IP address*, *Source Port*, *Direction*, *Destination IP address*, *Destination Port*, *State*, *SToS*, *Total Packets* and *Total Bytes*.

Table 2 – Characteristics of the botnet scenarios. (CF: ClickFraud, PS: Port Scan, FF: FastFlux, US: Compiled and controlled by us.)

Id	IRC	SPAM	CF	PS	DDoS	FF	P2P	US	HTTP	Note
1	√	√	√							
2	√	√	√							
3	√			√				√		
4	√				√			√		UDP and ICMP DDoS.
5		√		√					√	Scan web proxies.
6				√						Proprietary C&C. RDP.
7									√	Chinese hosts.
8				√						Proprietary C&C. Net-BIOS, STUN.
9	√	√	√	√						
10	√				√			√		UDP DDoS.
11	√				√			√		ICMP DDoS.
12							√			Synchronization.
13		√		√					√	Captcha. Web mail.

Table 3 – Amount of data on each botnet scenario.

Id	Duration(hrs)	# Packets	#NetFlows	Size	Bot	#Bots
1	6.15	71,971,482	11,231,035	52 GB	Neris	1
2	4.21	71,851,300	7,037,972	60 GB	Neris	1
3	66.85	167,730,395	15,202,061	121 GB	Rbot	1
4	4.21	62,089,135	4,238,045	53 GB	Rbot	1
5	11.63	4,481,167	7,710,910	37.6 GB	Virut	1
6	2.18	38,764,357	2,579,105	30 GB	Menti	1
7	0.38	7,467,139	454,175	5.8 GB	Sogou	1
8	19.5	155,207,799	11,993,935	123 GB	Murlo	1
9	5.18	115,415,321	8,087,513	94 GB	Neris	10
10	4.75	90,389,782	5,180,852	73 GB	Rbot	10
11	0.26	6,337,202	40,836	5.2 GB	Rbot	3
12	1.21	13,212,268	1,262,790	8.3 GB	NSIS.ay	3
13	16.36	50,888,256	6,425,345	34 GB	Virut	1

6.2.1. *Ground-truth labels assignment*

The assignment of ground-truth labels is a very important part of the dataset creation process (Fontugne et al., 2010). However, it can be complex and difficult to do (Davis and Clark, 2011). For example, a wrongly assigned label might produce unreliable results (Maloof, 2006).

Our labeling strategy assigns three different labels: background, botnet and normal. The priority to assign the labels is the following:

1. Assign the Background label to the whole traffic.
2. Assign the Normal label to the traffic that matches certain filters.
3. Assign the Botnet label to all the traffic that comes from or to any of the known infected IP addresses.

The filters used to assign normal labels were created from the known and controlled computers in the network, such as routers, proxies, switches, our own computers in the laboratory, etc.

The distribution of labels on each experiment is shown in Table 4. It can be seen that most of the traffic was labeled as Background. This majority class may add a natural bias to the dataset, however one of the ways to avoid this is to capture a large dataset, as it was sated in by Kotsiantis et al. (2006).

Table 4 – Distribution of labels for each scenario in the dataset.

Id	Background	Botnet	Normal
1	10,124,854 (95.40%)	94,972 (0.89%)	392,433 (3.69%)
2	6,071,419 (95.59%)	54,433 (0.85%)	225,336 (3.54%)
3	14,381,899 (94.60%)	75,891 (0.49%)	744,270 (4.89%)
4	3,895,469 (91.91%)	6466 (0.15%)	336,103 (7.93%)
5	416,267 (91.37%)	2129 (0.46%)	37,144 (8.15%)
6	2,031,967 (94.12%)	4927 (0.22%)	121,854 (5.64%)
7	425,611 (93.71%)	293 (0.06%)	28,270 (6.22%)
8	11,451,205 (95.47%)	12,063 (0.10%)	530,666 (4.42%)
9	6,881,228 (90.22%)	383,215 (5.02%)	362,594 (4.75%)
10	4,535,493 (87.54%)	323,441 (6.24%)	321,917 (6.21%)
11	119,933 (29.33%)	277,892 (67.97%)	11,010 (2.69%)
12	119,933 (29.33%)	277,892 (67.97%)	11,010 (2.69%)
13	1,218,140 (93.76%)	21,760 (1.67%)	59,190 (4.55%)

6.2.2. *Dataset separation into training, testing and cross-validation*

To correctly create the classification models used in the BClus and CAMNEP methods, we need to first separate the dataset.

For the CAMNEP method the training consisted in the first 25 min of each scenario, so it was not necessary to further separate them.

For the BClus method, it was necessary to separate the dataset into training and cross-validation, and testing. The separation criteria was carefully evaluated, because the following constrains must be met:

- The training and cross-validation datasets should be approximately 80% of the dataset.
- The testing dataset should be approximately 20% of the dataset.
- None of the botnet families used in the training and cross-validation dataset should be used in the testing dataset. This ensures that the methods can generalize and detect new behaviors.

However, it is not clear which feature should be used for the 80%–20% separation criteria. It is not the same to have an 80% of the amount of packets than of the amount of bytes. We made the separation by carefully selecting the scenarios so that the 80% of the following features are considered: the *Duration in minutes*, the *Number of clusters*, the *Number of NetFlows* and the *Number of aggregated NetFlows* of the scenarios.

The final separation of the scenarios for the datasets is shown in Table 5. The problem of the imbalanced amount of labels on each dataset was reduced, as stated in Kotsiantis et al. (2006), by carefully selecting the training and testing datasets. Also, as the majority label is *Background*, the bias toward the majority class reported in Li et al. (2010) is avoided.

All the three methods compared used only the testing scenarios for obtaining results.

Table 5 – Dataset separation into training, testing and cross-validation.

Scenario	Dataset
1,2,6,8,9	Testing
3,4,5,7,10,11,12,13	Training and cross-validation

6.3. Dataset publication

The thirteen scenarios of our dataset were published in the web site https://mcfp.felk.cvut.cz/(García, 2013). Each scenario includes the botnet pcap file, the labeled NetFlow file, a README file with the capture time line and the original malware executable binary. It was not possible to publish the complete pcap file with the background and normal packets because they contain private information. However, both of our methods use only the NetFlows files. The correspondence between the number of scenario and the name of the capture in the web page is:

- Scenario Id 1 is CTU-Malware-Capture-Botnet-42.
- Scenario Id 2 is CTU-Malware-Capture-Botnet-43.
- Scenario Id 3 is CTU-Malware-Capture-Botnet-44.
- Scenario Id 4 is CTU-Malware-Capture-Botnet-45.
- Scenario Id 5 is CTU-Malware-Capture-Botnet-46.
- Scenario Id 6 is CTU-Malware-Capture-Botnet-47.
- Scenario Id 7 is CTU-Malware-Capture-Botnet-48.
- Scenario Id 8 is CTU-Malware-Capture-Botnet-49.
- Scenario Id 9 is CTU-Malware-Capture-Botnet-50.
- Scenario Id 10 is CTU-Malware-Capture-Botnet-51.
- Scenario Id 11 is CTU-Malware-Capture-Botnet-52.
- Scenario Id 12 is CTU-Malware-Capture-Botnet-53.
- Scenario Id 13 is CTU-Malware-Capture-Botnet-54.

7. Comparison methodology and new error metric

To compare several detection methods it is necessary to have a methodology, so the comparisons can be repeated and extended. For this purpose we created a simple methodology and a new error metric. The methodology may be used by other researchers to add the results of their methods and obtain a new comparisons. Section 7.1 presents the methodology and Section 7.2 presents the error metric.

7.1. Comparison methodology

When a new botnet detection method using a new dataset needs to be compared with a third-party method, the most usual approach is to try to run the third-party method on the new dataset. However, obtaining the original implementation of a third-party method may be difficult or even impossible due to copyright issues.

The comparison methodology used in this paper is simpler. Instead of trying to implement a third-party method on our dataset, we propose that the researchers first download a common dataset with labels, execute their methods on this common dataset, add its results to the common dataset and then publish the common dataset back.

A dataset made of NetFlows lines with ground truth labels can be easily modified to add a new column with the method's predictions for each NetFlow. In this way, more and more methods will publish their results and more comparisons can be made. The main advantage of this approach is that the details of the methods remain private.

To implement this methodology we created and published a new tool called *Botnet Detectors Comparer* (García, 2014) that it is publicly available for download.[13]

This tool reads the dataset NetFlow file and implements the following steps:

- Separates the NetFlow file in *comparison time windows*.
- Compares the ground-truth NetFlow labels with the predicted labels of each method and computes the TP, TN, FP and FN values.
- After the *comparison time window* ended, it computes the error metrics: FPR, TPR, TNR, FNR, Precision, Accuracy, ErrorRate and FMeasure1 for that time window.
- When the dataset ends, it computes the final error metrics.
- The error metrics are stored in a text file and plotted in a eps image.

