

Automatic Diagnosis of Security Events in Complex Infrastructures using Logs

Daniel Gonçalves¹

¹INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
daniel.dias.g@gmail.com

Abstract—The management of complex network infrastructures continues to be a difficult endeavor today. These infrastructures can contain a huge number of devices that may misbehave in unpredictable ways. Many of these devices keep logs that contain valuable information about the infrastructures’ security, reliability, and performance. However, extracting information from that data is far from trivial. In this paper we present a novel approach to assess the security of such an infrastructure using its logs, inspired on data from a real telecommunications network. We use machine learning and data mining techniques to analyze the data and semi-automatically discover misbehaving hosts, without having to instruct the system about how hosts misbehave.

I. INTRODUCTION

Over the past few years, many organizations with large network infrastructures started to record huge amounts of data in logs. These logs contain data about the behavior of users, computers and network devices. Telecommunication companies such as Vodafone PT have complex infrastructures in order to support their business. These infrastructures include a multitude of network devices such as routers, switches, firewalls, authentication servers, and base stations. Log data can be used for different purposes including application debugging, performance monitoring, fault diagnosis, and security.

Extracting useful information from these logs is far from trivial [1]. Their sheer size turns tasks such as collecting and moving the logs to machines that will process them a time-consuming operation that requires considerable bandwidth. Moreover, today’s logs are highly susceptible to data repetition, inconsistency, and noise, due to their large size, complex structure, and multiple heterogeneous sources.

This paper presents an approach for analyzing large logs for extracting security information. The size of the data involved has led the problem to be called *big data analytics for security* [2], [3] and data mining and machine learning techniques to be used to tackle it.

Log analysis is being increasingly used for intrusion detection, i.e., to identify malicious activity [4], [1], [5]. The most common forms of analysis of logs for security are misuse detection – looking for known suspicious patterns (signatures) in the logs –, anomaly detection – looking for deviations in relation to what is considered good behavior – and policy-violation detection – looking for violations of a certain policy [1]. Despite their interest, these approaches require manual definition of what is good/bad behavior (misuse and policy-violation detection) or training the detector with large datasets of good behavior

(anomaly detection). Worse, the most common practices are manual analysis and signature-based detection [4].

In the paper we present an approach to detect misbehavior in logs without these shortcomings. *The approach combines data mining and both supervised and unsupervised machine learning in order to make the detection process as automatic as possible, without having to instruct the system about how entities misbehave*, although humans cannot be entirely removed from the loop. The approach can be applied to detect misbehavior of different entities, e.g., of hosts, routers, users, base stations, firewalls, etc. In this paper, *hosts* were chosen as they are the most common targets of intrusion in an organization and can misbehave in several different ways: participating in distributed denial of service attacks, exfiltrating confidential data, sending spam mail, mapping the network, attacking other nodes, etc.

The approach involves: (1) data mining a set of *features* from the logs (e.g., bytes sent/received, number of sessions); (2) using an unsupervised machine learning technique – *clustering* – to create sets of entities (of hosts in this paper) with similar behaviors; (3) using a supervised machine learning technique – *linear classification* – to detect misbehaving sets of entities. Sets have to be classified manually the first time clustering is done and whenever the classification fails for some set. However, the fact that the process is done for sets of entities about which much information was collected (features) instead of isolated entities, greatly simplifies manual classification. Moreover, this approach does not require defining how a host misbehaves, just to classify clusters as misbehaving given the values observed for the features.

The main contributions of this work are: (1) an approach for detecting host misbehavior in large logs using a combination of data mining and both supervised and unsupervised machine learning; (2) a set of features relevant for detecting misbehaving hosts; (3) an experimental evaluation of the approach using large logs from a telecommunications infrastructure.

II. NETWORK INFRASTRUCTURE LOGS

This section provides some background on logs and specific information on the logs used in the experimental evaluation of Section IV, which were supplied by Vodafone PT. The fact that we describe these logs does not mean that the approach is specific for them – it is not – only that more detail is useful for the reader to understand the approach. The logs we consider were taken from DHCP servers, authentication servers, and firewalls, all from major vendors.

