TÉCNICO LISBOA

Automatic Diagnosis of Security Events in Complex Infrastructures using Logs

Daniel Dias Gonçalves

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Miguel Nuno Dias Alves Pupo Correia
Eng. João Duarte Parrinha Bota

Examination Committee

Chairperson: Prof. José Carlos Alves Pereira Monteiro
Supervisor: Prof. Miguel Nuno Dias Alves Pupo Correia
Member of the Committe: Prof. Fernando Manuel Valente Ramos

May 2015

Privacy is not for the passive.

Jeffrey Rosen

Acknowledgments

I would like to thank to my supervisor Miguel Correia and my co-supervised João Bota for all granted help and support in the course of this work.

I warmly thank João Gomes, Naércio Magaia and Pedro Peixe Ribeiro for their assistance throughout of this work. To my friends Daniel Mendes, Nadiya Kyryk, Nelson Sales, Mariana Silva, Rodrigo Lourenço, Duarte Pompeu and Jorge Santos, thank you all for the support and companionship in my academic progress.

Lastly, I am grateful to my family, Mário Gonçalves, Anabela Gonçalves, Vânia Gonçalves and André Gonçalves as without them I would never achieve this goal.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 and by Vodafone Portugal.

Thank you all!

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.

This dissertation presents a novel solution to assess the security of such an infrastructure using its logs, inspired on data from a real telecommunications network. The solution uses 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.

Keywords

Big Data, Security, Feature Selection, Detecting Host Misbehavior, Scalability and Log File Analysis.

Resumo

A gestão de infraestruturas de redes complexas continua a ser uma tarefa difícil hoje em dia. Estas infraestruturas podem conter um grande número de dispositivos que podem comportar-se de maneiras imprevisíveis.

Muitos destes dispositivos mantêm registos com informações importantes sobre segurança, fiabilidade e desempenho da infraestrutura. No entanto, extrair informação destes dados está longe de ser trivial.

Esta dissertação apresenta uma solução para avaliar a segurança de infraestruturas usando *logs*, inspirada em dados de uma rede de telecomunicações real. São utilizadas técnicas de aprendizagem e mineração de dados para analisar os dados e semi-automaticamente descobrir se um determinado dispositivo tem um comportando fora do normal, sem ter de instruir o sistema sobre o que é considerado um mau comportamento.

Palavras Chave

Big Data, Selecção de Características, Deteção de Computadores com Mau Comportamento, Escalabilidade, Análise de *Logs*.

Contents

List of Figures

List of Tables

Abbreviations

API Application Programming Interface

AS Autonomous System

CPU Central Processing Unit

CRLF Carriage Return Line Feed

CSV Comma-Separated Values

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

Dryad Distributed Data-Parallel Programs

EM Expectation-Maximization

FIFO First In-First Out

FTP File Transfer Protocol

HDFS Hadoop Distributed File System

HTTP HyperText Transfer Protocol

ICMP Internet Control Message Protocol

ID Identifier

IDS Intrusion Detection System

IETF Internet Engineering Task Force

IP Internet Protocol

MAC Media Access Control

RFC Requests for Comments

SLD Second Level Domain

SMTP Simple Mail Transfer Protocol

SQL Structured Query Language

SVM Support Vector Machines

TCP Transmission Control Protocol

UDP User Datagram Protocol

URL Uniform Resource Locator

List of Symbols

1

Introduction

Contents

This chapter starts by presenting the problem of log analysis for security, then summarizes the approach proposed in the dissertation. Finally, the chapter details the goals and contributions of this work and the structure of the document.

1.1 Motivation

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.

Traditional Intrusion Detection Systems (IDSs) are critical components in many organizations today, but there is an increasing need for approaches that gather security data from a wider variety of heterogeneous sources and correlate it in order to gain better situational awareness [2].

Extracting useful information from various heterogeneous sources for the purpose of correlating it involves several steps [3]:

1. Remove noise and inconsistent data;
2. Combine multiple data sources;
3. Select data to be analyzed;
4. Consolidate data into forms appropriate for mining;
5. Data mining in order to extract patterns;
6. Patterns evaluation;
7. Presentation of the knowledge extracted.

This dissertation 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, 4, 5] 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 [1, 6, 7]. 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 [6].

1.2 Goals and Contributions

This dissertation presents an approach to detect misbehavior in logs without the previously mentioned 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 dissertation, *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.

Consider the Table 1.1 as an example of data about a misbehaving host. The detection of abnormal

Timestamp	Protocol	Source Host	Destination Host
1.1	TCP	Host_A	Malware Domain
1.2	TCP	Malware Domain	Host_A
1.3	TCP	Host_A	Malware Domain
1.4	TCP	Malware Domain	Host_A
1.5	TCP	Malware Domain	Host_A
...	...	...	...
3.1	UDP	A	B
3.2	TCP	B	D

Table 1.1: Entries set that describes a misbehavior host sending spam mail.

behavior in this example is trivial, since few log records are represented. Moreover, the represented records explicitly inform that the connection is made with a malware domain. However, huge infrastructures contain a large number of records and only references to Internet Protocol (IP) addresses. This approach aims to solve this kind of problems.

The approach proposed in this dissertation involves:

- Data mining a set of *features* from the logs (e.g., bytes sent/received, number of sessions);
- Using an unsupervised machine learning technique – *clustering* – to create sets of entities (of hosts in this dissertation) with similar behaviors;
- 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.

1.3 Thesis Outline

Chapter 2 describes several existing approaches related to analysis of huge logs that are relevant to this work, as well as what differentiates them with this work.

The approach is detailed in Chapter 3, describing all the details from the pre-processing to the classification model.

The evaluation results are presented and discussed in Chapter 4.

Chapter 5 concludes the dissertation and discusses future work.

2

State of the Art

Contents

The purpose of this chapter is to describes problems and existing approaches for log analysis to a better understanding of the presented approach.

The remaining chapter is divided as follow: Section 2.1 will address the issue of pre-processing data; in Section 2.2 are presented various data mining techniques in order to obtain patterns; in Section 2.3 are presented various methods of analysis of these patterns in terms of security; and in Section 2.4 are discussed in general other possible analysis of these patterns.

2.1 Preparation and Basic Analysis of Logs

In a complex system logs have a wide variety of sources, therefore they may be differently structured and have various formats and sizes. Moreover, in most of the cases repetitive and periodic data exists. A major challenge is to accomplish the removal of inconsistent records and obtain all relevant information to be analyzed.