Also, this tool computes the new error metric that we propose in next Section 7.2.

The *comparison time window* is the time window used for computing the error metrics and it is not related with the methods. It is the time that the network administrator may wait to have a decision about the traffic. In our methodology the width of the *comparison time windows* is five minutes.

Using this methodology, researchers can now add its own predictions to the NetFlows files of our dataset and use this tool to compute the error metrics. To tell the tool which labels the new method uses for its predictions, they should be added to the header of the NetFlow file as a new column with the format "NameOfNewMethod(NormalLabelUsed: BotnetLabelUsed: BackgroundLabelUsed)".

The next Subsection describes the new error metric proposed to compare botnet detection methods.

7.2. New error metric

The error metrics usually used by researchers to analyze their results (e.g. FPR, FMeasure) were historically designed from a statistical point of view, and they are really good to measure differences and to compare most methods. But the needs of a network administrator that is going to use a detection method are slightly different. These error metrics should have a meaning that can be translated to the network. This has been called the semantic gap by Rossow et al. (2012). It is possible that the common error metrics are not enough for a network administrator (García et al., 2013).

For example, according to the classic definition, a False Positive should be detected every time that a normal NetFlow is detected as botnet. However, a network administrator might want to detect a small amount of infected IP addresses instead of hundreds of NetFlows. Furthermore, she may need to detect them as soon as possible. However, these needs are not satisfied in the classic error metrics.

[13] http://downloads.sourceforge.net/project/botnetdetectorscomparer/BotnetDetectorsComparer-0.9.tgz.

The type of error metric that may be useful for a network administrator may be also useful for comparing the methods that she is going to use.

Therefore, we have created a new set of error metrics in an attempt to solve this issue that adhere to the following principles:

- Errors should account for IP addresses instead of NetFlows.
- To detect a botnet IP address (TP) early is better than latter.
- To miss a botnet IP address (FN) early is worst than latter.
- The value of detecting a normal IP address (TN) is not affected by time.
- The value of missing a normal IP address (FP) is not affected by time.

The first step is to incorporate the time to the metrics by computing the errors in *comparison time frames*. These time frames are only used to compute the errors and are independent of the detection methods.

The second step was to migrate from a NetFlow-based detection to an IP-based detection. The classical error values (TP, FP, TN, FN) were redefined as follows:

- c_TP: A True Positive is accounted when a Botnet IP address is detected as Botnet at least **once** during the comparison time frame.
- c_TN: A True Negative is accounted when a Normal IP address is detected as Non-Botnet during the **whole** comparison time frame.
- c_FP: A False Positive is accounted when a Normal IP address is detected as Botnet at least **once** during the comparison time frame.
- c_FN: A False Negative is accounted when a Botnet IP address is detected as Non-Botnet during the **whole** comparison time frame.

The third step was to modify the values of the error metrics by adding a time-based *correcting_function* that is defined as follows:

$$correcting\_function = e^{(-\alpha \ * \ N^{\circ} \ \ comparison \ \ time \ \ frame)} + 1 \quad (1)$$

This function depends on the ordinal number of the time frame were it is computed and on the α value. Fig. 4 shows this monotonically decreasing function that weights the values according to the time frame number. The main idea is to weight the values more on the firsts time frames and to weight them less on the lasts ones. The α value is used to manually fit the function according to the capture length. The α value used for our comparison was 0.01.

Using this *correcting_function*, four time-dependent error metrics were created, called tTP, tTN, tFP and tFN. Note that they use the previously defined IP-based error metrics. They are computed as follows:

- tTP:

$$\frac{c\_TP * correcting\_function}{N^{\circ} \text{ of unique botnet IP addresses in the comparison time frame}} \quad (2)$$

- tFN:

$$\frac{c\_FN * correcting\_function}{N^{\circ} \text{ of unique botnet IP addresses in the comparison time frame}} \quad (3)$$

- tFP:

$$\frac{c\_FP}{N^{\circ} \text{ of unique normal IP addresses in the comparison time frame}} \quad (4)$$

- tTN:

$$\frac{c\_TN}{N^{\circ} \text{ of unique normal IP addresses in the comparison time frame}} \quad (5)$$

These time-based error metrics allow for a more realistic comparison between detection algorithms. An algorithm weights better if it can detect sooner all the infected IP addresses without error. To miss an infected IP address at the

Table 6 – References for algorithms names.

Algorithm name	Reference
Flags-FOG-srcIP.src.fog-1.00	Fs1
Flags-FOG-srcIP.src.fog-1.50	Fs1.5
Flags-FOG-dstIP.dst.fog-1.00	Fd1
Flags-FOG-srcIP.src.fog-2.00	Fs2
Flags-FOG-dstIP.dst.fog-1.50	Fd1.5
Flags-FOG-dstIP.dst.fog-2.00	Fd2
Minds-1.00	Mi1
Xu-1.00	X1
Xu-1.50	X1.5
Minds-1.50	Mi1.5
Minds-2.00	Mi2
LakhinaEntropyGS-1.00	Le1
KGBFog-sIP.src.fog-1.00	Ko1
KGBFog-sIP.src.fog-1.50	Ko1.5
MasterAggregator-1.00	CA1
TAPS3D-1.50	T1.5
TAPS3D-1.00	T1
KGBF-sIP.src.f-1.00	K1
KGBF-sIP.src.f-2.00	K2
AllNegative	AllNeg
AllPositive	AllPo
BClus	BClus
XuDstIP-1.50	Xd1.5
XuDstIP-2.00	Xd2
TAPS3D-2.00	T2
LakhinaVolumeGS-1.50	Lv1.5
MasterAggregator-1.50	CA1.5
LakhinaVolumeGS-2.00	Lv2
MasterAggregator-2.00	CA2
Xu-2.00	X2
KGBF-sIP.src.f-1.50	K1.5
XuDstIP-1.00	Xd1
LakhinaEntropyGS-1.50	Le1.5
LakhinaEntropyGS-2.00	Le2
LakhinaVolumeGS-1.00	Lv1
KGBFog-sIP.src.fog-2.00	Ko2
AllBackground	AllBac
BotHunter	BH

Table 7 – Comparison of error metrics for the methods in Scenario 1.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
AllPo	65.5	0	69	0	1	0	1	0	0.4	0.4	0.5	0.65
BClus	**30.2**	**41.3**	**27.6**	**35.3**	**0.4**	**0.5**	**0.4**	**0.5**	**0.5**	**0.5**	**0.4**	**0.48**
Fs1	7.8	66.4	2.5	57.5	0.1	0.9	<0.0	0.8	0.7	0.5	0.4	0.20
Fs1.5	6.3	67.2	1.7	59.1	<0.0	0.9	<0.0	0.9	0.7	0.5	0.4	0.17
Fd1	6.8	54.2	14.6	58.6	0.1	0.7	0.2	0.8	0.3	0.4	0.5	0.15
Fs2	4	67.6	1.3	61.4	<0.0	0.9	<0.0	0.9	0.7	0.5	0.4	0.11
Fd1.5	4.6	57.5	11.4	60.8	<0.0	0.8	0.1	0.9	0.2	0.4	0.5	0.11
Fd2	2.2	59.8	9.1	63.2	<0.0	0.8	0.1	0.9	0.1	0.4	0.5	0.05
Mi1	2.3	52.3	16.6	63.1	<0.0	0.7	0.2	0.9	0.	0.4	0.5	0.05
X1	1.7	68.6	0.3	63.6	<0.0	0.9	<0.0	0.9	0.8	0.5	0.4	0.05
X1.5	1.5	68.6	0.3	63.9	<0.0	0.9	<0.0	0.9	0.8	0.5	0.4	0.04
BH	**1.59**	**73.8**	**0.18**	**109**	**0.01**	**0.9**	**<0.0**	**0.9**	**0.8**	**0.4**	**0.5**	**0.02**
Mi1.5	1	56.9	12	64.4	<0.0	0.8	0.1	0.9	<0.0	0.4	0.5	0.02
Mi2	0.6	63.1	5.8	64.8	<0.0	0.9	<0.0	0.9	<0.0	0.4	0.5	0.01
Le1	0.2	68.1	0.8	65.2	<0.0	0.9	0.01	0.9	0.2	0.5	0.4	0.007
Ko1	0.1	68.7	0.1	65.3	<0.0	0.9	<0.0	0.9	0.4	0.5	0.4	0.004
Ko1.5	0.08	68.9	0.02	65.3	<0.0	1	0	0.9	0.7	0.5	0.4	0.002
CA1	**0.005**	**68.7**	**0.2**	**65.4**	**0**	**0.9**	**<0.0**	**1**	**<0.0**	**0.5**	**0.4**	**<0.00**
T1.5	0.005	68.9	0	65.4	0	1	0	1	1	0.5	0.4	<0.00
T1	0.005	68.9	0	65.4	0	1	0	1	1	0.5	0.4	<0.00

beginning is more costly than to falsely detect an infected IP address at the beginning. After some time frames, all the error values are weighted the same.