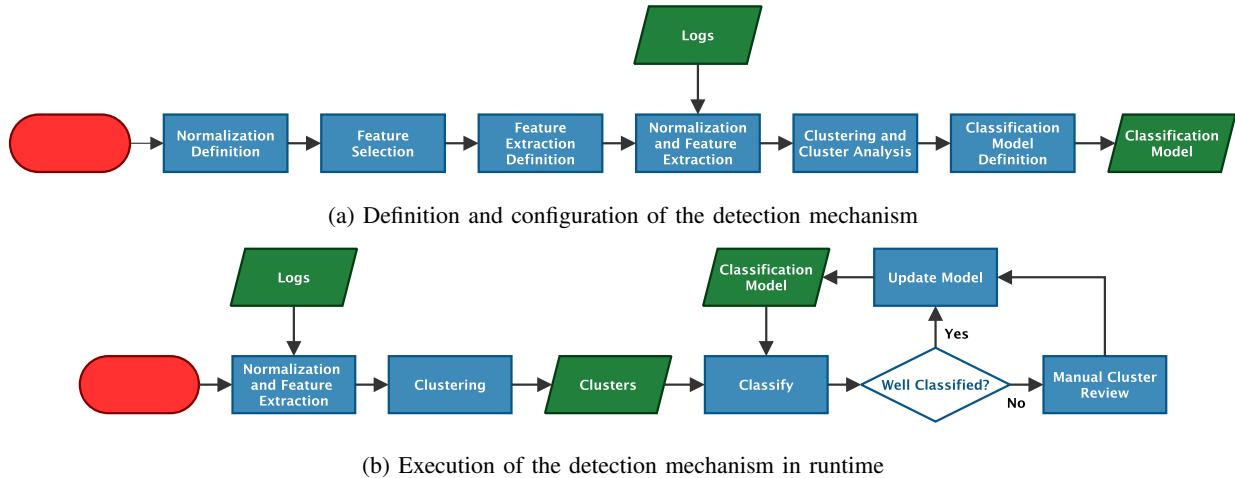


Figure 1: Fluxograms of the two main phases of the proposed approach

Logs are typically text files with an entry (an event) per line, each line with several fields separated by white spaces or commas (e.g., in comma-separated values format [6]). The actual fields found depend on the type of device and the way logging is configured. Common fields are IP and MAC addresses, timestamps, and human-readable messages. The format of logs can vary, but this is the most common format and the one we found in practice.

The entry of a *DHCP server log* has the format “*ID, date, time, description, IP address, host name, MAC address*”, where ID identifies the event, date and time indicate when the event happened, description is text that explains what the event was, IP address, host name, and MAC address identify the host running the DHCP client. DHCP server logs play an important role in the approach as they tell the period when each IP address was used by each host.

A second case are logs from *authentication servers*. These servers are contacted by hosts joining the network as part of an 802.1X authentication process. These logs keep information about each host that attempts to join the network, including a timestamp, a sequence number, host MAC and IP addresses, number of bytes the host sent/received in the session, session duration, and several other items. Examples of data extracted from these logs are how often users log in and the data flow of each session.

Firewalls logs have much data about communication, e.g., send/receiver IP/ports, action (accepted/dropped), connection status, connection type, number of bytes, reason for alert, rule or security policy applied to the connection, textual comment, etc.

III. THE APPROACH

This section presents our approach, starting with an overview of how it works (Section III-A), then detailing its several aspects in the rest of the sections.

A. Overview of the approach

The approach can be divided in two main phases. The first consists in executing a set of steps for *defining and configur-*

ing the detection mechanism and the second in *executing the detection mechanism in runtime*.

The first phase particularizes the approach for a certain *entity* of interest (hosts in our case) and a set of *logs* (those of Section II in our case). Figure 1a shows the steps of this phase. The first (left of the figure) defines how data in the logs will be normalized, e.g., how inconsistent/redundant data will be removed and data from multiple sources will be combined. The second consists in selecting the features that will be used to characterize the available data. This step also involves defining the *time period* (T_f) from which the features are extracted (e.g., $T_f = 1$ day). The third step defines how the features are extracted from the logs.

After these initial steps of phase 1, a first processing of logs is performed (fourth to sixth steps). If the features depend on a single period T_f , then only the parts of logs for that period are analyzed. However, there can be some features that depend also on the previous period (or more), so two periods (or more) may have to be analyzed. The fourth step consists in extracting the features from the logs while also normalizing them. The fifth in using clustering to group entities (e.g., hosts) based on the features and analyzing them manually to classify them in categories such as “behaving normally” and “misbehaving”. Finally, the sixth step consists in defining the classification model, which is the output of this phase (right of the figure).

The second phase is the normal period of execution (Figure 1b). For every time period the following sequence of steps is executed. First, the features are extracted from the logs jointly with normalization and, second, they are clustered producing clusters, similarly to what happens in the first phase. The third step consists in classifying the clusters based on the classification model (which was the output of the first phase). Then, if the classification is successful (Yes), the model is updated; otherwise (No) the clusters not correctly classified have to be analyzed manually and the model is also updated. This process may have to be iterated until the classification process is able to classify all the clusters automatically.

B. Data normalization

As already mentioned, the nature of log data requires normalization. In this section we explain briefly a few normalization techniques, mostly borrowed from [4], [7]. Due to their size, logs should not be normalized before feature extraction, but during feature extraction (see Figure 1).

Hosts are identified in logs in several ways: domain names, IP addresses, and MAC addresses. Moreover, some IP addresses are assigned dynamically using DHCP so they can be used by different hosts. To identify hosts with dynamic addresses we used their MAC addresses, extracting (IP, MAC) mappings from the DHCP logs.

Mapping from static IPs to MACs cannot be extracted from DHCP logs. In the infrastructure we observed that all hosts with static IPs had domain names, so we used them as unique identifiers. If there were hosts with static IPs but no names, the IPs could be used as identifiers.

Some servers store in logs repeated entries, or very similar entries. These repetitions or almost repetitions can be removed to normalize logs (we removed only strict repetitions).

Some networks have facilities in more than one time zone so timestamps may be inconsistent in different logs. In that case timestamps have to be normalized to one time zone, but this was not our case.

C. Feature selection

A critical part of our approach is to have a set of features that allow distinguishing well-behaved entities (hosts) from misbehaving ones. We spent a few months devising a list of features as long as possible, combining features found in the literature with many others inspired by the logs we had and others we discovered. Notice that the goal is not to select features that are known to separate good from bad behavior, but to select features that are candidates to do that separation; the approach works even if some features are bad choices, and making good choices would require knowing a priori how entities misbehave, which is something we do not want to make assumptions about, as already explained.

The features use data from: (1) one or more log entries; (2) other features; (3) one period alone or in combination with the previous one; (4) external datasets (e.g., lists of suspicious IP addresses).

