NetFlow Based Framework for Identifying Anomalous End User Nodes

Risto Vaarandi[1] and Mauno Pihelgas[1,2]
[1]Centre for Digital Forensics and Cyber Security, Tallinn University of Technology, Estonia
[2]Technology Branch, NATO CCDCOE, Estonia
firstname.lastname@taltech.ee

Abstract: During the last two decades, cyber attacks against end users have grown significantly both in terms of number and sophistication. Unfortunately, traditional signature-based technologies such as network IDS/IPS and next generation firewalls are able to detect known attacks only, while new attack types not matching any signatures remain unnoticed. Therefore, the use of machine learning for detecting anomalous network traffic of end user nodes has become an important research problem. In this paper, we present a novel NetFlow based framework for identifying anomalous end user nodes and their network traffic patterns, and describe experiments for evaluating framework performance in an organizational network.

Keywords: detection of anomalous end user nodes, network anomaly detection, NetFlow based network monitoring

1. Introduction

For protecting workstations and laptops in corporate networks, security-aware organizations are employing specialized technologies like gateways for filtering web and e-mail traffic. Also, for lessening the risk of infection, end user nodes are often centrally managed, with unmanaged computers being blocked from connecting to corporate network. Unfortunately, many smaller organizations are using simple NAT firewalls that do not support any filtering of malicious application layer traffic. According to recent report by Symantec (2019), employees of smaller institutions are more likely to be hit by e-mail threats. Also, many organizations have no centralized patching routines and have adopted bring-your-own-device policy which makes their end user nodes much more vulnerable. This problem is exacerbated by insufficient cyber security awareness and lack of relevant training (SANS, 2018).

After an end user node has been infected, attackers can harness it for various purposes like compromising other devices in private network, launching attacks against other organizations, etc. For detecting such activities, network IDS is often used. Since most network IDS are signature based and thus able to identify previously known malicious traffic patterns only, a number of NetFlow based anomaly detection algorithms have been suggested in recent papers. However, apart from few exceptions (e.g., (Grill et al, 2015)), most previously suggested methods have not focused on detection of anomalous end user nodes in organizational networks. Furthermore, many methods can only raise an alarm about anomaly for some node or network segment without the ability to highlight malicious traffic patterns. Nevertheless, providing such information to network administrators would significantly reduce incident resolution time (Zhou et al, 2015). Finally, only few works have considered the use of multiple classifiers for anomaly detection from NetFlow data (Hou et al, 2018).

This paper addresses above research gaps and presents an unsupervised framework for detecting anomalous end user nodes in organizational networks. The framework employs several anomaly detectors for finding a total anomaly score for each node in hourly time windows, and uses LogCluster algorithm for finding network traffic patterns for anomalous nodes from NetFlow data. We have evaluated the framework during 5 months in a network of an academic institution. The remainder of this paper is organized as follows – section 2 discusses related work, section 3 presents our framework, section 4 describes performance evaluation of the framework, and section 5 outlines future work.

2. Related Work

Cisco NetFlow is a protocol which defines *flow* as a sequence of packets that share a number of common properties, most notably the source IP address, source port, destination IP address, destination port, and transport protocol. For monitoring network traffic, NetFlow *exporter* maintains a record for each flow in a memory-based cache, with the record holding counters for packets and bytes in the flow, the union of all observed TCP flags, start and end time of the flow, and other data. NetFlow exporter sends a flow record to NetFlow *collector* when some flow-based timer expires (e.g., no packets have been seen during 60 seconds), when the flow ends (e.g., TCP connection is closed), or when flow cache becomes full. Many modern network devices can act as NetFlow exporters, and there are many commercial and open-source NetFlow exporter and collector implementations (Hofstede et al, 2014). Also, a number of approaches have been suggested for anomaly detection from NetFlow data.

Brauckhoff et al (2012) have proposed an algorithm where histograms are built from flow features (e.g., source port) during measurement intervals, where each histogram represents a flow feature distribution. At the end of each interval, Kullback-Leibler distance between distributions for current and previous interval is calculated. If the distance exceeds a threshold, alarm is raised and Apriori frequent itemset mining algorithm is used for mining patterns from suspicious flow records. Kind, Stoecklin and Dimitropoulos (2009) have described a supervised method where the training phase involves building histograms and clustering similar histograms together, in order to create models of normal behavior. For anomaly detection, a vector is computed that encodes online network behavior, and a distance of the vector from clusters is calculated.