With these time-based error metrics we can now compute new corresponding rates. They are like the classic ones, but are redefined to use the time-based values:

- $FPR = \frac{tFP}{tTN+tFP}$
- $TPR = \frac{tTP}{tTP+tFN}$
- $TNR = \frac{tTN}{tTN+tFP}$
- $FNR = \frac{tFN}{tTP+tFN}$
- $Precision = \frac{tTP}{tTP+tFP}$
- $Accuracy = \frac{tTP+tTN}{tTP+tTN+tFP+tFN}$
- $ErrorRate = \frac{tFN+tFP}{tTP+tTN+tFP+tFN}$
- $F\ Measure1 = 2 * \frac{Precision*TPR}{Precision+TPR}$ (FMeasure with beta = 1)

This error metric is implemented in the *Botnet Detectors Comparer* tool that it is publicly available for download and described in previous Subsection.

8. Comparison of the results of the detection methods

The three detection methods were executed on each of the five testing datasets described in Section 6.2.2. Each method added its flow predictions to each dataset file, so there are five files to compare using the methodology described in Section 7.

To better understand the implications of comparing these results, the following baseline algorithms were added: the *AllPositive* algorithm, that always predicts Botnet, the *AllNegative* algorithm that always predicts Normal and the *AllBackground* algorithm that always predicts Background. They are analyzed alongside with the BClus, CAMNEP and BotHunter algorithm. The names of all the algorithms compared are encoded in Table 6.

The next Subsections compare the results on each of the five testing scenarios of the dataset.

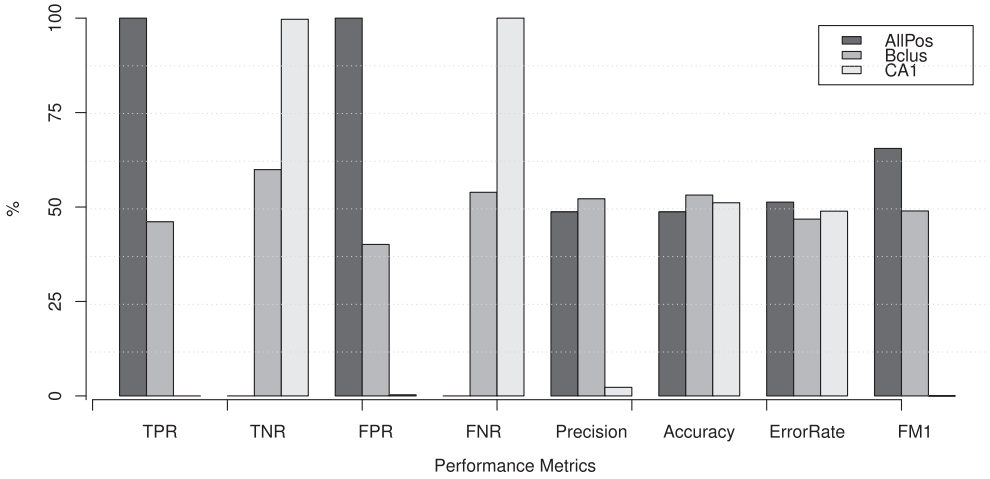

Fig. 5 – Simplified comparison of error metrics for the BClus, CAMNEP and AllPositive algorithms on Scenario 1.

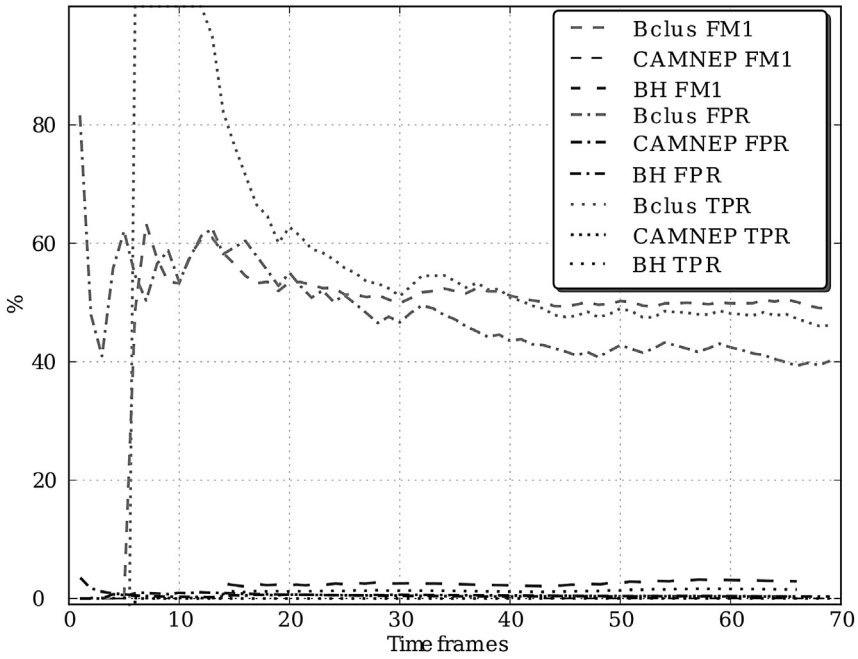

Fig. 6 – Comparison of the running error metrics for Scenario 1.

8.1. *Adaptation of the BotHunter results*

The comparison of the results is done by reading the ground-truth label of each NetFlow and comparing it to the predicted label of each NetFlow. However, BotHunter does not read NetFlows files and does not output a prediction label for each NetFlow, making the comparison more difficult.

To solve this issue we run BotHunter on the original pcap files and we obtained, for each pcap, a list of alerts. These alerts include the date, the name of the alert, the protocol, the source IP address, source port, the destination IP address and destination port. Then, we searched which was the NetFlow corresponding to that alert and we assigned it the label *Botnet*. The rest of the NetFlows were labeled as *Normal*.

With this labels assignment procedure it was possible to add the BotHunter method to the comparison.

Next Subsections compare the results on each scenario.

8.2. *Comparison of results in Scenario 1*

This scenario corresponds to an IRC-based botnet that sent spam for almost six and a half hours.

Table 8 – Comparison of error metrics for the methods in Scenario 2.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
AllPo	49.9	0	47	0	1	0	1	0	0.5	0.5	0.4	0.68
BClus	**15.6**	**37.1**	**9.8**	**34.2**	**0.3**	**0.7**	**0.2**	**0.6**	**0.6**	**0.5**	**0.4**	**0.41**
Fd1	14.4	36.5	10.4	35.5	0.2	0.7	0.	0.7	0.5	0.5	0.4	0.38
Fd1.5	9.3	39.1	7.8	40.5	0.1	0.8	0.1	0.8	0.5	0.5	0.5	0.27
Fd2	7.9	40.7	6.2	42	0.1	0.8	0.1	0.8	0.5	0.5	0.4	0.24
Fs1	6.8	45.9	1	43	0.1	0.9	<0.0	0.8	0.8	0.5	0.4	0.23
Fs1.5	6	46.3	0.6	43.8	0.1	0.9	<0.0	0.8	0.8	0.5	0.4	0.21
X1	5.3	46.7	0.2	44.5	0.1	0.9	<0.0	0.8	0.9	0.5	0.4	0.19
X1.5	4.3	46.8	0.1	45.6	<0.0	0.9	<0.0	0.9	0.9	0.5	0.4	0.15
Fs2	4.2	46.5	0.4	45.7	<0.0	0.9	<0.0	0.9	0.9	0.5	0.4	0.15
BH	**1.65**	**46.9**	**0.05**	**75**	**0.02**	**0.99**	**<0.0**	**0.9**	**0.9**	**0.3**	**0.6**	**0.04**
Mi1	1.1	35.8	11.1	48.8	<0.0	0.7	0.2	0.9	<0.0	0.3	0.6	0.03
Mi1.5	0.6	39.1	7.8	49.2	<0.0	0.8	0.1	0.9	<0.0	0.4	0.5	0.02
X2	0.5	46.9	0.02	49.4	<0.0	1	0	0.9	0.9	0.4	0.5	0.02
CA1	**0.2**	**46.9**	**0.04**	**49.6**	**<0.0**	**0.9**	**<0.0**	**0.9**	**0.8**	**0.4**	**0.5**	**0.01**
Ko1	0.1	46.9	0.08	49.8	<0.0	0.9	<0.0	0.9	0.6	0.4	0.5	0.005
Mi2	0.1	46.3	0.6	49.8	<0.0	0.9	<0.0	0.9	0.1	0.4	0.5	0.005
Le1	0.1	46.1	0.8	49.8	<0.0	0.9	<0.0	0.9	0.1	0.4	0.5	0.005
Lv1	0.1	46.6	0.3	49.8	<0.0	0.9	<0.0	0.9	0.2	0.4	0.5	0.004
Xd1	0.07	44	2.	49.8	<0.0	0.9	<0.0	0.9	<0.0	0.4	0.5	0.003
T1	0.03	47	0	49.9	<0.0	1	0	0.9	1	0.4	0.5	0.001