Currently all features we consider are numeric and we observed three patterns that span several: (1) some features count the number of occurrences of something, so they begin with *Number of*; (2) others correspond to the number of occurrences of a particular characteristic of the feature divided by the total number of occurrences, so they begin with *Fraction of* (their values fall in the interval $[0, 1]$); (3) a third group counts the number of different windows of T_w units of time where a threshold K_w is exceeded, with the window being shifted T_s at a time.

Some features use two other terms. *Internal IP addresses* are those that were correctly authenticated through the 802.1X protocol (i.e., hosts from the organization, outsourced personnel and other authorized machines). The rest are called *external IP addresses*.

#	Feature
Session-based	
1	Number of sessions
2	Number of long sessions
3	Fraction of sessions of long duration
4	Burst bytes sent
5	Burst bytes received
Authentication-based	
6	Number of admin authentications tries
7	Number of failed admin authentications tries
8	Fraction of admin authentications tries
9	Burst of admin authentications tries
10	Number of authentication tries
11	Number of failed authentication tries
12	Fraction of failed authentication tries
13	Burst of authentication tries
Connection-based	
14-15	Number of packets sent blocked/allowed
16-17	Number of packets received blocked/allowed
18	Burst of packets sent
19	Burst of packets received
20	Fraction of packets sent blocked
21	Fraction of packets received blocked
22-24	Number of TCP/UDP/ICMP packets sent blocked
25-27	Fraction of TCP/UDP/ICMP packets sent blocked
Endpoint-based	
28	Number of IP addresses in the top of malicious subnets
29	Number of IP addresses with bad reputation
30	Number of external IP addresses not contacted last T_f period
31	Number of internal IP addresses not contacted last T_f period
32	Number of external IP address locations not found last T_f period
33	Number of external IP addresses in the malicious AS list
34	Number of external IP addresses in the spam AS list

Table I: Extracted features

Table I shows the features we defined for classification of hosts. We divided them in four groups. Session- and authentication-based features are extracted from authentication logs, while connection- and endpoint-based features are extracted from firewalls logs. Some of these features depends on information from the previous period, e.g., numbers 30 to 32. We did not define any, but there might be features representing whitelists [4].

The features that use external data are: 28 – uses data from the Internet Storm Center (ISC) list of the top malicious subnets¹; 29 – uses IPs from the AlienVault list of IP addresses with bad reputation²; 33 – uses data from the Sitevet list of worse Autonomous Systems (ASs)³; 34 – based on the CleanTalk list of ASs with more spam activity⁴. Other sources of similar data may be used instead of these or to define similar features.

D. Feature extraction

Big data requires time and bandwidth to be moved to different machines. Therefore, it is advisable to process large logs in the devices that generate them, as close as possible to them (in terms of number of network hops), or in a distributed fashion.

The MapReduce paradigm was introduced by Google for processing data with these characteristics [8] and its open implementation in Hadoop [9] is widely adopted. This model consists of two phases: map and reduce. Inputs are handled as sequences of key/value pairs. In the first phase, the mappers

¹<https://isc.sans.edu/block.txt>

²<http://reputation.alienvault.com/reputation.data>

³<http://sitevet.com/hosts/>

⁴<https://cleantalk.org/blacklists/asn>

transform these pairs in other, intermediate, key/value pairs. In the second, reducers receive intermediate pairs grouped by key and produce the result of the processing. Although simple, this model has been shown to be powerful enough to process many types of large data.

Some examples of feature extraction aspects follow.

A preliminary bulk of log processing has to be done to obtain a table that maps dynamic IP addresses to MAC addresses per period of time $[t_a, t_r]$ (from IP assignment to release). Recall that features are extracted for a certain period of time with duration T_f , say $[t_i, t_i + T_f]$. The table is obtained by analyzing DHCP logs following these rules: (1) an IP assignment entry in the log defines t_a for an (IP, MAC) mapping; (2) a release entry in the log defines t_r for a mapping; (3) an assignment for a mapping without a later release entry, sets t_r to $t_i + T_f$ for that mapping; (4) an IP renew entry to which there is no corresponding assignment entry, sets t_a to t_i .

Feature extraction is done by a small number of MapReduce jobs, although each job extracts many features. We illustrate the process of extracting a feature using a MapReduce job with an abstract example of a mapper and a reducer to count bursts:

- Mapper
 - Objective: extract fields of each log’s entry
 - Input: {offset, line}
 - Output: {(IP, timestamp), (type, MAC, hostname, timestamp)}
- Reducer
 - Objective: count burst of specific feature
 - Input: {(MAC, timestamp), List(field 1, field 2, ..., timestamp)}
 - Output: {MAC, burst count}

Accessing external data sources may be a major cause of log processing delay. Whenever possible this data should be copied to the servers where log processing is done. However, sometimes this is not possible, e.g., when the objective is to verify if a certain IP address belongs to an AS. We implemented a cache to avoid repeating lookups for the same pair (IP address, AS number).

E. Clustering

Machine learning techniques are often classified in two categories. Unsupervised learning algorithms do not require pre-classified input data, whereas supervised learning algorithms take as input data sets with classified data. Classifying data is an onerous process and we do not want to define what misbehavior is, so we use unsupervised machine learning.

We use an unsupervised machine learning technique denominated clustering. The idea of clustering is to group related entities based on a set of features. In our case, for each entity (host) there is a vector with one value for each of the features of Table I. Based on these vectors, related or similar entities are grouped in clusters. Then, these clusters have to be classified but, on the contrary of supervised learning, only a few clusters have to be classified, instead of many individual instances. Moreover, for this kind of application it is reasonable to make the assumption that larger clusters contain well-behaved entities, so the priority is to analyze the smallest ones.

