Detection of Unknown Network Attackers Through Flow Analysis

Luís Côrte-Real Machado Sacramento

Thesis to obtain the Master of Science Degree in Telecommunications and Informatics Engineering

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

Examination Committee
Chairperson: Prof. Dr. Paulo Jorge Pires Ferreira
Supervisor: Prof. Dr. Miguel Nuno Dias Alves Pupo Correia
Member of the Committee: Prof. Dr. Henrique Manuel Dinis Santos

October 2016
Acknowledgments

First and foremost I would like to sincerely thank my advisor Professor Miguel Pupo Correia for introducing me to this topic and for his guidance during this whole year, for being always available to help as much as possible, and without whom this work would have not been possible; and my co-advisor Eng. João Bota, for the all the support given and for the conditions provided to the development of this work.

I would also like to thank my family who has always inconditionally supported and motivated me throughout this whole process, and to whom I owe my every achievement. Also to to my friends and loved ones who have always been by my side and kept me going through the hardest times.

And last but not least, I would like to thank Vodafone Portugal for providing me with means to finish this dissertation, and Arash Habibi Lashkari from Information Security Centry of Excelence for providing me with their UNB Intrusion Detection System dataset, which was fulcral for this dissertation.
Abstract

Network security threats are becoming more and more prominent and the connection links at the Internet Service Provider (ISP) are getting faster. Traditional Network Intrusion Detection System (NIDS) are either signature- or behavior-based, which means looking up for known intrusion signatures in the packets, or detecting deviations from normal behavior, respectively. These NIDS are not able to cope with the high increase of network security threats, as the majority of them are designed to detect specific intrusions that are previously known, or trained with intrusion-free traffic. Also, most of them rely on payload inspection, which can create a bottleneck in real-time detection. Furthermore, now that frequently every communication is done through encrypted messages, it is almost impossible to interpret the observed payload. In order to counter these limitations, we propose a NIDS that detects malicious hosts by using flows, which are defined as a set of packets with common characteristics that pass a specific observation point in a given time period. This work will therefore present a proposal for a flow-based NIDS capable of detecting malicious hosts without any a priori knowledge. The results from this proposed system were validated with a realistic dataset provided by the network operator Vodafone Portugal.

Keywords: Network Intrusion Detection Systems, Network Flows, Machine Learning
**Resumo**

As ameaças à segurança de redes informáticas estão a tornar-se cada vez mais proeminentes, e as capacidades das ligações dos ISP estão a crescer a um ritmo acelerado. Os Sistemas de Deteção de Intrusões de Redes (SDIR) tradicionais baseiam-se em abordagens *signature-* ou *behavior-based*, i.e. procuram determinados conjuntos de bits nos pacotes que sabem corresponder a uma intrusão, ou procuram desvios de padrões de comportamento normais por parte dos utilizadores, respectivamente. Estes SDIR não conseguem acompanhar a crescente tendência de ameaças à segurança, uma vez que a maioria destes sistemas estão desenhados de forma a detectar instruções que já são conhecidas, ou requerem tráfego de treino sem ataques. Para além disto, estes sistemas geralmente dependem da inspeção do conteúdo dos pacotes, o que pode representar um estrangulamento na deteção em tempo real, e actualmente a grande maioria das comunicações é feita através de mensagens cifradas, o que torna quase impossível interpretar qualquer conteúdo dos pacotes observados. De forma a ultrapassar estas limitações nos sistemas tradicionais, desenvolvemos um SDIR baseado na análise de *flows*, conceito este que pode ser definido como sendo um conjunto de pacotes com características comuns entre si que passa por um determinado ponto de monitorização da rede num dado período de tempo. Este trabalho apresenta então um SDIR baseado em *flows*, capaz de detectar utilizadores maliciosos sem qualquer conhecimento prévio dos mesmos. Os resultados são validados com dados de rede recolhidos pela operadora de telecomunicações Vodafone Portugal.

**Palavras-chave:** Sistema de Deteção de Intrusões em Redes, Fluxos de Rede, Aprendizagem Automática
Contents

Acknowledgments iii
Abstract v
Resumo vii
List of Figures xi
List of Tables xiii
Acronyms xv

1 Introduction 1
  1.1 Background ........................................... 1
  1.2 Proposed Solution ...................................... 2
  1.3 Outline .................................................. 3

2 Related Work 5
  2.1 Network Flows and Basic Flow Tools ..................... 5
  2.2 Flow-based Network Monitoring .......................... 8
  2.3 Intrusion Detection based on Network Flows ............. 11
    2.3.1 Port Scanning ........................................ 12
    2.3.2 Denial of Service ..................................... 13
    2.3.3 Worms ................................................ 18
    2.3.4 Botnets ................................................. 21
  2.4 Intrusion Detection Based on Machine Learning ........ 23

3 Network Intrusion Detection Approach 28
  3.1 Approach Overview ....................................... 28
  3.2 Data Gathering and Preprocessing ........................ 30
3.2.1 Feature Extraction ................................................. 31
3.2.2 Data Normalization ................................................. 33
3.3 Clustering .......................................................... 36
  3.3.1 K-Means Clustering .............................................. 36
  3.3.2 Mini Batch K-Means .............................................. 37
3.4 Supervised Cluster Classification ............................... 38
  3.4.1 Support Vector Machine ........................................ 38

4 Implementation ...................................................... 40
  4.1 Map Reduce ....................................................... 40
  4.2 Unsupervised Learning .......................................... 43
    4.2.1 Choosing the number of clusters .......................... 44
    4.2.2 Generating the clusters ................................... 46
  4.3 Supervised Learning ............................................. 47

5 Evaluation .......................................................... 49
  5.1 Evaluation with ISCX dataset ................................. 49
    5.1.1 Cluster Analysis ............................................ 49
    5.1.2 Supervised Cluster Classification ......................... 54
    5.1.3 Results Validation ......................................... 55
  5.2 Vodafone Data Analysis ......................................... 56

6 Conclusions ......................................................... 62

Bibliography .......................................................... 66
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A typical NetFlow architecture</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>TCP Three-way handshake</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Graph inflation in the left side and component inflation on the right side</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>An overview of the intrusion detection approach</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Flow collection scenario considered in the experimental evaluation of the NIDS approach</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Map Reduce Algorithm Overview</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Terminal user interface for the unsupervised learning module</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Interactive visualization of a cluster</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Plot of the Elbow Method</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>Example of parallel coordinates visualization</td>
<td>47</td>
</tr>
<tr>
<td>4.5</td>
<td>Terminal user interface for the supervised learning module</td>
<td>48</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Summary of fields in flow headers that can trigger attacks ................... 16
3.1 Features used .................................................................................. 33
3.2 Raw Features .................................................................................. 35
3.3 Normalized Features ....................................................................... 35
5.1 Mapping of feature names to numbers for source and destination aggregation keys, respectively .............................................................. 50
5.2 Saturday cluster content .................................................................. 51
5.3 Sunday clustering content ................................................................. 51
5.4 Monday clustering content ............................................................... 52
5.5 Tuesday clustering content ............................................................... 53
5.6 Thursday clustering content ............................................................. 54
5.7 Classification Scores ....................................................................... 55
5.8 Results validation ............................................................................ 56
5.9 Source key clustering content of a morning of October, 2016 using Mini Batch K-Means ................................................................. 59
5.10 Source key clustering content of a morning of October, 2016 using K-Means ................................................................. 59
5.11 Destination key clustering content of a morning of October, 2016 using Mini Batch K-Means ................................................................. 60
5.12 Destination key clustering content of a morning of October, 2016 using K-Means ................................................................. 60
5.13 Vodafone data analysis results ......................................................... 61
List of Acronyms

BSS  Between Sum of Squares

DNS  Domain Name System

DoS  Denial of Service

DDoS  Distributed Denial of Service

DPI  Deep Packet Inspection

ED  Euclidean Distance

HDFS  Hadoop Distributed File System

HTTP  Hyper Text Transfer Protocol

ICMP  Internet Control Message Protocol

IDS  Intrusion Detection Systems

IEEE  Institute of Electrical and Electronics Engineers

IP  Internet Protocol

IRC  Internet Relay Chat

ISP  Internet Service Provider

ISCX  Information Security Centry of Excelence

k-NN  k-Nearest Neighbors

L2R  Local to Remote

L2L  Local to Local

MPLS  Multi Protocol Label Switching

NIDS  Network Intrusion Detection System

PCA  Principal Component Analysis

PoVE  Percentage of Variance Explained
RBF  Radial Basis Function

SCTP  Stream Control Transmission Protocol

SDIR  Sistemas de Deteção de Intrusões de Redes

SMTP  Simple Mail Transfer Protocol

SSH  Secure Shell

SVM  Support Vector Machines

TCP  Transfer Control Protocol

TSS  Total Sum of Squares

UDP  User Datagram Protocol

WSS  Within Sum of Squares
Chapter 1

Introduction

This section will start by providing some background on how tradicional NIDS operate and which drawbacks they present, followed by a proposal to counter such issues. Before moving onto the next chapter, there will also be a roadmap to the remainder of the document.

1.1 Background

Preserving the security of a network is becoming increasingly important nowadays. The number of security threats is growing day by day, and the networks’ security systems must be able to keep up with this. Moreover, Internet Service Providers are increasing the capacity of their backbone links, now operating in the order of 1 to 10 Gbps. Traditional NIDS usually operate by using Deep Packet Inspection (DPI) methods, meaning that they look up on the payload of the packets passing through specific points of the network (e.g. an edge or border router), looking for a certain signature or a certain behavioral pattern. This would be feasible for link connections that were rather slow, but now it is very difficult to analyze the payload of every packet passing through the routers and be able to process them in real-time, without creating a bottleneck. Also, nowadays most of the traffic payload is encrypted, which makes this kind of detection even harder.

An alternative to this approach is the use of flows. This concept was first proposed by Cisco, and is defined as being a sequence of packets with the same features passing through an observation point, in a given period of time. This allows to observe the communication between hosts, rather than the content of the exchanged packets, opening a whole new world for NIDS. Traditionally, a flow is described by a 5-tuple, composed by Source IP address, Destination IP address, Source Port, Destination Port and Transport Protocol.

There are basically two main approaches when it comes to classifying a NIDS: 1. Signa-
ture-based detection; 2. Behavior-based detection. In the former, the system looks for a certain sequence of bits in the observed traffic in order to match it with a known intrusion; using DPI in such a system, would mean to look for sequences of bytes in the payload in order to find a signature of a known intrusion, whilst when using a flow-based approach it would be, for example, to look for flows with an active SYN flag. In the case of behavior-based detection, the system first builds what would be a normal behavioral profile, and then looks for traffic that deviates from such profile (normally certain thresholds are imposed, in order to quantify what separates normal behavior from that which is suspicious).

However, both these two approaches require a priori knowledge: signature-based Intrusion Detection Systems (IDS) require knowledge about the attack itself, i.e. a signature that defines it; behavior-based IDS do not require information regarding the attack, but are trained with intrusion-free datasets. Even though these may present good results, they do not scale well with the increasing number of novel threats. In order for these systems to be effective, it would be necessary to be constantly updating the system with information regarding these new kinds of attacks or intrusions, or in case of the anomaly-based systems, to maintain a good dataset that distinguishes clean data from some kind of novel attack.

1.2 Proposed Solution

The main goal of this dissertation is to present a solution that improves these limitations. Such solution features a combination of the two major machine learning techniques:

1. Unsupervised Learning

2. Supervised learning

Both these techniques aim to provide knowledge to the machine, allowing it to discover some hidden patterns in the data and correctly classify a set of input data, respectively. The difference between these two is that the former does not need ground truth, i.e. it does not require any labeling of the data: the machine is able to provide results based on the raw data alone; the latter, on the other hand, does require labeling of the data, as the algorithm will take as input a set of training data that will teach it to make the right labeling to the remainder of the data it will receive from there on.

With this dissertation, we aim to present a flow-based NIDS that is able to overcome the drawbacks mentioned in Section 1.1. This means creating an hybrid NIDS operating on a flow-level - rather than packet-level, and that is able to detect intrusions without a priori knowledge. This is done by combining the two techniques mentioned above. By achieving this, we have a
system that provides increased autonomy and is, in a way, self-learning, which may reduce a lot of the network managers’ constant need for intervention, although not being completely free from human intervention, as no NIDS is.

This approach will allow the detection of the source of the attacks, rather than to identify which kind of attack it is, i.e. it detects the devices (be them desktops, laptops, mobile phones or compromised servers) that are behaving maliciously. Also, this system focuses on detecting volumetric attacks, i.e. it allows to unveil relevant patterns in large scale and intensity network attacks, therefore not allowing to discover attacks such as Buffer Overflows, SQL Injections, Phishing, Cross Site Scripting, and so on.

To achieve so, the first step consists in preprocessing the incoming flows, by mining a set of features from the raw dataset, therefore eliminating features that are not needed, such as its payload. Upon this filtering, the flows will be aggregated in order to give the program a wider and more granular view of what is happening, as it will be explained further ahead. Having the flows preprocessed and aggregated, the two machine learning techniques that were previously mentioned will come into play. For the unsupervised learning, a clustering algorithm will be used, which will group together flows that share similar patterns. This step allows to have a clearer and graphical view of what is going on with our dataset. Not only, this is a crucial step for the intrusion detection, as the solution is based on the assumption that there is a much bigger percentage of normal traffic vs. malicious traffic. This means that the bigger clusters will correspond to regular traffic, and the clusters that appear much smaller are the ones we have to worry about, thus reducing drastically the amount of time spent in analyzing the flows. Having the clusters been generated, there will be a need for an expert intervention, in order to label the clusters. This is necessary for the next step, which applies supervised learning, in which Support Vector Machines (SVM) will be used. The SVM will then receive the labeled flows, and will be able to indicate whether the input flows are perceived by the system as being malicious, or benign behavior, therefore highlighting the intrusions present in the dataset.

1.3 Outline

This document describes the research and work developed and it is organized as follows:

- **Chapter 2** aims to explore relevant studies on this field.
- **Chapter 3** provides the backbone structure, requirements and the solution’s architecture.
- **Chapter 4** describes the implementation of the solution as well as the technologies involved in its development.
• **Chapter 5** describes the evaluation tests performed and the corresponding results.

• **Chapter 6** summarizes the developed work, as well as some future work.
Chapter 2

Related Work

This section provides an overview of some major contributions in this area. Section 2.1 provides an insight on some of the existing tools to perform flow analysis. Section 2.2 proceeds to point out some network monitoring applications that were built in a flow-based fashion. Section 2.3 gives an overview of some of the most addressed network intrusions and respective works that show how to detect them using a flow-level analysis rather than payload inspection. At last, Section 2.4 gives a brief explanation of what machine learning is and how it can be used to achieve our goal.

2.1 Network Flows and Basic Flow Tools

As stated previously, network flows allow a different approach in analyzing, monitoring and securing a network. While deep packet inspection allows for signature-based approaches, making it easier to detect some kinds of attacks, its not scalable for high speed networks, e.g. 10 Gbit/s [SSS+10] - the packets’ payloads can not be analyzed in real time, for such speeds. Also, nowadays most packets exchanged have their payload encrypted, making it even more difficult to inspect, even if it was possible to process that many packets in real time.