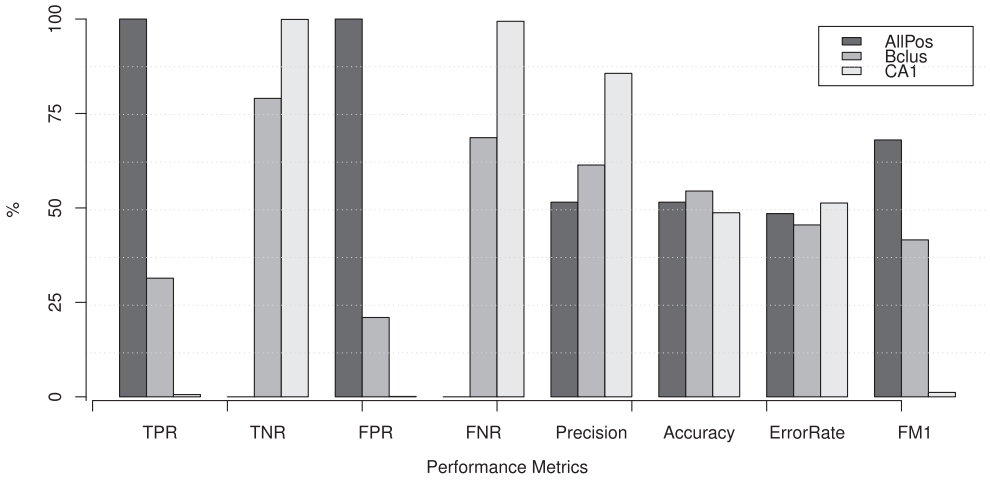

Fig. 7 – Simplified comparison of error metrics for the BClus, CAMNEP and AllPositive algorithms on Scenario 2.

The error metrics for this scenario are shown in Table 7, which is ordered by FMeasure1. This Table shows that the AllPositive algorithm had the best FMeasure, although it had a 100% FPR. The BClus algorithm had a FMeasure of 0.48 and an FPR of 40%. The BotHunter algorithm had a FMeasure of 0.02 and an FNR of 98%. The CAMNEP (CA1) algorithm had a low FMeasure and low FPR. The bold text in Tables 7,8,9,10 and 11 identifies the main algorithms compared in this work, i.e. BClus, CAMNEP and BotHunter. The rest are the internal CAMNEP algorithms and the All Positive algorithm.

A simplified comparison between the BClus, CAMNEP and AllPossitive algorithms is shown in Fig. 5. Although the AllPositive algorithm had a 100% TPR and FPR, it can be seen that it had a Precision, Accuracy and ErrorRate around 50% and a FMeasure of more than 60%. The BClus algorithm had between 40% and 60% for the TPR, FPR, TNR and FNR metrics and nearly 50% for the FMeasure. The CAMNEP algorithm had a value near 0% for the TPR, near 1% for the TNR and near 0% for the FMeasure.

The apparently good results of the AllPositive algorithm may have an explanation. This algorithm predicts always Botnet, which gives a TPR of 100%, a precision of 50% and a FMeasure of 66%. However, these metrics were computed using only the Botnet and Normal labels and omitting the Background labels. The Background labels were not used for computing the error metrics because they were neither Normal nor Botnet. Therefore, the only traffic that this algorithm can mis-classify is the Normal traffic. However, the amount of Normal traffic in the dataset is considerably

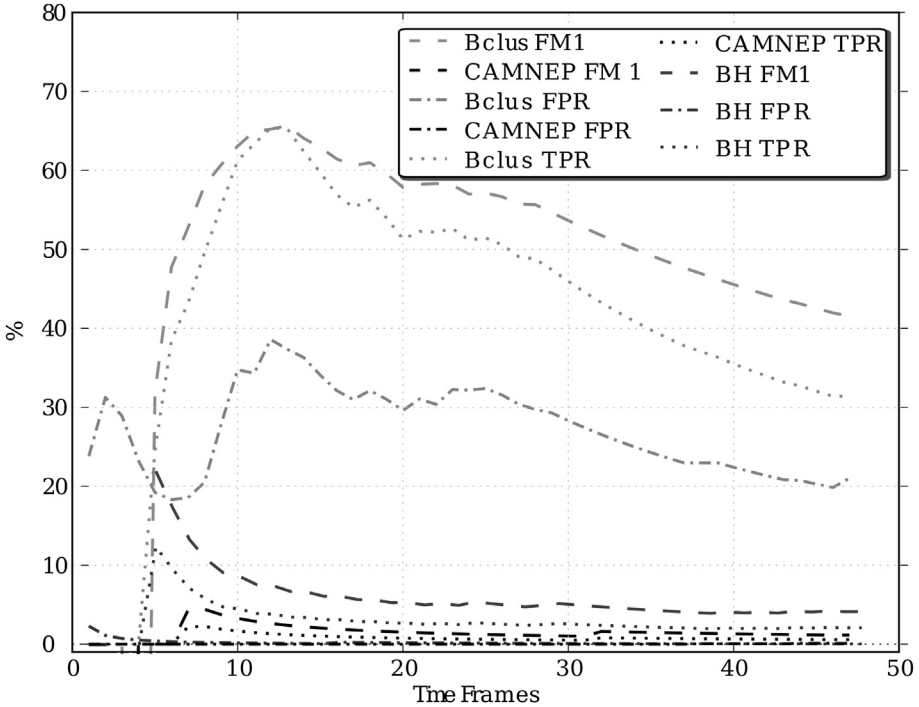

Fig. 8 – Comparison of the running error metrics for Scenario 2.

Table 9 – Comparison of error metrics for the methods in Scenario 6.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
Fs1	22.5	20.7	0.2	6.8	0.7	0.9	<0.0	0.2	0.9	0.8	0.1	0.86
Fd1	23.2	17.6	3.3	6	0.7	0.8	0.1	0.2	0.8	0.8	0.1	0.83
Fs1.5	20.8	20.8	0.1	8.5	0.7	0.9	<0.0	0.2	0.9	0.8	0.1	0.82
Fd1.5	19.9	18.6	2.3	9.3	0.6	0.8	0.1	0.3	0.8	0.7	0.2	0.77
Fd2	19.1	19.3	1.6	10.1	0.6	0.9	<0.0	0.3	0.9	0.7	0.2	0.76
Fs2	17.7	20.8	0.1	11.5	0.6	0.9	<0.0	0.3	0.9	0.7	0.2	0.75
AllPo	29.3	0	21	0	1	0	1	0	0.5	0.5	0.4	0.73
X1	5.7	20.9	0.04	23.6	0.1	0.9	<0.0	0.8	0.9	0.5	0.4	0.32
Xd1.5	3.6	17.3	3.6	25.7	0.1	0.8	0.1	0.8	0.5	0.4	0.5	0.19
BH	**2.53**	**20.9**	**0.02**	**37.3**	**0.06**	**0.99**	**<0.0**	**0.93**	**0.98**	**0.38**	**0.61**	**0.11**
Xd2	3.6	17.3	3.6	25.7	0.1	0.8	0.1	0.8	0.5	0.4	0.5	0.19
Xd1	3.6	17.3	3.6	25.7	0.1	0.8	0.1	0.8	0.4	0.4	0.5	0.19
X1.5	1.4	21	0	27.8	<0.0	1	0	0.9	1	0.4	0.5	0.09
CA1	**0.6**	**20.9**	**0.07**	**28.6**	**<0.0**	**0.9**	**<0.0**	**0.9**	**0.9**	**0.4**	**0.5**	**0.04**
BClus	**0.6**	**20.2**	**0.7**	**28.6**	**<0.0**	**0.9**	**<0.0**	**0.9**	**0.4**	**0.4**	**0.5**	**0.04**

smaller than the rest. This imbalance made the AllPositive have better results than it should. This algorithm is useful as a baseline for evaluating detection methods and datasets, but it is useless in a real network.

To better appreciate the inner workings of the detection methods during the analysis of this scenario, we plotted the accumulated and running error metrics for each comparison time frame in Fig. 6. This Figure shows that the FPR of the BClus method was high on the first time frames, but after that it kept going down until the final 40%. On the sixth time frame the BClus method started to detect botnets with a 100% TPR until near the twelfth time frame. While still having a huge amount of FP, the BClus method managed to have a final FMeasure of 48%. The CAMNEP and BotHunter algorithms had low values during all the scenario.

8.3. *Comparison of results in Scenario 2*

In this scenario, the same IRC-based botnet as scenario 1 sent SPAM for 4.21 h.