Grill et al (2015) have suggested a method for discovering hosts with domain generation algorithm (DGA) based malware by measuring the ratio of DNS requests to the number of unique IP addresses contacted by the host. According to experiments, the ratio is high for infected hosts. Our past work (Vaarandi, 2013) proposes two unsupervised anomaly detection algorithms for organizational private networks. The first algorithm maintains behavior profile for each node which describes recently used services, and raises an alarm if the node connects to unusual service. The second algorithm clusters nodes on daily basis, in order to find node groups that consistently use the same services, and raises an alarm if node behavior deviates from the rest of the group. Muhs et al (2018) have developed a method for detecting P2P botnets which creates a communication graph from NetFlow data. Each graph node represents a host and each graph edge a probability of communication between relevant hosts. The method conducts a large number of random walks (traversals over *k* nodes), and calculates the probability of reaching each end node. Resulting probability distribution is then clustered with Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm (Ester et al, 1996), and according to authors, P2P bots form dense clusters.

Hofstede et al (2013) have proposed exponentially weighted moving average (EWMA) based algorithm for DDoS detection which tracks the number of flows. If unexpected change in the number of flows is detected, the algorithm can create firewall rules for blocking malicious traffic. Hou et al (2018) have suggested random forest classifiers for DDoS detection, and have found their performance superior to C4.5, SVM, and Adaboost classifiers. Paredes-Oliva et al (2012) have proposed a supervised algorithm which first identifies traffic patterns with FPmax frequent itemset mining algorithm, and then uses C5.0 classifier for finding anomalous patterns. Finally, Zhou et al (2015) have developed ENTvis tool which divides NetFlow data into timeframes, and for each timeframe calculates entropies for source IP, source port, destination IP and destination port. This information is then visualized with several techniques (e.g., visual clustering) which allows human analysts to spot anomalies and understand their nature.

3. Framework for Detecting Anomalous End User Nodes

3.1 Overview of Anomaly Detection Framework

If an end user node communicates with some port at remote node, we define *peer* as a tuple *(transport protocol ID, IP address of remote node, port number at remote node)*. Also, *peer port* is defined as a tuple *(transport protocol ID, port number at remote node)*. For protocols without ports (e.g., ICMP), 0 is used for remote port number.

For studying typical network usage patterns of end user nodes, we analyzed a 1-month NetFlow data set for 78 workstations. We divided this data set into 1 hour timeframes and investigated how many peers and peer ports each node had accessed during hourly timeframes of its activity. Firstly, total numbers of peers for the entire month remained relatively modest, and an average node accessed 1213.2 peers (largest number of peers per node was 4688).

Secondly, significant part of the flows were associated with a small fraction of frequently used peers (see Table 1). For example, an average node contacted only 51.7 peers during at least half of its activity hours, but 69.9% of flows were associated with these peers.

Table 1: Network usage patterns of end user nodes

N	*Number of peers contacted during at least N% of hourly timeframes when node was active (average over all nodes)*	*Fraction of flows associated with peers (average over all nodes)*
10	292.6	92.4%
25	145.9	83.4%
50	51.7	69.9%

Also, we discovered similar trends for peer ports – for example, an average node contacted 8.9 peer ports during at least half of its activity hours, and 96.6% of flows were associated with these peer ports. Finally, we have observed similar regularities for NetFlow data sets collected in different environments (Vaarandi, 2013).

These findings suggest that if a new flow is observed for an end user node, it is likely that the flow is associated with a peer and peer port the node has already accessed in recent past. In other words, *recent communication patterns of end user node can be harnessed for predicting its future behavior*, and we have developed an anomaly detection framework which relies on that assumption (see Figure 1).

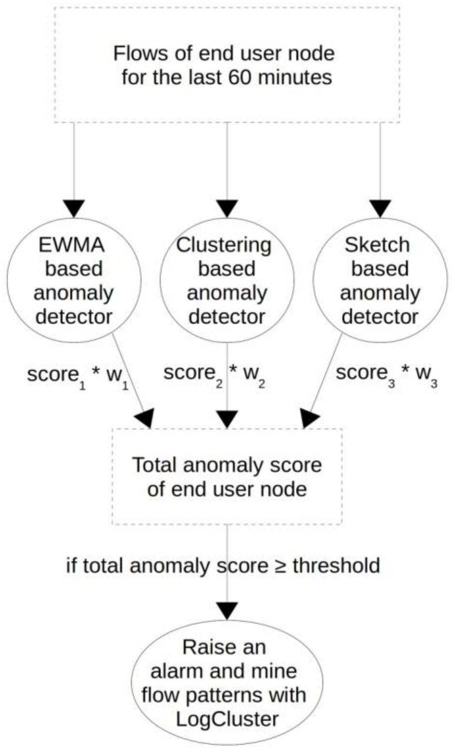

Flows of end user node for the last 60 minutes
EWMA based anomaly detector
Clustering based anomaly detector
Sketch based anomaly detector
score1 * w1
score2 * w2
score3 * w3
Total anomaly score of end user node
if total anomaly score ≥ threshold
Raise an alarm and mine flow patterns with LogCluster