It is important to follow some pattern when registering data in logs. There was one attempt to keep all logs homogeneous, without success, that resulted in a Internet Engineering Task Force (IETF) draft [8], whose name is Universal Format for Logger Messages. However, more and more the Comma-Separated Values (CSV) format is used to represent sets or sequences of data. Each record is composed always with the same fields, so import and export data in this format is easy. As the name suggest, it contains multiple records (one per line), and each field is delimited by a comma. There is no definitive standard for CSV, however the most commonly accepted definition is RFC4180 [9]. Thus, the CSV format has the following proprieties:

- Each record is located in a different line and is delimited by a line break (CRLF);
- The last record may not have an ending line break;
- There may be an initial line that contains names corresponding to the fields of each record;
- The fields of the initial line and of each record are separated by commas. Each line has exactly the same number of fields and the last field in the record must not be followed by a comma;
- Each field may or may not be enclosed in double quotes;
- Fields containing line breaks, double quotes and commas should be enclosed in double-quotes;
- Whether double quotes are used to enclosed fields, then a double-quote appearing inside a field must be escaped by preceding it with another double quote.

2.1.1 Data Filtering

Complex infrastructures can generate huge logs, so it is important to make them smaller in order to facilitate the analysis and obtain timely results. The LogMaster approach identified the following forms of inconsistent data in various devices logs, which should be filtered for further analysis [10]:

Redundant data. Log records which represent the generation of a series of error messages from the same source. As an example, a switch which produce a huge amount of error records in a short time until it is repaired.

Periodic data. All data that is always recorded after a certain time interval, in other words periodically. As an example a synchronization service running and producing a record every day at 6 pm.

In order to filter that kind of inconsistent data, Fu et al. proposed two solutions based on statistic algorithms [10]. For removal of redundant data the interval between the current record and the previous one with the same log identifier is calculated, and if the current record has an interval smaller than a threshold, it is identified as redundant data and removed. For the periodic data, each time interval has a counter which represents the number of occurrences. Whether a specific time interval has the counter and the percentage of incident data greater than a threshold, it is considered a fixed cycle, thus periodic data.

LogMaster aims to mine correlations of events but, on the contrary of the approach of this dissertation, it takes into account the order between events by analyzing n-ary sequences. This allows to find interesting forms of misbehavior but limits scalability. LogMaster also does not aim to identify misbehaving entities specifically.

2.1.2 Data Normalization

Another important factor is the coherence of the data, an aspect considered in Beehive [6]. 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. The system uses several logs from different security solutions in order to find patterns and detect malicious activities within the organization.

Beehive uses the following strategies in order to normalize the records along the logs:

Timestamps. The devices within an organization may be located geographically at different points, thus the timestamps not always are recorded with the same format across the logs. The devices which sent logs have specific timestamps $t_{d,i}$, as well as the receiver device $t_{rec,i}$. For each device the difference between the sent timestamps and the actual time of the receiver device is performed ($\Delta_{diff} = t_{d,i} - t_{rec,i}$). After a specific time period (e.g., $T_{corr} = 1$ day) it is defined the correlation value of each device ($\Delta_{corr,i}$) as the majority within the set of values previously obtained. The normalized timestamp values are obtained applying this correlation value, i.e., $t_{normalized,d} = t_{rec,i} + \Delta_{corr,i}$.

Mapping IP addresses to hosts. In the majority of cases, IP addresses are assigned to devices in a dynamic way, i.e., throw a Dynamic Host Configuration Protocol (DHCP) server. The IP addresses can be assigned to different devices over time, thus being necessary to follow some strategy in order to obtain the IP address assigned to a host in a specific time. The proposed solution by Chen et al. is analyze the DHCP server logs to create an IP addresses database assigned to a host given a time interval. This process is executed in a daily basis [11].

Detecting static IP addresses. The previous strategy does not take into account the mapping between a static IP address and a host. Thus, a different strategy was adopted. In a first step, the set of all IP addresses found in all logs are stored, denoted by A. Then the set of IP addresses of the previous strategy (D) are removed from this set ($S = A - D$). To the new set, the name of each host is found by doing a reverse Domain Name System (DNS) lookup, storing the results. All these steps are performed periodically (e.g., once a day) in order to keep an updated set of static IP addresses. In each iteration the results are compared and whether some host name changed, then it is removed from set S.

Detecting dedicated hosts. Another important aspect is to identify all dedicated hosts, which means the hosts used by only one user. This allow to find the normal behavior of each one of these hosts. For that, the history of all users that log on to each host during a time period is kept, and it is considered a dedicated host, whether after this period of time an specific user is responsible for 95% of all authentications.

2.1.3 Features Extraction

The analysis of large data sets is a difficult endeavor. There are two main aspects related to quality of data: relevance and non-redundancy [12].

Feature extraction is the process of extracting new values from the initial set of data, i.e., of reducing the amount of data needed to describe a large set of data. The extracted set of features shall contain relevant and non-redundant information for the analysis. Learning algorithms may use these extracted features to find patterns in data and are divided in two main categories [13]:

Supervised Learning. These algorithms receive at an early stage a data set correctly classified. The classification of new data is based on this initial set.

Unsupervised Learning. Algorithms of this type do not receive any data set correctly classified, and new data is classified according to the classifications previously made.

Before applying the feature extraction process it is necessary to select what features to extract. There are two types of methods for automatic features selection [14]:

Filters. These methods apply a statistical measure to assign a score to each feature. The features are ranked by the score and the least interesting features are suppressed, i.e., the features smaller than a threshold are discarded. The subset selection procedure is independent of the learning algorithm.

Wrappers. A subset of features is selected estimating the accuracy of the learning algorithm. In each iteration, the procedure estimates the accuracy of adding each unselected feature to the subset and chooses the feature with a better increase on accuracy. Accuracy is estimated using cross-validation on the training set.

Wrappers are computationally very expensive and do not scale well to large data sets, as opposed to filters that are fast. On the other hand, using wrapper methods the features selection is more rigorous and therefore can be used for small datasets.

The approach of this dissertation does manual feature selection, since we have domain knowledge, i.e., it is possible to construct a better set of features than with the automatic methods [15]. The set of features to be extracted depends on the type of analysis to be achieved, thus the rest of this dissertation is focused on security.