In our approach, typically several features taking very different values will be used, so it is convenient to use normalization [10]. The idea is to normalize the values of every feature to be between $[0, 1]$. This can be done by dividing all features by the maximum value, but in some cases outliers with high values may push most values towards 0 so we may divide the feature values by a lower value and accept some saturation (see Section IV-A).

Our approach aims to separate well-behaved from misbehaving hosts, so the clustering algorithm has to separate hosts with different behavior, being different behavior expressed by different values of features. We use a probability-based clustering algorithm, the *Expectation-Maximization* (EM) algorithm [10]. In relation to other clustering algorithms such as *k*-means, EM has the benefit of not requiring prior knowledge of the distribution of each feature and other parameters. Therefore, EM has been shown to be adequate for clustering of large data sets [11]. EM works iteratively by starting with parameter guesses, uses them to calculate the probability of each entity/host being on a cluster, uses these probabilities to estimate the parameters again, and iterates. EM does not define the number of clusters, which is an input of the algorithm.

Setting the number of clusters is an important aspect. As mentioned, it is reasonable to assume that misbehaving hosts are a minority so the number of clusters has to be large enough for clusters of misbehaving hosts to appear. It has also to be larger than the number of classes of host behavior, which is unknown.

F. Cluster analysis

As explained in the previous section, clustering is done using the numeric, normalized, values of features. However, these values are hard to interpret by humans during manual analysis so we define qualitative feature values to help with that process. These qualitative values are defined at two levels.

First, features are labelled according to how well they characterize a cluster. For each cluster, we label its features with three tags: primary, secondary, non-relevant. *Primary features* are those that best characterize a given cluster. A feature is tagged as primary for a cluster if it has a small deviation within that cluster (e.g., the number of sessions is a primary feature in a cluster if all hosts in that cluster have similar numbers of sessions), i.e., if its standard deviation is less than α (e.g., $\alpha = 0.2$). *Non-relevant features* are those that do not characterize the cluster, i.e., those in which the mean and the deviation are very similar in all clusters (they differ less than a certain θ , e.g., $\theta = 0.001$). *Secondary features* also characterize the cluster but less strongly than primary clusters. A feature is tagged as secondary in a cluster if it is neither primary nor non-relevant.

Second, features are labelled according to their mean in the cluster. We classify qualitatively the values of the mean as: very high (VH) $]0.8, 1]$; high (H) $]0.6, 0.8]$; medium (M) $]0.4, 0.6]$; low (L) $]0.2, 0.4]$; very low (VL) $]0.0, 0.2]$.

With these qualitative values it is possible to assess what differentiates clusters and what their main characteristics are. For instance, it may be observed that nodes in a cluster are different because they send too much traffic (e.g., because

they participate in denial-of-service attacks or exfiltrate data) or access malicious hosts or domains (e.g., because they contain malware that contacts a command and control center).

G. Classification

The existing approaches for detecting malicious hosts based on clustering require intensive human labor, either to keep a database of what is considered malicious behavior, or to analyze the data obtained to know if there is malicious behavior [4]. We propose using classification of clusters to minimize this effort.

Classification algorithms assign entities to certain classes based on a set of features. These algorithms can be used in security to predict whether an entity (e.g., a host) is having malicious behavior or not given a set of features that allow discerning it. In our approach we have the features used to create the clusters so we use them also for classification of entities as malicious or not. More specifically, we aim to classify entities by classifying the clusters outputted by the clustering process as malicious or not, not individual hosts (see the second and third steps in Figure 1b). The reason is that it is simpler to examine a few clusters of entities instead of many individual entities (hosts in our case).

The clusters for each time period T_f are introduced into the classification algorithm, and after checking the existence of errors the model is updated (see figure). With a period of one day it takes long to obtain enough data to increase the accuracy of the model. Therefore, the classification algorithm used is Naive Bayes as it converges quicker than other models and needs less data to obtain accurate results [10].

Our approach requires considerable human labor in the first phase (Figure 1a) in order to classify the clusters and create the classification model. After that, human intervention is needed only if the classification results are wrong, updating the model. Thus over time the accuracy of the classification model increases.

The classification model is based on the mean and the standard deviation of each feature of each cluster. Each time the classification algorithm is executed, we obtain a *class* for the introduced clusters data. These classes can be as simple as “normal” and “abnormal”, or more detailed as: “abnormal-high-sender”, “abnormal-many-sessions”, “normal-workstation”, “normal-server”, “normal-printer”. In any case, the abnormal entities will normally be further investigated.

IV. EXPERIMENTAL EVALUATION

This section describes the experimental evaluation of our approach. The objective is threefold: (1) to show that our clustering mechanism can identify abnormal behavior (Section IV-A); (2) to show that the use of a classifier can automate the process of assigning classes to clusters (Section IV-B); (3) to assess the performance of the process (Section IV-C).

In order to evaluate our approach we used logs provided by Vodafone Portugal for 5 consecutive days (23-27 February 2015). We used a time period $T_f = 1$ day. Some features require data from the previous period (day) to be computed, so the first phase of the approach (recall Figure 1) was applied to the logs from the day 2. The second phase was applied to the other three days (3–5).