In 1996, Cisco developed the first network protocol to handle network flows: NetFlow [Cla04]. This consists in a built-in software in their routers, and is used to collect and export flow records. As the years went by, new versions arrived and were more and more complete. The most recent version - NetFlow v9 - includes integration with protocols such as Multi Protocol Label Switching (MPLS) that were not supported in the previous versions.

Before we proceed to detail this protocol, first we must explain what a flow is. A flow is defined as being an unidirectional sequence of packets, passing through an observation point that satisfies a set of common features in a given period of time [Cla04]; in this case, an observation
point would be a Cisco NetFlow-enabled router. These features must be defined *a priori*, in order for the device to perform the matching of features (an example of such features will be presented further ahead).

Being this technology built-in in network devices, it allows to select, from all the traffic passing through that device, what we want really to analyze. For example, by deploying this in a border router, all of the traffic going in and out of that network will be filtered by NetFlow. Upon the reception of an IP packet, the network device looks at that packet’s fields in order to find any matching feature with those previously defined. In case the packet’s features do match, then an entry is created in a data structure called *flow cache*, for that flow. Note that a flow may correspond to several packets, and many different flows can be collected. This process is depicted in Figure 2.1.

As this cache cannot be kept indefinitely, certain policies were defined for its management. According to those policies, when one of them is satisfied, the entries belonging to that cache will be exported to another device. So, a packet will be exported when: (i) the end of a flow is detected; (ii) a flow is inactive (i.e. when, for a given timeout, there are no longer packets belonging to a flow); (iii) belonging to a long-lasting flow, the timeout is reached; (iv) the device is in need of resources, e.g. internal memory.

![Figure 2.1: A typical NetFlow architecture](image)

The device to which the flow records are exported is referred to as the *Collector*, and is physically located in another device. In order to achieve maximum efficiency latency-wise, these records are encapsulated in a UDP datagram. However, NetFlow also provides the possibility of exporting the data through SCTP (Stream Control Transport Protocol), if we want to operate in a congestion-aware environment.

However, this technology initially did not have in consideration any security issue - no confidentiality, integrity or authentication is guaranteed. This was designed as such due to
efficiency and scalability issues - the deployment of such technology in large networks would not be able to provide real time measures. Instead, it assumed that the exporters (and also collectors) are deployed within a private, restricted and controlled network, rather than a public network, in which anyone could sniff these records, or even forge them.

Apart from NetFlow, many other vendors have their own implementation for flow collection and exporting. Examples of such implementations are NetFlow-lite, sFlow, NetStream, etc.

Due to the heterogeneous nature of these technologies from each of the vendors existing in this market, the Internet Engineering Task Force (IETF) joined forces to create a standard in flow collection and exportation, thus allowing for the clients to easily deploy their flow-based applications. This protocol was given the name IPFIX (IP Flow Information eXport) [CTA13]. As previously stated, packets which share common properties are grouped in flows, and in the IPFIX terminology they are referred to as flow keys and these can be, for example, a tuple such as:

(IP_source, IP_destination, port_source, port_destination, typeOfService)

In NetFlow, the flow exportation was done by encapsulating the records into UDP or SCTP datagrams. IPFIX provides, in addition to these protocols, the option of exporting flow records through TCP. Also, it fills the security gap in Cisco’s implementation: IPFIX provides both confidentiality, integrity, and authentication. These three properties are guaranteed by using secure channels - either through TLS (Transport Layer Security), if the records are exported through TCP, or through DTLS (Datagram Transport Layer Security) if the records are to be exported through UDP or SCTP datagrams. Furthermore, X.509 Certificates are also used, in order to reinforce the latter property.

In order to simplify the implementation of these frameworks, there are some tools available. Such is the case of nfdump [NFD], compatible with versions v5 v7 and v9 of NetFlow. It consists of six built-in functions:

- nfcapd, used to read the collected NetFlow data and store it into files
- nfdump, used to read the files generated by the previous function, displays the data and its able to create statistics on it
- nfprofile, also reads the data from the generated file, and filters it according to a user defined profile, storing into a file
- nfreplay, which reads the data from the file generated by nfcapd and sends it to another host
- nfclean.pl, a script to cleanup expired data
• *ft2nfdump*, used to convert other flow tools data into *nfdump* format

Although this is a command line based tool, there is a graphical web-based front-end application, made to provide a better view over the gathered data.

Another tool, that is widely deployed, is SiLK - System for Internet-Level Knowledge [SiL], a flow analysis tool developed by the CERT Network Situational Awareness Team. As stated in the official documentation, its ideal application is for traffic analysis on the backbone of a large enterprise or mid-sized ISPs. It is compatible with both IPFIX and NetFlow (versions v5 and v9).

This tool is divided in two categories of applications: 1. a packing system 2. an analysis suite. Upon the arrival of a flow record, the packing system converts that data into another format (so that it can be more easily processed by this tool) and saves this newly converted data into a binary file. From the new data, the analysis suite, which consists of small functions to read the generated file, can perform many operations that can go from filtering to statistical analysis of the records:

• *rwfilter*, which filters the gathered data, according to the specified conditions

• *rwcut*, which is able to convert the binary flow data contained in the generated file into a format that is perceptible by the human analyst

• *rwstats*, that generates the aforementioned statistical data

• *rwcount*, that summarizes the whole network traffic

### 2.2 Flow-based Network Monitoring

Amongst many other applications, such as the monitoring of applications, hosts, security, account and billing, the analysis of network flows has been widely deployed for network monitoring. In this subsection we refer some of the existing work done in this area.

[SLKM11] presented a flow-based network monitoring system called FACT - Flow-based Approach for Connectivity Tracking. Their goal was to deliver a monitoring system focused on remote hosts and networks, and to check if they are reachable from inside their (network operators) network or a costumer’s network, and to trigger an alarm in case there are any kinds of connectivity problems. Such problems could be for example an unusually high number of outgoing connections from the inside of the network to remote host (which could indicate an ongoing *port scan*).
First off, their flow collection was retrieved from all network border routers, in order to inspect every single packet that leaves or enters the network. Also, the flow traffic was divided in 5 different classes, namely *Traversing*, representing the traffic that comes from outside the network, passes through it and leaves, *InOut*, the traffic that goes outside and returns, *OnlyOut*, the traffic that only leaves the network, *OnlyIn*, the traffic that comes from outside specifically to the inside of the network, and *Internal*, the traffic that circulates inside the network, without leaving it. However, as this work’s goal was to monitor the availability of the connection to remote hosts and networks from the inside of the network, the traffic classes *Traversing* and *OnlyIn* were ignored in this process, and a special focus was given to *OnlyOut*. Also, there is a basic assumption that where flow is considered complete, i.e. for each outgoing flow there must be an incoming reply to that same flow.

In order to identify the critical events, connection-wise, they aggregate *OnlyOut* flows types across external hosts, /24 networks and prefixes observed in public BGP tables, and then they take into account the number of affected internal hosts. After this data collecting there is a pre-processing of the data, where there is a removal of some flows (e.g. blacklisted hosts) as a filtering for the flow cache where the data will actually processed, and will stay for a period of 5 minutes. When in the flow cache, the flows are stored as a tuple containing the IP addresses, the application ports and the protocol numbers. When the 5 period minute expires, the flows are divided in two groups: 1. *ConnSuccess* if the bidirectional *OnlyOut* flows start or end within the timeout interval (5 minutes) and they were initiated by an internal host; 2. *ConnFailed*, which includes only unidirectional *OnlyOut* flows that either end or start within the timeout interval. This way, they were able to identify which remote hosts or network were totally unreachable from the inside of the network, and providing the network managers an easy way to visualize and interpret the ongoing occurrences in the network.

Another possible usage of flow-based models in network monitoring is to diagnose backbone links for network operators, or Internet Service Providers, for instance. A work in this line is [SSG09]. In order to do so, they use a simple flow-based model to check for active flows in the network and identify the quality of their connection, making it possible to know when a link needs an upgrade.

Their flow-based model relies on an existing Poisson shot-noise model (a type of electric noise that can be modeled with the Poisson processes, and whose application can be, for example, the modeling of network traffic [KM95]). With the use of the following three parameters, it is able to make an approximation of the throughput of the backbone link: \( \lambda \) - the arrival rate of flows, \( E[S_n] \) - the average size of a flow, and \( E[S_n^2/D_n] \) - the average value for the ratio of the
square of a flow size and its duration. Also, it allows to characterize the data rate existing in a backbone link, with the following input: 1. session arrivals for any period where the traffic intensity is nearly constant, and are accurately modeled by a homogeneous Poisson process of finite rate $\lambda$ (which lasts for approximately 30 minutes) 2. distribution of the flow sizes and their durations (which are independent) 3. the flow rate function $\lambda$. Based on these values, it is possible to determine the total rate of data in the link at a given time $t$, and consequently its average total traffic and variance. However, if such value is calculated in function of the average size of the flow, that function is only applicable to the ideal case of a backbone link with infinite capacity, and that’s never achievable in a real network. So, in order to acquire the real network state, Little’s Law is used, and such is given by $N = \lambda E[D_n]$, where $N$ is the mean of active network flows.

They analyzed the connection quality at the backbone link by investigating graphical dependencies between the link utilization and the number of active flows, i.e. a graphic in which the $x$ axis represents the number of active flows and the $y$ axis represents the link utilization. They managed to observe that there are three real network states: the first is called the operational region, and corresponds to a nearly ideal behavior, as the link utilization grows linearly with the number of active flows; the second is when the network starts getting moderately loaded and diverging from the ideal behavior; the first and last state is when the network is completely disabled, and the packet losses begin, as well as the decrease of its throughput. Also, they presented an equation which allows to obtain a good confidence interval, based on the flow performance, the standard deviation of the link utilization and a parameter $\alpha$ obtained from data processing.

These theoretical models were evaluated through experiments made from two different networks, with a data set consisting of data collected every 30 minutes throughout a week (for the first network) and every 5 minutes throughout 72 hours (for the second network). With this model, they were able to diagnose the backbone link, and check when the link utilization was beginning to leave the operation region, making possible to the network managers to know when to upgrade the link’s capacity, and to steadily dimension the connections.

A study on the effects of DDoS attacks on network flow monitoring applications is reported in [SSP12]. Their goal was to show how a flow monitoring reacts in the the presence of such an attack. The proposed scenario was a traditional flow-based setup, where all the traffic in the network is filtered by an exporter placed in an observation point (a Cisco NetFlow-enabled router). It then proceeds to export the collected packets with its flow records to a device in charge of generating flow records - the Collector - which, on its turn, redirects these records
to the monitoring application (already described in the previous section). They break down their approach in two parts: they first studied the impact of the attack in the flow exporter (the monitoring probe), and then they study the impacts on the flow monitoring application.

Due to the great number of flow records generated by DDoS attacks, a strategy to manage the flow record cache must be chosen. Upon the arrival of a new flow record, one of three actions can be taken: 1. prematurely export the stored records; 2. directly export the new record; 3. prevent flow from being created – being the first one elected. As a consequence, there will be an increase in the rate of exported flow records, both for normal and attack traffic. Not only the rate will increase, but also, as the records have a smaller lifetime, these flows will have fewer packets – this is due to the fact that this DDoS attack consists in flooding the network with SYN Request, and each one of these requests creates a new unique flow. One other result of such decision is that, since the rate of exportation rises, there is a decrease in the delay between flow expiration and flow exportation.

Then, they used a simple single queuing model in order to study to effects on the flow monitoring application. All of the following work and conclusions was based on three assumptions: 1. the exporter is not a bottleneck in the system; 2. the network connection between exporter and collector is well dimensioned; 3. incoming records are buffered by the application, if needed. This queue operates in a FIFO fashion, and each job represents a flow packet, which, in its turn, only carries the number of normal and attack flow records. They point out to the fact that if such a system is not sufficiently well dimensioned, such an attack could easily overload it: if the service rate $\mu$ is much smaller than the arrival rate $\lambda$, the systems is easily flooded. Furthermore, they showed that the effective rate of processed flows is a function of the rate arrival rate of attack records, and not only the normal ones. This means that during an attack, the normal flow traffic is also affected, which means that, for our case, the development of a NIDS, it might be interesting to also analyze the "normal" traffic in order to detect suspicious activity.

### 2.3 Intrusion Detection based on Network Flows

Nowadays, there are a vast variety of network attacks. From simple port or network scans to complex botnet infrastructures, there are numerous types of different attacks, both in type, in scale or severity of impact. However, as the variety of the attacks is indeed enormous, we cannot focus on the detection of all of them. Moreover, a flow-based intrusion detection approach is not able to detect all kinds of attacks, as it relies on the inspection of header information. Logic attacks such as Buffer Overflows or SQL injections can only be detected by inspecting the payload of the packets in the network, which represents a major limitation if one is to use an
exclusively flow-based NIDS.

With the help of some of the technologies and tools referred in Section 2.1, we present some approaches on the detection of these attacks, namely Port Scans, Denial of Service, Worms and Botnets.

### 2.3.1 Port Scanning

A port scan is defined as the act of consistently probing a target host (which may be either a single machine or an entire network - network scan), by sending a large amount of generally small packets.

Althought not being considered an attack, it is generally the first step of almost every network attack (such is the case in DoS/DDoS, worms and botnets), thus being its detection a crucial step for a NIDS. As the attacker sends a great number of packets, even though these may be very small, it will produce many flows, therefore making it possible to rely on flow-based approach to detect it, as it can be very easily addressed. [SP11] divides this attack in three distinct categories:

- *Horizontal scan*, in which a single host scans multiple ports in a single machine
- *Vertical scan*, in which a single host scans one single port in multiple machines
- *Block scan*, a combination of the two scans above - a single host scanning multiple ports in multiple machines

Whatever category the attack falls in, this may create an anomaly in the normal network traffic pattern, and many kinds of different flows can be observed.

Most of these attacks are investigated by observing a flow characteristic that registers the most significant difference when compared to normal traffic: the unusually high number of incoming and outgoing connections in a host. This is due to the fact that the attacker, or attackers, probe many different ports and machines, therefore generating an anomalous amount of new flows.

A type of attack that falls into this category is the brute-force SSH network attack [HHH+12]. This is a particularly interesting attack in this field of study. The attack consists in three phases (that will be explained better further ahead): it begins by consistently scanning a certain number of victims, until a running port is found; then, it attempts to login in those victims through a brute-force dictionary attack; once it gains access, the attacker can do whatever he wants with the victim, and also to others that might belong to the same network. Therefore, SSH attacks can be potential harmful not only to the host individually, but also to the network it is connected to.
However, the detection of these attacks, as they rely mainly on scanning, can be addressed by performing an analysis of the network traffic at a flow-level. [HHH+12] presented a flow-based Intrusion Detection System called *SSH Cure*, which allows for real-time detection of brute-force SSH network attacks.