2.1.4 Log Analysis Tools

This section presents log analysis tools used by many organizations to process large amounts of data in order to find hidden patterns and correlations between them. This process is generically called *big data analytics* [5, 16, 17]. Once organizations have a large amount of data, the facility of having a good decision making relies on a good analysis. Several tools have been appearing for processing large logs, some of them described bellow.

Hadoop Hadoop is a tool for processing large volumes of data. In order to support parallel execution, it follows MapReduce programming model, which was originally implemented by Google [18], being this implementation not available. The MapReduce has the more generic objective of processing large data sets, but the original paper mentions the specific case of web logs [19]. Hadoop has been much used to process logs [20]. This model consists of two phases: one phase of *Mapping* and another of *Reducing* (see figure 2.1). The input data are divided into a list of key/value pairs, where the processes of the first phase, the mappers, transform the received pairs in others intermediate pairs. Then, each process of the second phase, reducers, receives the intermediate pairs, that contain the same key in order to carry out the reduction in parallel. Hadoop uses the Hadoop Distributed File System (HDFS) file system which is an implementation of the Google File System. The files are divided and stored as fixed-size blocks except for the last block.

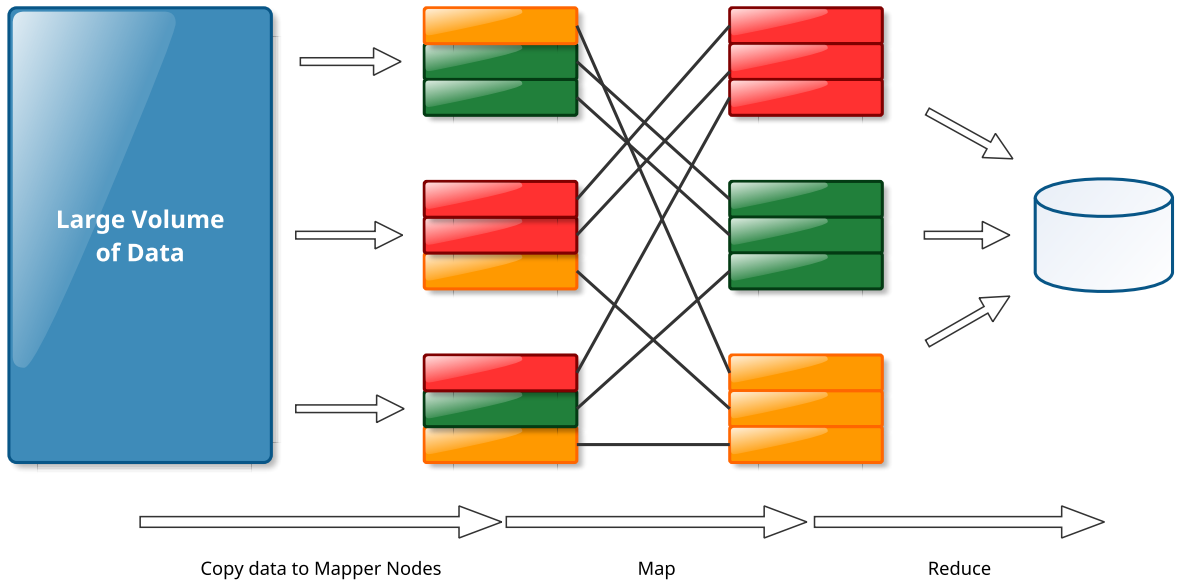


Figure 2.1: Hadoop execution system.

The architecture of this tool contains a master node and multiple slaves. The master node performs

two services: the Jobtracker which manages all tasks of a job, and the Namenode that manages the file system and its access. Each slave also has two services running: the TaskTracker running tasks on your node according to Jobtracker's instructions, and the DataNode that provides data blocks that are stored locally to HDFS clients.

Shang et al. demonstrate how this tool works, giving an example that counts the size frequency of the words of a document [16]. In the first phase, each process receives a word document and generates an intermediate pair key/value with the format word size/word (e.g., 5/hello). Then, all pairs with the same key value are sent to one reducer of the second phase, which in turn makes it counting pairs with the same key value, generating as result with the format word size/number of words (e.g. whether the document has three words of size 5, one result is 5/3).

Distributed Data-Parallel Programs (Dryad) The MapReduce model with its mappers and reducers is simple, so similar but more versatile models have been proposed. Dryad is a tool from Microsoft that supports parallel executions in distributed systems [21]. An application developed to Dryad is modeled on the form of a graph. The combination of the vertices using a number of different communication channels originates a data flow graph. Each vertice of the graph defines a set of operations to be executed sequentially in the data set. These vertices are then spread over a number of computers in order to be executed in parallel. Dryad is able to modify the graph created in order to achieve greater efficiency of available resources. By definition each communication channel is implemented using a temporary file, however it can also be set to use Transmission Control Protocol (TCP) pipes or First In-First Out (FIFO) shared memory. Despite the Application Programming Interface (API) being written in C++, vertices can be implemented in other languages, for example C#. Two of the major differences compared to MapReduce is the support of more than two phases in the graph and having multiple options for communication in addition to temporary files. Pig Latin and Pregel also do more generalized graph processing [22, 23].

Splunk There are also tools that are specific for processing logs. Splunk is one of the most adopted [24]. Splunk[1] is a tool that handles logs as searchable text indexes and generates multiple views of a log. It has powerful visualization, indexing, search, and reporting mechanisms that can be used for security and other kinds of analysis. Splunk has the ability to combine data in terms of security in real time, add context to events, generate reports and generate threat detection alerts. This tool uses the MapReduce model previously mentioned in order to be able to process large amounts of data and display the results efficiently. In order to facilitate the searches it has its own language so that a higher level of abstraction is provided to the user. There is also the possibility of generating reports of all data against a particular search, and there are some predefined templates, such as reports that search by failures, i.e., search for words such as "error" and "failed" in the data.

Splunk is composed by three main components: forwarder, which forwards the data to the remote crawlers; indexer, which stores the data and answers the searches; search head, which is the front-end

[1] http://www.splunk.com

usually accessed through a web interface. It is also possible to integrate this tool with others through an API provided for this purpose.