Our current prototype has basically two components. First, the normalization and feature extraction are performed by a set of mappers and reducers that are executed by Hadoop MapReduce. We have 10 mappers and 10 reducers, adding up to 4205 lines of code written in Java. Second, the clustering and classification are done using WEKA, a well-known data mining software package written in Java [10]. Table II shows the number of lines of code for each map-reduce job plus the auxiliary classes.

Log	Extracted Data	Mapper Size	Reducer Size	Others
Firewall	Previous Day Input	358	86	290
DHCP	IP-MAC Mapping	83	106	162
Authentication serv.	Session Features	145	199	274
Authentication serv.	Authentication Features	65	136	81
Firewall	Connection Features	120	172	242
Firewall	End-point Features	226	528	502
Total		997	1227	1551
Lines of code common to all jobs				430

Table II: MapReduce lines of code for normalization and feature extraction (Java).

We run Hadoop in a single server due to practical reasons. Nevertheless, we setup Hadoop to use the 32 cores of the server that correspond to 16 servers with 2 cores, which is the default configuration of Hadoop. In order to have an efficient configuration, we followed the Hortonworks recommendations⁵. Moving the data into the server was as considerable effort that took several days, showing the importance of running at least the mappers as close as possible to the logs’ locations.

A. Discovering intrusions with clustering

After extracting the features using Hadoop, they have to be normalized to the interval $[0, 1]$. To do so we have to define the maximum for each feature, i.e., the value by which the values of each feature will be divided. As mentioned, it is not necessarily convenient to use the actual maximum as that value may push values towards 0, making it difficult to differentiate them. Therefore, we choose the maximum for each feature by trial and error, analyzing the percentage of computers within each qualitative range (VH, H, M, L, VL) for a tentative maximum, and trying with another value until an acceptable distribution is reached. The final values we selected for each feature are in Table III. The features relative to fractions were excluded because of the bad results we obtained when applying the clustering algorithm to our logs (but we expect they may be useful in other cases). In addition, there are two columns which represent the percentage of hosts that exceed the maximum value of the feature (saturated), and the percentage of computers whose value is 0 (zero). Some features have a value of 100% in column VL as all hosts had the feature equal to zero.

The next step is to choose the number of clusters. We executed the clustering algorithm varying the number of clusters from 15 to 25, then choose the number to use with the following criteria: (1) the percentage of hosts in each cluster shall be small with exception of the clusters that represent the most common behavior; (2) the number of primary features in each cluster shall be sufficient to characterize the hosts.

⁵http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html

Cluster	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
#	1	55	566	9	207	72	78	61	697	25	86	27	53	1091	54	109	83	459	169	35	34	205	89
%	0.02	1.29	13.27	0.21	4.85	1.69	1.83	1.43	16.34	0.59	2.02	0.63	1.24	25.58	1.27	2.56	1.95	10.76	3.96	0.82	0.80	4.80	2.09
Feature																							
1	VL	VL	VL	VL	VL	VL		VL	VL	VL		VL	VL	VL	VL	VL		L	VL		VL	VL	
2	VL	VL	VL	VL	VL	VL	L	VL	VL	VL		VL	VL	VL	VL	L		L	VL		VL	VL	L
4	VL	VL	VL	VL	VL	VL		VL	VL	VL		VL	VL	VL	VL	VL		VL	VL		VL	VL	
5	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL			VL	VL		VL	VL	VL		VL	VL	VL
6																							
7																							
9																							
10	L		VL	VL	VL		VH		VL	VL	VH			VL	VH			VH	VL		VL	VL	VL
11	VL	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
13	VL	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
14	VL	VL	VL	VL	L		VL	VL	VL	VL	VL	VL	VL	VL	VH	VL	L	VL	VL	VH	VL	VL	VL
15	VL	VL	VL	VL	VH		VL	VL	VL	VL	VL	VL	VH	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
16	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
17	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	L	VL	VL	VL	VL	VL
18	VL	VL	VL	VL	VH		L	H	VL	VL	VL		VH	VL	VH	VL		M	VL		VH	VL	VL
19	VL	VL	VL	VL	VH	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	M	VL
22	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VH	VL	VL
23	VL	VL	VL	VL	L	VL	VL	VL	VL	VL	VL	VL	VL	VL	VH	VL	VL	VL	VL	VL	VL	VL	VL
24	VL	L		VL	VL	VL	VL	VL	VL	VL	VL	L	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
28																							
29	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
30	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
31	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
32	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
33	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
34	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL	VH	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL

Table IV: Cluster description in terms of hosts it contains (total 4265) and primary features.

Feature	Maximum	VL %	L %	M %	H %	VH %	Saturated %	Zero %
1	10	82.02	10.88	4.34	1.20	1.57	1.13	54.70
2	5	88.25	8.70	2.27	0.49	0.28	0.14	58.94
4	550	95.62	0.61	1.01	1.01	1.76	1.31	93.81
5	550	98.59	0.16	0.33	0.38	0.54	0.45	97.84
6	5	100.00	0.00	0.00	0.00	0.00	0.00	100.00
7	5	100.00	0.00	0.00	0.00	0.00	0.00	100.00
9	5	100.00	0.00	0.00	0.00	0.00	0.00	100.00
10	5	66.12	17.47	2.72	5.98	7.71	5.89	58.01
11	5	98.57	0.82	0.12	0.16	0.33	0.33	96.30
13	50	97.89	0.35	0.87	0.23	0.66	0.56	97.68
14	1000	88.70	4.74	1.62	0.91	4.03	3.19	45.30
15	1000	80.84	2.46	2.56	2.98	11.16	8.79	36.39
16	500	98.94	0.14	0.05	0.14	0.73	0.70	94.11
17	1000	86.89	4.41	3.35	1.59	3.75	2.04	68.89
18	545	49.12	12.43	10.86	12.94	14.65	2.79	37.49
19	500	82.39	3.31	3.45	6.66	4.20	2.16	79.70
22	600	90.29	2.51	2.18	1.03	3.99	3.38	54.35
23	500	92.78	3.12	0.75	0.47	2.88	2.49	58.80
24	150	88.42	9.80	0.61	0.28	0.89	0.47	80.54
28	5	100.00	0.00	0.00	0.00	0.00	0.00	100.00
29	10	99.62	0.07	0.05	0.05	0.21	0.21	99.44
30	100	97.07	0.96	0.45	0.28	1.24	1.08	91.18
31	100	99.84	0.05	0.02	0.00	0.09	0.09	98.80
32	100	98.80	0.28	0.16	0.26	0.49	0.38	96.51
33	300	97.63	0.45	0.23	0.19	1.50	1.48	93.18
34	250	94.30	1.62	0.82	0.26	3.00	2.77	79.23