Figure 1: Overview of the framework

The framework executes once in every hour on NetFlow collector node, and processes the flows of last 60 minutes for each end user node. If the node has no flows, it will be skipped due to its inactivity. The framework consists of three anomaly detectors which calculate anomaly scores *score*$_1$, *score*$_2$ and *score*$_3$ from range 0..1 for a node. The scores are aggregated into total anomaly score $score_1*w_1 + score_2*w_2 + score_3*w_3$ with non-negative weights w_1, w_2 and w_3, where $w_1 + w_2 + w_3 = 1$ (therefore, total anomaly score ranges from 0 to 1). If node's total anomaly score exceeds a given threshold, an alarm is raised and flows of that node for the last 60 minutes are mined with LogCluster algorithm, in order to identify anomalous network traffic patterns (LogCluster is known to be well suited for analyzing security logs (Vaarandi, Kont and Pihelgas, 2016)). The following subsections provide a detailed discussion of each anomaly detector from Figure 1.

3.2 EWMA Based Anomaly Detector

Since it has been shown that EWMA based anomaly detection methods are efficient for network traffic analysis (Hofstede et al, 2013), one of anomaly detectors employs a similar approach. If $X = \{x_1, x_2, ...\}$ is a time series, EWMA μ and exponentially weighted moving standard deviation σ are calculated according to Equation 1.

$$\begin{cases} \mu_1 = x_1 \\ \mu_i = \alpha * x_i + (1-\alpha) * \mu_{i-1} \ , \ i > 1 \end{cases} \qquad \begin{cases} var_1 = 0 \\ var_i = (1-\alpha) * (var_{i-1} + \alpha * (x_i - \mu_{i-1})^2) \ , \ i > 1 \\ \sigma_i = \sqrt{var_i} \end{cases}$$

Equation 1: Exponentially weighted moving average and standard deviation

In Equation 1, α ranges from 0 to 1, with larger values of α giving more weight to recent observations of *X*. Values close to 0 are distributing weight more evenly, and EWMA is known to estimate the average of last $(2/\alpha)$-1 values from *X*. For detecting anomalies for time series *X*, the following method is often used:
if $|x_i - \mu_{i-1}| > m*\sigma_{i-1}$, then x_i is regarded anomalous (*m* is a user defined constant and commonly set to 3).

The above approach has motivated EWMA based anomaly detector for identifying unexpected increases in the number of peers and peer ports, and the volume of traffic exchanged with peers (according to section 3.1, end user nodes communicate with modest number of peers). For each node *E*, the anomaly detector tracks the following six features:

- *Peers* – number of unique peers for node *E* per 1 hour
- *RarePeers* – number of unique peers for node *E* per 1 hour, so that *E* has not communicated with these peers during the last *N* hours of its activity (during our experiments, we have set *N* = 50)
- *PeerPorts* – number of unique peer ports for node *E* per 1 hour
- *LogFlows* – $log_{10}M$, where *M* is the number of flows for node *E* per 1 hour
- *LogPackets* – $log_{10}M$, where *M* is the number of packets sent and received by node *E* per 1 hour
- *LogBytes* – $log_{10}M$, where *M* is the number of bytes sent and received by node *E* per 1 hour

The anomaly detector raises an alarm for any of above features if $x_i - \mu_{i-1} > m*\sigma_{i-1}$, i.e., the value of a feature increases significantly. We have employed the settings $m = 3$ and $\alpha = 0.05$ for anomaly detector, and with α=0.05, μ_{i-1} estimates the average feature value over previous 39 hours (that is the approximate working week length for end users, and end user nodes are often switched off outside office hours). The purpose of the *RarePeers* feature is to detect unexpected increase of the number of unusual peers, even if the overall number of peers stays within expected boundaries. Also, instead of tracking the number of flows, packets, and bytes, these values have been converted to logarithmic scale for lessening the number of false positives. Finally, each feature contributes equally to the anomaly score reported by the detector. For example, if for node *E* alarms have been raised for *RarePeers* and *LogFlows* features, anomaly score 1/3 will be reported for node *E*.