Their solution was based on the observation made by [SP11]. They observed that the behavior of the attacks over time, in terms of flows, follows a pattern of evolution, and it can be identified in three distinct phases, as described below:

1. Scanning phase, in which the attacker performs a *port scan* for a certain IP address block, in order to find running SSH daemons in a host (SSH daemons use TCP port 22)

2. Brute-force phase, in which the attackers tries to login to a certain number of hosts, by means of a dictionary attack - various combinations of usernames and passwords

3. Die-off phase, in which after successfully login into the victim host, the traffic volume is drastically reduced, leaving only residual traffic

By tracking these three phases in a flow pattern, this attack can easily be identified. Moreover, [KKH+04] also proposed a solution to detect this intrusion by, once again, observing flow-based traffic patterns. This approach will be explained in more detail in the next section.

### 2.3.2 Denial of Service

A *Denial of Service (DoS)*, is the attempt of an attacker to make a certain server, firewall, or even a whole network unable to reply, by flooding it with several requests and thus draining all of its resources. In a DoS attack there is generally one single attacker. This has some limitation to the attacker, such as the fact that it may not scale when targeting a big number of victims, or even the fact if one single attacker is producing all the traffic, it is easier to detect, due to the large anomalous volume of traffic. Another kind of DoS attacks is the Distributed Denial of Service, in which now there as various sources that are producing this large volume of traffic, thus making it more difficult to detect, as the traffic is now more evenly distributed and hidden (this was already discussed in Section 2.2).

According to Peng *et al.* [PLR07] and Kim *et al.* [KKH+04], there can be various types of DoS/DDoS attacks:

- *SYN flood* - this attack aims to explore the vulnerability in the *three-way handshake*, the connection establishment procedure in the TCP protocol. As depicted in Figure 2.2, to establish a TCP session, the one initiating the connection (the client) will send a packet
with an active SYN flag, to which the entity on the other end (the server) replies with a SYN/ACK flagged packet. The session is successfully established when the client replies to the latter packet with one that has an ACK flag. Whenever a client initiates the three-way handshake, the server will store this request information in its memory, and will keep it until it receives an acknowledge. Usually, in these attacks, the attacker sends packets with an active SYN flag and an nonexistent IP source address, therefore making the server unable to respond with the SYN/ACK packet; this can also be done with a source that exists, but that machine has to be configured such that they ignore any correspondent packet originated from the server. By send multiple packets with active SYN flags, the server’s memory will be exhausted, and be unable to respond to any other requests.

- **ICMP flood** - this attack is labeled as a bandwidth attack, as its goal is to congest the communication channels in a network. When a broadcast message is sent, there will be as many packets being sent in the network as the number of connected hosts; by sending multiple broadcast messages, the bandwidth can easily be depleted.

- **Smurf** - this attack consists in sending an ICMP Echo Request packet whose destination source is the broadcast address of a network. Also, the source address is external to the compromised network and is propagated with the use of intermediates (who can also be victims in this attack).

- **Fraggle** - this attack is performed like the **Smurf** attacks, except now the UDP protocol is used, instead of ICMP.

- **Ping of Death** - this attack consists in sending multiple (e.g. thousands per second) spoofed pings with an abnormally large size.

- **Land** - this attack is performed by sending packets whose both source IP address and port are the same as both destination IP address and port, through TCP. This way, the victim continuously reply to themselves, resulting in a Local Area Network Denial (hence the name of the attack).

- **Ping Pong** - this attack consists in sending packets whose both destination and source ports are reflecting ports, which are used for services such as **echo**, in UDP.

Apart from these, nowadays there are more variations of the Distributed Denial of Service, namely the DDoS HTTP Flood and the DDoS DNS Amplification attacks. The former, such as the SYN Flood attack, makes use of a vulnerability in the protocol used. Located in the network’s Layer 7 (the Application Layer), the attacker send an HTTP request to the server.
- it may be a POST or GET message, aiming to make the server reply with messages having the maximum resources possible, in order to drain its resources, e.g. by sending requests of downloads of very large files. If this attack is well coordinated, by sending these multiple request from multiple different users, the server will exhaust its resources, not being able to respond to legitimate requests. The latter makes explore a vulnerability in the Domain Name System (DNS) protocol behavior. The attacker perpetrates his attacks by sending a spoofed Source IP address - being this address the IP address of the victim - to various DNS servers, and the query contains an Internet domain that has many different DNS records for the same name. This way, the DNS servers will all respond these multiple records to the victim, therefore flooding it with a large volume of traffic. However, for the best of our efforts, no literature was found where these attacks were detected using flow-based approaches.

An interesting work presented a system for detecting abnormal traffic in a network, using a flow-based approach, extracting the traditional 5-tuple from the flow extractor [KKH+04]. The authors define the term abnormal as traffic that induces a malicious purpose, as is the case of the traffic generated by DoS/DDoS, worms or even scanning. Moreover, this work is particularly interesting as they focus their effort on detecting DoS/DDoS attacks, as we will be describing below.

Their detection model consists in two parts, namely flow header and traffic pattern detection modules. Whenever one of these two modules detects an attack, an alarm is triggered, indicating the network manager that an attack might be occurring, very much like a signature-based NIDS would operate. The first module is in charge of inspecting the fields of the gathered flow headers. This allows the detection of some flooding attacks that possess specific values, e.g. having Broadcast as the IP destination address. Table 2.1 presents an extensive list of such values that will be validated and the respective attack that they might point out to. When validating the protocol fields, the detection is pretty straightforward, needing only to inspect the corresponding
fields and check is they match or not, leading to a possible trigger. However, when referring to the packet count and flow size, the term *large* is rather ambiguous, as they depend on the volume of traffic at the time, and the kind of traffic that is being produced. And so, to address this, this value is obtained by calculating a percentile threshold of the observed values.

During a scan, an attacker makes a great amount of connection attempts, therefore generating many flows in which the packet count is small (approximately 40 bytes). If this scanning falls in the category of *port scanning*, the attacker will sweep a great amount of ports in the target host, therefore generating traffic in which the destination IP address is constant. However, if it is a *network scanning*, the attacker will make many connections with many different destination IP addresses, searching for a service availability in one of them. With this said, we can conclude that both the total packet count and total bandwidth consumption can be either small or large, depending on the number of host connections and the number of ports, therefore making it impossible to use these two values to detect scanning attacks. Both Smurf and Fraggle attacks force traffic gathered by the victim, using a third party, which will create as many flows as the number of hosts in the third party used, consequently increasing both total packet count and total bandwidth consumption. And, because the number of repetitions of the transmission (being the destination address the Broadcast address) of spoofed packets determines the packet count for each flow, these parameters are also unavailable for detecting attacks.

The need for a second detection module derives from the fact that some attacks (and consequently, their patterns) can not be detected only by inspecting the flow headers fields and analyzing flows individually. In order to detect these, they need the traffic information capable of identifying patterns in it, and this can be achieved by aggregating related flows. From these aggregations, it is now possible to detect both flooding and scanning attacks (not only port scans,
but also network scans), by checking the information sent and received from a host. They use two hash tables, in which the traffic pattern data aggregated for the same Destination or Source IP address is recorded. The detector checks if a large number of flows appears, the flow size of an individual flow is small, and also if the number of packets per flows is also small. If these conditions are verified, and if the number of distinct destination ports is high and the number of source IP address traffic generated is small, then the traffic is assumed to be a host scan. Else, if there is a small number of destination ports and a small fraction of the ratio \( \frac{n(ACK)}{n(SYN)} \), it implies that we are in the presence of a TCP SYN flooding attack.

Another approach for detecting DoS attacks was done using 2D sketches. Not only this detects such attacks (their main focus relies on TCP SYN flooding detection), but also can cope with the detection of the various types of port scans [GLC06]. A sketch is an hash table used to quickly store information, mainly counting the occurrences of a given event. When this concept first came up, these hash tables had only one dimension, but the work presented by [GLC06] introduces the use of a two-dimensional table. With this data structure it is possible to characterized the observed traffic, in order to draw some conclusions, which in this case is the detection of port scans and DoS attacks.

As it was just mentioned, this system is able to detect both port scans (which is also extensible to detect most of large-scale worm propagations) and TCP SYN flooding attacks, using sketches as a base for statistical intrusion detection. To achieve so each of the network routers (may be either edge or backbone routers) are configured to record network traffic into sketches. From the multiple created sketches (one for each router), they summarize all of them into one aggregated sketch, therefore allowing to distinguish from many different attacks observed. Then, they apply time series algorithms on these in order to obtain a forecast sketch, which will be used for the change detection. By subtracting this forecast sketch to the current one (i.e. in the present time), it is obtained a forecast error, which, if sufficiently large, indicates that there is an anomaly. By reversing the sketches, they obtain the key characteristics of the recorded flows, allowing to mitigate the attacks. In case of the values recorded trespass a certain defined threshold, the system triggers an alarm, indicating an attack.

The first step in their algorithm was to create a reversed sketch with keys \( DIP, Dport \) and whose feature is \( \#SYN - \#SYN/ACK \), which allows to detect SYN flooding attacks, as the attacker would be targeting a certain service in a specific destination port on a small set of hosts. The feature value \( \#SYN - \#SYN/ACK \) means that on every incoming SYN packet, the sketch entry would be incremented by one, and by every incoming SYN/ACK packet the sketch entry would be decremented by one. This set of DIPs will be stored in a set called
FLOODING\_DIP\_SET, to be used in the following set. Then, another reversible sketch is to be applied, only this time with the keys SIP, DIP, which could detect any intruder that was trying to attack a particular IP address. These attacks could either be a non-spoofed SYN flooding attack, or a vertical scan. With this data, the algorithm would then proceed to check if, for every pair in this sketch, the DIP is present in the FLOODING\_DIP\_SET, and if so, it would store the SIP in a set called \#SYN − \#SYN/ACK. If not, this is the attackers IP address and this is considered a vertical scan. The third step was to use a reversible sketch with keys SIP, Dport and feature values \#SYN − \#SYN/ACK, once again. This allows to detect any large number of unfinished connections for a specific destination port. For each of the key pairs, it checks if the key SIP belongs to \#SYN − \#SYN/ACK, and if so, concludes it is a non spoofed SYN flooding, and if not, it is a horizontal scan. After these three steps, the system then applies the time series analysis from which will result the error function used to detect anomalies.

2.3.3 Worms

A worm consists on a harmful software that, unlike the well-know case of a virus, has the capability to autonomously explore software vulnerabilities, thus making it capable of replicating itself throughout a network.

This specific attack is usually divided in two distinct phases: (i) a scanning phase, in which the worms probes several machines in order to find a vulnerability and then proceed to spread the infection; (ii) the transfer phase, in which the worm proceeds to send the harmful code and infect the victim. As the first phase is a well know case (as described above, in the port scanning phase), the discovery of the scan can be crucial in identifying the attacker. Due to the fact that most of the network traffic relies on secure connections, and therefore the payload of traveling packets in most of the times is encrypted (e.g. TCP traffic), a major emphasis is given to the detection of the first phase, as is would be very difficult to detect malicious content of packets in a flow-based analysis.

From using protocol graphs [CR07], to extending port scan detection approaches, to assigning hosts to sets of classes, many can be the strategies to identify worms in a network, based on the analysis of network flows.

The authors of [CR07] developed a system capable both of detecting hit-list (a list of target servers that were previously identified) worms and identifying bots (next section), using protocol graphs. A protocol graph is a representation of traffic logs, in which its vertices (or nodes, as sometimes referred in literature) are representations of IP addresses and the edges are the connections existing between those two entities. The system’s detection approach is based on
graph size and largest connected component properties. The first is the total number of connected
vertices, and the latter is the number of vertices in the largest connected component of the graph,
i.e. the vertex with the most connections overall.

In that research, only four protocols were considered:

- **HTTP**, identified by observing a connection in which one of the peers uses port 80 (either
destination or source); this represents the majority of the traffic observed

- **SMTP**, identified by observing connections where one of the peers uses port 25; this is
the second most active protocol producing traffic in the network

- **FTP**, which can be identified by observing the usage of port 20 by one of the entities in a
connection

- **Oracle**, identified when one of the peers is using port 1521. As this protocol requires a
login and password to authenticate an user, it's expected that users connected to a smaller
number of servers

In order to construct the graph, flow records were extracted from Cisco NetFlow-enabled
routers, in a large intercontinental network. These records were collected throughout a period of
5 days. As stated by the authors, many worm detection systems on the detection of an unusual
high number of frequent connections between peers, and track them by inspecting connection
evidences such as half-open TCP connections. So, to avoid this issue, the attackers can use hit
lists, which consists in a list of previously identified servers. Using this, they need not to contact
random servers across the network and can focus on these known ones, making it harder for the
systems to detect this scanning phase.

The detection model consists in the hypothesis that an attacker that contacts different servers
through a hit list will affect the graph in two ways (as depicted in Figure 2.3): first, when an
attacker is communicating with servers that were not active during the observation period \( \pi \),
there will be a graph inflation, i.e. there will be an increase in the number of vertices in the graph.
The other way is when the attacker communicates with already active servers in the
observation period \( \pi \), and this will affect the component inflation, i.e. increasing the number of
connections in one vertice. The system will raise an alarm whenever one of two conditions are
verified:

1. The total size of the graph is bigger than its mean value in the observation period \( \pi \) along
with its standart deviation over time
2. The largest component size is bigger than its mean value in the observation period $\pi$ along with its standard deviation over time.

By observing the two previously mentioned possible inflation in the graph, they were able to detect when an attack is on course, given that the first hypothesis will raise a trigger for condition (1) - a graph inflation will result in a considerably relevant increase of the total graph size, and the second hypothesis will raise a trigger for condition (2) - the component inflation reflects a big increase of the largest component size value.

One other, simpler, approach to detecting network worms was addressed by Dubendorfer et al. [DWP05]. This came with the development of their framework for both worm attack detection in real-time and the network backbone monitoring, UPFrame. In this framework, they developed algorithms for host behavior, network activity and traffic characterization for attack detection.

The host behavior detection module is the one used to identify worms in the network. The hosts are assigned to a set of classes, and this is updated periodically (in the paper the defined interval was 1 minute). However, host will only belong to one of these classes if it presents an unusual behavior, only belonging to any of them when a certain threshold is trespassed (note that they are not restricted to belong to only one class, can belong to more than can). This leads to the concluding that the rapid change in the cardinality of a set indicates a significant change in the network behavior. The classes are the following:

- **Traffic**, for hosts with $\frac{\text{bytes}\_\text{send}}{\text{bytes}\_\text{received}} > 3$

- **Connector**, for hosts with $\#\text{outgoing\_connections} > 10$

- **Responder**, for hosts with $\#\text{bidirectional\_connections} > 1$

The hosts in the **Traffic** class are the one that send much more traffic than what they receive. This is the kind of behavior that is expected for code exploiting worms or worms that spread...
on email attachments, for example [DP05]. The hosts in Responders are those that maintain bidirectional connections, which the author define as flow in opposite directions between a pair of hosts, that are spaced from each other in less than 50 ms. Hosts that respond to TCP scans are usually in this class. And, at last, hosts in Connector are those that initiate many outgoing connection without maintaining a bidirectional connection, which is the typical behavior for a scanning. With the definition of these classes, the attack detection is based on the assumption that if a host tries to infect others during a worm outbreak, the behavior of many hosts will change, causing great variation in the cardinality of the sets. This approach, however, is applicable mainly to networks that rarely see its infrastructure changing, therefore not being scalable for dynamic networks, i.e. for networks that are constantly both growing and decreasing its dimension, as the graphs would keep on being updated, thus not being able to reflect its proper results.