ElasticSearch ElasticSearch is an open source software based on the Apache Lucene text search library, written in Java, capable of indexing logs and performing searches [25]. It aims to be distributed and scalable. Data is stored in indexes (equivalent to a Structured Query Language (SQL) database), where each index can contain multiple types of documents (equivalent to a column in SQL). A document has zero or more fields and is associated with a certain type (like a table in SQL). The search is done almost in real time, which means that although these documents are indexed as they are created, these will only appear in search results after the index is updated (by definition in 1 second intervals). This tool has its own language to perform a search, and can also be used for writing filters. The main difference between searches and filters is that the first return results depending on the values relevance while filters do not.

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 the presented approach.

2.2 Data Mining

Data mining can be defined as extracting useful information from large data sets [3]. This section introduces important data mining concepts. It presents four techniques: clustering, classification, sequence analysis, and association rules.

Clustering Clustering is the process of organizing objects into groups where members of each one have similar features. In this work we are interested in organizing hosts in several groups.

In [26], Cohen et al. present a method to automatically extract signatures of a specific system. A signature of a system is defined as a set of features that describe the system state at a given time. In order to identify sets of signatures which characterize different behaviors of the system, clustering techniques are used. It is performed a comparison between the actual state of the system and the signature database, in order to detect abnormal behaviors previously identified.

[11] presents a log analysis technique applied in a large organization in order to identify whether the system is in a normal or failure state, and categorize this state according to the previous states. It used hierarchical clustering, which consists to begin with each record as a cluster and then identify the closest cluster and merge them. This process continues until all records of logs belong to a single cluster. This operation results in a tree where the internal nodes represent nested clusters of various sizes and the leaf nodes are the original data, i.e., all leaf nodes are at the same level and the number of nodes in each level represents the number of clusters.

Classification A process that uses a classification of various association rules to identify each type of intrusion is presented in [27]. To apply this process, the first step is to derive association rules. After

this step, it is used a classifier called RIPPER [28], which ranks in classes the status of the devices based on the association rule set found. It aims to identify intrusions. As an example, a telnet connection that when the number of authentication failures exceeds 5 is considered a dictionary attack.

Sequence Analysis The purpose of sequence analysis is to find relevant event patterns in the data taking into account the order of the entries (e.g., every time an user logs on to a host, a connection to the mail server is made).

Sequence analysis consists of two main phases: connection between the logs and simplification of the sequences [16]. Having already performed preprocessing of the log, where static values (e.g., the task type) and dynamic values (e.g., timestamps and identifiers (IDs) of tasks) have been identified, the first step, connection between logs consists in using the values to create a dynamic sequence of events (e.g., to create a sequence having timestamps as a function of the task IDs). Finally, in simplification of the sequences, repetitions and permutations of events are removed.

In [29] it is presented a classifier to predict the root cause of a problem in a system. The classifier is trained with data from a set of problems already solved. That mechanism uses a Support Vector Machines (SVM) classifier developed by Vapnik [30]. The representation of the features was based on n-grams. A n-gram is a sequence of N successive symbols in a given string of symbols. The advantage of this representation is that it retains information about order.

Association Rules The purpose of association rules is to infer the correlation of various features in a database [3]; on the contrary of sequence analysis the order is not important. In [31] is presented the following example: consider a supermarket where all customer activities are recorded; after analyzing the data we can know that 90% of the time that a customer buys bread and butter is also buys milk.

In [27] is explained how to use association rules in order to detect intrusions. Given a set of records, where each one is a set of items, we define $support(X)$ as the percentage of records which contain a set of items X. An association rule is defined by the expression $X \rightarrow Y, [c, s]$. Where X and Y are set of items, $X \cap Y = \emptyset, s = support(X \cup Y)$ is the support of the rule and $c = \frac{support(X \cup Y)}{support(X)}$ is the confidence. Not all features are essential, thus there is the concept of the most relevant association rules. The essential features are used as the axis (assigned a fixed value) in order to restrict the items.

In [10], Fu et al. use acyclic graphs in order to obtain association rules to represent correlations between events described by logs. It is used the Apriori-simiLIS algorithm, that is an improved version of Apriori-LIS, for learning association rules in logs, i.e., mine event rules. In this approach is used a time window that varies depending on the event that is to be analyzed, i.e., each event are used for events that are before and after a certain time with respect to its timestamp. After this event is analyzed, this window is advanced according to the new event to be analyzed. Apriori is a classic algorithm for mining frequent items. Before executing the algorithm, data within the time window is filtered so that only events that occur in the same nodes in applications of the same type are analyzed.

These works are interesting but they are not about security and use very different approaches than the one here proposed.

2.3 Log Analysis for Security

As in this work we are interested in analyzing logs for security, this section is more focused in understanding how to use the data stored by the various devices of an organization, also providing information considered important for understanding the topic.

2.3.1 Detection of Attacks and Intrusions

Log-based intrusion detection can be classified in three categories [1, 3, 7]:

Detection based on anomalies. Detection systems based on anomalies are mainly intended to observe activities that deviate from what is considered normal use of the system, marking them as a possible anomaly. In such systems there is the difficulty of defining what is considered normal, leading for flagging inexistent intrusions (false positives) or missing existing intrusions (false negatives).

Detection based on misuse. The system detects behavior that is known to be malicious. Such methods can easily detect attacks that match the signatures defined in the system, but are insensitive to those who have no associated signature. For this reason the signature database has to be constantly updated.

Detection based on policy-violation. In this case the system aims to detect activities of users that go against the rules of the organization. The rules of the organization must be easily modifiable and updated regularly.

Supervised machine learning and data mining techniques have been much adopted for anomaly-based intrusion detection [32–34]. 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 this approach, this form of detection involves having a data set that does not contain misbehavior.

It is important to mention the existence of various attack data planes, as an attack may use more than one in order to achieve a certain goal, hindering its detection. In [7] the following data planes were identified, with the possibility of the existence of others:

Physical. Related with all physical systems of an organization. In this layer all data of these devices are recorded (e.g., data from the entry of the buildings, etc).

User. Related with all social information of the systems users (e.g., update of the users' contacts, etc.).

Network. This data plane is concerned with all information related to the network infrastructure (e.g. firewalls' data, etc).

Application. This data plane is related with all applications of the organization. These application contain servers and systems used by users records (e.g., data authentication, etc.).

In [7], Giura and Wang present an approach to detect advanced persistent threats in logs. The work is quite ambitious as it aims to do near real-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.