Table III: Maximum values of each feature.

Table IV shows the clusters obtained by applying the clustering algorithm set for 23 clusters to the logs of day 2. The table shows also the qualitative values of the primary features. The values for secondary and non-relevant features were left blank. The qualitative values were defined using the values in Section III-F (including $\alpha = 0.2$ and $\theta = 0.001$).

An immediate observation is that the largest cluster is number 14 with 25.58% of the hosts. For this cluster, all features related to firewalls fall in the VL range. We used the MAC addresses of a sample of these hosts to find what kind of hosts they were and found devices such as workstations. The second largest cluster is number 9 with 16.34% of the hosts. The features of this cluster are almost identical to those of cluster 14, all VL, with the exception of bursts sent (H). By inspecting a sample of the hosts we found again well-behaved machines (bursts do not mean much traffic but peaks of traffic). These are clearly normal hosts in the company. The rest of the clusters correspond to hosts with less frequent, possibly malicious, behavior.

We did not investigate all the smaller clusters, only the three that looked most suspicious. Cluster 13 has a few problematic features with value VH: number of packets sent allowed; bursts

of packets sent; number of external IP addresses contacted in the malicious AS and spam AS lists. The last two are conspicuous as all the other clusters have VL in these two features, so we assigned the cluster the class label *abnormal-suspicious-ASs*.

We analyzed in more depth some of the hosts in this cluster and confirmed that there are reasons to suspect of malicious behavior. We found communication with IP addresses reported to distribute malware and alarms in an anti-virus about a couple of Trojan horses. Moreover, it was possible to understand that some of those hosts were not under tight control of the company, e.g., because they belonged to external companies (suppliers).

Cluster 15 has also some problematic features with VH: number of authentication tries; number of packets sent blocked by the firewall; bursts of packets sent; number of UDP packets sent blocked by the firewall. Cluster 20 has also a VH number of packets sent blocked by the firewall and TCP packets sent blocked by the firewall. We set the class labels for these two clusters to *abnormal-authentications-blockedUDP* (cluster 15) and *abnormal-blockedTCP* (cluster 20).

B. Classifying clusters

As explained, the initial classifier is defined during the first phase of the approach. An option is to define a class for each cluster, then define the classifier based on these classes. Following this path we have 23 classes, although in the previous section we gave names just to 3 of them. This classifier is then refined in the following iterations of the process for the next time periods (i.e., days).

Table V shows the values of the features for example hosts classified in these 3 classes (for lack of space we reduced the names of the classes and omitted features 6-9 that were never primary). Each host is shown in a row. Each day the wrong labels were corrected and the classification model was updated with the new information. Discrepancies in the early days are considered normal as the classification model has limited information. As our approach is incremental, the accuracy of the predictions will increase over time.

Class \ Day		day 3	day 4	day 5
abnormal-suspicious-ASs	classified in class	0	1	1
	false positives	0	0	0
	false negatives	1	1	1
abnormal-authentications-blockedUDP	classified in class	3	6	1
	false positives	1	6	0
	false negatives	2	1	1
abnormal-blockedTCP	classified in class	0	0	1
	false positives	0	0	0
	false negatives	1	2	1

Table VI: Clusters classified in the 3 suspicious classes per day.

Log \ Day	1	2	3	4	5
Firewall (previous day input)	1h52m	1h54m	2h39m	2h38m	
DHCP (IP-MAC Mapping)		24s	24s	25s	24s
Authentication serv. (Session)		48s	49s	49s	50s
Authentication serv. (Authentication)		27s	27s	28s	28s
Firewall (Connection)		55m24	1h36m	1h32m	1h17s
Firewall (End-point)		1h09m	1h38m	1h34m	1h21m
Total	1h52m	3h40m3s	5h54m40s	5h45m42s	2h39m42s

Table VII: Feature extraction time per day and per log.

Table VI summarizes the classification for the 3 days to which the second phase was applied, again just for the three suspicious classes. The term *classified in class* is used to mean the number of clusters classified in that class. The term *false positives* denominates the clusters wrongly classified in that class. Finally, *false negatives* means the clusters that should have been classified in that classes automatically but were not, so they had to be reclassified manually. Ideally the false positives and false negatives would be zero, which was not the case. However, these numbers tended to lower with the number of days, but the total number of days was too small for them to reach zero.

C. Performance

The last part of the experimental evaluation is an assessment of its performance, i.e., of time required to run the approach. The approach has several steps but, disregarding manual interventions, the bottleneck is the extraction of the features from the logs, so the section focus on this aspect. The rest of the steps take at most 2 minutes.