### 2.3.4 Botnets

A botnet is a network of infected hosts (referred to as bots) that is controlled by an entity that is referred to as master or botmaster. The master controls its bots through a Command and Control - C&C - infrastructure, and uses them as a third party to launch attacks and intrusions in the network, such as Spam, Phishing, DDoS and Identity Theft. This C&C infrastructure could either be an IRC (Internet Relay Chat) channel, or an HTTP server. However, by identifying it, most of the problem is solved [SSS+10].

Botnets are considered to the date as one of the biggest security threats, and the most difficult to detect, as the botmaster is not easy to identify, and requires a long period of observation.

An important contribution to this field was made by Zeng et al. [ZHS10], who developed a protocol-independent framework to detect botnets, using both host-level and network-level information. They constructed their detection model based on the fact that bots within the same botnet are likely to receive the same input from their botmaster and to take similar action, while benign hosts are seldom to behave as such. The system architecture can be divided in three parts: 1. Host analyzer; 2. Network analyzer; 3. Correlation engine.

The first part can be broke down in two modules: 1. In-host monitor; 2. Suspicion-level generator. The first one monitors in run-time the host behavior. By studying contemporary botnets, the authors found that these act in three different sectors in the hosts: the registry, the file system, and the network stack. When a computer is infected (note that this study refers only to Windows operating systems), the botnet begins by generating an exe or dll file in the system directory. Then, it registers an autorun key in registry to run automatically when system boots. Also it injects its code into other processes in order to hide its presence, and disables any
anti-virus software and task manager, if needed, and may open one or more ports for further communication, establishing connections with botmaster or peers in order to launch its attacks. The combination and aggregation of these mentioned behaviors is a strong indicator that the host might be infected. This run-time behavior is transformed into a *behavior vector*, consisting of 9 features. All together are intrinsic to a botnet behavior. As the network analyzer is able to gather network-level information, the in-host analyzer need to focus on aspects the can’t be observed externally, and also features that can, and so the network stack features like Number of Suspicious Ports used, Number of Unique IP addresses contacted and Number of SMTP Flows are accounted. The suspicion-level generator uses a Support Vector Machine, a supervised machine learning algorithm to quantify the suspicion level. The training data fed to the system consists in both benign and malicious behavior profiles. Based on this training process, a hyperplane is created, which will correspond the the classification. When a new behavior vector arrives, the Support Vector Machine will compute its distance from the hyperplane, and decide whether to classify it as a benign or a malicious behavior.

The second part of the system architecture is also divided in two parts: 1. flow analyzer; 2. clustering module. The flow analyzer receives flow data as input, and begins filtering it. This filtering process consists in removing internal and legitimate flows, which is the traffic within the network and traffic to well known destinations, respectively. Then, it processes the remaining flow records of all hosts in a network to extract trigger-action patterns of interest in each time interval. As stated previously, bots receive the same input, and so, the analyzer looks for suspicious flow with the same destination IP address and transport protocol across all hosts, and labels them as triggering flows. Once again, by studying contemporary botnets, the authors found that bots usually connect to same group of C&C servers or peers to communicate with their *botmaster*, and execute the commands immediately. As the benign hosts rarely visit same IP with same protocol, upon filtering legitimate and internal flows, it is possible to distinguish them from the infected ones. From the observed traffic, 17 flow features are extracted into a feature vector. The first 14 are for classifying traffic pattern, and last 3, which are common to those also extracted from in-host monitoring, are used to measure botnet’s malicious intent to some degree. Then, a hierarchical clustering algorithm is used upon these feature values for each time interval, in order to group hosts that might be considered botnets.

After all this data is extracted both from the network analyzer and the in-host analyzer, the correlation engine performs a last analysis to check if a botnet is or not present. When a group of hosts are clustered, the respective host analyzers are requested the suspicion-level together with some network statistic. There are two possible outcomes from this step:
1. The network statistics sent from the in-host analyzer may differ from those observed by the network analyzer, implying that the host if sending falsified data, therefore triggering immediately an alarm.

2. If the results are consistent from both analysis, then the suspicion-level and clustering quality must be analyzed by the correlator. This is done by a function with two parameters. The first is the suspicion-level, which itself is a quantitative measure. The second is the distance between the hosts presented in the same cluster, which is computed by a simple Euclidean Distance.

With these three modules, the authors managed to track, from real-world data, different kinks of botnets, achieving low false-positive and false-negative rates. However, it is made clear that flow-based NIDS alone are not able to detect Botnets, as it is still necessary to gather information that is not provided by network flows.

### 2.4 Intrusion Detection Based on Machine Learning

An increasing trend in IDS is the use of machine learning techniques [LSBG13, SSS+10].

*Machine Learning* can be defined as a collection of methods that aim to attain knowledge by building an intelligent system through the observation of patterns in a given environment [LSBG13]. This knowledge may be refined and improved at each iteration, by learning from previous experiences and observations. Such method has been used in an enormous number of different applications, in many different fields of science, such as natural language processing, speech recognition, bioinformatics, spam detection, network intrusion, among many others.

Machine learning algorithms can be divided into two major fields:

- Supervised learning
- Unsupervised learning

The first one relies on a labeled training dataset. The dataset consists in an extensive list of input data that aims to train the system, making correspondences between keywords and their meaning or interpretation that is expected to the system. After this training phase, the system is ready to classify data based on the training set it was trained to. Examples of this method are the algorithms Naïve Bayes and Support Vector Machine. While Supervised methods relies initially on the introduction of training data, Unsupervised methods only receives as input a feature vector without any kind of labeling, and is used to means such as discovering similar groups within a data set. Clustering is an example of this kind of learning.
In the field of network intrusion detection, machine learning has been able to classify network traffic and identify both anomalous patterns and potentially harmful machines. When it comes to embed this technology in an Intrusion Detection Systems, generally this is the strategy adopted:

- **Anomaly-based detection (or behavior-based)**, in which normal traffic patterns are differentiated from anomalous ones. It focuses its attention on finding patterns that would not be expected from the machine’s behavior. Unlike **misuse-based** NIDS, these patterns are unknown to the system.

These two make up the classical approaches, being that machine learning is applied to the anomaly-based approach, rather than the misuse-based one. However, there are some variants to these, as we will now be discussing. Eskin et al. were one of the first to address unsupervised learning in intrusion detection systems [PES01]. They present a new technique, which they entitle *Unsupervised Anomaly Detection*. This technique allows to train the system with a dataset of completely unlabeled data, providing the chance of detecting unknown attacks to the network, which would not be possible when training the system with labeled data - in this case, the system is only able to recognize those labeled intrusions; and also, the manual classification of data can be very hard and tiresome. They used the famous KDD99 dataset for training the system. This dataset is a labeled dataset that was built in a military environment, and is provided to the intrusion detection research community (note that nowadays this dataset is almost deprecated, as it was collected in 1999). The features were extracted from connection record the raw data gathered throughout the simulated intrusions present in the dataset. This included features such as the basic components of a TCP connection (duration time, protocol type, etc.), some others that were not so trivially obtained (number of file creation operations, number of failed login attempts, etc.), and some other features captured in a small two-second time windows (number of connections to the same host as the current connection in this timespan, percentage of connections that have SYN and REJ errors, etc.). All summed up, the authors considered a total of 49 features. Also, the dataset was filtered, so that there would only exist a percentage of 1 to 1.5% of attacks vs. 98.5 to 99% normal traffic instances. This is done because of the need of the system to learn to distinguish intrusion instances from the normal ones, and the original dataset was composed mainly of intrusions.

Their solution was based on two assumptions:¹

1. The number of normal instances greatly outnumbers from the number of intrusions

2. The intrusions themselves are qualitatively different from the normal instances

¹These two assumptions are extremely important, as these will be the foundation for the intrusion detection approach.
The system clusters the collected data through an algorithm that computes a distance-based metric. However, because of the different distributions that each feature vector may have, these have to be normalized in order to apply the same metric to all of the vectors. After computing the clustering algorithm, these new clusters can now be classified as being normal traffic instances or an intrusion. The first assumption implies that small clusters correspond to the intrusion instances, as opposed to the bigger clusters that represent normal instances; the second assumption implies that normal and intrusion instances will not be under the same clusters because of their qualitative differences.

In order to measure the performance of the system, they used the following metrics:

- **Detection rate**, which represents that ratio of intrusions detected by the system, by the intrusions present in the dataset
- **False positives**, which is the ratio of the total number of intrusions that were incorrectly detected by the system, by the total number of normal instances (as defined by the author [PES01])

In this solution, there is an inevitable trade off between these two indicators, as one scales with the other. However, they managed to obtained a fairly reasonable ratio of detection rate and false positives.

This paper presents mainly two advantages to the traditional intrusion detection systems. The first, is that it does not require any kind of manual classification, and the second is that the system is capable of detect intrusion that were previously unknown.

Similar to this work, and more recently, a system that goes by the name of UNIDS (Unsupervised Network Intrusion Detection System) was developed [CMO12], which is able to detect unknown attacks without requiring any labeling, signatures or training. In order to understand the results obtained, they always rely on the assumption, just like the previously stated work, that the vast majority of the observed traffic is considered normal, rather than anomalous.

From a higher level perspective, the architecture of the solution can be divided in the main components: 1. the detection of anomalous time slots, resorting to change-detection algorithms; 2. the processing of the flow records present in the identified anomalous time slots, creating clusters of the gathered data; 3. the ranking of outliers generated from the clustering algorithm

In the first part of the solution, upon the extraction of network flow records - which are 5-tuples: the source and destination IP addresses and ports, and the protocol used - these are aggregated by unique aggregation keys ($l_i$): 1. traffic per time slot ($l_1$); 2. source network prefixes (IP source /8, /16, /24) ($l_2$); 3. destination network prefixes (IP destination /8, /16, /24) ($l_4$); 4. source IP addresses ($l_5$); 5. destination IP addresses ($l_9$). The first seven are used
exclusively to the change-detection phase, while the last two are used on the clustering algorithm. In order to detect anomalous time-slots, they construct time-series $Z_t^i$ for traffic metrics like the number of packets and IP flows per time slot. For each time slot, a change-detection algorithm is used to analyze the different time-series associated with each one of the aggregation keys (from $l_1$ to $l_7$), and the respective time slot is considered to be anomalous if the change-detection algorithm triggers an alarm for any of the defined traffic metrics. The fact that multiple aggregation levels are being used, allows them to have a higher reliability, since they are able to analyze different perspectives of the traffic, going from finer to more coarse aggregation levels.

Upon the discovery of the anomalous time slot, the abnormality degree will be calculated for each and every one of them, using both clustering and outliers analysis techniques. An outlier is any sample that does not belong to a cluster, and these will correspond to the anomalies in the traffic, due to the assumption that the vast majority of the observed traffic is normal. The traffic anomalies can be grouped in two different classes: the first one is a $1$-$to$-$N$ anomaly, in which an IP flow from a single source infects $N$ target hosts, and such is the case of port scans, worms and viruses. The other one is $N$-$to$-$1$, which, analogous to the previous one, has IP flows from $N$ sources infecting one single destination. The aggregation key $l_8$ (IPsrc) allows to identify the first class, while $l_9$ allow to identify the latter.

Each of the aggregated flows is described by $x_i \in \mathbb{R}^m$, a set of $m$ traffic features, such as the number of sources or destination ports, and a component of $X \in \mathbb{R}^{m \times n}$, the complete matrix of all the features, being $n$ the number of aggregated flows. The previously mentioned clustering algorithm will be performed the feature space $X$. First, an algorithm called Sub-Space Clustering will be applied to create $N$ data partitions $X_i$ out of $X$, by selecting $k$ features from the complete set of $m$ features. From these partitions, another clustering algorithm called Density-Based Clustering will be applied to each one of them, generating partitions $P_i$. Apart from the partitions, there will also be generated the previously mentioned outliers, which will later on be ranked in order to find classify the anomalies. To this, they use the concept of Evidence Accumulation, which uses the clustering results of the multiple partitions, and produces similarity measures that are able to better reflect their groupings.

Now that we have presented a couple of unsupervised NIDS, we present one that is built in a supervised fashion. Winter et al. [WHZ11] developed a system that uses the network data as training for a One-Class Support Vector Machine (OC-SVM) algorithm.

Based on the fact that machine learning methods achieve better results when classifying inliers than when classifying outliers, the training set consisted only in malicious data, rather than benign data. This way, the inliers will the be the malicious data, for which the training
algorithm will achieve better results. The first step in their system was, upon the receiving of a flow stream, the processing of data. In this step, they filter the flows from the dataset in order to achieve a feasible value that would not overload the learning process. Then, they needed to tune the OC-SVM, in order to achieve the best results possible (in this case, the false negative and miss rate were taken into account). To do so, they needed to achieve the best trade off between the parameters $v$ and $\gamma$, which will determine the fraction of outliers in the set and the width of the RBF (Radial Basis Kernel) - that was added to the algorithm, respectively. Also, in this step, the most valuable features will be selected. This optimization was done first in a coarse grained fashion, to determine the best feature subset and the best trade off between the two parameters; and then, in a finer grained fashion, that would further explore the relation between the two parameters to achieve better results. From this optimization, the most relevant attributes were: Packets per Flow, Source Port, Destination Port, TCP flags and the IP protocol.

With this setup, they managed to achieve a 0% false alarm rate, and a miss rate of only 2%, approximately. This shows that this approach might be interesting to consider to our solution, as the results obtained were extremely good.
Chapter 3

Network Intrusion Detection Approach

In this section we describe a flow-based NIDS capable of detecting unknown network attacking hosts. This section starts with an overview of the system’s idea, followed by the architecture of the solution, proceeding to a more detailed view of each module. Moreover, this chapter provides a description of the new ideas introduced and to be further implemented.

This work aims to cover two major drawbacks of traditional NIDS:

1. The inability to react to an unknown pattern, given a real-world dataset, i.e. containing both clean and malicious traffic

2. The slow processing and analysis of the traffic payload, as well as the inability to interpret its content

The first drawback may be countered by using an unsupervised machine learning algorithm, and the second is tackled by performing the analysis at a flow-level. In order for the system to provide accurate results, it must be provided with an adequate dataset. If so, and combining these two ideas we are able to design such a system that overcomes these issues.

3.1 Approach Overview

Figure 3.1 provides an overview of the structure of the system, as it will now be explained, following the workflow.

First of all, upon receiving the gathered flows (step 1 of the figure), a filtering is performed. This filtering will consist of removing unnecessary features from the flows, e.g. its payload content and date (as the flows are organized in such way that each flow is stored in a file whose
name has information regarding the latter). Therefore, apart from the traditional 5-tuples, three more features will be extracted (more details in Section 4.1), as depicted by step 2 in the figure.