2.3.2 Features

As noted before, feature extraction is an important process to do data mining. In [6] is presented a system for automatic mining logs produced by various security software packages in order to identify malicious users. The authors identified 15 features to characterize communications between each host organization and abroad, presented in Table 2.1.

Feature Type	#	Description
Destination-Based	1	New destinations
	2	New destinations without whitelisted referer
	3	Unpopular IP destinations
	4	Fraction of unpopular raw IP destinations
Host-Based	5	New user-agent strings
Policy-Based	6	Blocked domains
	7	Blocked connections
	8	Challenged domains
	9	Challenged connections
	10	Consented domains
	11	Consented connections
Traffic-based	12	Connection spikes
	13	Domain spikes
	14	Connections bursts
	15	Domain bursts

Table 2.1: Features identified by Beehive system.

The identified features are described in the following subsections.

A – Destination-based features

New destinations The first feature presented in the table is the number of new foreign destinations contacted per host per day. Over time, a history of connections is built, and updated at the beginning of each day with the connections of the previous day. A connection is considered new, in a particular day, whether it was never contacted by any host within a specified time period.

Due to the huge size of the logs this approach is not scalable. As such different data reduction techniques are referred. First destinations are filtered by popularity, i.e., each host will have a list of allowed destinations that contain IP addresses of services already well known (Google, Facebook, Twitter). To these allowed destinations are also added addresses which exceed a certain number of connections for one week.

Many services (e.g., clouds) use several strings in their subdomains, thus addresses are filtered by the Second Level Domain (SLD) name, i.e., by the second string of the domain name that is located

immediately to the left of the dot and domain name extension (e.g., in example.com, example is the second level domain).

New destinations without whitelisted referer This feature is an extension of the first, with the difference that are counted only the new destinations that do not have a whitelisted HyperText Transfer Protocol (HTTP) referer, i.e., the Uniform Resource Locator (URL) that originated the request of the new page, in the permissions list.

Unpopular IP destinations The third characteristic shown in the table is the number of IP addresses that are considered unpopular. Although access to non popular addresses is normal, the excess may indicate suspicious as such in the fourth characteristic is the fraction of the IP addresses not popular that day.

B – Host-based features

These features allow us to detect when the hosts install new applications, where these can be unauthorized. Due to the difficulty of obtaining the software settings and only having access to the logs, this data is obtained from the user-agent strings included in HTTP requests headers. The user-agent strings contain the application name that is making the request, the version and the environment information.

A history of user-agent strings is built for a period of one month. After that a user-agent strings is considered new if it is sufficiently distinct from all others in history. In order to know what is considered sufficiently distinct is used the Levenshtein distance technique [35], which calculates the distance based on the number of insertions, deletions and substitutions of characters between the strings.

C – Policy-based features

Within an organization a connection can be blocked whether it has a bad reputation or is blocked for use by employees. Thus the number of domains and blocked connections can indicate bad behavior by a host. When connecting to an unknown destination is required, the user has to accept the organization's policies before proceeding. It is counted for each host the number of domains and blocked, unknown and allowed connections.

D – Traffic-based features

A significant increase in the volume of traffic can be caused by malware, so it is important to monitor network activity.

A domain spike is considered to happen when the number of domain accesses exceeds a threshold in a time period T_f (e.g., $T_f = 1\ minute$). The same applies to the number of connections. It is considered a burst if within a time period all minutes contain spikes.

While we can manually identify the features of greatest interest in terms of security, it is important to be able to select them and set them in an automated way by analyzing the logs.

An algorithm is proposed in [27] which uses patterns of frequent sequential events, called frequent episodes, as guidelines to construct additional features. First the raw data is summarized into a set of predefined features. In this case, they characterize connections with several predefined features (see Table 2.2). Then a set of frequent episodes is created using these records, and only the ones that represent intrusions patterns are considered in the process (e.g., Table 2.3). Each of these patterns is used for adding additional features, with the aim of obtaining a better learning algorithm model. Considering these patterns, the procedure to find additional features that characterize each connection are as follows:

1. Take a feature F_0 (e.g., destination host) as a reference and a time window T_w, i.e., each episode represents T_w seconds of the log;

2. Add the following three additional features examining only the connections of the frequent episode and sharing the same value, $F_0 = V_0$:

   (a) Add a count of connections with $F_0 = V_0$ in the past T_w seconds;

   (b) Let F_1 be a different feature (e.g., service), if the same F_1 value is in all records of the episode with $F_0 = V_0$, add a percentage of connections sharing the same F_1 value as the current connection; otherwise, add a percentage of different values of F_1;

   (c) Let V_2 be a value of a feature F_2 other than F_0 and F_1. Whether V_2 is in all records of the episode, add a percentage of connections that have the same value V_2; otherwise, whether F_2 is a numerical feature, add an average of the F_2 features.

The pattern described in Table 2.3 may result in the following additional features that characterize each connection record: (1) A count of connections to the same "Destination Host" in the past T_w seconds; (2) Among these connections, a percentage of those that have the same service; (3) A percentage of those that have the "S0" flag.

A problem of this model is the selection of the right reference features to generate useful intrusion patterns and the right size of the time window.

Timestamp	Duration	Service	Source host	Destination host
1.1	0	HTTP	spoofed_1	victim
1.1	0	HTTP	spoofed_2	victim
1.1	0	HTTP	spoofed_3	victim
1.1	0	HTTP	spoofed_4	victim
1.1	0	HTTP	spoofed_5	victim
1.1	0	HTTP	spoofed_6	victim
1.1	0	HTTP	spoofed_7	victim
...	...	...	...	...
10.1	2	FTP	A	B
12.3	1	SMTP	B	D

Table 2.2: Set of connection records that represents an attack syn flood.

Frequent Episode	Meaning
(service = HTTP, destination host = victim), (service = HTTP, destination host = victim) → (service = HTTP, destination host = victim) [0.93,0.03,2]	93% of the time, after two HTTP connections to the victim, within 2 seconds in relation to the first one, a third connection is done, and this pattern occurs in 3% of the records.

Table 2.3: Example of an obtained intrusion pattern by mining frequent episodes.

2.4 Log Analysis for other Purposes

The information in the logs of the various systems of an organization is also valuable for purposes other than security. This section presents several purposes for which the logs can be used:

Create User Profiles. Increasingly logs are used for creating user profiles in order to facilitate systems management. As an example, logs of a web-server featuring the users of a particular site. With the use of different data mining techniques noted before it is possible to get information about various patterns of users, which facilitates marketing (for example to know what consumers buy together with another product). In [36], the authors present a method that creates user profiles analyzing their behavior.