The overall execution time per day and per log needed to normalize and extract the features are shown in Table VII. For day 1, only one job was executed to create the input necessary to obtain the features of day 2 that depend on the previous day (features 30–32), so the rest of the rows are empty. It is also noteworthy that the extraction of data from the DHCP and authentication server logs took just seconds, as these logs are small (see Table VIII). The rest of the logs took approximately 1 to 3 hours to be processed.

To understand if the processing time is proportional to the size of the log we plotted the time to extract features from logs versus the size of the logs in Figure 2. We considered only the logs from the first type of firewalls, which were the largest (tens of gigabytes). The figure suggests that indeed the processing time increases approximately linearly with the size of the log.

V. RELATED WORK

Several tools have been appearing for processing large logs. MapReduce has the more generic objective of processing large

Log Source \ Day	1	2	3	4	5
Firewall type 1	51 GB	44 GB	69 GB	68 GB	59 GB
Firewall type 2	18 MB	18 MB	18 MB	18 MB	18 MB
DHCP	14 MB	14 MB	14 MB	14 MB	14 MB
Authentication serv.	222 MB	202 MB	201 MB	197 MB	210 MB

Table VIII: Log size per day per log source.

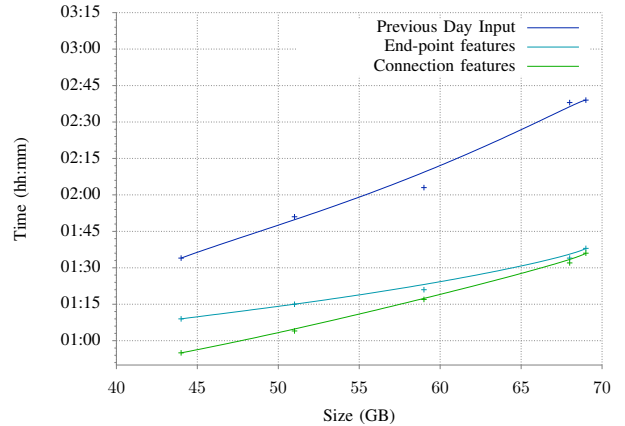


Figure 2: Time for feature extraction versus size of the logs.

data sets, but the original paper mentions the specific case of web logs [8]. Hadoop has also been much used to process logs [9]. The MapReduce model with its mappers and reducers is simple, so similar but more versatile models have been proposed. For instance, Dryad models processing as a graph with different possible interconnections (vertices of the graph) between processing units (nodes of the graph) [12]. Pig Latin and Pregel also do more generalized graph processing [13], [14]. There are also tools that are specific for processing logs. Splunk is one of the most adopted [15]. It has powerful visualization, indexing, search, and reporting mechanism that can be used for security and other kinds of analysis. ElasticSearch is a similar open source tool based on the Apache Lucene text search library [16]. There are many similar tools available. Although able to analyze logs, even for security, these tools do not discover misbehaving entities, which is the purpose of our approach.

Data mining and machine learning have been used to analyze logs before. Chen et al. use hierarchy clustering to analyze telephony logs to discover faulty states [17]. They do not use a rich set of features but a single one: pair-wise dissimilarity between log entries. Shang et al. aim to diagnose big data analytics applications developed in pseudo-cloud environments moved to the cloud [18]. They compare logs from both environments to look for discrepancies and help diagnosis problems that happen during that transfer of environments. Data mining has been used to extract user profiles from logs, e.g., for marketing purposes [19]. Similarly, data mining has been used to extract performance information from logs [1]. These works are interesting but they are not about security and use very different approaches.

We are aware of three papers about large-scale log analysis for security. Beehive is probably the closest to ours as it also uses clustering, but it does not do supervised classification of the clusters, considers much less features, and detects malicious users instead of hosts [4]. LogMaster aims to mine correlations of events but, on the contrary of Beehive and our approach, takes into account the order between events by analyzing n-ary sequences [7]. This allows to find interesting forms of misbehavior but limits scalability. LogMaster also does not aim to identify misbehaving entities specifically. Giura and Wang present an approach to detect advanced persistent threats in logs [5]. The work is quite ambitious as it aims to do near real-