![Diagram of intrusion detection approach]

**Figure 3.1:** An overview of the intrusion detection approach

The extracted tuples will be fed to a clustering algorithm (step 3), which separates the whole traffic into groups of hosts that share a common pattern. We will take into account the assumptions that the majority of the observed traffic is benign rather than malicious, as well as that malicious traffic is qualitatively different from the regular, normal traffic, when analyzing the data. Upon the clustering of the data, a manual intervention takes place. This manual intervention is performed on the outlier clusters produced by the algorithm, in order to better perceive and analyze the characteristics of this traffic, ultimately leading to the production of a labeled dataset that will serve as training for the next step. Once the flows are properly classified, they are then passed on to the next step.

Step 4 corresponds to a supervised learning module. In this module, the system runs a supervised learning algorithm that is trained with the labeled data produced by the former step, and will proceed to classify the traffic that was perceived as being outlier by the clustering algorithm. Upon this classification, the system should be able to correctly identify the observed malicious traffic, thus allowing the detection of the malicious hosts. This process is to be performed on a daily basis. If the analysis period was smaller, some attacks would not be possible to detect, as some of them last for long periods of time; if it was longer than a day, the obtained values would become noisy, as the flows are aggregated, some IP addresses may be reused from one day to another, therefore achieving very high feature values, misleading us to think that it is indeed an attack.

The first step - filtering and feature extraction - will be crucial for the performance of the whole system, as it will be the input for the rest of it. If these features are carefully chosen, the system may be able to produce good results, performing an accurate clustering
of data; if not, the clustering result might be completely inconclusive, thereby impairing the remaining modules. Furthermore, the decision of using aggregated flows rather than simply analyzing individual flows, was due to the observation made in the intrusion detection community [CMO12,SLKM11,SSP12,KKH+04] that aggregated flows provide a more precise analysis, e.g. in the case of detecting DDoS. If some attacks were to be analyzed using individual flows, it would be much harder to detect them. The choice of using the source and destination IP addresses as aggregation keys was inspired by [CMO12], that used them to distinguish groups of $1$-to-$N$ and $N$-to-$1$ anomalies.

The unsupervised clustering algorithm will be applied on these aggregated features, which will proceed to form various large groups of hosts, and some outliers. According to the previously mentioned assumption, these outliers may represent an attack, although this may not always be the case. Such outliers could also represent, for instance, some applications that are less frequently used, or even a machine whose characteristics are not very common, therefore producing flow features that are different from regular traffic that is found in bigger clusters. So, it is of utmost importance to analyze them, in order to differentiate the actual attacks from these benign outlier traffic patterns. In a first run of the system, the supervised learning module has not yet any knowledge at all, and so there is a need for a manual intervention that will classify and label these outliers. This manual classification will serve as input to the supervised learning algorithm, that, over time, will come more and more capable of classifying on its own. In the following runs, there may not be a need for manual intervention if the classification produced by this learning algorithm is feasible (which is to be validated with ground truth, as will be discussed in the next section); else, an expert must classify this traffic manually and feed the algorithm once again. The supervised learning algorithm that we will be using is the SVM. For the unsupervised learning we will use, as previously mentioned, a clustering algorithm, namely K-Means (although the Mini Batch K-Means algorithm was also tested).

The remainder of this section will describe the core of these modules, and how they relate to each other.

### 3.2 Data Gathering and Preprocessing

Our NIDS approach assumes that data is collected using NetFlow-enabled routers, e.g. placed at the border routers between the core network and the connection to the ISP. Figure 3.2 depicts the specific scenario considered in our study. All the data that reaches the network arises from the ISP, that connects to it via routers that are placed at the borders of the core network. These routers are NetFlow-enabled, and are protected by firewalls, in order to filter
any wanted data, according to their security policies, therefore ensuring that only supposedly clean data reaches its clients. However, not all bad traffic is filtered, which is why there is a need for extra security measures, such as NIDS. So, the data that reaches these routers (i.e. the data incoming/outcoming from the ISP) is collected by NetFlow, for further analysis. For the sake of analyzing and treating these flows, all of this data was converted to the SiLK format. Having the flows in SiLK format, these will go through a number of modules and transformations, before reaching the clustering and supervised learning sections. The first step is to aggregate the flows and, apart from the ones that are already provided by the dataset itself, extract some new features, in order to have a more rich and descriptive dataset. Having extracted these features, there is then the need to normalize the data.

![Flow collection scenario considered in the experimental evaluation of the NIDS approach](three.prop/two.prop/one.prop)

Figure 3.2: Flow collection scenario considered in the experimental evaluation of the NIDS approach

### 3.2.1 Feature Extraction

In order to obtain an overall improvement on the system’s performance, we use parallel processing of the data, thus reducing the computational processing time.

To achieve so, the system implements a Map Reduce algorithm. Map Reduce, which was first introduced by Google [DG08], is a Big Data programming paradigm whose goal is to provide maximum scalability, in order to process massive amounts of data. It achieves so by dividing the task in two parts: the Mapper and the Reducer phases. The Mapper phase takes as input a set of data blocks and divides it into <key,value> pairs, as specified by the programmer; the Reducer phase receives as input the output generated by the former phase, and combines key pairs with the same keys, performing some operation to their respective values. Figure 3.3 depicts the basic workflow of the algorithm.

As it was previously mentioned, the flows are aggregated, which means that all the flows that have the same IP address (Destination and Source addresses for destination and source aggregations, respectively) will be merged into one. This is when the Map Reduce algorithm will be applied. The mapper nodes will, for every entry in the dataset, extract the Source and
Destination IP addresses, Source and Destination ports, number of sent packets, which protocol was used, TCP flag (if any), number of exchanged bytes and its duration. Each of these values will be sent individually to the reducer (i.e., for every entry, 9 records will be sent), being the key a string in the following form: Source/Destination, feature, IPAddress, and the value will be the correspondent value of the feature in question. For example, in order to aggregate the number of bytes of a certain Source IP destination, the Mapper would produce a tuple such as: `<S,bytes,192.168.2.105', 64>`, where 192.168.2.105 would be one of the key IP addresses, and 64 the number of bytes sent in that flow. The reducer nodes, on the other hand, will receive these records, and sum the value of all the records that have the same key. For example, if it receives two tuples `<S,bytes,192.168.2.105', 64>` and `<S,bytes,192.168.2.105', 128>`, it would produce as output `<S,bytes,192.168.2.105', 192>`.

The Map Reduce algorithm calculates the values (these sums) of a set of features, and produces some more feature that derive from these. These features are presented in Table 3.1. The Aggregation Key feature is obtained simply by creating a new tuple for each different IP address, both for source and destination. When the reducer receives tuples with an Aggregation Key that already has been seen before, it will generate the remaining features described in Table 3.1. Features NumSIPs/NumDIPs, NumSports, NumDports, TotalNumBytes and TotalNumPks are obtained simply by summing all the features values in each flow, that is, for each flow the algorithm will produce a sum of all the different IP addresses contacted, all the different Source Ports and Destination Ports, number of packets and number of bytes, respectively. There are also features that will count the occurrences of contacting ports 80, 194 or 6667, 25 and 22, i.e. of using protocols Hyper Text Transfer Protocol (HTTP), Internet Relay Chat (IRC), Simple Mail Transfer Protocol (SMTP) and Secure Shell (SSH), namely NumHTTP, NumIRC, NumSMTP.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation Key</td>
<td>The IP address that will be used as an identifier, to which the below features relate to</td>
</tr>
<tr>
<td>NumSIPs / NumDIPs</td>
<td>The number of IP addresses contacted</td>
</tr>
<tr>
<td>NumSports</td>
<td>The number of different source ports contacted</td>
</tr>
<tr>
<td>NumDport</td>
<td>The number of different destination ports contacted</td>
</tr>
<tr>
<td>NumHTTP</td>
<td>The number of packets to/from port 80 (HTTP)</td>
</tr>
<tr>
<td>NumIRC</td>
<td>The number of packets to/from ports 194 or 6667 (IRC)</td>
</tr>
<tr>
<td>NumSMTP</td>
<td>The number of packets to/from port 25 (SMTP)</td>
</tr>
<tr>
<td>NumSSH</td>
<td>The number of packets to/from port 22 (SSH)</td>
</tr>
<tr>
<td>TotalNumPkts</td>
<td>The total number of packets exchanged</td>
</tr>
<tr>
<td>PktRate</td>
<td>The ratio of the number of packets sent and its duration</td>
</tr>
<tr>
<td>ICMPRate</td>
<td>The ratio of ICMP packets, and total number of packets</td>
</tr>
<tr>
<td>SynRate</td>
<td>The ratio of packets with a SYN flag and the total number of packets</td>
</tr>
<tr>
<td>TotalNumBytes</td>
<td>The overall sum of bytes</td>
</tr>
<tr>
<td>AvgPktSize</td>
<td>The average packet size</td>
</tr>
<tr>
<td>BadSubnet</td>
<td>This field expresses whether the IP address belongs to a blacklisted subnet</td>
</tr>
<tr>
<td>MaliciousIP</td>
<td>This field expresses whether the IP address is blacklisted</td>
</tr>
<tr>
<td>OpenVaultBlacklistedIP</td>
<td>Same as the above, but checked from a trusted and well know threat database [Ope]</td>
</tr>
<tr>
<td>MaliciousASN</td>
<td>This field shows if the IP address belongs to a blacklisted ASN</td>
</tr>
<tr>
<td>LocationCode</td>
<td>Code for the country associated with the address</td>
</tr>
</tbody>
</table>

Table 3.1: Features used

and NumSSH. By dividing the total number of packets by the total duration of the Aggregation Key, we obtain the PktRate. As we mentioned earlier, one of the features that composes a flow is what TCP flag is being sent, if any. There is also a summation for all the times that a SYN flag is sent, and in the end of the processing, this value is divided by the total number of packets for that key, resulting in the feature SYNRate, and the same applies to ICMPRate, where the total number of times the Internet Control Message Protocol (ICMP) protocol is used is divided by the total number of packets. From the last 5 features, the first 4 were generated based on online databases, whose goal is to identify known threats, while the last one was fetched from a Python IP tracker module.¹

3.2.2 Data Normalization

This step is again crucial. As we will be working solely with numerical data, we need to keep every value in one common scale. Moreover, there are some features that are not expressed in a numerical manner, such as the IP addresses of the IP’s associated country. In these cases, these features are mapped to numerical values, which can be reversed to text.

Normalizing a dataset means mapping a set of values to a specific range. In our case, all the values will belong to the interval [0,1], where 0 is the minimum, and 1 the maximum. To achieve so, given a dataset entry in the form \( X = (x_1, ..., x_n) \), the correspondent normalized \( Y \) vector is

obtained using:

$$y_i = \frac{y_i - \min(x)}{\max(x) - \min(x)}, y_i \in [0, 1]$$

Tables 3.2 and 3.3 describe such a normalization.
### Table 3.2: Raw Features

<table>
<thead>
<tr>
<th>Aggregation Key</th>
<th>NumSIPs / NumDIPs</th>
<th>NumSports</th>
<th>NumDport</th>
<th>TotalNumPkts</th>
<th>PktRate</th>
<th>ICMPRate</th>
<th>SynRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.3.115</td>
<td>154</td>
<td>1506</td>
<td>25</td>
<td>134662</td>
<td>1.0938</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TotalNumBytes</th>
<th>AvgPktSize</th>
<th>BadSubnet</th>
<th>MaliciousIP</th>
<th>OpenVaultBlacklistedIP</th>
<th>MaliciousASN</th>
<th>LocationCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>389105930</td>
<td>2889.5006</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>United States</td>
</tr>
</tbody>
</table>

### Table 3.3: Normalized Features

<table>
<thead>
<tr>
<th>Aggregation Key</th>
<th>NumSIPs / NumDIPs</th>
<th>NumSports</th>
<th>NumDport</th>
<th>TotalNumPkts</th>
<th>PktRate</th>
<th>ICMPRate</th>
<th>SynRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.863979570845325</td>
<td>0.025844930417495</td>
<td>0.24591381872214</td>
<td>0.0527752502274795</td>
<td>0.190662253708506</td>
<td>0.0123052238027187</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TotalNumBytes</th>
<th>AvgPktSize</th>
<th>BadSubnet</th>
<th>MaliciousIP</th>
<th>OpenVaultBlacklistedIP</th>
<th>MaliciousASN</th>
<th>LocationCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.279399032125611</td>
<td>0.0352953096554911</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
3.3 Clustering

As stated in the previous section, machine learning algorithms can be divided roughly in two categories – supervised and unsupervised learning. In this section we will focus on the latter, as the first one will be discussed further ahead.

The idea behind clustering is to group different instances of a dataset into \( k \) distinct groups, i.e. clusters, according to their characteristics. For instance, applying a clustering algorithm to a dataset of network traffic would generate \( k \) clusters, where one would be representative of regular DNS traffic, another one would be simple SMTP traffic, and so on. This is done by feeding a set of vectors to the algorithm, which will then proceed to obtain groups of elements for that set of vector. The previously mentioned data normalization was performed specifically for this step.

Depending on the algorithm used, the value of \( k \) may or may not be chosen automatically. For example, the DBSCAN algorithm [EKS+96] does not need a predefined value for the number of clusters; on the contrary, K-Means requires it. Two different clustering algorithms have been used throughout this work:

- K-Means
- Mini Batch K-Means

3.3.1 K-Means Clustering

This algorithm starts by receiving a set of vectors \( X = (x_1, ..., x_n), x_n \in \mathbb{R}^d \), where \( d \) is the number of features. Based on the \( k \) predefined number of clusters, K-Means initializes a set of \( k \) random centroids (cluster centers) \( C = (c_1, ..., c_k), c_k \in \mathbb{R}^d \). The first step is, for each point \( x_i \) in the dataset, it will compute its nearest centroid, using an Euclidean Distance (ED), which is computed using \( d(a, b) = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \). This step is followed by another iteration, this time for each and every centroid. Each centroid will be updated so that its coordinates are the result of the mean value of all points belonging to that cluster. The algorithm will come to an end when the variations in the new centroid position are no longer significant, i.e. when \( \Delta c_i > \xi \). Such behavior is described algorithmically in Algorithm 3.3.1.

Although K-Means was proven to be an efficient algorithm, and widely implemented [KMN+02], it may not always be the best fit for every dataset. Its complexity grows with the number of clusters \( k \), the number of dimensions \( d \) and the number of instances of the dataset, \( n \). As stated by [KMN+02], briefly, its complexity grows as this:

- The time needed to assign the first round of points to its centroids is \( O(n \times d \times k) \)
Algorithm 1 Pseudo Code for the K-Means algorithm

1: procedure K-Means(X, k)
2:   Generate random $C = (c_1, ..., c_k), c_k \in \mathbb{R}^d$
3:   for $\forall x_i \in X$ do
4:     Find closest centroid using $\arg\min_p d(x_i, c_p)$
5:     $x_i$ is associated to cluster $p$
6:   for $\forall c_j \in C$ do
7:     while $\Delta c_j > \xi$ do
8:        $c_j \leftarrow \frac{1}{n_i} \sum_{x_j \in X} x_j$
9:   return $C$

- Computing time of each centroid is $O(n \times d)$
- And, at last, the time spent in calculating the error function is $O(n \times d)$ as well

So we see here that the computation time grows almost exponentially as we increase the dataset and its dimension.