The error metrics for this scenario are shown in Table 8. The AllPositive algorithm had the best FMeasure. The BClus algorithm had a FMeasure of 0.41 and an FPR of 20%. The BotHunter algorithm had an FMeasure of 0.04 and an FNR of 97%. The CAMNEP algorithm had a FMeasure of 0.01 and a very small FPR.

The simplified comparison for this scenario is shown in Fig. 7. Although it was the same bot that scenario 1 and it performed almost the same actions, all the algorithms gave different results. The CAMNEP method still have a large amount of tFN, but despite the low 1% FMeasure, it was 55 times better than itself on scenario 1. Its Precision was high because there were almost no tFP, independently of the amount of tTP. Regarding the BClus method, it had a lower TPR than on scenario 1, but also a lower FPR, which lead to a comparatively better FMeasure value. This scenario is a good example of the variability in the network due to the presence of Background traffic. The same bot, generating the same type and amount of traffic obtained different error metrics.

The inner workings of the algorithms can be seen in the running metrics shown in Fig. 8. The BClus method started with a large FPR, but after the fifth time frame it started to detect botnets correctly and its FMeasure value improved. The TPR, FPR and FMeasure values for the BClus method decreased until the end of the scenario, suggesting that the final values could be even lower. The BotHunter algorithm had an FM1

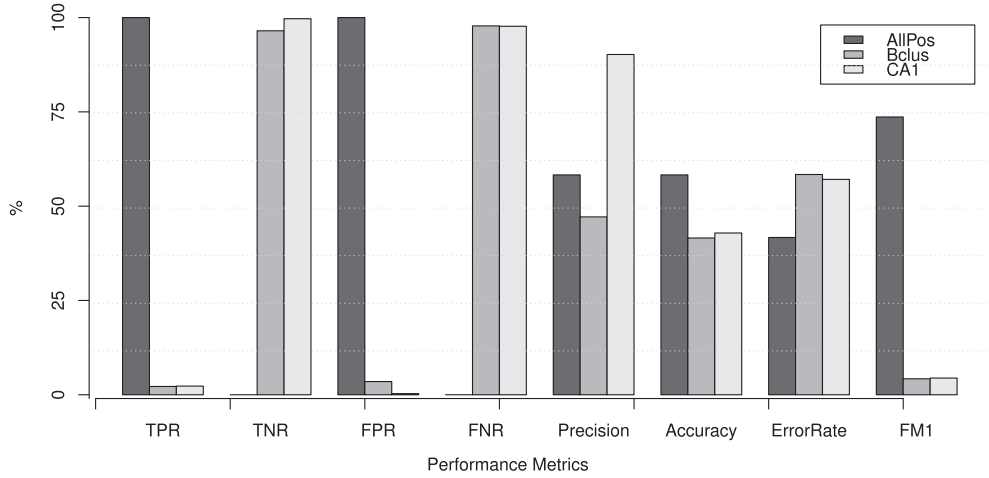

Fig. 9 – Simplified comparison of error metrics for the BClus, CAMNEP and AllPositive algorithms on Scenario 6.

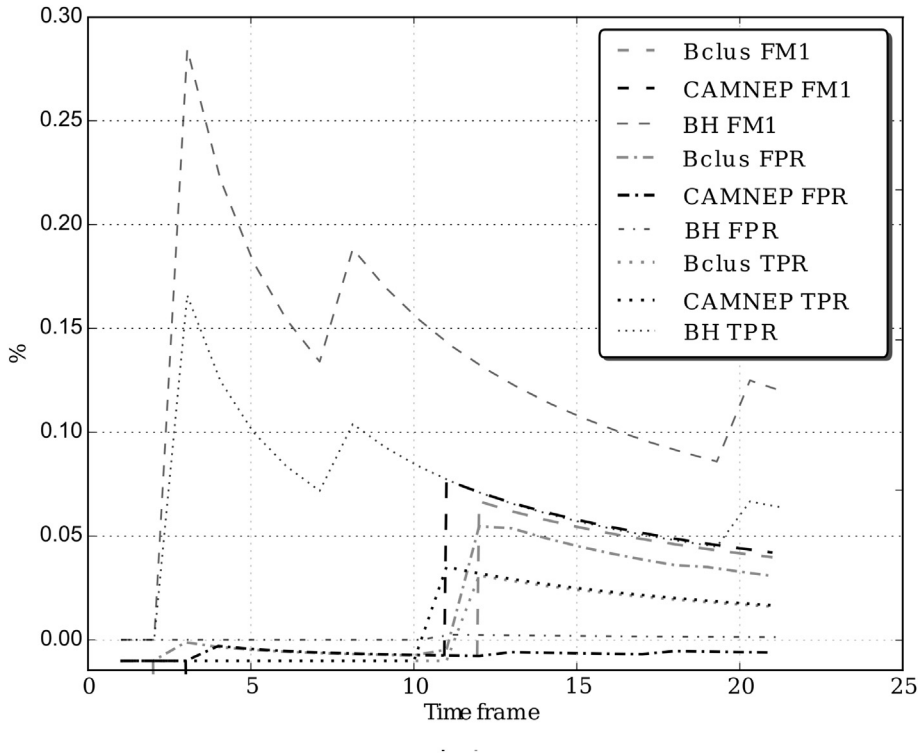

Fig. 10 – Comparison of the running error metrics for Scenario 6.

close to 20% on the first time frames but then it quickly dropped to 0.04. The CAMNEP error metrics remained low during the whole scenario.

8.4. *Comparison of results in Scenario 6*

The botnet in this scenario scanned SMTP (Simple Mail Transfer Protocol) servers for two hours and connected to several RDP (Remote Desktop Protocol) services. However, it did not send any SPAM and did not attack. The C&C server used a proprietary protocol that connected every 33 s and sent an average of 5500 bytes on each connection.

The error metrics for this scenario can be seen in Table 9. The AllPositive algorithm had a Fmeasure of 0.73, far better than the FMeasures of BClus and CAMNEP, which were both 0.04. The BotHunter algorithm had a better FMeasure than the

Table 10 – Comparison of error metrics for the methods in Scenario 8.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
AllPo	233.4	0	230	0	1	0	1	0	0.5	0.5	0.4	0.67
Fs1	74.4	220.7	9.2	159	0.3	0.9	<0.0	0.6	0.8	0.6	0.3	0.46
Ko1	70.7	228.9	1.08	162.7	0.3	0.9	<0.0	0.6	0.9	0.6	0.3	0.46
Ko1.5	68.4	229.3	0.6	164.9	0.2	0.9	<0.0	0.7	0.9	0.6	0.3	0.45
Ko2	53.4	229.5	0.4	180	0.2	0.9	<0.0	0.7	0.9	0.6	0.3	0.37
Fs1.5	54.6	222.7	7.2	178.8	0.2	0.9	<0.0	0.7	0.8	0.5	0.4	0.36
Fs2	28	224.9	5	205.3	0.1	0.9	0.02	0.8	0.8	0.5	0.4	0.21
BClus	**23.5**	**152.8**	**77.1**	**209.9**	**0.1**	**0.6**	**0.3**	**0.8**	**0.2**	**0.3**	**0.6**	**0.14**
Fd1	15.6	196.1	33.8	217.7	<0.0	0.8	0.14	0.9	0.3	0.4	0.5	0.1
Fd1.5	14.1	203.9	26	219.2	<0.0	0.8	0.1	0.9	0.3	0.4	0.5	0.10
CA1	**10.8**	**229.8**	**0.1**	**222.6**	**<0.0**	**0.9**	**<0.0**	**0.9**	**0.9**	**0.5**	**0.4**	**0.08**
Fd2	11.6	209.8	20.1	221.8	<0.0	0.9	<0.0	0.9	0.3	0.4	0.5	0.08
X1	8.5	229.7	0.2	224.9	<0.0	0.9	<0.0	0.9	0.9	0.5	0.4	0.07
Mi1	4.6	213.1	16.8	228.7	<0.0	0.9	<0.0	0.9	0.2	0.4	0.5	0.03
Xd1.5	3.9	194.1	35.8	229.5	<0.0	0.8	0.1	0.9	<0.0	0.4	0.5	0.02
Xd2	3.9	194.4	35.5	229.5	<0.0	0.8	0.1	0.9	<0.0	0.4	0.5	0.02
Xd1	3.9	193.5	36.4	229.5	<0.0	0.8	0.1	0.9	<0.0	0.4	0.5	0.02
Mi1.5	1	219.2	10.7	232.4	<0.0	0.9	<0.0	0.9	<0.0	0.4	0.5	0.008
X1.5	0.5	230	0	232.8	<0.0	1	0	0.9	1	0.4	0.5	0.005
BH	**0**	**229**	**0.11**	**309**	**0**	**0.99**	**<0.0**	**1**	**0**	**0.42**	**0.57**	–