3.3 Clustering Based Anomaly Detector

Unfortunately, EWMA based anomaly detector is not aware of the surrounding context which might influence the severity of detected anomalies. For example, while downloading unusually large amount of data is an anomaly if observed for one node only, the same simultaneous behavior change by many nodes can have a benign root cause (e.g., centralized patching of all workstations). The purpose of the clustering based anomaly detector is to find groups of similarly behaving nodes and report outliers as anomalous.

For clustering purposes, each node is represented by a vector with the following attributes – the number of peers, rare peers, and peer ports (defined like *Peers*, *RarePeers*, and *PeerPorts* features in the previous section); total number of bytes, packets, and flows; total number of bytes and packets for outgoing traffic; total number of bytes and packets for incoming traffic. Before clustering, all features are standardized by removing the mean and scaling to unit variance. For clustering the nodes, DBSCAN algorithm (Ester et al, 1996) is used which takes *minPts* and ε input parameters.

DBSCAN regards data point as *core point* if it has at least *minPts* points (including itself) within distance ε (we have used Euclidean distance for anomaly detector). A point q is *reachable* from core point p if either: (1) q is within distance ε from p, or (2) there exist core points $p_1,...,p_k$, so that:

- $p_1 = p$,
- p_i is within distance ε from p_{i-1} $(1 < i \leq k)$,
- q is within distance ε from p_k.

If p is a core point, DBSCAN creates a cluster from p and all points that are reachable from it (therefore, each cluster will contain at least *minPts* points). Points that are not reachable from any other point are regarded outliers.

For favoring the creation of larger clusters with many nodes, the anomaly detector executes DBSCAN with settings *minPts* = 10 and $\varepsilon = 5$. For nodes belonging to clusters, anomaly score 0 is returned. Also, if there are k outliers among n nodes, anomaly score 1-(k/n) is returned for each outlier node. Therefore, if there are only few outlier nodes, their anomaly scores are close to 1, while with many outliers (i.e., when anomalous behavior is more common) their anomaly scores will be significantly lower.

3.4 Sketch Based Anomaly Detector

Unlike two previous anomaly detectors that are based on well-known algorithms, this subsection presents a novel anomaly detector which employs one-row sketches for summarizing past communication patterns of each end user node, in order to predict its future behavior. The anomaly detector has three components with the following goals:

- Similarity and entropy based anomaly detection for peers
- Similarity and entropy based anomaly detection for peer ports
- Entropy based anomaly detection for local ports

Each component can raise an alarm which will contribute equally to node's anomaly score (e.g., if two components raise an alarm for a node, its anomaly score is 2/3). The following paragraphs will outline the work of the first peer-related component for node E (the second component works similarly, and the third will be described in the end of this section).

The anomaly detector will start its work with two sketches $S = (s_1,...,s_n)$ and $V = (v_1,...,v_n)$ which are vectors of n counters (we have set n = 100000 for peers, and n = 10000 for peer ports and local ports). The purpose of vector S is to capture recent average communication patterns with peers for E, while V captures this information for the last hour.

When anomaly detector is executed for the flows of E from last 60 minutes, counters of vector V are initialized to zero and updated in the following way:

1. for each flow, a peer is extracted and hashed into a value from range 1..n,
2. if the value is k, counter v_k is incremented.

If $V_1, V_2, \ldots$ denote above hourly vectors in the order of creation, and v_{ji} denotes the i^{th} counter of vector V_j, the i^{th} counter of vector S is maintained as an EWMA for time series $\{v_{1i}, v_{2i}, \ldots\}$ (see Equation 1). In other words, each subsequently calculated vector $V = (v_1, \ldots, v_n)$ is used for updating $S = (s_1, \ldots, s_n)$ according to Equation 2.

$$s_i := \alpha * v_i + (1 - \alpha) * s_i \ , \ 1 \leq i \leq n$$

Equation 2: Vector update procedure

Therefore, counter v_i is equal to the number of flows during the last hour for a group of one or more peers, while counter s_i estimates the average number of flows during last $(2/\alpha)$-1 hours for the same peer group. Thus, vectors S and V represent distributions of flows over peer groups. Since vector S reflects recent communication patterns with peers that serve as a good predictor for the future behavior of the node (see section 3.1), significant difference between vectors S and V can be regarded as an anomaly.