3.3.2 Mini Batch K-Means

In order to overcome the computational cost of the K-Means algorithm when applied to large datasets, a new algorithm was proposed by Sculley et al.: Mini Batch K-Means [Scu10].

Algorithm 2 Pseudo Code for the Mini Batch K-Means Algorithm

1: procedure K-Means(X, k, i, b)
2:   Generate random $C = (c_1, ..., c_k), c_k \in \mathbb{R}^d$ with an $x$ randomly picked from $X$
3:   $i \leftarrow 1$
4:   $v \leftarrow 0$
5:   while $i < n$ do:
6:      $M \leftarrow b$ instances randomly picked from $X$
7:      for $\forall x_i \in M$ do
8:         Find closest centroid using $\arg\min_p d(x_i, C)$
9:         $x_i$ is associated to cluster $p$
10:    for $\forall c_j \in C$ do
11:       while $\Delta c_j > \xi$ do
12:          $c_j \leftarrow \frac{1}{n_i} \sum_{x_j \in X} x_j$
13:       $i \leftarrow i + 1$
14:   return $C$

While K-Means is applied to all of the instances of the dataset at once, Mini Batch K-Means takes randomly chosen smaller batches, and performs its computation from these smaller batches. For each of these batches, a cluster is assigned to each instance. Instead of using the mean of the values for each instance of the cluster, it uses a gradient descent update, which also speeds up the computational time. In addition to what the previous algorithm received as input, Mini Batch K-Means also requires the maximum number of iterations and the size of the batches.
Although this technique can be much faster than the standard K-Means algorithm, it produces slightly different results. For this reason, both techniques were tested and compared, in order to analyze which one was the best fit for our purpose.

### 3.4 Supervised Cluster Classification

The malicious flows, i.e. the flows that correspond to intrusions, will be placed in the clusters with smaller size. Once the clusters are generated, each one of them will be manually inspected, in order to identify if they are malicious. In order to obtain a coarse grained overview of each cluster’s content, each feature of each cluster will be described by its mean value and standard deviation. This way, it is possible to have an idea of each cluster’s behavior. Of course, this is only a preliminary method to indicate each cluster’s feature distribution. Then, we focus on the clusters with smaller size and higher feature values, and proceed to label one of the flows that they contain. Each of these labeled flows is then fed to the SVM, that will learn from these examples.

Supervised learning problems can be divided into two different categories: 1. Regression, and 2. Classification. The former produces outputs for $\mathbb{R}$, while the latter produces outputs for $\mathbb{N}$. Since our goal is to detect malicious hosts, we want our outputs to be discrete values, identifying if one is or is not malicious, we fall therefore under the first category. Examples of algorithms for this problem are Naïve Bayes, Decision Trees, Perceptrons, Bayesian Networks, k-Nearest Neighbors (k-NN), Support Vector Machine (SVM), among many others.

Having the data separated into different groups according to their respective behavior and being properly labeled, from the last module, we have now gathered the necessary conditions to apply such a technique. For the purpose of this work, we chose to apply a SVM. This decision was due to the fact that this algorithm is proved to behave well when it comes to intrusion detection [WHZ11, ZHS10].

#### 3.4.1 Support Vector Machine

The goal of the SVM algorithm is to find an optimal hyperplane for patterns that are linearly separable. An hyperplane can be defined as an Euclidean space that has one less dimension than the space provided for the problem, e.g. for a 2-dimensional space, the hyperplane would simply be a line. The optimal hyperspace is that which maximizes the margins between itself and the vectors provided, i.e. the classes that will be used for classification. The SVM receives as input $n \times f$ training vectors $X = (x_1, x_2, \ldots, x_f)$, where $f$ is the number of features in the dataset, and a $n \times 1$ vector, which will be the corresponding class for each of the input $X$ vectors. It will
produce as output a set of weights $w$, whose linear combination will predict the value of $y$, for each $X$ provided from now on.

For the purpose of this work, the SVM will have two class, namely 0 and 1 – a binary output. For each input flow, it will label it as 0 if the flow is benign, or as 1 if the flow is malicious. The reason why there is only one class is that the goal of this work is not to classify, but to identify attacks, and therefore there is no need to differentiate them. Our aim is to identify malicious hosts, regardless of the type of attack they conduct, and therefore the SVM output will identify each flow as being malicious or benign.
Chapter 4

Implementation

This section will describe the implementation of the NIDS. All of the following modules were developed in Python, as it has a wide open source community, with a number of machine learning libraries available, already optimized for the purpose of this work.

As the unsupervised learning and the classification modules are the sections that require the more interaction with the user, a terminal interface was developed in order to facilitate such tasks. The interface for the unsupervised learning module features a menu that allows the user to: 1. obtain a brief description of the dataset; 2. plot graphics to aid in the choice of the number of clusters; 3. generate and visualize the clusters; 4. analyze each of the generated clusters; 5. interactively visualize the generated clusters; 6. access a database to validate the attacks, if possible; 7. plot two features, one against the other; 8. alternate between datasets; 9. change clustering algorithm. As for the supervised learning module, the interface features a menu that allows to: 1. predict values for a given day 2. train the system. With these interfaces, it was much simpler to analyze and interpret the results obtained in this section thus allowing to properly classify the observed traffic, as it allows for an easier interaction between the user and the data. The remainder of the system does not feature a terminal interface, but it does have scripts that automate the processes, as it did not require so much interaction as the former modules, it can be made automatically.

4.1 Map Reduce

As it was mentioned, the first step is to gather the flow collection. These flows are collected using NetFlow-enabled routers placed at the border routers between the core network and the connection to the ISP. For the sake of analyzing and treating these flows, all of this data is converted to a SiLK format. Once the data is converted, it is fed to a Map Reduce algorithm
developed to be able to do parallel processing of the data.

To achieve this goal, the system will be using the Hadoop framework [DG08,SKRC10]. Hadoop is an open-source framework that features both distributed storage and parallel processing of Big Data, making it very scalable to very large amounts of data. As it operates in a distributed fashion, it can take advantage of a cluster of computers, using its computational power to achieve a higher speed during the data processing. To support the parallel data processing, Hadoop implements Google’s Map Reduce algorithm [DG08]. This model operates on a virtual environment called Hadoop Distributed File System (HDFS), which has both Mapper and Reducer nodes. This model can be divided in two main steps: Mapping and Reducing. First and foremost, the input data is split in equal sized blocks (64 MB by default), and each of these blocks is fed to a Mapper node. Each of these nodes divides the received chunks into <key,value> pairs, that will then be sent across the network to the Reducer nodes. Before proceeding to execute the algorithm, a filtering to the dataset is performed, maintaining only the following features:

- Source Internet Protocol (IP) address
- Destination IP address
- Source Port
- Destination Port
- Number of packets sent
- Protocol used
- Which TCP flag was sent, if any
- Number of bytes sent
- Duration of the flow

Other features such as the Starting and End Time, and payload content were removed from the dataset.

Having the dataset filtered, the system proceeds to execute the Map Reduce algorithm. This consists in two distinct parts, as the name suggests: the Mapping and the Reducing phases. The goal of the former is to aggregate values by a specific key; the goal of the latter is to perform some sort of operation on the aggregated keys generated by the Mapping phase. In our case, the aggregation keys will be both Source IP and Destination IP. However, for each of these keys, a number of operations on each feature must be performed. Therefore, the Mapper consists in two functions:
1. Source Features

2. Destination Features

Both will produce \(<key,value>\) pairs for each one of the features mentioned above. Apart from these, the mapper will also produce some more features:

- A counter for the number of times the port 80 was used
- A counter for the number of times the ports 194 and 6667 were used
- A counter for the number of times the port 22 was used
- A counter for the number of times the port 25 was used

These ports relate to the protocols HTTP, IRC, SMTP and SSH, respectively.

The goal here is to count the occurrences of each one, whilst preserving the aggregation key.

For the Source IP as aggregation key, the mapper will produce the following outputs:

\(<"S,bytes,sip,file",#Bytes>\)
\(<"S,dstIP,sip,file",DstIP>\)
\(<"S,dstPort,sip,file",DstPort>\)
\(<"S,duration,sip,file",Duration>\)
\(<"S,packets,sip,file",#Packets>\)
\(<"S,protocol,sip,file",Protocol+Flag>\)
\(<"S,srcPort,sip,file",SrcPort>\)
\(<"S,HTTPPort,sip,file",Yes/No>\)
\(<"S,IRCPort,sip,file",Yes/no>\)
\(<"S,SSHPort,sip,file",Yes/No>\)

The outputs produced are analogous to the destination function, also present in the Mapper phase, simply replacing the 'S' key by 'D', and 'sip' by 'dip'.

In the Reducer phase the algorithm will proceed to receive as input the outputs produced by the Mapper and process them in order to produce the features that are described in Table 3.1. Once these features are computed, the final output of the program will be written in two files:
one containing the data *per se*, including the file where each of the flows is located; another where
the file is excluded and the IP addresses and the Location Codes are converted to a numerical
format, in order to be able to perform some mathematical operations to the data, which will be
necessary for remainder of the program.

### 4.2 Unsupervised Learning

For the third step of Figure 3.1, the user must run the unsupervised learning module. Once the
flows are aggregated, the unsupervised learning module may take place, in order to cluster the
flows.

![Terminal user interface for the unsupervised learning module](image)

**Figure 4.1: Terminal user interface for the unsupervised learning module**

This module presents a terminal interface to the user, which consists in a group of task
implemented in the NIDS (see Figure 4.1):

1. Obtain a statistical description of the dataset
2. Obtain a graphical representation of the optimal number of the clusters for the dataset
   presented
3. Generate the clusters and visualize them (either in $\mathbb{R}^2$ or $\mathbb{R}^n$)
4. Obtain a statistical description of each cluster
5. Launch an interactive representation of the clusters, to aid visualization
6. Validate attacks in the dataset, from a database

7. Generate a graphic which maps just 2 features

8. Alternate between datasets (source or destination)


Figure 4.2: Interactive visualization of a cluster

The first option provides the user with a brief description of the data being analyzed, featuring the number of entries of the dataset, and, for each feature, it also presents its minimum and maximum value, and also the mean value and standard deviation. This way, we get a simple overview of the data, statistically. The fourth option presents, for each cluster and feature, its mean value and standard deviation. As for the fifth option, window is opened in the browser, that allows to choose which cluster to analyze, and provides an interactive visualization of the data it contains, as seen in Figure 4.2

4.2.1 Choosing the number of clusters

As it was discussed in Section 3.3, both K-Means and Mini Batch K-Means algorithms require a number $k$ clusters to be specified. Upon the receipt of the dataset, we do not know exactly into how many clusters we want to divide our data. However, there are some techniques that give us a hint of what the value of $k$ should be. Such is the case of the Elbow Method (see Figure 4.3). These clustering algorithms converge when the variation of the distance between the data points and the clusters centers start converging to 0. With this in mind, the Elbow Method starts
by computing the error function that is used as a stopping criterion in the algorithm, know as Within Sum of Squares (WSS), which is mathematically defined as such:

\[ WSS = \sum_{i=1}^{k} \sum_{x \in c_i} \text{dist}(x, c_i)^2 \]

This equation will produce values for \( k \) in a specified range, which will be provided by the user. By plotting this values against its respective \( k \) value, we obtain a graphic that will be decreasing with the increase of \( k \). Theoretically, the optimal value of the WSS would be 0, but this value is only obtained when the number of \( k \) clusters is equal to the number of entries in the dataset, which would mean that each data point would be in its own clusters, and this process would provide us no information at all. Instead, the \textit{Elbow Method} tells us that the best value of \( k \) is when the slope of the WSS has a sudden break. Apart from the WSS, the \textit{Elbow Method} also plots a metric called Percentage of Variance Explained (PoVE) for each value of \( k \). This metrics reflects the ratio of the Between Sum of Squares (BSS) and Total Sum of Squares (TSS), which will indicate an optimal \( k \) when it suffers an abrupt change.

![Figure 4.3: Plot of the Elbow Method](image)

Although this method provides us with some more insight than what we previously had, this is an heuristic method, which may be statistically correct, but may not fit our dataset with maximum precision. With this in mind, this tool is used \textit{only} to give a hint of where a correct \( k \) value might be. After plotting the graphic, we still have to try several runs with values around the suggested \( k \). Accord to the Elbow Method, the optimal \( k \) would be 2 – in Figure 4.3 we see that the biggest slope is found where the x-axis is equal to 2 –, which is not correct. There is yet a rule of thumb often used, which consists in starting off with \( k = \sqrt{n} \), where \( n \) is the number of entries in the dataset. Ultimately, the finding the optimal number of clusters can not be an automated method, as human intervention is always necessary, due to the fact that the number
of clusters will always be dependent on our type of data, and the purpose of clustering. However, we do know that the value lies nearby 2, as shown be the Elbow Method, and after several runs with different values for \( k \), we found that \( k = 10 \) is a number of clusters that successfully and coherently divides the different datasets of various sizes. Also, if we analyze Figure 4.3, we see that around the value of 10 the values start to stabilize, and its variation is close to 0.

### 4.2.2 Generating the clusters

This task corresponds to the third option prompted in the interface – **Generate the clusters and visualize them**. When the data is described in more than three dimensions, it becomes a difficult task to represent it visually. For this reason, a 2-Dimensional representation of the data is also provided. Instead of dropping several features until only two are left, the data is rearranged such that there are now two features that describe the original ones. This process is called dimensionality reduction, and may be achieved through Principal Component Analysis (PCA) [WEG87]. PCA is a statistical technique that is used to identify hidden patterns in the data, by identify a number of uncorrelated features smaller than the original dataset. These smaller number of features are called the Principal Components. This method is able to reduce the number of features in a dataset by computing its eigenvectors and respective eigenvalues from a correlation matrix. This allows to retrieve the information that best describes the correlation between features, therefore synthesize the information into a smaller number of features.

As for plotting the dataset in its original dimension, a technique called Parallel Coordinates was used [ID91]. With this technique, the \( x \)-axis is divided into \( f \) sections, being \( f \) the number of features, while the \( y \)-axis is maintained as if in a normal 2D projection. This technique provides a visual aid for better understanding of our data, as the data can now be fully plotted in its entirety, therefore allow to easily identify which features have significantly higher values than others. Such representation is depicted in Figure 4.4.

Upon the creation of the clusters, a file for each one of them will be generated, in order to further analize them in depth. All of the data that is generated in this module is stored in an SQLite database, in order to maintain a persistent storage of the data. With the database, it is possible to traceback all the information regarding the flows. This is necessary, given that the data handled by the algorithm is strictly numerical, and the raw data contains textual information as well, namely the file where the flow is contained, the aggregation and also the location of the flow (i.e. its respective country).
Figure 4.4: Example of parallel coordinates visualization

4.3 Supervised Learning

For the fourth step of the figure, the user must run the supervised learning module, which is described below.