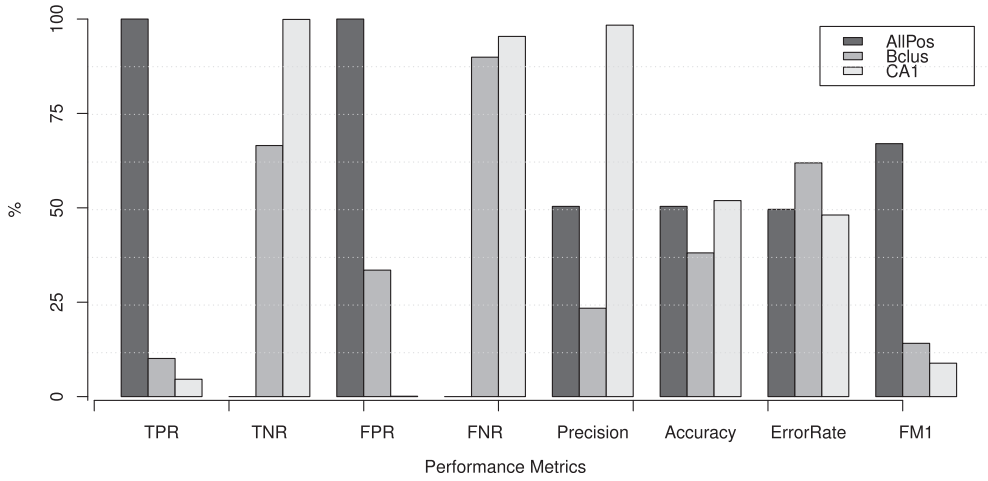

Fig. 11 – Simplified comparison of error metrics for the BClus, CAMNEP and AllPositive algorithms on Scenario 8.

BClus and CAMNEP algorithms because some of the IP addresses used in the SMTP connections where blacklisted in its static detection rules as part of the RBN (Russian Business Network). It should be noted that the scenarios were captured on August 2011 and the BotHunter rules are from January 2013, so it is possible that these IP addresses were blacklisted after the capture.

The simplified comparison for this scenario is shown in Fig. 9. The CAMNEP and the BClus methods behaved almost identically. The only difference is that the BClus method had ten times more FPR and therefore the CAMNEP method had a better Precision and a slightly better FMeasure.

The inner workings of the algorithms can be seen in the running metrics shown in Fig. 10. This Figure shows that both BClus and CAMNEP methods detected tFP values until half of the scenario, and then they had some tTP. However, the tTPs were not enough to improve the FMeasure1 significantly.

8.5. *Comparison of results in Scenario 8*

In this scenario, the botnet contacted a lot of different Chinese C&C hosts and received large amounts of encrypted data. It also scanned and cracked the passwords of machines using the DCERPC protocol both on Internet and on the local network for 19 h. There were more attacks during more time.

The error metrics for this scenario can be seen in Table 10. The AllPositive algorithm had the best FMeasure. Six algorithms were better than BClus, who had a FMeasure of 0.14

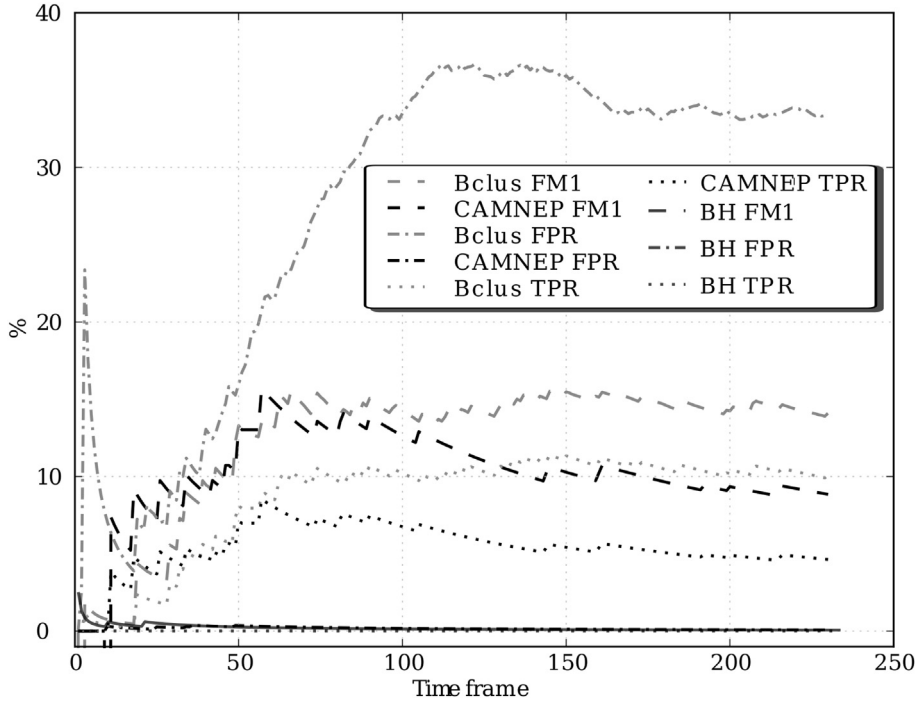

Fig. 12 – Comparison of the running error metrics for Scenario 8.

Table 11 – Comparison of error metrics for the methods in Scenario 9.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
AllPo	58.3	0	58	0	1	0	1	0	0.5	0.5	0.4	0.66
Fd1	21.7	47.1	10.8	36.6	0.3	0.8	0.1	0.6	0.6	0.5	0.4	0.47
Fd1.5	17.7	49.2	8.7	40.6	0.3	0.8	0.1	0.6	0.6	0.5	0.4	0.41
Fd2	13.9	50.9	7.01	44.4	0.2	0.8	0.1	0.7	0.6	0.5	0.4	0.35
Xd1	10.3	48.5	9.4	48.0	0.1	0.8	0.1	0.8	0.5	0.5	0.4	0.26
BClus	**10.1**	**46.4**	**11.5**	**48.2**	**0.1**	**0.8**	**0.2**	**0.8**	**0.4**	**0.4**	**0.5**	**0.25**
Fs1	8.3	55.2	2.7	50	0.1	0.95	<0.0	0.8	0.7	0.5	0.4	0.23
Fs1.5	7.8	55.7	2.2	50.4	0.1	0.9	<0.0	0.8	0.7	0.5	0.4	0.23
Xd1.5	7.04	53.3	4.6	51.3	0.1	0.9	<0.0	0.8	0.6	0.5	0.4	0.2
CA1	**5.5**	**57.7**	**0.2**	**52.8**	**<0.0**	**0.9**	**<0.0**	**0.9**	**0.9**	**0.5**	**0.4**	**0.17**
Fs2	4.3	56.6	1.3	54	<0.0	0.9	<0.0	0.9	0.7	0.5	0.4	0.13
X1	2.8	56.9	1.09	55.5	<0.0	0.9	<0.0	0.9	0.7	0.5	0.4	0.09
Mi1	2.9	47.3	10.6	55.4	<0.0	0.8	0.1	0.9	0.2	0.4	0.5	0.08
CA1.5	2.3	57.8	0.1	55.9	<0.0	0.9	<0.0	0.9	0.9	0.5	0.4	0.07
Mi1.5	2.2	51.3	6.6	56.1	<0.0	0.8	0.1	0.9	0.2	0.4	0.5	0.06
BH	**1.76**	**57.9**	**0.06**	**86.9**	**0.02**	**0.9**	**<0.0**	**0.9**	**0.9**	**0.4**	**0.5**	**0.03**
X1.5	0.9	57.3	0.6	57.4	<0.0	0.9	<0.0	0.9	0.6	0.5	0.4	0.03
Mi2	0.4	57.2	0.7	57.9	<0.0	0.9	<0.0	0.9	0.3	0.4	0.5	0.01
Ko1	0.2	57.8	0.1	58.1	<0.0	0.9	<0.0	0.9	0.6	0.4	0.5	0.008
Le1	0.1	57.2	0.7	58.1	<0.0	0.9	<0.0	0.9	0.1	0.4	0.5	0.006
X2	0.1	57.9	0.05	58.2	<0.0	0.9	<0.0	0.9	0.6	0.4	0.5	0.004
Ko1.5	0.06	57.9	0.03	58.3	<0.0	0.9	<0.0	0.9	0.6	0.4	0.5	0.002
Lv1	0.04	57.6	0.3	58.3	<0.0	0.9	<0.0	0.9	0.1	0.4	0.5	0.001
T1	0.04	58	0	58.3	<0.0	1	0	0.9	1	0.4	0.5	<0.00
CA2	0.04	58	0	58.3	<0.0	1	0	0.9	1	0.4	0.5	<0.00
Le1.5	0.03	57.7	0.2	58.3	<0.0	0.9	<0.0	0.9	0.1	0.4	0.5	<0.00
T1.5	0.02	58	0	58.3	0	1	0	1	1	0.4	0.5	<0.00
Ko2	0.02	57.9	0.01	58.3	0	1	0	1	0.5	0.4	0.5	<0.00

and an FPR of 30%. The CAMNEP algorithm had a FMeasure of 0.08 and a very low FPR. The BotHunter algorithm could not detect a single TP, so it was not possible to compute its FMeasure.