Class \ Feature	1	2	4	5	10	11	13	14	15	16	17	18	19	22	23	24	28	29	30	31	32	33	34
Day 2																							
suspicious-ASs	0.4	0.0	0.0	0.0	0.0	0.2	0.2	1.0	1.0	0.0	0.0	0.6	0.0	1.0	1.0	0.0	0.0	0.1	1.0	0.0	0.78	1.0	1.0
authentications-blockedUDP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.48	0.0	0.0	0.8	0.0	0.04	0.92	0.0	0.0	0.0	0.16	0.0	0.0	0.0	0.30
blockedTCP	0.4	0.0	0.0	0.0	0.0	0.2	0.2	0.03	1.0	0.19	0.0	0.65	0.53	1.0	0.02	0.0	0.0	0.0	1.0	0.0	0.0	0.00	1.0
Day 3																							
authentications-blockedUDP	0.2	0.0	0.0	0.0	0.0	0.4	0.2	0.00	1.0	0.0	0.0	0.073	0.0	1.0	0.36	0.0	0.0	0.0	0.26	0.0	0.01	0.03	1.0
authentications-blockedUDP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.14	1.0	0.0	0.0	1.0	0.01	0.16	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
Day 4																							
suspicious-ASs	1.0	0.0	0.0	0.0	0.0	0.4	1.0	1.0	0.05	0.00	0.00	1.0	0.0	0.04	0.04	0.0	0.0	0.0	0.24	0.01	0.96	1.0	1.0
authentications-blockedUDP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.08	0.0	0.03	0.0	0.26	0.06	0.0	0.0	0.0	0.0	0.0	0.49	0.24	0.0	0.0	0.0
authentications-blockedUDP	0.4	0.0	0.0	0.0	0.0	0.0	0.2	0.04	0.63	0.0	0.00	0.91	0.0	0.87	0.09	0.45	0.0	0.0	0.0	0.0	0.0	0.0	0.11
Day 5																							
suspicious-ASs	0.4	0.0	0.0	0.0	0.0	0.2	0.2	1.0	1.0	0.0	0.0	0.69	0.0	1.0	1.0	0.0	0.0	0.5	0.79	0.0	1.0	1.0	1.0
authentications-blockedUDP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	1.0	0.0	0.0	1.0	0.0	0.09	1.0	0.0	0.0	0.0	0.21	0.0	0.01	0.0	0.36
blockedTCP	0.8	0.0	0.0	0.0	0.0	0.4	0.4	0.14	1.0	0.01	0.0	1.0	0.0	1.0	0.44	0.0	0.0	0.0	0.42	0.0	0.03	0.11	1.0

Table V: Feature values for example hosts classified in the 3 suspicious classes (normalized values).

time detection and considers several data planes (physical, user, network, application), but mining techniques are applied only to time and plane correlation, as detection is based on the three known techniques mentioned above (misuse detection, anomaly detection, and policy-violation detection). In summary, neither of these three works combine data mining and both supervised and unsupervised machine learning for misbehavior detection in large logs as we do.

Supervised machine learning and data mining techniques have been much adopted for anomaly-based intrusion detection [20], [21], [22]. The generic approach consists in defining a model of normal behavior based on a set of training data, collected in real-time or taken from logs; then, a metric of distance is used to assess if runtime behavior deviates from that model. On the contrary of our approach, this form of detection involves having a data set that does not contain misbehavior.

VI. CONCLUSION

We present an approach to identify malicious entities based on large logs from several devices. We consider in more detail the specific case of identifying malicious hosts. Our approach uses a combination of data mining and both supervised and unsupervised machine learning to make the detection process as automatic as possible, without having to instruct the system about how entities misbehave, only defining a set of features that may be used to distinguish good from bad behavior. We present an experimental evaluation of our approach with a large data set from a telecommunications company (approx. 290 GB).

The output of the process we present is not accurate enough to take automatic actions, e.g., to quarantine the hosts in a cluster classified as suspicious. However, it extracts relevant security information from the logs that otherwise is not directly observable. That information is valuable to take actions in combination with other information available in the company (e.g., type of device, internal or external host, anti-malware alarms). Although the process is not fast, having this information in a few hours is useful for many purposes. The outputs can also be stored in a database for historical analysis of the behavior of hosts and eventual actions.

REFERENCES

- [1] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, 2012.

- [2] A. Cárdenas, P. Manadhata, and S. Rajan, "Big data analytics for security," *IEEE Security and Privacy*, vol. 11, no. 6, 2013.
- [3] —, "Big data analytics for security intelligence," Cloud Security Alliance, 2013.
- [4] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th ACM Annual Computer Security Applications Conference*, 2013.
- [5] P. Giura and W. Wang, "Using large scale distributed computing to unveil advanced persistent threats," *Science Journal*, vol. 1, no. 3, 2012.
- [6] Y. Shafranovich, "Common format and MIME type for comma-separated values (CSV) files," Request for Comments (RFC) 4180, Oct. 2005.
- [7] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu, "LogMaster: mining event correlations in logs of large-scale cluster systems," in *Proceedings of the 31st IEEE Symposium on Reliable Distributed Systems*, 2012.
- [8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2004.
- [9] T. White, *Hadoop: The Definitive Guide*. O'Reilly, 2009.
- [10] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.
- [11] P. S. Bradley, U. Fayyad, and C. Reina, "Scaling EM (expectation-maximization) clustering to large databases," Microsoft Research, Tech. Rep. MSR-TR-98-35, 1998.
- [12] M. Isard, M. Budi, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, 2007.
- [13] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008.
- [14] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010.
- [15] J. Stearley, S. Corwell, and K. Lord, "Bridging the gaps: joining information sources with Splunk," in *Proceedings of the 2010 Workshop on Managing Systems via Log Analysis and Machine Learning Techniques*, 2010.
- [16] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey, "Mining modern repositories with Elasticsearch," in *Proceedings of the 11th IEEE Working Conference on Mining Software Repositories*, 2014.
- [17] C. Chen, N. Singh, and S. Yajnik, "Log analytics for dependable enterprise telephony," in *Proceedings of the 9th European Dependable Computing Conference*, 2012.
- [18] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on Hadoop clouds," in *Proceedings of the 2013 IEEE International Conference on Software Engineering*, 2013.
- [19] G. Stermsek, M. Strembeck, and G. Neumann, "A user profile derivation approach based on log-file analysis," in *Proceedings of the International Conference on Information and Knowledge Engineering*, 2007.
- [20] D. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, 1987.
- [21] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003.
- [22] G. Nascimento and M. Correia, "Anomaly-based intrusion detection in software as a service," in *IEEE/FIP International Conference on Dependable Systems and Networks Workshops*, 2011.