In order to generate the hyperplane that divides the data into different classes (in our case, the classes are either 0 or 1), the SVM algorithm uses kernel functions, which allow to divide it in linearly or non-linearly separable classes. A kernel function is a similarity function, i.e. a function that describes how similar two inputs are between each other. These kernel functions can be either Linear or Radial Basis Function (RBF). While the former is simply a linear combination of weights and is used for linearly separable datasets and requires no additional inputs, the latter will decided which is the optimal hyperplane by using either a Polynomial or Gaussian approximation. Also, this latter function requires an additional parameter $\gamma$, which will determine the width of the function. The system was tested using both Linear and RBF kernel functions, and when using a linear function the machine was able to produce more accurate results.

The results of the SVM are stored persistently, in a file. This way there is no need to train the system everytime we want to use the SVM. The system starts by verifying if this file exists, and, if not, it trains itself with the data from the first day of the analysis period – if the system is to be evaluated in a period of one week, it the SVM is trained with the first day of the week. Upon this training it is possible (see Figure 4.5):

1. predict results for a given day
2. further train the SVM with new data

In the first case, the system will present the user the flows (if any) that were labeled as malicious, for that day. These flows are identified, once again, by their unique ID, thus making it possible to traceback it back to its IP address stored in the database mentioned in Section 4.2.

![Image]

Figure 4.5: Terminal user interface for the supervised learning module

For the second case, more knowledge will be added to the SVM. This new data that will train the system must first be merged with the data that was already present, and then the system is trained with this new set of data. With this feature, it is possible to refine the results generated, as it allows to give new insight to the SVM. This means that, as we add more data to the system, it will have a better understanding of it, and will allow for it to detect malicious intent more accurately according to our analysis. For example, if in a first analysis some sort of malicious activity was not detected, and the system as trained with this data, it may not detect intrusions that follow that mislabeled pattern; but if later on we perceive that activity as malicious, it is possible to train the system once again with this new information. This process therefore allows to refine the SVM accuracy throughout the analysis period.
Chapter 5

Evaluation

In order to validate the performance of the system, two datasets were used. First, a dataset provided by the Information Security Centery of Excellence (ISCX)\(^1\) was used. This dataset consists in flows collected during one week, and aims to provide a complete testbed for IDS [SSTG12]. All of the flows in this dataset are properly labeled, therefore allowing for a validation of the accuracy of the system. Upon the cluster generation and respective manual labeling of this data, the results were compared to the ground truth provided by the original dataset. After the data has been labeled, we proceeded to train the SVM, which will be the classifier for further flows to be analyzed. Also, an analysis is performed upon real data provided by the portuguese ISP Vodafone Portugal.

5.1 Evaluation with ISCX dataset

5.1.1 Cluster Analysis

Once the data is both filtered and extracted, it must be normalized in order to obtain coherent results throughout the different days during which the analysis is performed. To do so, the absolute maximum of the features, as shown in Equation 1, will correspond to the absolute maximum observed throughout the six days, rather than the absolute maximum of each day. This way, we ensure that the cluster algorithm is able to detect when a feature is indeed presenting higher values. If this was not taken into account, every day there would be flows present the maximum feature value, as it would correspond to the maximum value observed in that day. For example, even during days that would contain no malicious activity, we would observe flows having the maximum value possible for its features, because these were not being put in context, and this could lead to false positives.

\(^1\)http://www.unb.ca/research/iscx/dataset/iscx-IDS-dataset.html
The ISCX IDS dataset is divided in 6 days, namely 1. Saturday, 2. Sunday, 3. Monday, 4. Tuesday, 5. Wednesday and 6. Thursday. Attacks were detected in all of these days, except for Wednesday, that was found to be a regular intrusion-free day. We will proceed to give the details of the analysis for the days that contained these attacks.

Table 5.1 contains the features that were used when representing the clusters, as the ones that are not present were found to always have a value of 0. Furthermore, for each cluster and feature, if the mean value of a feature is bigger or equal to 0.1, it has been represented alongside with its standard deviation. This way we are able to better perceive the content of each generated cluster, as they are not always composed by one single aggregated flow (from now, we will refer to the flows in the clusters as entries). Please note that this dataset is composed of hundreds of thousands of flows per day, but once the flows are aggregated, the number of flows is drastically reduced. This is due to the fact that all the traffic is produced by a small number of source IP addresses, thus drastically reducing its number of flows. In fact, this represented a major drawback when evaluating the system, as this the feature values tended to inflate quite a lot because all the traffic for a whole day was reduced to only a few flows.

Table 5.2 shows the clustering content for the first day, Saturday.

By looking at cluster 7 (8th column of the table), which is the cluster with the greatest amount of entries, we can see that all of its features present very low values, so they are omitted. Also, the clusters smaller in size are those where some variations in the features values can be observed, satisfying our assumption that the smaller clusters are the ones that have the most irregular traffic. By analyzing these, there is one cluster that present features that are rather alarming, which is cluster 3. In this, the number of different source ports used and number of
connections through the SSH port are highlighted, being the number of connections through the SSH port at its absolute maximum value. In Section 2.3.1, we presented studies that have shown that these two features together are representative of a Brute-Force SSH attack. Given that the rest of the traffic presents feature values that are rather normal (i.e. none of them is indicating the presence of an intrusion), we considered that the flow present in cluster 3 was performing such an attack, therefore highlighting it as an intrusion flow, as the remainder of the traffic was considered to be normal.

On the second day, Sunday, the results showed a rather different pattern, as shown in Table 5.3.

<table>
<thead>
<tr>
<th>Clusters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># Entries</td>
<td>6</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>20</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Features</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.172</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.102</td>
<td>-</td>
<td>0.110</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0.438</td>
<td>0.132 ; 0.045</td>
<td>0.885</td>
<td>-</td>
<td>0.216 ; 0.0002</td>
<td>0.190</td>
<td>-</td>
<td>0.216</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>0.318</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.224</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>0.325 ; 0.120</td>
<td>-</td>
<td>-</td>
<td>0.985 ; 0.189</td>
<td>0.412</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>0.871</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>0.871</td>
<td>0.245 ; 0.147</td>
<td>0.181</td>
<td>-</td>
<td>0.389 ; 0.171</td>
<td>0.905</td>
<td>-</td>
<td>0.873</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>0.944</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>0.479</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.111 ; 0.051</td>
<td>0.230</td>
<td>-</td>
<td>0.230</td>
</tr>
<tr>
<td>16</td>
<td>0.109</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.2: Saturday cluster content

Table 5.3: Sunday clustering content
Unlike the previous day, now we see that almost every cluster presents very high feature values. Features such as the average packet size, the number of source ports and the number of HTTP connection are high in the great majority of the clusters. Also, the number of SMTP connections was also found to be very high in two different clusters. This behavior shows us that something is not right, as the SMTP connections are usually grouped together in a single cluster, and this analysis shows us that two different clusters have these characteristics. Taking this into account we assume that these flows, although having this feature with very high values, were grouped into different clusters because they have a different behavior, and therefore showing us that these flows are not normal. As for the remaining clusters, we can see that clusters 3, 6, 7 and 9 have very high values for the the number of HTTP connections, alongside with the number of different source ports and average packet sizes. These three features together may indicate that a large volume attack is being perpetrated, exploring the HTTP protocol, therefore also labeling these clusters as attacks.

Moving on to Monday, the cluster analysis was shown to be, once again, different from the previous two days, as shown in Table 5.4.

Table 5.4: Monday clustering content

<table>
<thead>
<tr>
<th>Clusters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># Entries</td>
<td>375</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Features</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>0.239</td>
<td>-</td>
<td>-</td>
<td>0.282 ; 0.127</td>
<td>-</td>
<td>-</td>
<td>0.214</td>
<td>0.166 ; 0.054</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.204</td>
<td>0.204</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>0.241</td>
<td>-</td>
<td>-</td>
<td>0.439 ; 0.120</td>
<td>-</td>
<td>-</td>
<td>0.329</td>
<td>0.195 ; 0.060</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.545</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.532</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>0.290 ; 0.099</td>
<td>-</td>
<td>-</td>
<td>0.164</td>
<td>0.131 ; 0.066</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.345 ; 0.149</td>
<td>-</td>
<td>-</td>
<td>0.298 ; 0.116</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>0.998 ; 0.090</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.441 ; 0.139</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.327</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>0.109 ; 0.085</td>
<td>0.581</td>
<td>0.759 ; 0.478</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.158</td>
<td>-</td>
</tr>
</tbody>
</table>

When looking at their content, clusters 1 and 4 immediately distinguish themselves from the rest. Cluster 1 has a mean value of 0.998 for the ICMP Rate, being it the cluster with the biggest dimension (it contains 375 flows); and cluster 4 has the number of destination ports and number of SMTP and IRC connections at its highest value possible. However, this is not considered an alarming behavior, because even though the value for ICMP Rate in indeed at a very high value, no other feature in that cluster was showing a high value; as for cluster 4, throughout the whole evaluation of the system, that there is always one cluster with such characteristics, and we can infer that this cluster corresponds only to regular clients using email services. Apart from these two, clusters 3, 6, 9 and 10 also present an alarming pattern. All these clusters share high values...
for the number of different source ports, number of HTTP connection and also for the average packet sizes. Apart from these features, clusters 3 and 9 also present a high value for the average packet size, which adds up to a suspicious pattern. Such patterns may be attributed to a Denial of Service (DoS) attack, as each host is send a great amount of packets from many different ports, all direct to the port 80 (or port 8080, in some cases), with an high average packet size. This is the case of the DoS HTTP Flood attack, discussed in Section 2.3.2. However, this is an attack that is easily identifiable by inspecting its payload, and this flow-based approach does no allow us to perform such an analysis, being these features our only way to hint the presence of such an attack.

The clustering content for the next day, Tuesday, may be found in Table 5.5.

<table>
<thead>
<tr>
<th>Clusters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># Entries</td>
<td>1219</td>
<td>205</td>
<td>1</td>
<td>171</td>
<td>287</td>
<td>1</td>
<td>1</td>
<td>236</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Features</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.234</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.249 ; 0.111</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.597</td>
<td>-</td>
<td>-</td>
<td>0.624 ; 0.275</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.436 ; 0.312</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.807</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.107 ; 0.180</td>
</tr>
<tr>
<td>13</td>
<td>0.997 ; 0.015</td>
<td>-</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.370</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.153 ; 0.190</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>0.148 ; 0.183</td>
</tr>
</tbody>
</table>

Table 5.5: Tuesday clustering content

We can see right away that there are multiple clusters with a very high value for the ICMP Rate, namely clusters 1, 3, 4, 5, 8 and 9. However, these features appear alone, i.e. its the only feature in these clusters that has a relevant high value, as no other features show up, apart from cluster 3 that also has a high value for the average packet size. However, this pattern does not arouse suspicion, and therefore will not be labeled as malicious, as it could possibly represent just a burst of traffic. From all these clusters, the one that grabs our attention is the tenth, which features a high value for the number of source ports, HTTP connections and average packet size. Not only it has these features, but it also has a high packet rate and total number of bytes. From what we have seen so far, this can only correspond to a malicious behavior, and therefore the content of this cluster was labeled as an attack.
Reaching the last day with intrusions, Thursday, the clusters can be observed in Table 5.6.

<table>
<thead>
<tr>
<th>Clusters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># Entries</td>
<td>14</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td>12</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Features</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 1 | - | - | - | - | - | - | - | - | - | - |
| 2 | - | 0.260 | - | - | 1.0 | - | 0.125 ; 0.65 | 0.157 ; 0.084 | 0.234 ; 0.001 | 0.216 |
| 3 | - | - | - | - | - | - | - | - | - | - |
| 4 | - | - | - | 0.614 ; 0.094 | - | - | 0.218 ; 0.068 | 0.443 ; 0.101 | - | 0.102 |
| 5 | - | - | - | - | - | - | - | - | - | - |
| 6 | - | - | - | 1.0 | - | - | - | - | - | - |
| 7 | - | 1.0 | - | - | - | - | - | - | - | - |
| 8 | - | - | - | - | - | - | - | - | - | - |
| 9 | - | - | - | - | - | - | - | - | - | - |
| 10 | - | - | - | - | - | - | - | - | - | - |
| 11 | - | - | - | - | - | - | - | - | 0.312 ; 0.056 | - | 0.127 |
| 12 | - | - | - | - | - | - | - | - | - | - |
| 13 | - | - | 0.939 ; 0.104 | - | - | 0.278 ; 0.083 | - | - | - | - |
| 14 | - | - | - | - | - | - | - | - | - | - |
| 15 | - | - | - | - | - | - | - | - | - | - |
| 16 | - | - | - | - | - | - | - | - | - | - |

Table 5.6: Thursday clustering content

Just like when we analyzed Saturday, a cluster is found to have an absolute maximum value for the number of SSH connection alongside with a high value of number of different source ports, thus indicating us the presence of a Brute-Force SSH attack. Also, clusters 7, 8 and 10 have high values for the number of different source ports, number of HTTP connections and also a high average packet size, also possibly indicating the presence of an attack. Therefore these three clusters were also labeled as malicious.

5.1.2 Supervised Cluster Classification

Parallel to this daily analysis, the system may also autonomously identify malicious activities using the SVM, which predicts the results based on its training. Before the system is able to classify data it is needed, at least, labeled data from the first day, which results from the manual intervention described in the previous section. From this day on, the classifier is able to produce results on its own, and these results may be refined with every iteration of the system (for the purposes of this work, an iteration corresponds to the period of one day), by training the system again, as new patterns are identified and manually labeled. As the purpose of this work is not to identify specific attacks, one single class was considered when classifying the data. As each day represents a different attack, the data available in not sufficient to train a system able to differentiate attacks from each other, and therefore the one class being considered is attributed to being or not a malicious activity.

The supervised learning algorithm was trained for the first day with data from the analysis for Saturday, as seen in Section 5.1.1. When asked to predict the results for that same day the result was accurate, the SVM correctly identified the malicious flow. However, when trying to
predict the results for Sunday, the SVM did not found any sort of malicious activity. This is due to the fact that the system’s only knowledge to that point was the intrusion seen during Saturday, which does not give sufficient information to the system to detect other attacks. After training the system once again with the analysis done also in the previous section, the SVM was now able to identify the malicious activities, although it could not identify them all. This same behavior was found when classifying the remainder of dataset throughout the rest of the days. Table 5.7 describes this analysis in further detail.

The flows that were classified as positive correspond to those whose were perpetrating the attack with greater intensity, i.e. producing large volumes of traffic, whereas the remainder of the attacks were not successfully identified. Also, on Wednesday, the system misclassified 5 flows as being malicious, when the traffic relative to Wednesday is all normal, intrusion-free traffic. This is due to the fact that Wednesday was one of the day that had the largest amount of traffic, and therefore the flows that belong to it also produced higher features values, leading to it being perceived as malicious and therefore raising false positives.