Performance. Another use for the logs of an organization is to check how resources are used in systems [1]. Over time several information is recorded about the load of different systems, such as CPU and memory levels. This data can be used to discover patterns in the various activities of the organization in order to detect anomalies.

Debug. Another usage for log analysis is to ease debugging of applications that process large amounts of data and use parallel execution techniques, e.g., tools that use Hadoop. Imagine that a developer has to analyze information from several users in a particular social network. Having already developed the application, tested at home with a small volume of data and tested in a large-scale cloud with large volume of data we now want to make sure that the application is waiting behavior compared to the two tests. The solution proposed in [16] consists of three steps, the first is the development of application that gives use of large volumes of data and test it in a cloud of small size with a small amount of data, the second step involves implementing the application in a large cloud of large volume of data, and finally deriving the execution sequences of the steps one and two in order to compare the patterns.

3

Detection Approach

Contents

This section presents the approach starting with an overview of how it works (Section 3.2), then detailing its several aspects in the remaining sections. For a better understanding of the approach, Section 3.1 provide some background on the logs supplied by Vodafone PT. Nevertheless, the approach is not specific to them.

3.1 Network Infrastructure Logs

This section provides some background on logs and specific information on the logs used to apply this approach in the experimental evaluation of Section 4, which were supplied by Vodafone PT. The fact that this section 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.

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 [37]). The actual fields found depend on the type of device and the way logging is configured. Common fields are IP and Media Access Control (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 "*Identifier (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.

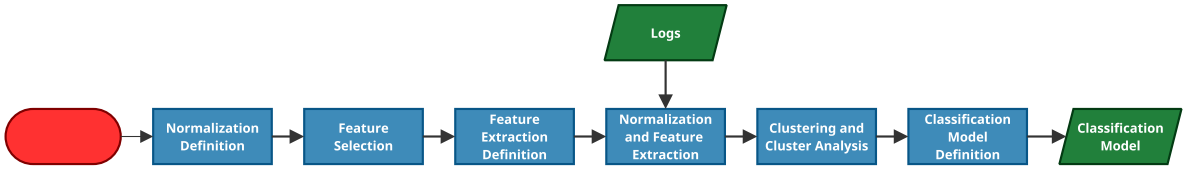
Firewalls logs have much data about communication, e.g., send/receiver IP/ports, action (accepted/dropped), connection status, connection type, number of bytes, reason for alert, rule or security policy applied to the connection, textual comment, etc.

```
11,10/27/14,23:16:42,Renew,X.X.X.X,MYDOMAIN,XX:XX:XX:XX:XX:XX,
10,10/27/14,23:16:42,Assign,X.X.X.X,MYDOMAIN,XX:XX:XX:XX:XX:XX,
30,10/27/14,23:16:42,DNS Update Request,X.X.X.X,MYDOMAIN,,
11,10/27/14,23:16:41,Renew,X.X.X.X,MYDOMAIN,XX:XX:XX:XX:XX:XX,
30,10/27/14,23:16:41,DNS Update Request,X.X.X.X,MYDOMAIN,,
11,10/27/14,23:16:41,Renew,X.X.X.X,MYDOMAIN,MYMAC,
30,10/27/14,23:16:41,DNS Update Request,X.X.X.X,MYDOMAIN,,
32,10/27/14,23:16:35,DNS Update Successful,X.X.X.X,MYDOMAIN,,
```

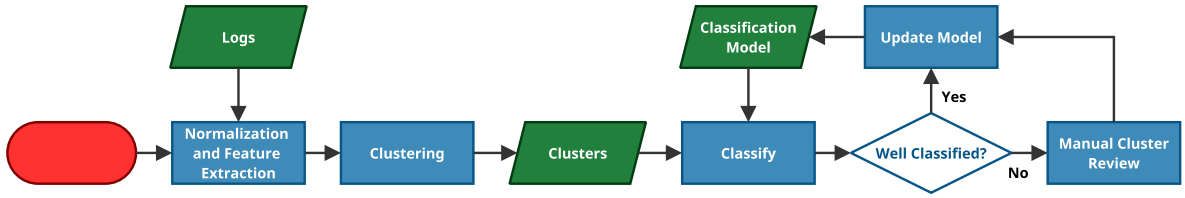
Figure 3.1: Example of a DHCP Server Log (IP and MAC addresses were anonymized).

An example of a DHCP log is shown in Figure 3.1. As noted, correlating thousands of records to obtain a specific host IP address within a period of time is a hard task.

3.2 Overview of the approach



(a) Definition and configuration of the detection mechanism



(b) Execution of the detection mechanism in runtime

Figure 3.2: Fluxograms of the two main phases of the proposed approach

The approach can be divided in two main phases. The first consists in executing a set of steps for *defining and configuring 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 this case) and a set of *logs* (those of Section 3.1 in our case). Figure 3.2(a) 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 3.2(b)). 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.

3.3 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 [6, 10]. Due to their size, logs should not be normalized before feature extraction, but during feature extraction (see Figure 3.2).

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 (See Table 3.1).

IP	MAC	Start Time	End Time
IP_1	MAC_1	02/24/15,08:48:18	ND
IP_2	MAC_2	02/24/15,08:56:34	ND
IP_3	MAC_3	02/24/15,09:06:02	ND
IP_4	MAC_4	ND	02/24/15,12:05:51
IP_5	MAC_5	02/24/15,09:09:20	ND
IP_6	MAC_6	ND	02/24/15,12:05:51
IP_7	MAC_7	02/24/15,09:10:18	ND

Table 3.1: Table with (IP, MAC) mappings in a specific time. The *ND* value means that the assignment does not have a start or/and end time for that host within the time period (T_f) considered.

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.

3.4 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 3.2: Extracted features.

Table 3.2 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 [6].

The features that use external data are:

- Number of IP addresses in the top of malicious subnets (28) – uses data from the Internet Storm Center (ISC) list of the top malicious subnets[1];
- Number of IP addresses with bad reputation (29) – uses IPs from the AlienVault list of IP addresses with bad reputation[2];
- Number of external IP addresses in the malicious Autonomous System (AS) list (33) – uses data from the Sitevet list of worse ASs[3];
- Number of external IP addresses in the spam AS list (34) – based on the CleanTalk list of ASs with more spam activity[4].