For measuring the difference between two vectors, we experimented with several distance functions and discovered that cosine similarity produces best results (see Equation 3).

$$cosim(S, V) = \frac{\sum_{i=1}^{n} s_i * v_i}{\sqrt{\sum_{i=1}^{n} s_i^2} * \sqrt{\sum_{i=1}^{n} v_i^2}}$$

Equation 3: Cosine similarity

Cosine similarity measures the angle between two vectors, and ranges from 0 to 1 since counters of S and V are non-negative. If $cosim(S,V) = 1$, vectors S and V have the same direction (e.g., vectors (1,0,23,0) and (2,0,46,0)), and if $cosim(S,V) = 0$, vectors are orthogonal. Therefore, values close to 1 indicate that during the last 60 minutes distribution of flows over peer groups is similar to distribution of previous observations, while smaller values of cosine similarity indicate deviations from past measurements.

The use of cosine similarity introduces the following issue – if node behavior changes frequently and has no clear baseline, similarity between S and V remains low and becomes meaningless for anomaly detection. For avoiding false alarms for such nodes, the anomaly detector calculates the similarity $cosim(S,V)$ before each update of S with Equation 2, and maintains EWMA *cavg* for past similarity values according to Equation 1. Before each update of *cavg* with current similarity value $cosim(S,V)$, anomaly detector raises an alarm only if $cosim(S,V) < T_{sim}$ and $cavg > T_{avg}$. In other words, alarm is generated if similarity between S and V falls below threshold T_{sim}, with average past similarity *cavg* being sufficiently high and exceeding T_{avg} (we have used the settings $\alpha = 0.05$, $T_{sim} = 0.5$ and $T_{avg} = 0.8$ for anomaly detector).

In addition, we have also implemented entropy based anomaly detection for vector S, in order to identify nodes with constantly changing behavior (since *cavg* is usually smaller than threshold T_{avg} for such nodes, similarity based anomaly detection tends to be disabled for them). According to section 3.1, end user nodes are mostly communicating with relatively small number of peers which results in relatively few counters of vector S having larger values, while many counter values remain zero. On the other hand, if node behavior is frequently changing, counter values in S would be less different. In previous research papers, entropy based techniques have been successfully used for capturing such regularities (Zhou et al, 2015).

If X is a discrete random variable that can take k values with probabilities $p_1, \ldots, p_k$ ($p_1 + \ldots + p_k = 1$), *normalized information entropy* of X is defined by Equation 4.

$$norment(X) = \frac{-\sum_{i=1}^{k} p_i * log(p_i)}{log(k)}$$

Equation 4: Normalized information entropy

Note that *norment*(*X*) ranges from 0 to 1, with values close to 1 indicating that distribution of *X* is close to uniform. Also, smaller values indicate that *X* takes some values with significantly higher probabilities. In order to employ entropy based anomaly detection for *S*, we first find its normalized vector $\hat{S} = (1/j) * S$, where $j = s_1 + \ldots + s_n$ (note that $\hat{s}_1 + \ldots + \hat{s}_n = 1$). After that, the entropy of *S* is calculated according to Equation 5.

$$norment(S) = \frac{-\sum_{i=1}^{n} \hat{s}_i * log(\hat{s}_i)}{log(n)}$$

Equation 5: Calculating sketch entropy

The anomaly detector calculates the entropy of *S* after each update with Equation 2, and raises an alarm if entropy is higher than T_{ent} (for peers and peer ports, we have set T_{ent} to 0.5 and 0.3 respectively). Finally, since typical end user nodes do not use fixed local ports for communicating with remote services but ports are selected randomly, the anomaly detector also maintains vector *S* for local ports of each end user node. Since *S* is expected to have a high entropy, the anomaly detector raises an alarm if the entropy falls below T_{ent2} (we have set T_{ent2} to 0.2 during experiments).

3.5 Mining Flow Patterns with LogCluster

If node's total anomaly score for last 60 minutes exceeds a given threshold, flows of last 60 minutes for this node are mined with LogCluster algorithm, in order to detect prominent traffic patterns that capture the nature of anomaly. Since LogCluster is designed for mining textual log files, each binary flow record is converted into a textual line with the following keyword-value based format: *proto <transport protocol ID> srcip <source IP address> srcport <source port> dstip <destination IP address> dstport <destination port> tcpflags <flagstring>*. For easing the conversion process, we have used Perl based LogCluster implementation with advanced input preprocessing features (https://ristov.github.io/logcluster/).