### Table 5.7: Classification Scores

<table>
<thead>
<tr>
<th></th>
<th>Saturday</th>
<th>Sunday</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>True Positive</strong></td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>False Positive</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td><strong>False Negative</strong></td>
<td>0</td>
<td>17</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

5.1.3 Results Validation

As it was said previously, a database is maintained, which holds information for the ground truth validation of the dataset. This way, it is possible to validate the results of this system. Each of these clusters is composed of flows which are identified by a unique ID. With this ID, we are able to traceback each ID to its IP address, which is stored in the database, and this way we are able to identify the malicious hosts.

When looking at the first day, Saturday, it was found that there was indeed a Brute-Force SSH attack, and that attack only. Therefore, for this case, the system was shown to be accurate. Analyzing Sunday, it was an intrusion from inside the network, therefore gaining access to a group of hosts and later on attacking the network from these compromised hosts. The clusters that we had classified as malicious were indeed correctly labeled, but there was one cluster that did not have high feature values, still being an attack. This cluster was cluster 5, which contained 7 intrusions, out of 20 entries. When it comes to Monday, we found that this was a DoS attack. The system was able to correctly identify clusters 6, 9 and 10, but cluster 3 was misclassified, as it did not correspond to any malicious activity at all. As for Tuesday, the attack that was taking...
place was a Distributed Denial of Service (DDoS) IRC Botnet attack, which is a botnet that takes control of its bots through IRC C&C servers, and uses them as a third party to induce DDoS attacks. On this day, the system was only able to correctly identify one malicious flow (only when manually analyzing the cluster, this SVM was not able to detect it), as the rest of them were not discovered. The remainder of the malicious flows were able to mask themselves among clusters that did not have high feature values, therefore making them indistinguishable, especially having this clusters a very high number of entries. Moreover, although this attack was an IRC Botnet, the values for the number of IRC communications remained undetectable, once again proving that the majority of the attacks was able to mask themselves. At last, Thursday, just as Saturday, had a Brute-Force SSH attack, which as correctly identified. However, in this day, the system led was to think that there were also two other clusters that also corresponded to an attack, and in fact they were not.

The tables shown in this evaluation were all for the source aggregation shown, as it was found that the flow direction of all the attacks was either Local to Remote (L2R) or Local to Local (L2L). This way, all the attacks are identifiable by tracking the Source IP addresses, and not the destination IP addresses, since we aim to find the malicious hosts; by analyzing the destination aggregation key, in this case, we would find the victims of the attacks, rather than the attackers, which is not the goal of this work. A general view of the of the system’s detection results can be seen in Table 5.8

The reason that some attack were not identified, is that this system’s focus is on large volumetric attacks, i.e. attacks that occur in large volumes, that exhaust the bandwidth of a network, and with feature values that tend to inflate. In the case of the DDoS attack, the system was only able to identify the one host that had the largest attack volume, as the rest of the host were producing a silent attack, that the system was not able to detect.

### 5.2 Vodafone Data Analysis

For a set of reasons, we were not able to analyse real data corresponding to several days. The real data was obtained in the scenario of Figure 3.2 and corresponds to part of the morning of a day in October 2016. We collected around 1GB of data, which corresponded to around 4.6

<table>
<thead>
<tr>
<th>Problematic Cluster #</th>
<th>True Positives</th>
<th>False Positives</th>
<th>False Negatives</th>
<th>Total Number of Flows</th>
<th>Ground Truth Attack Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>46</td>
<td>Brute-Force SSH</td>
</tr>
<tr>
<td>Sunday</td>
<td>2, 3, 4, 6, 7, 9</td>
<td>3</td>
<td>1</td>
<td>17</td>
<td>44</td>
</tr>
<tr>
<td>Monday</td>
<td>3, 6, 9, 10</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>421</td>
</tr>
<tr>
<td>Tuesday</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>213</td>
</tr>
<tr>
<td>Thursday</td>
<td>2, 7, 8, 10</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 5.8: Results validation
Unlike the ISCX dataset, this real data is not labelled, i.e., there is no ground truth on existing attacks. Therefore, we have analyzed both source and destination keys. Moreover, the number of clusters generated for each of the aggregation keys was 30 instead of 10 of the ISCX dataset. The reason for this was that the volume of the data was much than immensely bigger than the previously analyzed data, and so the number of clusters could have not been the same. After running the system with several different values, we found that 30 clusters was a good value for the number of clusters.

For both source and destination aggregation keys, the clustering was performed with the K-Means and Mini Batch K-Means algorithms. By looking at Table 5.9, Table 5.10, Table 5.11 and Table 5.12, we can see that although the K-Means algorithm has a bigger computational complexity and slower performance, it provides more detail on the data. It allows to unveil patterns that were hidden when using Mini Batch K-Means, and so we will focus our analysis on the clusters generated by the K-Means algorithm.

By looking the source aggregation key clustering content (Table 5.10), we see that clusters 13, 15, 17, 21 and 30 are those who present the higher feature values. Cluster 13 presents a high number of different source ports, as well as a high number of total bytes sent. However, such pattern was not found to be suspicious, as the number of source ports itself does not represent an alarming network trait, as opposed to the number of destination ports, and no address found in this cluster was found to be in any IP blacklist. When analyzing cluster 15, we see that is presents a high connectivity to various users, under various ports, receiving communication on an IRC port, and communicating through HTTP, with a high number of packets sent, as well as a high number of bytes. This leads us to assume that this machine is either a major spammer, or it could be a DoS attack, given its traffic pattern, and it was thus labeled as being a malicious host! Moving on to cluster 17, it was found to have a high number of SSH communications alone, which could represent a Brute-Force SSH attack, in just like had observed in the previous section, thus also being labeled as malicious hosts. Cluster 17 shows a high number of IRC (which, as we discussed earlier, is a way of botnet’s C&C communications) communications, alongside with an high average packet size. This feature distribution led us to consider that this could a Botnet communicating, and thus labeling it as malicious hosts. At last, cluster 30 presents a high number of SMTP communications, but when analyzing its IP addresses, we found that these were only mail servers communication, and we found no harm in it. Prior to this analysis, all of the IP addresses present in the malicious clusters, were found to be present in several blacklist, thus confirming our suspicion.
When looking at the destination aggregation key clustering content (Table 5.12), we see that clusters 16, 20, 22, 25 and 29 are those whose features are the most alarming. Analyzing cluster 16, we see that it has a feature distribution that is similar to what we had understood as a DoS HTTP Flood attack when analyzing the ISCX data, except that this cluster is missing a high value for the number of HTTP connections. Therefore, this could also represent a DoS attack, but directed to other applications, e.g. DNS. We cannot be sure of this attack, because none of the monitored ports are presenting high values, and so we can’t infer anything more about it. Cluster 20 presents a high number of different source IP addresses, destination ports and number of bytes. Because these flows do not have a high average packet size, it could possibly indicate that this a network scan, as these flow contacted many different port of many different IP addresses, resulting in a high value of bytes sent throughout this process. Cluster 22, on the other hand, presents a feature distribution that is similar to what had previously perceived as a DoS attack: it has a high number of different source IP addresses, number of source ports, number of HTTP connections, and a number of bytes sent. However, it still lacks a high value for the average packet size. Therefore, this may be, just like cluster 20, a network scan, but this time directed to the HTTP application, i.e. it may be a probing of a website in order to locate some vulnerability, for example. Cluster 25 presents a high number of different IP addresses, average packet sizes and number of bytes sent. These features alone do not seem to correspond to a malicious behavior, as they may be interpreted as a burst of traffic. At last, cluster 29 hold a have number of source IP addresses, number of destination ports, number of source HTTP connections, average packet sizes and number of bytes sent. This pattern is very similar to what we have seen for the DDoS IRC botnet attacks, expect for the number of IRC connections. Therefore, this may also correspond to infected hosts that are being used as a third party for attacks, but contacting its botmaster through a C&C server other than an IRC, or they could be victims of an attacker who is using spoofed IP addresses to use them as a third party.
Clusters

<table>
<thead>
<tr>
<th>/one.prop /two.prop /three.prop /four.prop /five.prop /six.prop /seven.prop /eight.prop /nine.prop /one.prop/zero.prop /one.prop/three.prop /one.prop/four.prop /one.prop/five.prop</th>
</tr>
</thead>
<tbody>
<tr>
<td>/seven.prop /eight.prop /one.prop/zero.prop /one.prop/three.prop /one.prop/four.prop /one.prop/five.prop /six.prop/two.prop/six.prop /two.prop/seven.prop /two.prop/eight.prop /two.prop/nine.prop /three.prop/zero.prop /three.prop/three.prop</td>
</tr>
<tr>
<td>/seven.prop /eight.prop /one.prop/zero.prop /one.prop/three.prop /one.prop/four.prop /one.prop/five.prop /six.prop/two.prop/six.prop /two.prop/seven.prop /two.prop/eight.prop /two.prop/nine.prop /three.prop/zero.prop /three.prop/three.prop</td>
</tr>
<tr>
<td>/seven.prop /eight.prop /one.prop/zero.prop /one.prop/three.prop /one.prop/four.prop /one.prop/five.prop /six.prop/two.prop/six.prop /two.prop/seven.prop /two.prop/eight.prop /two.prop/nine.prop /three.prop/zero.prop /three.prop/three.prop</td>
</tr>
<tr>
<td>/seven.prop /eight.prop /one.prop/zero.prop /one.prop/three.prop /one.prop/four.prop /one.prop/five.prop /six.prop/two.prop/six.prop /two.prop/seven.prop /two.prop/eight.prop /two.prop/nine.prop /three.prop/zero.prop /three.prop/three.prop</td>
</tr>
<tr>
<td>/seven.prop /eight.prop /one.prop/zero.prop /one.prop/three.prop /one.prop/four.prop /one.prop/five.prop /six.prop/two.prop/six.prop /two.prop/seven.prop /two.prop/eight.prop /two.prop/nine.prop /three.prop/zero.prop /three.prop/three.prop</td>
</tr>
</tbody>
</table>

Features

<table>
<thead>
<tr>
<th>/five.prop /nine.prop /five.prop/eight.prop /four.prop</th>
</tr>
</thead>
<tbody>
<tr>
<td>/five.prop /nine.prop /five.prop/eight.prop /four.prop</td>
</tr>
<tr>
<td>/five.prop /nine.prop /five.prop/eight.prop /four.prop</td>
</tr>
<tr>
<td>/five.prop /nine.prop /five.prop/eight.prop /four.prop</td>
</tr>
<tr>
<td>/five.prop /nine.prop /five.prop/eight.prop /four.prop</td>
</tr>
</tbody>
</table>

Table 5.9: Source key clustering content of a morning of October, 2016 using Mini Batch K-Means

| Clusters | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| /Flows   | 1050 | 5171 | 6485 | 13395 | 529 | 1970 | 1220 | 21507 | 8523 | 5722 | 1498 | 6586 | 10 | 5693 | 7 | 924 | 5 | 5651 | 1041 | 1876 | 12 | 101 | 13 | 2211 | 2264 | 8395 | 10 | 21507 | 16341 | 38 |
| Features | - | - | - | - | - | - | - | - | - | - | - | - | 1.0 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 0.237 | 0.158 |

Table 5.10: Source key clustering content of a morning of October, 2016 using K-Means
Table 5.11: Destination key clustering content of a morning of October, 2016 using Mini Batch K-Means

| Clusters | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|----------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Features| 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 | 1040 | 1041 | 1042 | 1043 | 1044 |
| Distance| 0.391 | 0.019 | 0.115 | 0.036 | 0.218 | 0.031 | 0.189 | 0.256 | 0.199 | 0.248 | 0.179 | 0.189 | 0.081 | 0.338 | 0.033 | 0.448 | 0.030 | 0.751 | 0.053 | 0.316 | 0.118 | 0.248 | 0.030 | 0.036 | 0.004 | 0.003 | 0.001 | 0.003 |

Table 5.12: Destination key clustering content of a morning of October, 2016 using K-Means
<table>
<thead>
<tr>
<th>Cluster #</th>
<th>Aggregation Key</th>
<th>Highlighted Features</th>
<th>Type of Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Source</td>
<td>1, 3, 5, 8, 11, 15</td>
<td>Spam / DoS</td>
</tr>
<tr>
<td>16</td>
<td>Destination</td>
<td>1, 3, 6</td>
<td>DoS</td>
</tr>
<tr>
<td>17</td>
<td>Source</td>
<td>10</td>
<td>Brute-Force SSH</td>
</tr>
<tr>
<td>20</td>
<td>Destination</td>
<td>1, 2, 15</td>
<td>Network Scan</td>
</tr>
<tr>
<td>21</td>
<td>Source</td>
<td>9, 16</td>
<td>Botnet Communication</td>
</tr>
<tr>
<td>22</td>
<td>Destination</td>
<td>1, 3, 8, 15</td>
<td>Web Application Probing</td>
</tr>
<tr>
<td>27</td>
<td>Source</td>
<td>1, 2, 5, 8, 11, 15</td>
<td>DDoS IRC Botnet</td>
</tr>
<tr>
<td>29</td>
<td>Destination</td>
<td>1, 2, 4, 11, 15</td>
<td>DDoS Botnet</td>
</tr>
</tbody>
</table>

Table 5.13: Vodafone data analysis results

Table 5.13 summarizes the information regarding the intrusions detected throughout this analysis. For the feature numbers, refer to Table 5.1.
Chapter 6

Conclusions

The main goal of this work was to present a system that would be capable of detecting malicious hosts without requiring previous knowledge about what we were looking for or clean training data. The solution features a combination of data mining techniques for the feature extraction, and also machine learning techniques, that allowed to analyze the data. It requires no specific training for the system to detect malicious behavior, except for the inevitable human intervention in a first run of the system. Furthermore, the solution was designed in a way that it allowed for the detection of large volumetric attacks, attacks that would produce very high feature values, so that the hosts producing such traffic, were easily distinguished from the remainder of the traffic, and producing patterns that allowed us to detect such an intrusion. However, as shown, this approach allows us only to detect a small portion of the attacks that may be going through a network, as many of them are done almost silently, making this approach infeasible. Still, the attacks that were perpetrated in a big scale were correctly detected and distinguished from the regular traffic, as the system detected both DoS and a portion of DDoS attacks, Brute-Force SSH attacks, and was also able to detect part of an intrusion from the inside of a network. When analyzing the data from Vodafone Portugal, although we do not have ground truth to perform a validation of the system's performance, the NIDS was able to unveil some interesting pattern. Even with the great amount of data, it was able to isolate a small number of flows that presented alarming patterns, that ultimately led to identifying them as being malicious.

In this data, the system was able to locate a machine producing major amounts of traffic, leading us to believe that it was either a major spammer or perpetrating a DoS attack, small DoS attacks, a few network scans, and indentify the perpetrators of these attacks, thus accomplishing our goal of detecting malicious hosts. Still, it does not allow to identify every network intrusion event, as some of them are performed with low intensity, thus being able to evade the system.
Bibliography