The simplified comparison for this scenario is shown in Fig. 11. The BClus method had a low TPR of about 10%, however it was more than twice of CAMNEPs TPR value. The TNR value was near 60% for BClus and near 90% for CAMNEP. The FPR of BClus was high, near 30%, however it had a FMeasure value that is twice the value for CAMNEP.

The inner workings of the algorithms can be seen in the running metrics shown in Fig. 12. This Figure shows that none of the error metrics exceeded 40%. It means that this scenario was very difficult for the methods. Until the fiftieth time frame both BClus and CAMNEP TPR values grew at almost the same rate. However, after that, the BClus method grew a little faster. The FPR of the BClus method was very high almost from the start of the scenario. The BotHunter algorithm had very low measurements during the whole scenario.

8.6. *Comparison of results in Scenario 9*

In this scenario, ten host were infected using the same Neris botnet as in scenario 1 and 2. For five hours, more than 600 SPAM mails were successfully sent.

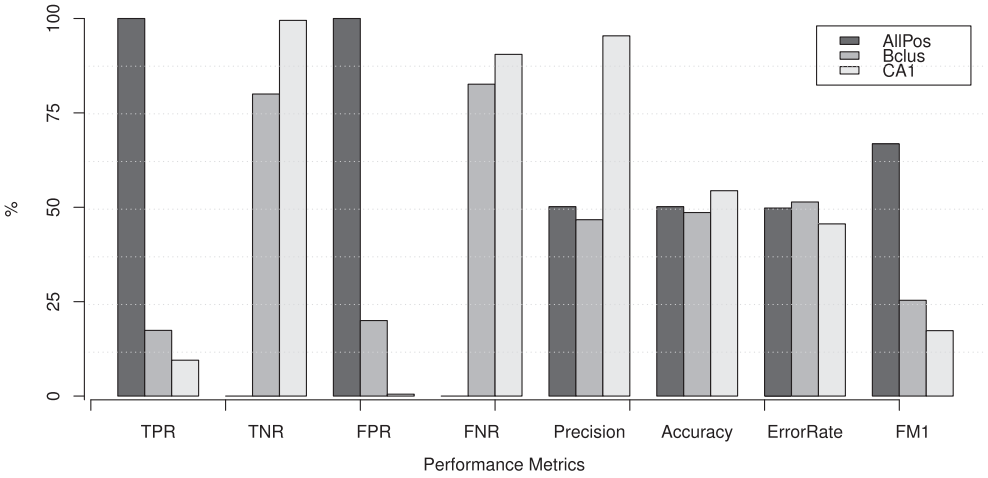

Fig. 13 – Simplified comparison of error metrics for the BClus, CAMNEP and AllPositive algorithms on Scenario 9.

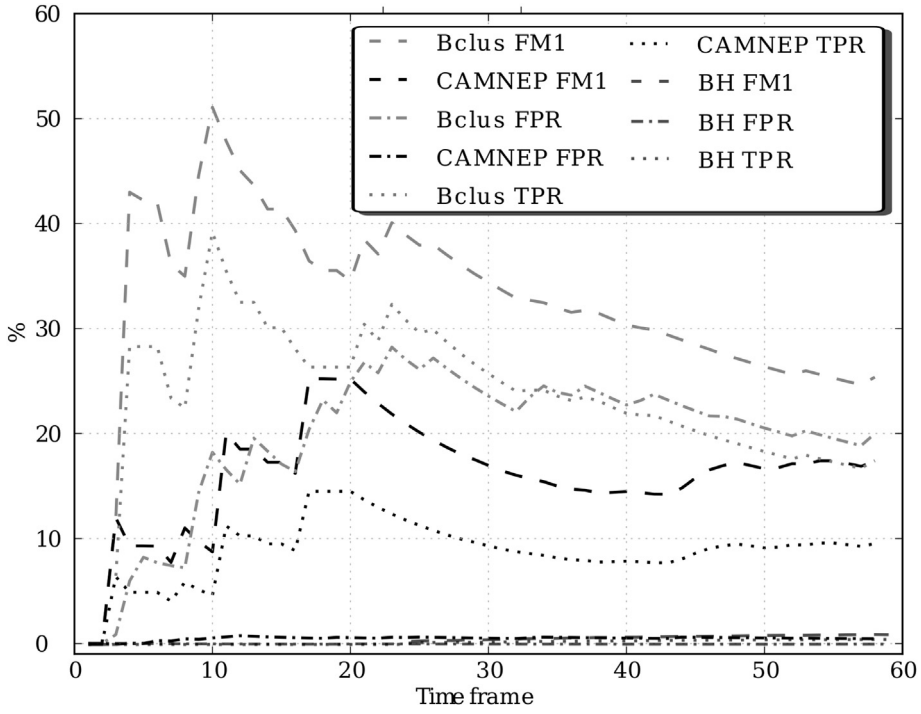

Fig. 14 – Comparison of the running error metrics for Scenario 9.

The error metrics for this scenario can be seen in Table 11. The AllPositive algorithm had the best FMeasure. The BClus algorithm had a FMeasure of 0.25, an FPR of 20% and a TPR of 10%. The CAMNEP algorithm had a FMeasure of 0.17, a very low FPR and a very low TPR. The BotHunter algorithm had a low FMeasure of 0.03 and high FNR of 98%.

The simplified comparison for this scenario is shown in Fig. 13. It can be seen that the TPR value for the BClus method was almost twice the value for CAMNEP. Also, the FPR value of BClus was 40 times larger than the CAMNEP value. However, the FMeasure value of CAMNEP was almost 70% the value of BClus.

The inner workings of the algorithms can be seen in the running metrics shown in Fig. 14. Almost from the start of the scenario, the TPR and FMeasure of the BClus and CAMNEP methods grew fast. However, after the twentieth time frame, both FMeasures values started to decrease. The FPR value of BClus was relatively low compared to the previous scenarios. The BotHunter algorithm presented very low values during the whole scenario despite that there were ten bots being executed.

9. Conclusions

We conclude that our comparison of detection methods using a real dataset greatly helped to improve our research. It showed us how and why the methods were not optimal, which botnet behaviors were not being detected and how the dataset should be improved. Also, it show us the need for a comparison methodology and a proper error metric.

We also conclude, as it was recommended by Aviv and Haeberlen (2011), that a join effort to create a comparison platform of detection methods could greatly enhance the results achieved in the area. We believe that such a platform could take advantages of our comparison methodology.

The usage of a large and real dataset, despite not having a great amount of different botnets, show us which phases of the botnet behavior were more easy to detect by the methods, and the difficulties of working with unknown background data.

Regarding our detection methods, BClus showed large FPR values on most scenarios but also large TPR, we are already working on improving it. The CAMNEP method had a low FPR during most of the scenarios but at the expense of a low TPR. Each of them seems best for a different botnet behavior. The comparison against the BotHunter method showed that in real environments it could still be useful to have blacklists of known malicious IP addresses.

Despite being biased by the really small amount of labeled normal traffic, the AllPositive baseline algorithm was useful to visualize how the error metrics should be always carefully considered.

We also conclude that, although useful and enough for our purposes, the comparison methodology can be improved to show how many of the infected IP addresses were detected by the algorithms. The new error metric proposed, that takes into consideration the IP addresses and time, allowed us to easily compare the algorithms from the perspective of a network administrator.

The dataset created, despite being paramount for the comparison, should be improved. We are already working on adding more botnets, more diverse attacks and more normal labels. A better and larger dataset is already being built.

Acknowledgment

This work was supported by the project of the Czech Ministry of Interior No. VG20122014079.