For detecting patterns from log file with LogCluster, the user has to specify *support threshold* value which ranges from 1 to *n*, where *n* is the number of lines in the log file. If support threshold is *s*, LogCluster identifies patterns that match at least *s* lines in the log file. During the first pass over the data set, LogCluster will split each line into words, and identify *frequent words* in the data set (words which appear in at least *s* lines). During the second data pass, LogCluster extracts all frequent words from each line, so that the combination of frequent words identifies a *cluster candidate* for the given line. For each cluster candidate, LogCluster memorizes the number of lines for this candidate, and summary information about the location of infrequent words for all lines of the candidate. After the data pass, cluster candidates with at least *s* lines are selected as clusters, and reported to the end user as line patterns. For example, the following pattern represents DNS queries from node 10.3.7.22 to server 192.168.1.1 port 53/UDP (note that **{1,1}* is a wildcard which matches exactly one word):

*proto 17 srcip 10.3.7.22 srcport *{1,1} dstip 192.168.1.1 dstport 53 tcpflags NA*

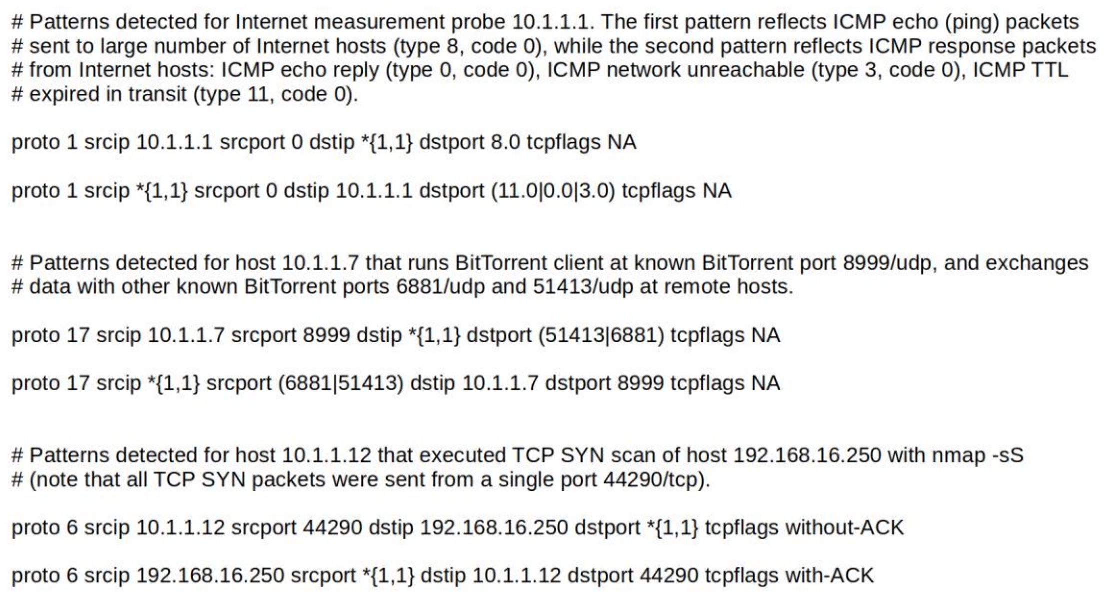
As discussed in (Vaarandi, Blumbergs and Kont, 2018), finding a good support threshold value for LogCluster is a non-trivial issue. However, we have discovered that if NetFlow data set contains *n* flow records, support threshold value $\sqrt{n}$ allows to adequately highlight the nature of anomalous traffic in most cases. Also, we have configured LogCluster to use word weight based heuristics for joining patterns with the same nature (Vaarandi, Kont and Pihelgas, 2016). For example, in the case of two similar patterns that reflect DNS queries from 10.1.1.1 to servers 192.168.1.1 and 192.168.1.2, LogCluster would join them into the following single pattern:

*proto 17 srcip 10.1.1.1 srcport *{1,1} dstip (192.168.1.1|192.168.1.2) dstport 53 tcpflags NA*.

4. Evaluation

We have implemented our anomaly detection framework in Perl and Python, and measured its performance during 5 months (October 2018 – February 2019, 151 days) in a network of an academic institution with over 200 workstations and laptops. The framework used *softflowd* NetFlow exporter (https://code.google.com/archive/p/softflowd/) on a dedicated Linux host for monitoring Internet traffic of all end user nodes without sampling, and flow data was collected with *nfdump* NetFlow collector (https://github.com/phaag/nfdump).