Other sources of similar data may be used instead of these or to define similar features.

3.5 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 [19] and its open implementation in Hadoop [20] 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 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 preliminar 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:

[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

- 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 a an AS. We implemented a cache to avoid repeating lookups for the same pair (IP address, AS number).

3.6 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 3.2. 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 [38]. 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 4.1).

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 [38]. 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 [39]. 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.

3.7 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).

3.8 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 [6]. 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 3.2(b)). 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 [38].

Our approach requires considerable human labor in the first phase (Figure 3.2(a)) 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.

4

Experimental Evaluation

Contents

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 4.1); (2) to show that the use of a classifier can automate the process of assigning classes to clusters (Section 4.2); (3) to assess the performance of the process (Section 4.3).

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 3.2) 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 [38]. Table 4.1 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 4.1: 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 recommendations[1]. 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.

4.1 Discovering intrusions with clustering

After extracting the features using Hadoop, they have to be normalized to fall in 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 4.2. The features relative to fractions were excluded because of the bad results we obtained when applying the

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

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.

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 4.2: Maximum values of each feature.

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.

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	
5	VL	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	
11	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	
14	VL	VL		VL	L		VL	VL	VL	VL	VL	VL		VL	VH	VL	L	VL	VL	VH	VL	VL	
15	VL	VL	VL	VL	VH		VL	VL	VL	VL	VL	VL	VH	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
17	VL	VL	VL	VL		VL	VL	VL	VL	VL	VL	VL		VL	VL	VL		L			VL	L	VL
18	VL		VL	VL	VH			L	H	VL	VL		VH	VL	VH			VL	M		VH		
19	VL	VL	VL	VL	VH	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	VH	VL	VL	
23	VL	VL	VL	VL	L		VL	VL	VL	VL	VL	VL		VL	VH	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	
28																							
29	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
31	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
33	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL	VL	VH	VL	VL	VL	VL	VL	VL	VL	VL	VL	VL
34	VL	VL	VL	VL	VL		VL	VL	VL	VL	VL	VL	VH	VL		VL		VL	VL			VL	VL

Table 4.3: Cluster description in terms of hosts it contains (total 4265) and primary features.

Table 4.3 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 3.7 (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 User Datagram Protocol (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).

The feature values for each cluster are represented in Figure 4.1.

4.2 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 4.4 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

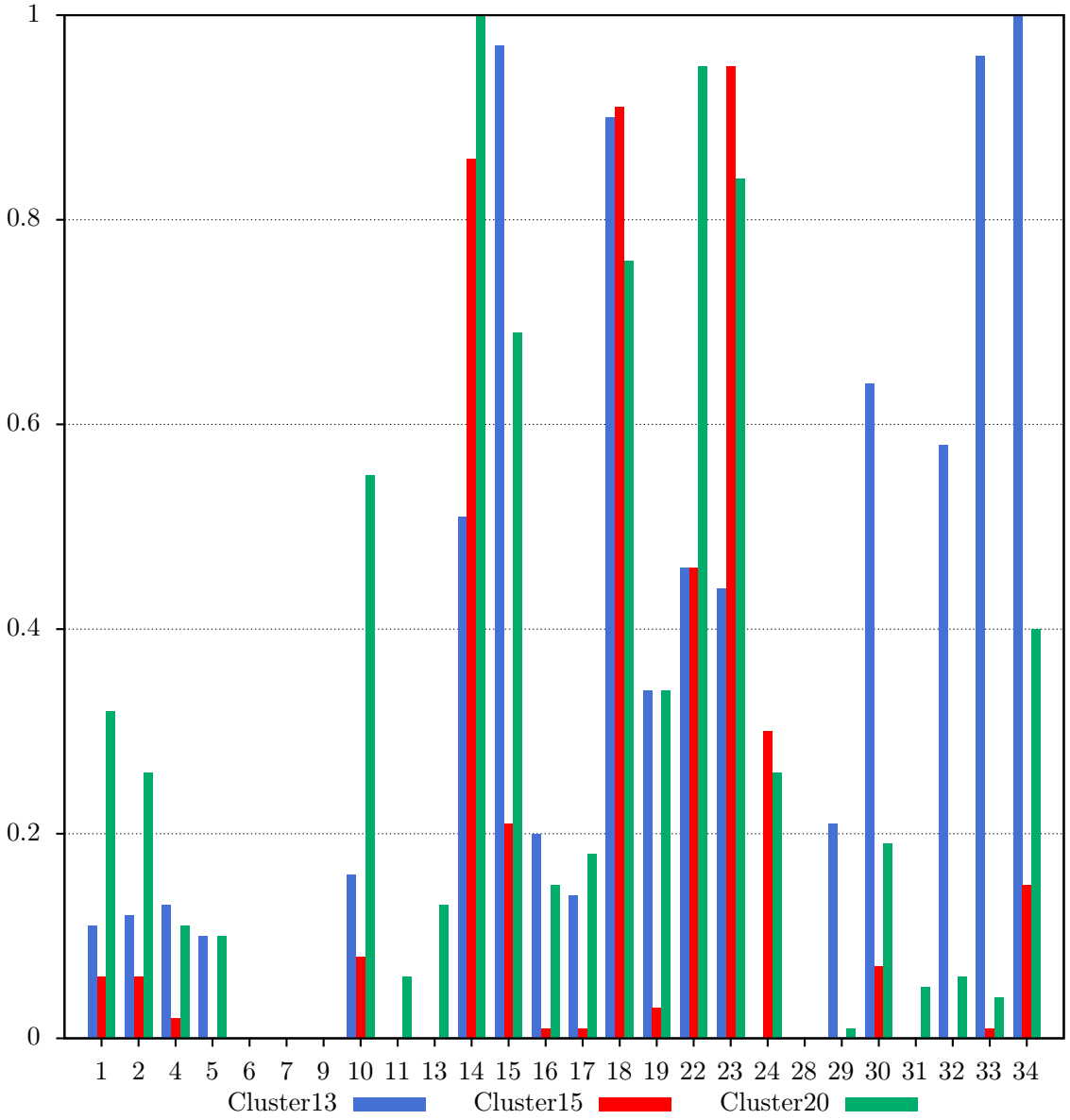


Figure 4.1: Feature values of the 3 identified clusters (normalized values).

over time.