REFERENCES

Argus. Auditing Network Activity http://qosient.com/argus/; 2013.
AsSadhan B, Moura JMF, Lapsley D. Periodic behavior in botnet command and control channels traffic. In: GLOBECOM: Global Telecommunications Conference IEEE; 2009. pp. 1–6.
Aviv A, Haeberlen A. Challenges in experimenting with botnet detection systems. In: USENIX 4th CSET Workshop, San Francisco, CA; 2011 [p. 6–6].
Bayer U, Comparetti PM, Hlauschek C, Kruegel C, Scalable Kirda E. Behavior-based Malware clustering. In: NDSS: Network and Distributed System Security Symposium; 2009.
Cho K, Mitsuya K, Kato A. Traffic data repository at the WIDE project. In: ATEC '00: Annual conference on USENIX Annual Technical Conference; 2000. pp. 51–2.
Claise B. Specification of the IP Flow Information Export (IPFIX) protocol for the exchange of IP traffic flow information http://tools.ietf.org/html/rfc5101; 2008.
Dainotti A, King A, Papale F, Pescapè A. Analysis of a/0 stealth scan from a botnet. In: IMC '12: ACM conference on Internet measurement conference; 2012. pp. 1–4.
Davis JJ, Clark JA. Data preprocessing for anomaly based network intrusion detection: a review. J Comput Secur 2011;30:353–75.
Ertoz L, Eilertson E, Lazarevic A, Tan PN, Kumar V, Srivastava J, et al. Minds-minnesota intrusion detection system. In: Next generation data mining. MIT Press; 2004. pp. 199–218.
Fontugne R, Borgnat P, Abry P, Fukuda K. Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In: CoNEXT 2010: ACM Conference on Emerging Networking Experiments and Technology; 2010 [p. 8–8].
García S. Malware capture facility project https://mcfp.felk.cvut.cz/; 2013 [accessed 06.03.14].
García S. Botnet detectors comparer http://sourceforge.net/projects/botnetdetectorscomparer/; 2014 [accessed 06.03.14].
García S, Zunino A, Campo M. Botnet behavior detection using network synchronism. In: Privacy, Intrusion Detection and Response: Technologies for Protecting Networks. IGI Global; 2012. pp. 122–44.
García S, Zunino A, Campo M. Survey on network-based botnet detection methods. J Secur Commun Networks 2013;7:878–903. John Wiley & Sons.
Gu G, Porras P, Yegneswaran V, Fong M, Lee W. Bothunter: detecting malware infection through ids-driven dialog correlation. In: Proceedings of 16th USENIX Security Symposium; 2007. pp. 1–16.
Gu G, Zhang J, Lee W. BotSniffer: detecting botnet command and control channels in network traffic. In: NDSS: Proc. 15th Network and Distributed System Security Symposium; 2008.
Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA data mining software: an update. ACM SIGKDD Explor Newsl 2009;11.
Hegna A. Visualizing spatial and temporal dynamics of a class of IRC-based botnets [PhD thesis]; 2010.
Jacobson V., Leres C., McCanne S. (1997). tcpdump/libpcap www.tcpdump.org Lawrence Berkeley Laboratory.
Kotsiantis S, Kanellopoulos D, Pintelas P. Handling imbalanced datasets : a review. J GESTS Int Transactions Comput Sci Eng 2006;30:25–36.
Lakhina A, Crovella M, Diot C. Diagnosing network-wide traffic anomalies. ACM SIGCOMM Comput Commun Rev 2004;34:357–74.
Lakhina A, Crovella M, Diot C. Mining anomalies using traffic feature distributions. ACM SIGCOMM Comput Commun Rev 2005;35:217–28.
Lee K, Kim J, Kwon K, Ha Yn, Kim S. DDoS attack detection method using cluster analysis. Expert Syst Appl 2008;34:1659–65.
Li P, Liu L, Gao D, Reiter MK. On challenges in evaluating malware clustering. In: Proceedings of the 13th international conference on recent advances in intrusion detection; 2010. pp. 238–55.
Lu W, Tavallaee M, Rammidi G, Ghorbani A. BotCop: an Online botnet traffic classifier. In: 2009 Seventh Annual Communication Networks and Services Research Conference; 2009. pp. 70–7.
Maloof MA. Some basic concepts of machine learning and data mining machine learning and data mining for computer security. London: Springer; 2006. pp. 23–43.
Moon TK. The expectation-maximization algorithm. Signal Process Mag IEEE 1996:47–60.
NexGinRC. Endpoint worm scan dataset [accessed 06.03.14], http://nexginrc.org/Datasets/DatasetDetail.aspx?pageID=24; 2013.
Pevný T, Rehák M, Grill M. Identifying suspicious users in corporate networks. In: Proceedings of workshop on information forensics and security; 2012. pp. 1–6.
Ramchurn SD, Jennings NR, Sierra C, Godo L. Devising a trust model for multi-agent interactions using confidence and reputation. Int J Appl Artif Intell 2004;18:833–52.
Rehak M, Pechoucek M, Gregor M, Reh M. Trust modeling with context representation and generalized identities. In: Proceedings of the 11th international workshop on Cooperative Information Agents XI; 2007. pp. 298–312.
Rehak M, Pechoucek M, Grill M, Stiborek J, Bartos K, Celeda P. Adaptive multiagent system for network traffic monitoring. Intell Syst IEEE 2009;24:16–25.
Rettinger V, Nickles M, Tresp V. Learning Initial trust Among Interacting agents. In: Proceedings of the 11th international workshop on Cooperative Information Agents XI; 2007. pp. 313–27.
Rossow C, Dietrich CJ, Grier C, Kreibich C, Paxson V, Pohlmann N, et al. Prudent practices for designing malware experiments: status quo and outlook. In: IEEE Symposium on Security and Privacy; 2012. pp. 65–79.
Rubinstein BIP, Nelson B, Huang L, Joseph AD, Lau S, Rao S, et al. Stealthy poisoning attacks on PCA-based anomaly detectors. SIGMETRICS Perform Eval Rev 2009;37:73–4.
Saad S, Traore I, Ghorbani A, Sayed B, Zhao D, Lu W, et al. Detecting P2P botnets through network behavior analysis and machine learning. In: Privacy, security and trust (PST), Ninth Annual International Conference on; 2011. pp. 174–80.
Sabater J, Sierra C. Review on computational trust and reputation models. J Artif Intell Rev 2005;24:33–60.
Salgarelli L, Gringoli F, Karagiannis T. Comparing traffic classifiers. In: ACM SIGCOMM Computer Communication Review; 2007. pp. 65–8.
Scarfone K, Mell P. Guide to Intrusion Detection and Prevention Systems (IDPS). Technical report. NIST; 2007.
Shiravi A, Shiravi H, Tavallaee M, Ghorbani A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. J Comput Secur 2012;31:357–74.
Sperotto A, Sadre R, Vliet FV, Pras A. A labeled data set for flow-based intrusion detection. In: IPOM '09 Proceedings of the 9th IEEE international Workshop on IP Operations and Management; 2009. pp. 39–50.

Sridharan A, Ye T, Bhattacharyya S. Connectionless port scan detection on the backbone. In: IPCCC 2006: Performance, Computing, and Communications Conference; 2006. p. 576.
Szabó G, Orincsay D, Malomsoky S, Szabó I. On the validation of traffic classification algorithms. In: PAM 2008: 9th International Conference, Passive and Active Network Measurement; 2008. pp. 72–81.
Tavallaee M, Stakhanova N, Ghorbani A. Toward credible evaluation of anomaly-based intrusion-detection methods. IEEE Transactions Syst Man Cybern Part C Appl Rev 2010;40:516–24.
Wurzinger P, Bilge L, Holz T, Goebel J, Kruegel C, Kirda E. Automatically generating models for botnet detection; 2010. pp. 232–49.
Xu K, Zhang ZL. Profiling internet backbone traffic: behavior models and applications. In: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications; 2005. pp. 169–80.
Xu K, Zhang Z, Bhattacharyya S. Reducing unwanted traffic in a backbone network. In: SRUTI 05: steps to reducing unwanted traffic on the internet workshop; 2005. pp. 9–15.
Yager RR. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. Syst Man Cybern IEEE Transactions 1988:183–90.
Zhao D, Traore I, Sayed B, Lu W, Saad S, Ghorbani A, et al. Botnet detection based on traffic behavior analysis and flow intervals. J Comput Secur 2013;39:2–16.

Sebastián García is a PhD student in UNICEN University (Argentina) and a researcher in the ATG group at the Czech Technical University. He is also a research fellow at the National Scientific and Technical Research Council of Argentina (CONICET) and a teacher in the UFASTA University. His research interests include network-based botnet behavior detection, anomaly detection, penetration testing, honeypots, malware detection, keystroke dynamics and machine learning. His recent projects focus on using unsupervised and semi-supervised machine learning techniques to detect botnets on large networks based on their behavioral models.

Martin Grill holds master degree in Software development at the Faculty of Nuclear Sciences and Physical Engineering of the Czech Technical University in Prague. At the present time he is a member of the Agent Technology Center, a researcher at CESNet, and a PhD student at the Department of Cybernetics of Czech Technical University in Prague.

Jan Stiborek holds master degree in Software development at Faculty of Nuclear Sciences and Physical Engineering of the Czech Technical University in Prague. At the present time, he is pursuing PhD degree in Artificial Intelligence and Biocybernetics at Department of Cybernetics, FEE CTU. His current profesional interests focus on network security, network simulation and autonomous adaptation of intrusion detection systems.

Alejandro Zunino (http://www.exa.unicen.edu.ar/~azunino) received a Ph.D. degree in Computer Science from the National University of the Center of Buenos Aires (UNICEN), in 2003, and his M.Sc. in Systems Engineering in 2000. He is a full Adjunct Professor at UNICEN, member of the ISISTAN Research Institute and Independent Researcher of the National Scientific and Technical Research Council (CONICET). His research areas are Distributed Computing and Software Engineering. Contact him at azunino@conicet.gov.ar.