When evaluating the framework, we used each anomaly detector with an equal weight of 1/3 and set anomaly score threshold to 0.5 (see Figure 1). During 5 months, the framework generated 1026 alarms about 33 end user nodes. However, over 90% of alarms were triggered for 4 hosts, and for 25 hosts only 1-3 alarms were raised. Figure 2 depicts some anomalous traffic patterns the framework discovered with LogCluster.



```
# Patterns detected for Internet measurement probe 10.1.1.1. The first pattern reflects ICMP echo (ping) packets
# sent to large number of Internet hosts (type 8, code 0), while the second pattern reflects ICMP response packets
# from Internet hosts: ICMP echo reply (type 0, code 0), ICMP network unreachable (type 3, code 0), ICMP TTL
# expired in transit (type 11, code 0).

proto 1 srcip 10.1.1.1 srcport 0 dstip *{1,1} dstport 8.0 tcpflags NA

proto 1 srcip *{1,1} srcport 0 dstip 10.1.1.1 dstport (11.0|0.0|3.0) tcpflags NA


# Patterns detected for host 10.1.1.7 that runs BitTorrent client at known BitTorrent port 8999/udp, and exchanges
# data with other known BitTorrent ports 6881/udp and 51413/udp at remote hosts.

proto 17 srcip 10.1.1.7 srcport 8999 dstip *{1,1} dstport (51413|6881) tcpflags NA

proto 17 srcip *{1,1} srcport (6881|51413) dstip 10.1.1.7 dstport 8999 tcpflags NA


# Patterns detected for host 10.1.1.12 that executed TCP SYN scan of host 192.168.16.250 with nmap -sS
# (note that all TCP SYN packets were sent from a single port 44290/tcp).

proto 6 srcip 10.1.1.12 srcport 44290 dstip 192.168.16.250 dstport *{1,1} tcpflags without-ACK

proto 6 srcip 192.168.16.250 srcport *{1,1} dstip 10.1.1.12 dstport 44290 tcpflags with-ACK
```

Figure 2: Flow patterns for anomalous network traffic

When investigating the alarms more closely, we found that 638 of them (62.2%) were generated for 5 hosts which were running BitTorrent client for file sharing purposes, although such activity is not allowed by organizational policies (see the second example in Figure 2). Also, 327 alarms (31.9%) were triggered for an Internet measurement probe which was accidentally connected to end user network (see the first example in Figure 2). According to the developer of the probe, it needs at most 100 Kbit/s of bandwidth and typical consumption does not exceed couple of Kbit/s (Internet Measurement Project, 2017). Despite its negligible network footprint, the anomaly detection framework was able to flag the probe as anomalous. Also, we found that the framework generated 46 false positive alarms (4.5%), with most of them being triggered by downloads of large files.

For estimating the precision and recall of the framework for attack traffic, we conducted network scanning with *nmap* tool from one of the end user nodes during 15 hours. Altogether, the node was active for 548 hours during 5 month timeframe, and no malicious network traffic was observed for this node during remaining 533 hours. Network scanning was conducted against test targets in controlled environment, and scanning scenarios involved different scans of a single host (4 hours) and a network of 2048 hosts (11 hours). Scanning types included lightweight reconnaissance scans, TCP SYN, Xmas, Fin and Null scans, and scans for detecting operating system and service versions. We also simulated TCP and UDP flooding attacks against the network of 2048 hosts. All 15 hours of malicious activity were correctly flagged as anomalous (see the third example in

Figure 2 for some detected patterns), and no false alarms were raised for 533 hours of legitimate network activity, yielding the precision and recall of 100%.

Finally, we measured what is the impact of using multiple classifiers in the anomaly detection framework, and executed each anomaly detector from Figure 1 separately with weight 1 and anomaly score threshold set to 0.5. The EWMA based anomaly detector triggered 570 alarms, clustering based detector 4020 alarms, and sketch based detector 1696 alarms. Table 2 depicts the performance of individual detectors for a node which we used for *nmap* scanning.

Table 2: Precision and recall of individual detectors for *nmap* scanning

	Precision	*Recall*
EWMA based detector	50%	20%
Clustering based detector	37.5%	100%
Sketch based detector	88.2%	100%

The primary reason for poor recall of EWMA based detector is the fact that abnormally large values can significantly increase the average, so that the following similarly large values will no longer be seen as anomalous (this condition will persist until a sequence of smaller values will lower the average again). Also, although sketch based detector had the best precision and recall, it triggers more false positive alarms than the framework of three detectors. We have made a similar observation for other nodes in the data set, and the use of multiple classification methods is thus beneficial over a single classifier.