Table 4.5 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.

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

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 4.5: Clusters classified in the 3 suspicious classes per day.

4.3 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 4.6. 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 4.7). 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 4.2. 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.

Log \ Day	1	2	3	4	5
Firewall (previous day input)	1h52m	1h34m	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 4.6: Feature extraction time per day and per log.

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 4.7: Log size per day per log source.

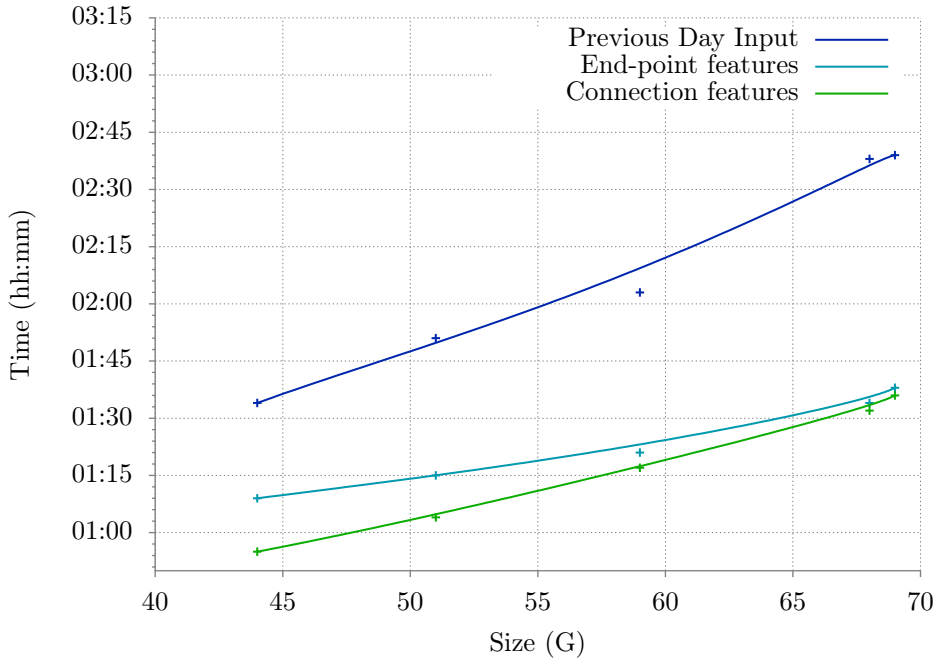


Figure 4.2: Time for feature extraction versus size of the logs.

5

Conclusions and Future Work

Contents

5.1 Conclusions

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 (approximately 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.

5.2 Future Work

Log analysis involves many phases, thus this work does not cover all of it in depth. We found the following possible additions and enhancements:

Whitelists. Whitelists have a huge impact on data reduction, however creating these lists manually requires a lot of human labor. Applying learning algorithms to logs may allow to create lists automatically (e.g., whitelist with allowed connections).

Addition of more features. The results led us to remove all features that represent *Fraction of*. However, these features have information about the number of occurrences of a particular characteristic of the feature in relation with the total. It would be interesting to add this information without compromising the remaining results. More features may be added using the whitelists previously mentioned.

Take automatic actions. Combine the approach results with other organization databases may be useful to take automatic actions (e.g, remove network access to a host), enabling a complete automatically process to detect and prevent network intrusions.

Bibliography

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

[2] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data*, vol. 2, no. 1, pp. 1–41, 2015.

[3] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

[4] A. Cárdenas, P. Manadhata, and S. Rajan, "Big data analytics for security," *IEEE Security and Privacy*, vol. 11, no. 6, 2013.

[5] ——, "Big data analytics for security intelligence," Cloud Security Alliance, 2013.

[6] 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.

[7] P. Giura and W. Wang, "Using large scale distributed computing to unveil advanced persistent threats," *Science Journal*, vol. 1, no. 3, 2012.

[8] J. Abela, T. Debeaupuis, and E. Guttman, "Universal format for logger messages," *Expired IETF draft draft-abela-ulm-05.txt*, 1997.

[9] Y. Shafranovich, "Common format and mime type for comma-separated values (csv) files," 2005.

[10] 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.

[11] C. Chen, N. Singh, and S. Yajnik, "Log analytics for dependable enterprise telephony," in *Proceedings of the 9th European Dependable Computing Conference*, 2012.

[12] G. Qu, S. Hariri, and M. Yousif, "A new dependency and correlation analysis for features," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 9, pp. 1199–1207, 2005.

[13] R. Sathya and A. Abraham, "Comparison of supervised and unsupervised learning algorithms for pattern classification," *International Journal of Advanced Research in Artificial Intelligence*, pp. 34–38, 2013.

[14] S. Das, "Filters, wrappers and a boosting-based hybrid for feature selection," in *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann, 2001, pp. 74–81.

[15] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[16] 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.

[17] EMC, "Rsa-pivotal security big data reference architecture," 2014. [Online]. Available: http://www.emc.com/collateral/white-paper/h12878-rsa-pivotal-security-big-data-reference-architecture-wp.pdf

[18] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[19] ——, "MapReduce: simplified data processing on large clusters," in *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2004.

[20] T. White, *Hadoop: The Definitive Guide*. O'Reilly, 2009.

[21] M. Isard, M. Budiu, 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5. ACM, 2005, pp. 105–118.

[27] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on.* IEEE, 1999, pp. 120–132.

[28] C. William *et al.*, "Fast effective rule induction," in *Twelfth International Conference on Machine Learning*, 1995, pp. 115–123.

[29] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma, "Automated known problem diagnosis with event traces," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4. ACM, 2006, pp. 375–388.

[30] V. Vapnik, "Principles of risk minimization for learning theory," in *NIPS*, J. E. Moody, S. J. Hanson, and R. Lippmann, Eds. Morgan Kaufmann, 1991, pp. 831–838.

[31] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD Record*, vol. 22, no. 2, 1993, pp. 207–216.

[32] D. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, 1987.

[33] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003.

[34] G. Nascimento and M. Correia, "Anomaly-based intrusion detection in software as a service," in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, 2011.

[35] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1091–1095, 2007.

[36] 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.

[37] Y. Shafranovich, "Common format and MIME type for comma-separated values (CSV) files," Request for Comments (RFC) 4180, Oct. 2005.

[38] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[39] 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.