5. Future Work

The framework presented in this paper assumes that each node belongs to one person (or few similarly behaving persons), so that network usage habits of node owners create behavior baselines for nodes which can be employed for anomaly detection. However, if many persons with different assignments are sharing the nodes (e.g., classroom computers in a university network), it is often difficult to identify clear behavior baselines for them. A similar issue arises if IP addresses of nodes are frequently changing (e.g., due to DHCP based IP address allocation) and the same address is reused for different nodes. In order to overcome these issues, the framework needs to identify the end user node not by IP address, but rather by unique ID of the person that operates the node (e.g., an organizational user account name). Augmenting the framework with a monitoring module for detecting user IDs has been left for future work.

As for other future work, we plan to integrate other anomaly detectors into the framework and experiment with methods for adjusting detector weights dynamically. We are also planning to research other approaches for mining traffic patterns from NetFlow data sets, and algorithms for identifying node types in organizational networks. Finally, we are considering to study methods for fingerprinting end users by their network usage patterns in large organizational networks.

Acknowledgements

The authors thank Prof. Olaf M. Maennel and Prof. Rain Ottis for supporting this work.

References

Brauckhoff, D., Dimitropoulos, X., Wagner, A. and Salamatian, K. (2012) “Anomaly Extraction in Backbone Networks Using Association Rules,” IEEE/ACM Transactions on Networking, vol. 20, no. 6, pp. 1788–1799.

Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. (1996) “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” Proceedings of the 1996 International Conference on Knowledge Discovery and Data Mining, pp. 226–231.

Grill, M., Nikolaev, I., Valeros, V. and Rehak, M. (2015) “Detecting DGA malware using NetFlow,” Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, pp. 1304–1309.

Hofstede, R., Bartoš, V., Sperotto, A. and Pras, A. (2013) “Towards Real-Time Intrusion Detection for NetFlow and IPFIX,” Proceedings of the 2013 International Conference on Network and Service Management, pp. 227–234.

Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A. and Pras, A. (2014) “Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX,” IEEE Communication Surveys and Tutorials, vol. 16, no. 4, pp. 2037–2064.

Hou, J., Fu, P., Cao, Z. and Xu, A. (2018) “Machine Learning based DDos Detection Through NetFlow Analysis,” Proceedings of the 2018 IEEE Military Communications Conference, pp. 565–570.

Internet Measurement Project (2017) “Internet Measurement Project FAQ,” [online], University of Nevada, Reno, https://im.cse.unr.edu/?page=FAQ.

Kind, A., Stoecklin, M. Ph. and Dimitropoulos, X. (2009) “Histogram-based traffic anomaly detection,” IEEE Transactions on Network and Service Management, vol. 6, no. 2, pp. 110–121.

Muhs, D., Haas, S., Strufe, T. and Fischer, M. (2018) “On the Robustness of Random Walk Algorithms for the Detection of Unstructured P2P Botnets,” Proceedings of the 2018 International Conference on IT Security Incident Management and IT Forensics, pp. 3–14.

Paredes-Oliva, I., Castell-Uroz, I., Barlet-Ros, P., Dimitropoulos, X. and Solé-Pareta, J. (2012) “Practical Anomaly Detection based on Classifying Frequent Traffic Patterns,” Proceedings of the 2012 IEEE INFOCOM Workshops, pp. 49–54.

SANS (2018) “2018 SANS Security Awareness Report: Building Successful Security Awareness Programs,” [online], https://www.sans.org/security-awareness-training/reports/2018-security-awareness-report.

Symantec (2019) “Internet Security Threat Report,” [online], Volume 24, February 2019, https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf.

Vaarandi, R. (2013) “Detecting Anomalous Network Traffic in Organizational Private Networks,” Proceedings of the 2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support, pp. 285–292.

Vaarandi, R., Kont, M. and Pihelgas, M. (2016) “Event Log Analysis with the LogCluster Tool,” Proceedings of the 2016 IEEE Military Communications Conference, pp. 982–987.

Vaarandi, R., Blumbergs, B. and Kont, M. (2018) “An Unsupervised Framework for Detecting Anomalous Messages from Syslog Log Files,” Proceedings of the 2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1–6.

Zhou, F., Huang, W., Zhao, Y., Shi, Y., Liang, X. and Fan, X. (2015) “ENTVis: A Visual Analytic Tool for Entropy-Based Network Traffic Anomaly Detection,” IEEE Computer Graphics and Applications, vol. 35, no. 6, pp. 42–50.